Manuel PostGIS 3.7.0rc1

Table des matières

1 Introduction **1**
1.1 What Is PostGIS? . . . 1
1.2 What Is in the PostGIS Distribution? . . . 1
1.3 The PostGIS Project Family . . . 1
1.4 How to Use This Manual . . . 2
1.5 Standards Support . . . 2
1.6 Project and Community . . . 3

2 Installation de PostGIS **4**
2.1 Version courte . . . 4
2.2 Compilation et installation depuis les sources . . . 4
2.2.1 Obtenir les Sources . . . 5
2.2.2 Pré requis à l'installation . . . 5
2.2.3 Configuration de la compilation . . . 6
2.2.4 Compiler . . . 7
2.2.5 Compiler les Extensions PostGIS et les déployer . . . 8
2.2.6 Tests . . . 10
2.2.7 Installation . . . 13
2.3 Problèmes courants pendant l'installation . . . 13

3 Administration de PostGIS **15**
3.1 Optimisation des performances . . . 15
3.1.1 Démarrage . . . 15
3.1.2 Temps d'exécution . . . 16
3.2 Configurer la prise en charge du raster . . . 16
3.3 Création de bases de données spatiales . . . 17
3.3.1 Base de données spatiale en utilisant EXTENSION . . . 17
3.3.2 Base de données spatiale sans utiliser EXTENSION (non recommandé) . . . 17
3.4 Mise à jour des bases de données spatiales . . . 18
3.4.1 Mise à niveau progressive (Soft upgrade) . . . 18
3.4.1.1 Mise à niveau progressive (Soft upgrade) utilisant des extensions . . . 18
3.4.1.2 Mise à niveau progressive (Soft Upgrade) sans extensions . . . 20
3.4.1.3 Mise à niveau complète (Hard upgrade) . . . 20

4 Gestion des données **22**

4.1 Modèle de données spatiales . . . 22
4.1.1 Géométrie OGC . . . 22
4.1.1.1 Point . . . 23
4.1.1.2 LineString . . . 23
4.1.1.3 LinearRing . . . 23
4.1.1.4 Polygon . . . 23
4.1.1.5 MultiPoint . . . 23
4.1.1.6 MultiLineString . . . 24
4.1.1.7 MultiPolygon . . . 24
4.1.1.8 GeometryCollection . . . 24
4.1.1.9 PolyhedralSurface . . . 24
4.1.1.10 Triangle . . . 24
4.1.1.11 TIN . . . 25
4.1.2 SQL/MM Part 3 - Courbes . . . 25
4.1.2.1 CircularString . . . 25
4.1.2.2 CompoundCurve . . . 25
4.1.2.3 CurvePolygon . . . 25
4.1.2.4 MultiCurve . . . 26
4.1.2.5 MultiSurface . . . 26
4.1.3 WKT et WKB . . . 26
4.2 Type de données Geometry . . . 27
4.2.1 PostGIS EWKB et EWKT . . . 28
4.3 Type de données Geography . . . 29
4.3.1 Création de tables géographiques . . . 30
4.3.2 Utilisation des tables géographiques . . . 31
4.3.3 Quand utiliser le type de données Geography . . . 33
4.3.4 FAQ Geography avancée . . . 33
4.4 Validation de la géométrie . . . 34
4.4.1 Géométrie simple . . . 34
4.4.2 Géométrie valide . . . 36
4.4.3 Gestion de la validité . . . 40
4.5 Systèmes de référence spatiale . . . 42
4.5.1 Table SPATIAL_REF_SYS . . . 43
4.5.2 Systèmes de référence spatiale définis par l'utilisateur . . . 44
4.6 Tables spatiales . . . 44
4.6.1 Créer une table spatiale . . . 44
4.6.2 Vue GEOMETRY_COLUMNS . . . 45
4.6.3 Enregistrement manuel des colonnes de géométrie . . . 46

4.7 Chargement des données spatiales . . . 49
4.7.1 Utilisation de SQL pour charger des données . . . 49
4.7.2 Utilisation de l'utilitaire qui permet de charger des fichiers Shapefile . . . 49
4.8 Extraction de données spatiales . . . 51
4.8.1 Utiliser SQL pour extraire des données . . . 51
4.8.2 Utilisation de Shapefile Dumper . . . 52
4.9 Index spatiaux . . . 53
4.9.1 Index GiST . . . 53
4.9.2 Index BRIN . . . 54
4.9.3 Index SP-GiST . . . 56
4.9.4 Optimisation de l'utilisation de l'index . . . 57

5 Requêtes spatiales **58**
5.1 Déterminer les relations spatiales . . . 58
5.1.1 Modèle à 9 intersections dimensionnellement étendu . . . 58
5.1.2 Relations spatiales nommées . . . 62
5.1.3 Relations spatiales générales . . . 62
5.2 Utilisation des index spatiaux . . . 64
5.3 Exemples de SQL spatial . . . 65

6 Conseils sur les performances **68**
6.1 Petites tables de grandes géométries . . . 68
6.1.1 Description du problème . . . 68
6.1.2 Solutions de contournement . . . 68
6.2 CLUSTER d'index géométriques . . . 69
6.3 Eviter les conversions de dimension . . . 69

7 Référence PostGIS **70**
7.1 Types de données PostGIS Geometry/Geography/Box . . . 70
7.1.1 box2d . . . 70
7.1.2 box3d . . . 71
7.1.3 geometry . . . 71
7.1.4 geometry_dump . . . 72
7.1.5 geography . . . 72
7.2 Fonctions de gestion des tables . . . 72
7.2.1 AddGeometryColumn . . . 72
7.2.2 DropGeometryColumn . . . 75
7.2.3 DropGeometryTable . . . 76
7.2.4 Find_SRID . . . 77
7.2.5 Populate_Geometry_Columns . . . 77

7.2.6 UpdateGeometrySRID . . . 79
7.3 Constructeurs de géométries . . . 80
7.3.1 ST_Collect . . . 80
7.3.2 ST_LineFromMultiPoint . . . 84
7.3.3 ST_MakeEnvelope . . . 85
7.3.4 ST_MakeLine . . . 86
7.3.5 ST_MakePoint . . . 90
7.3.6 ST_MakePointM . . . 92
7.3.7 ST_MakePolygon . . . 94
7.3.8 ST_Point . . . 98
7.3.9 ST_PointZ . . . 99
7.3.10 ST_PointM . . . 100
7.3.11 ST_PointZM . . . 100
7.3.12 ST_Polygon . . . 101
7.3.13 ST_TileEnvelope . . . 103
7.3.14 ST_HexagonGrid . . . 105
7.3.15 ST_Hexagon . . . 110
7.3.16 ST_SquareGrid . . . 111
7.3.17 ST_Square . . . 113
7.3.18 ST_Letters . . . 114
7.3.19 ST_MakeNurbsCurve . . . 116
7.4 Fonctions d'accès aux géométries . . . 119
7.4.1 GeometryType . . . 119
7.4.2 ST_Boundary . . . 121
7.4.3 ST_BoundingDiagonal . . . 127
7.4.4 ST_CoordDim . . . 128
7.4.5 ST_Dimension . . . 129
7.4.6 ST_Dump . . . 131
7.4.7 ST_DumpPoints . . . 134
7.4.8 ST_DumpSegments . . . 139
7.4.9 ST_DumpRings . . . 142
7.4.10 ST_EndPoint . . . 144
7.4.11 ST_Envelope . . . 148
7.4.12 ST_ExteriorRing . . . 152
7.4.13 ST_GeometryN . . . 155
7.4.14 ST_GeometryType . . . 159
7.4.15 ST_HasArc . . . 160
7.4.16 ST_InteriorRingN . . . 161
7.4.17 ST_NumCurves . . . 163

7.4.18 ST_CurveN . . . 164
7.4.19 ST_IsClosed . . . 165
7.4.20 ST_IsCollection . . . 170
7.4.21 ST_IsEmpty . . . 172
7.4.22 ST_IsPolygonCCW . . . 174
7.4.23 ST_IsPolygonCW . . . 175
7.4.24 ST_IsRing . . . 176
7.4.25 ST_IsSimple . . . 178
7.4.26 ST_M . . . 180
7.4.27 ST_MemSize . . . 181
7.4.28 ST_NDims . . . 182
7.4.29 ST_NPoints . . . 183
7.4.30 ST_NRings . . . 184
7.4.31 ST_NumGeometries . . . 185
7.4.32 ST_NumInteriorRings . . . 186
7.4.33 ST_NumInteriorRing . . . 187
7.4.34 ST_NumPatches . . . 187
7.4.35 ST_NumPoints . . . 188
7.4.36 ST_PatchN . . . 189
7.4.37 ST_PointN . . . 190
7.4.38 ST_Points . . . 193
7.4.39 ST_StartPoint . . . 194
7.4.40 ST_StartM . . . 199
7.4.41 ST_EndM . . . 202
7.4.42 ST_SetStartM . . . 205
7.4.43 ST_SetEndM . . . 208
7.4.44 ST_Summary . . . 211
7.4.45 ST_X . . . 212
7.4.46 ST_Y . . . 213
7.4.47 ST_Z . . . 214
7.4.48 ST_ZMFlag . . . 215
7.4.49 ST_HasZ . . . 216
7.4.50 ST_HasM . . . 217
7.4.51 ST_ControlPoints . . . 219
7.4.52 ST_Degree . . . 220
7.4.53 ST_Weights . . . 222
7.4.54 ST_Knots . . . 223
7.4.55 ST_NumControlPoints . . . 225
7.4.56 ST_NurbsCurveIsRational . . . 227

7.4.57 ST_IsNurbsCurve . . . 229
7.4.58 ST_Evaluate . . . 231
7.4.59 ST_NurbsToLineString . . . 232
7.5 Éditeurs de géométrie . . . 234
7.5.1 ST_AddPoint . . . 234
7.5.2 ST_CollectionExtract . . . 235
7.5.3 ST_CollectionHomogenize . . . 238
7.5.4 ST_CurveToLine . . . 242
7.5.5 ST_Scroll . . . 246
7.5.6 ST_FlipCoordinates . . . 247
7.5.7 ST_Force2D . . . 248
7.5.8 ST_Force3D . . . 250
7.5.9 ST_Force3DZ . . . 252
7.5.10 ST_Force3DM . . . 254
7.5.11 ST_Force4D . . . 256
7.5.12 ST_ForceCollection . . . 258
7.5.13 ST_ForceCurve . . . 261
7.5.14 ST_ForcePolygonCCW . . . 262
7.5.15 ST_ForcePolygonCW . . . 263
7.5.16 ST_ForceSFS . . . 265
7.5.17 ST_ForceRHR . . . 266
7.5.18 ST_LineExtend . . . 267
7.5.19 ST_LineToCurve . . . 268
7.5.20 ST_Multi . . . 270
7.5.21 ST_Normalize . . . 271
7.5.22 ST_Project . . . 272
7.5.23 ST_QuantizeCoordinates . . . 274
7.5.24 ST_RemovePoint . . . 276
7.5.25 ST_RemoveRepeatedPoints . . . 276
7.5.26 ST_RemoveIrrelevantPointsForView . . . 280
7.5.27 ST_RemoveSmallParts . . . 284
7.5.28 ST_Reverse . . . 286
7.5.29 ST_Segmentize . . . 287
7.5.30 ST_SetPoint . . . 290
7.5.31 ST_ShiftLongitude . . . 293
7.5.32 ST_WrapX . . . 295
7.5.33 ST_SnapToGrid . . . 296
7.5.34 ST_Snap . . . 299
7.5.35 ST_SwapOrdinates . . . 303

7.6 Validation de la géométrie . . . 304
7.6.1 ST_IsValid . . . 304
7.6.2 ST_IsValidDetail . . . 307
7.6.3 ST_IsValidReason . . . 309
7.6.4 ST_MakeValid . . . 311
7.7 Fonctions des systèmes de référence spatiale . . . 314
7.7.1 ST_InverseTransformPipeline . . . 314
7.7.2 ST_SetSRID . . . 316
7.7.3 ST_SRID . . . 318
7.7.4 ST_Transform . . . 319
7.7.5 ST_TransformPipeline . . . 323
7.7.6 postgis_srs_codes . . . 326
7.7.7 postgis_srs . . . 327
7.7.8 postgis_srs_all . . . 328
7.7.9 postgis_srs_search . . . 328
7.8 Import de géométrie . . . 330
7.8.1 Well-Known Text (WKT) . . . 330
7.8.1.1 ST_BdPolyFromText . . . 330
7.8.1.2 ST_BdMPolyFromText . . . 330
7.8.1.3 ST_GeogFromText . . . 331
7.8.1.4 ST_GeographyFromText . . . 332
7.8.1.5 ST_GeomCollFromText . . . 332
7.8.1.6 ST_GeomFromEWKT . . . 333
7.8.1.7 ST_GeomFromMARC21 . . . 336
7.8.1.8 ST_GeometryFromText . . . 340
7.8.1.9 ST_GeomFromText . . . 340
7.8.1.10 ST_LineFromText . . . 345
7.8.1.11 ST_MLineFromText . . . 346
7.8.1.12 ST_MPointFromText . . . 347
7.8.1.13 ST_MPolyFromText . . . 348
7.8.1.14 ST_PointFromText . . . 350
7.8.1.15 ST_PolygonFromText . . . 352
7.8.1.16 ST_WKTToSQL . . . 354
7.8.2 Well-Known Binary (WKB) . . . 354
7.8.2.1 ST_GeogFromWKB . . . 354
7.8.2.2 ST_GeomFromEWKB . . . 355
7.8.2.3 ST_GeomFromWKB . . . 357
7.8.2.4 ST_LineFromWKB . . . 359
7.8.2.5 ST_LinestringFromWKB . . . 360

7.8.2.6 ST_PointFromWKB . . . 362
7.8.2.7 ST_WKBToSQL . . . 363
7.8.3 Autres formats . . . 363
7.8.3.1 ST_Box2dFromGeoHash . . . 363
7.8.3.2 ST_GeomFromGeoHash . . . 364
7.8.3.3 ST_GeomFromGML . . . 365
7.8.3.4 ST_GeomFromGeoJSON . . . 370
7.8.3.5 ST_GeomFromKML . . . 372
7.8.3.6 ST_GeomFromTWKB . . . 373
7.8.3.7 ST_GMLToSQL . . . 375
7.8.3.8 ST_LineFromEncodedPolyline . . . 375
7.8.3.9 ST_PointFromGeoHash . . . 377
7.8.3.10 ST_FromFlatGeobufToTable . . . 379
7.8.3.11 ST_FromFlatGeobuf . . . 380
7.9 Export de géométrie . . . 380
7.9.1 Well-Known Text (WKT) . . . 380
7.9.1.1 ST_AsEWKT . . . 380
7.9.1.2 ST_AsText . . . 382
7.9.2 Well-Known Binary (WKB) . . . 384
7.9.2.1 ST_AsBinary . . . 384
7.9.2.2 ST_AsEWKB . . . 386
7.9.2.3 ST_AsHEXEWKB . . . 387
7.9.3 Autres formats . . . 388
7.9.3.1 ST_AsEncodedPolyline . . . 388
7.9.3.2 ST_AsFlatGeobuf . . . 390
7.9.3.3 ST_AsGeobuf . . . 390
7.9.3.4 ST_AsGeoJSON . . . 391
7.9.3.5 ST_AsGML . . . 393
7.9.3.6 ST_AsKML . . . 396
7.9.3.7 ST_AsLatLonText . . . 398
7.9.3.8 ST_AsMARC21 . . . 400
7.9.3.9 ST_AsMVTGeom . . . 402
7.9.3.10 ST_AsMVT . . . 404
7.9.3.11 ST_AsSVG . . . 405
7.9.3.12 ST_AsTWKB . . . 408
7.9.3.13 ST_AsX3D . . . 409
7.9.3.14 ST_GeoHash . . . 413
7.10 Opérateurs . . . 415
7.10.1 Opérateurs de Bounding Box . . . 415

7.10.1.1 && . . . 415
7.10.1.2 &&(geometry,box2df) . . . 416
7.10.1.3 &&(box2df,geometry) . . . 417
7.10.1.4 &&(box2df,box2df) . . . 418
7.10.1.5 &&& . . . 418
7.10.1.6 &&&(geometry,gidx) . . . 421
7.10.1.7 &&&(gidx,geometry) . . . 422
7.10.1.8 &&&(gidx,gidx) . . . 423
7.10.1.9 &< . . . 424
7.10.1.10 &<| . . . 425
7.10.1.11 &> . . . 426
7.10.1.12 << . . . 428
7.10.1.13 <<| . . . 429
7.10.1.14 = . . . 430
7.10.1.15 >> . . . 433
7.10.1.16 @ . . . 435
7.10.1.17 @(geometry,box2df) . . . 436
7.10.1.18 @(box2df,geometry) . . . 437
7.10.1.19 @(box2df,box2df) . . . 438
7.10.1.20 |&> . . . 438
7.10.1.21 |>> . . . 440
7.10.1.22 ~ . . . 441
7.10.1.23 ~(geometry,box2df) . . . 443
7.10.1.24 ~(box2df,geometry) . . . 443
7.10.1.25 ~(box2df,box2df) . . . 444
7.10.1.26 ~= . . . 445
7.10.2 Opérateurs de distance . . . 446
7.10.2.1 <-> . . . 446
7.10.2.2 |=| . . . 448
7.10.2.3 <#> . . . 449
7.10.2.4 <<->> . . . 451
7.11 Relations spatiales . . . 451
7.11.1 Relations topologiques . . . 451
7.11.1.1 ST_3DIntersects . . . 451
7.11.1.2 ST_Contains . . . 454
7.11.1.3 ST_ContainsProperly . . . 465
7.11.1.4 ST_CoveredBy . . . 468
7.11.1.5 ST_Covers . . . 470
7.11.1.6 ST_Crosses . . . 472

7.11.1.7 ST_Disjoint . . . 475
7.11.1.8 ST_Equals . . . 478
7.11.1.9 ST_Intersects . . . 479
7.11.1.10 ST_LineCrossingDirection . . . 482
7.11.1.11 ST_OrderingEquals . . . 485
7.11.1.12 ST_Overlaps . . . 488
7.11.1.13 ST_Relate . . . 494
7.11.1.14 ST_RelateMatch . . . 497
7.11.1.15 ST_Touches . . . 499
7.11.1.16 ST_Within . . . 506
7.11.2 Relations de distance . . . 509
7.11.2.1 ST_3DDWithin . . . 509
7.11.2.2 ST_3DDFullyWithin . . . 510
7.11.2.3 ST_DFullyWithin . . . 511
7.11.2.4 ST_DWithin . . . 513
7.11.2.5 ST_PointInsideCircle . . . 515
7.12 Fonctions de mesure . . . 517
7.12.1 ST_Area . . . 517
7.12.2 ST_Azimuth . . . 520
7.12.3 ST_Angle . . . 522
7.12.4 ST_ClosestPoint . . . 525
7.12.5 ST_3DClosestPoint . . . 527
7.12.6 ST_Distance . . . 530
7.12.7 ST_3DDistance . . . 532
7.12.8 ST_DistanceSphere . . . 534
7.12.9 ST_DistanceSpheroid . . . 535
7.12.10 ST_FrechetDistance . . . 537
7.12.11 ST_HausdorffDistance . . . 538
7.12.12 ST_Length . . . 540
7.12.13 ST_Length2D . . . 543
7.12.14 ST_3DLength . . . 544
7.12.15 ST_LengthSpheroid . . . 545
7.12.16 ST_LongestLine . . . 547
7.12.17 ST_3DLongestLine . . . 550
7.12.18 ST_MaxDistance . . . 553
7.12.19 ST_3DMaxDistance . . . 556
7.12.20 ST_MinimumClearance . . . 556
7.12.21 ST_MinimumClearanceLine . . . 558
7.12.22 ST_Perimeter . . . 559

7.12.23 ST_Perimeter2D . . . 563
7.12.24 ST_3DPerimeter . . . 563
7.12.25 ST_ShortestLine . . . 565
7.12.26 ST_3DShortestLine . . . 566
7.13 Fonctions de superposition . . . 569
7.13.1 ST_ClipByBox2D . . . 569
7.13.2 ST_Difference . . . 570
7.13.3 ST_Intersection . . . 572
7.13.4 ST_MemUnion . . . 575
7.13.5 ST_Node . . . 576
7.13.6 ST_Split . . . 578
7.13.7 ST_Subdivide . . . 581
7.13.8 ST_SymDifference . . . 584
7.13.9 ST_UnaryUnion . . . 586
7.13.10 ST_Union . . . 587
7.14 Traitement des géométries . . . 592
7.14.1 ST_Buffer . . . 592
7.14.2 ST_BuildArea . . . 601
7.14.3 ST_Centroid . . . 604
7.14.4 ST_ChaikinSmoothing . . . 609
7.14.5 ST_CatmullRomSmoothing . . . 613
7.14.6 ST_ConcaveHull . . . 615
7.14.7 ST_ConvexHull . . . 619
7.14.8 ST_DelaunayTriangles . . . 621
7.14.9 ST_FilterByM . . . 624
7.14.10 ST_GeneratePoints . . . 626
7.14.11 ST_GeometricMedian . . . 627
7.14.12 ST_LineMerge . . . 628
7.14.13 ST_MaximumInscribedCircle . . . 633
7.14.14 ST_LargestEmptyCircle . . . 635
7.14.15 ST_MinimumBoundingCircle . . . 637
7.14.16 ST_MinimumBoundingRadius . . . 639
7.14.17 ST_OrientedEnvelope . . . 640
7.14.18 ST_OffsetCurve . . . 641
7.14.19 ST_PointOnSurface . . . 648
7.14.20 ST_Polygonize . . . 653
7.14.21 ST_ReducePrecision . . . 656
7.14.22 ST_SharedPaths . . . 660
7.14.23 ST_Simplify . . . 662

7.14.24 ST_SimplifyPreserveTopology . . . 665
7.14.25 ST_SimplifyPolygonHull . . . 668
7.14.26 ST_SimplifyVW . . . 671
7.14.27 ST_SetEffectiveArea . . . 674
7.14.28 ST_TriangulatePolygon . . . 676
7.14.29 ST_VoronoiLines . . . 679
7.14.30 ST_VoronoiPolygons . . . 680
7.15 Couvertures . . . 682
7.15.1 ST_CoverageInvalidEdges . . . 682
7.15.2 ST_CoverageClean . . . 683
7.15.3 ST_CoverageSimplify . . . 685
7.15.4 ST_CoverageEdges . . . 686
7.15.5 ST_CoverageUnion . . . 688
7.16 Transformations affines . . . 689
7.16.1 ST_Affine . . . 689
7.16.2 ST_Rotate . . . 692
7.16.3 ST_RotateX . . . 695
7.16.4 ST_RotateY . . . 696
7.16.5 ST_RotateZ . . . 697
7.16.6 ST_Scale . . . 699
7.16.7 ST_Translate . . . 703
7.16.8 ST_TransScale . . . 706
7.17 Fonctions de clustering . . . 708
7.17.1 ST_ClusterDBSCAN . . . 708
7.17.2 ST_ClusterIntersecting . . . 710
7.17.3 ST_ClusterIntersectingWin . . . 711
7.17.4 ST_ClusterRelateWin . . . 713
7.17.5 ST_MinimumSpanningTree . . . 714
7.17.6 ST_ClusterKMeans . . . 715
7.17.7 ST_ClusterWithin . . . 720
7.17.8 ST_ClusterWithinWin . . . 721
7.18 Fonctions des boîtes de délimitation . . . 723
7.18.1 Box2D . . . 723
7.18.2 Box3D . . . 725
7.18.3 ST_EstimatedExtent . . . 727
7.18.4 ST_Expand . . . 728
7.18.5 ST_Extent . . . 730
7.18.6 ST_3DExtent . . . 732
7.18.7 ST_MakeBox2D . . . 733

7.18.8 ST_3DMakeBox . . . 734
7.18.9 ST_XMax . . . 734
7.18.10 ST_XMin . . . 736
7.18.11 ST_YMax . . . 738
7.18.12 ST_YMin . . . 741
7.18.13 ST_ZMax . . . 743
7.18.14 ST_ZMin . . . 745
7.18.15 ST_XSize . . . 748
7.18.16 ST_YSize . . . 749
7.18.17 ST_ZSize . . . 751
7.18.18 ST_MMin . . . 752
7.18.19 ST_MMax . . . 753
7.18.20 ST_MSize . . . 753
7.19 Référencement linéaire . . . 754
7.19.1 ST_LineInterpolatePoint . . . 754
7.19.2 ST_3DLineInterpolatePoint . . . 758
7.19.3 ST_LineInterpolatePoints . . . 759
7.19.4 ST_LineLocatePoint . . . 760
7.19.5 ST_LineSubstring . . . 763
7.19.6 ST_LocateAlong . . . 769
7.19.7 ST_LocateBetween . . . 770
7.19.8 ST_LocateBetweenElevations . . . 772
7.19.9 ST_InterpolatePoint . . . 774
7.19.10 ST_3DInterpolatePoint . . . 775
7.19.11 ST_AddMeasure . . . 777
7.20 Fonctions de trajectoire . . . 780
7.20.1 ST_IsValidTrajectory . . . 780
7.20.2 ST_ClosestPointOfApproach . . . 781
7.20.3 ST_DistanceCPA . . . 783
7.20.4 ST_CPAWithin . . . 784
7.21 Fonctions de version . . . 785
7.21.1 PostGIS_Extensions_Upgrade . . . 785
7.21.2 PostGIS_Full_Version . . . 786
7.21.3 PostGIS_GEOS_Version . . . 787
7.21.4 PostGIS_GEOS_Compiled_Version . . . 787
7.21.5 PostGIS_Liblwgeom_Version . . . 788
7.21.6 PostGIS_LibXML_Version . . . 788
7.21.7 PostGIS_LibJSON_Version . . . 789
7.21.8 PostGIS_Lib_Build_Date . . . 790

7.21.9 PostGIS_Lib_Version . . . 790
7.21.10 PostGIS_PROJ_Version . . . 790
7.21.11 PostGIS_PROJ_Compiled_Version . . . 791
7.21.12 PostGIS_Wagyu_Version . . . 792
7.21.13 PostGIS_Scripts_Build_Date . . . 792
7.21.14 PostGIS_Scripts_Installed . . . 793
7.21.15 PostGIS_Scripts_Released . . . 793
7.21.16 PostGIS_Version . . . 794
7.22 Variables PostGIS GUC (Grand Unified Custom Variables) . . . 795
7.22.1 postgis.gdal_datapath . . . 795
7.22.2 postgis.gdal_enabled_drivers . . . 796
7.22.3 postgis.enable_outdb_rasters . . . 797
7.22.4 postgis.gdal_vsi_options . . . 798
7.22.5 postgis.gdal_cpl_debug . . . 799
7.23 Fonctions de dépannage . . . 799
7.23.1 PostGIS_AddBBox . . . 799
7.23.2 PostGIS_DropBBox . . . 800
7.23.3 PostGIS_HasBBox . . . 801

8 Référence des fonctions SFCGAL 802
8.1 Fonctions de gestion de SFCGAL . . . 802
8.1.1 postgis_sfcgal_version . . . 802
8.1.2 postgis_sfcgal_full_version . . . 802
8.2 Fonctions d'accès et de modifications SFCGAL . . . 803
8.2.1 CG_ForceLHR . . . 803
8.2.2 CG_IsPlanar . . . 804
8.2.3 CG_IsSolid . . . 804
8.2.4 CG_MakeSolid . . . 806
8.2.5 CG_Orientation . . . 806
8.2.6 CG_Area . . . 807
8.2.7 CG_3DArea . . . 808
8.2.8 CG_Volume . . . 809
8.3 Fonctions de traitement et de relation SFCGAL . . . 810
8.3.1 CG_Intersection . . . 810
8.3.2 CG_Intersects . . . 811
8.3.3 CG_3DIntersects . . . 813
8.3.4 CG_Difference . . . 814
8.3.5 CG_3DDifference . . . 816
8.3.6 CG_Distance . . . 817

8.3.7 CG_3DDistance . . . 818
8.3.8 CG_3DConvexHull . . . 819
8.3.9 CG_3DIntersection . . . 822
8.3.10 CG_Union . . . 825
8.3.11 CG_3DUnion . . . 826
8.3.12 CG_AlphaShape . . . 828
8.3.13 CG_ApproxConvexPartition . . . 831
8.3.14 CG_ApproximateMedialAxis . . . 832
8.3.15 CG_GenerateFlatRoof . . . 835
8.3.16 CG_GenerateHippedRoof . . . 836
8.3.17 CG_GenerateGableRoof . . . 837
8.3.18 CG_GenerateSkillionRoof . . . 838
8.3.19 CG_GenerateRoof . . . 840
8.3.20 CG_PolygonRepair . . . 841
8.3.21 CG_ConstrainedDelaunayTriangles . . . 843
8.3.22 CG_Extrude . . . 845
8.3.23 CG_ExtrudeStraightSkeleton . . . 847
8.3.24 CG_GreeneApproxConvexPartition . . . 848
8.3.25 CG_MinkowskiSum . . . 850
8.3.26 CG_OptimalAlphaShape . . . 852
8.3.27 CG_OptimalConvexPartition . . . 854
8.3.28 CG_StraightSkeleton . . . 855
8.3.29 CG_Tesselate . . . 857
8.3.30 CG_Triangulate . . . 860
8.3.31 CG_Visibility . . . 861
8.3.32 CG_YMonotonePartition . . . 863
8.3.33 CG_StraightSkeletonPartition . . . 864
8.3.34 CG_3DBuffer . . . 866
8.3.35 CG_Rotate . . . 869
8.3.36 CG_2DRotate . . . 870
8.3.37 CG_3DRotate . . . 871
8.3.38 CG_RotateX . . . 872
8.3.39 CG_RotateY . . . 873
8.3.40 CG_RotateZ . . . 874
8.3.41 CG_Scale . . . 875
8.3.42 CG_3DScale . . . 876
8.3.43 CG_3DScaleAroundCenter . . . 877
8.3.44 CG_Translate . . . 878
8.3.45 CG_3DTranslate . . . 879

8.3.46 CG_Simplify . . . 880
8.3.47 CG_3DAlphaWrapping . . . 884
8.3.48 CG_NurbsCurveInterpolate . . . 885
8.3.49 CG_NurbsCurveApproximate . . . 887
8.3.50 CG_NurbsCurveDerivative . . . 889
8.4 Deprecated SFCGAL Function Aliases . . . 892
8.4.1 ST_3DIntersection . . . 892
8.4.2 ST_3DDifference . . . 893
8.4.3 ST_3DUnion . . . 893
8.4.4 ST_Tesselate . . . 893
8.4.5 ST_3DArea . . . 894
8.4.6 ST_Extrude . . . 894
8.4.7 ST_ForceLHR . . . 895
8.4.8 ST_Orientation . . . 895
8.4.9 ST_MinkowskiSum . . . 895
8.4.10 ST_StraightSkeleton . . . 896
8.4.11 ST_ApproximateMedialAxis . . . 896
8.4.12 ST_IsPlanar . . . 897
8.4.13 ST_Volume . . . 897
8.4.14 ST_MakeSolid . . . 898
8.4.15 ST_IsSolid . . . 898
8.4.16 ST_ConstrainedDelaunayTriangles . . . 899
8.4.17 ST_3DConvexHull . . . 899
8.4.18 ST_AlphaShape . . . 899
8.4.19 ST_OptimalAlphaShape . . . 900

9 Topologie **901**
9.1 Building Faces from Linework . . . 901
9.2 Topology Primitive Tables . . . 903
9.3 Les types associés à "Topology" . . . 903
9.3.1 getfaceedges_returntype . . . 903
9.3.2 TopoGeometry . . . 903
9.3.3 validatetopology_returntype . . . 904
9.4 Domaines de topologie . . . 905
9.4.1 TopoElement . . . 905
9.4.2 TopoElementArray . . . 906
9.5 Gestion de la topologie et de TopoGeometry . . . 907
9.5.1 AddTopoGeometryColumn . . . 907
9.5.2 RenameTopoGeometryColumn . . . 908

9.5.3 DropTopology . . . 908
9.5.4 RenameTopology . . . 909
9.5.5 DropTopoGeometryColumn . . . 909
9.5.6 FixCorruptTopoGeometryColumn . . . 910
9.5.7 Populate_Topology_Layer . . . 910
9.5.8 TopologySummary . . . 912
9.5.9 ValidateTopology . . . 913
9.5.10 ValidateTopologyRelation . . . 915
9.5.11 ValidateTopologyPrecision . . . 915
9.5.12 MakeTopologyPrecise . . . 916
9.5.13 FindTopology . . . 917
9.5.14 FindLayer . . . 918
9.5.15 TotalTopologySize . . . 918
9.5.16 UpgradeTopology . . . 919
9.6 Gestion des statistiques de topologie . . . 920
9.7 Constructeurs de topologie . . . 920
9.7.1 CreateTopology . . . 920
9.7.2 CopyTopology . . . 921
9.7.3 ST_InitTopoGeo . . . 922
9.7.4 ST_CreateTopoGeo . . . 922
9.7.5 TopoGeo_AddPoint . . . 923
9.7.6 TopoGeo_AddLineString . . . 924
9.7.7 TopoGeo_AddPolygon . . . 924
9.7.8 TopoGeo_LoadGeometry . . . 925
9.8 Éditeurs de topologie . . . 925
9.8.1 ST_AddIsoNode . . . 925
9.8.2 ST_AddIsoEdge . . . 926
9.8.3 ST_AddEdgeNewFaces . . . 927
9.8.4 ST_AddEdgeModFace . . . 927
9.8.5 ST_RemEdgeNewFace . . . 928
9.8.6 ST_RemEdgeModFace . . . 929
9.8.7 ST_ChangeEdgeGeom . . . 929
9.8.8 ST_ModEdgeSplit . . . 930
9.8.9 ST_ModEdgeHeal . . . 931
9.8.10 ST_NewEdgeHeal . . . 931
9.8.11 ST_MoveIsoNode . . . 932
9.8.12 ST_NewEdgesSplit . . . 933
9.8.13 ST_RemoveIsoNode . . . 934
9.8.14 ST_RemoveIsoEdge . . . 934

9.9 Accès à la topologie . . . 935
9.9.1 GetEdgeByPoint . . . 935
9.9.2 GetFaceByPoint . . . 936
9.9.3 GetFaceContainingPoint . . . 937
9.9.4 GetNodeByPoint . . . 938
9.9.5 GetTopologyID . . . 939
9.9.6 GetTopologySRID . . . 939
9.9.7 GetTopologyName . . . 940
9.9.8 ST_GetFaceEdges . . . 940
9.9.9 ST_GetFaceGeometry . . . 941
9.9.10 GetRingEdges . . . 942
9.9.11 GetNodeEdges . . . 943
9.9.12 FindVertexSegmentPairsBelowDistance . . . 943
9.10 Traitement de la topologie . . . 944
9.10.1 Polygonize . . . 944
9.10.2 AddNode . . . 944
9.10.3 AddEdge . . . 945
9.10.4 AddFace . . . 946
9.10.5 ST_Simplify . . . 948
9.10.6 RemoveUnusedPrimitives . . . 949
9.11 Constructeurs de TopoGeometry . . . 949
9.11.1 CreateTopoGeom . . . 949
9.11.2 toTopoGeom . . . 951
9.11.3 TopoElementArray_Agg . . . 953
9.11.4 TopoElement . . . 954
9.12 Editeurs de TopoGeometry . . . 954
9.12.1 clearTopoGeom . . . 954
9.12.2 TopoGeom_addElement . . . 955
9.12.3 TopoGeom_remElement . . . 955
9.12.4 TopoGeom_addTopoGeom . . . 956
9.12.5 toTopoGeom . . . 957
9.13 Accès aux TopoGeometry . . . 957
9.13.1 GetTopoGeomElementArray . . . 957
9.13.2 GetTopoGeomElements . . . 957
9.13.3 ST_SRID . . . 958
9.14 Sorties TopoGeometry . . . 959
9.14.1 AsGML . . . 959
9.14.2 AsTopoJSON . . . 961
9.15 Relations spatiales de topologie . . . 962

9.15.1 Equals . . . 962
9.15.2 Intersects . . . 963
9.16 Importer et exporter des topologies . . . 964
9.16.1 Utiliser l'exportateur de topologie . . . 964
9.16.2 Utiliser l'importateur de topologie . . . 965

10 Gestion des données raster, requêtes et applications 966
10.1 Chargement et création de rasters . . . 966
10.1.1 Utilisation de raster2pgsql pour charger des rasters . . . 966
10.1.1.1 Exemple d'utilisation . . . 966
10.1.1.2 options raster2pgsql . . . 967
10.1.2 Création de rasters à l'aide des fonctions raster de PostGIS . . . 969
10.1.3 Utilisation de rasters "out db" stockés sur le cloud . . . 969
10.2 Raster Coverages and Raster/Vector Analysis . . . 970
10.3 Catalogues Raster . . . 972
10.3.1 Catalogue des colonnes raster . . . 972
10.3.2 Aperçu des données raster . . . 973
10.4 Créer des applications personnalisées avec PostGIS Raster . . . 973
10.4.1 Exemple de sortie PHP utilisant ST_AsPNG avec d'autres fonctions raster . . . 974
10.4.2 Exemple ASP.NET C# Sortie utilisant ST_AsPNG en conjonction avec d'autres fonctions raster . . . 974
10.4.3 Application console Java qui produit une requête raster sous forme de fichier image . . . 976
10.4.4 Utiliser PLPython pour extraire des images via SQL . . . 978
10.4.5 Sortie de données raster avec PSQL . . . 978

11 Référence Raster 980
11.1 Types de données pour la prise en charge raster . . . 981
11.1.1 geomval . . . 981
11.1.2 addbandarg . . . 981
11.1.3 rastbandarg . . . 981
11.1.4 raster . . . 982
11.1.5 reclassarg . . . 982
11.1.6 summarystats . . . 983
11.1.7 unionarg . . . 984
11.2 Gestion raster . . . 984
11.2.1 AddRasterConstraints . . . 984
11.2.2 DropRasterConstraints . . . 986
11.2.3 AddOverviewConstraints . . . 987
11.2.4 DropOverviewConstraints . . . 988
11.2.5 PostGIS_GDAL_Version . . . 988

11.2.6 PostGIS_Raster_Lib_Build_Date . . . 989
11.2.7 PostGIS_Raster_Lib_Version . . . 990
11.2.8 ST_GDALDrivers . . . 990
11.2.9 UpdateRasterSRID . . . 993
11.2.10 ST_CreateOverview . . . 994
11.3 Constructeurs de raster . . . 994
11.3.1 ST_AddBand . . . 994
11.3.2 ST_AsRaster . . . 997
11.3.3 ST_AsRasterAgg . . . 1000
11.3.4 ST_Band . . . 1001
11.3.5 ST_MakeEmptyCoverage . . . 1005
11.3.6 ST_MakeEmptyRaster . . . 1006
11.3.7 ST_Tile . . . 1008
11.3.8 ST_Retile . . . 1010
11.3.9 ST_FromGDALRaster . . . 1010
11.4 Fonctions d'accès aux rasters . . . 1011
11.4.1 ST_GeoReference . . . 1011
11.4.2 ST_Height . . . 1012
11.4.3 ST_IsEmpty . . . 1013
11.4.4 ST_MemSize . . . 1014
11.4.5 ST_MetaData . . . 1014
11.4.6 ST_NumBands . . . 1015
11.4.7 ST_PixelHeight . . . 1016
11.4.8 ST_PixelWidth . . . 1017
11.4.9 ST_ScaleX . . . 1019
11.4.10 ST_ScaleY . . . 1019
11.4.11 ST_RasterToWorldCoord . . . 1020
11.4.12 ST_RasterToWorldCoordX . . . 1021
11.4.13 ST_RasterToWorldCoordY . . . 1022
11.4.14 ST_Rotation . . . 1023
11.4.15 ST_SkewX . . . 1024
11.4.16 ST_SkewY . . . 1025
11.4.17 ST_SRID . . . 1026
11.4.18 ST_Summary . . . 1026
11.4.19 ST_UpperLeftX . . . 1027
11.4.20 ST_UpperLeftY . . . 1028
11.4.21 ST_Width . . . 1028
11.4.22 ST_WorldToRasterCoord . . . 1029
11.4.23 ST_WorldToRasterCoordX . . . 1030

11.4.24 ST_WorldToRasterCoordY . . . 1030
11.5 Fonctions d'accès aux bandes raster . . . 1031
11.5.1 ST_BandMetaData . . . 1031
11.5.2 ST_BandNoDataValue . . . 1033
11.5.3 ST_BandIsNoData . . . 1033
11.5.4 ST_BandPath . . . 1035
11.5.5 ST_BandFileSize . . . 1036
11.5.6 ST_BandFileTimestamp . . . 1036
11.5.7 ST_BandPixelType . . . 1037
11.5.8 ST_MinPossibleValue . . . 1038
11.5.9 ST_HasNoBand . . . 1038
11.6 Fonctions d'accès et de modifications des pixels raster . . . 1039
11.6.1 ST_PixelAsPolygon . . . 1039
11.6.2 ST_PixelAsPolygons . . . 1040
11.6.3 ST_PixelAsPoint . . . 1041
11.6.4 ST_PixelAsPoints . . . 1042
11.6.5 ST_PixelAsCentroid . . . 1044
11.6.6 ST_PixelAsCentroids . . . 1045
11.6.7 ST_Value . . . 1047
11.6.8 ST_NearestValue . . . 1050
11.6.9 ST_SetZ . . . 1052
11.6.10 ST_SetM . . . 1053
11.6.11 ST_Neighborhood . . . 1055
11.6.12 ST_SetValue . . . 1057
11.6.13 ST_SetValues . . . 1058
11.6.14 ST_DumpValues . . . 1066
11.6.15 ST_PixelOfValue . . . 1067
11.7 Éditeurs de raster . . . 1068
11.7.1 ST_SetGeoReference . . . 1068
11.7.2 ST_SetRotation . . . 1069
11.7.3 ST_SetScale . . . 1070
11.7.4 ST_SetSkew . . . 1071
11.7.5 ST_SetSRID . . . 1073
11.7.6 ST_SetUpperLeft . . . 1073
11.7.7 ST_Resample . . . 1074
11.7.8 ST_Rescale . . . 1075
11.7.9 ST_Reskew . . . 1077
11.7.10 ST_SnapToGrid . . . 1078
11.7.11 ST_Resize . . . 1079

11.7.12 ST_Transform . . . 1081
11.8 Éditeurs de bandes raster . . . 1085
11.8.1 ST_SetBandNoDataValue . . . 1085
11.8.2 ST_SetBandIsNoData . . . 1086
11.8.3 ST_SetBandPath . . . 1088
11.8.4 ST_SetBandIndex . . . 1089
11.9 Statistiques et analyses des bandes raster . . . 1090
11.9.1 ST_Count . . . 1090
11.9.2 ST_CountAgg . . . 1091
11.9.3 ST_Histogram . . . 1092
11.9.4 ST_Quantile . . . 1094
11.9.5 ST_SummaryStats . . . 1096
11.9.6 ST_SummaryStatsAgg . . . 1098
11.9.7 ST_ValueCount . . . 1100
11.10 Import de raster . . . 1102
11.10.1 ST_RastFromWKB . . . 1102
11.10.2 ST_RastFromHexWKB . . . 1103
11.11 Export de raster . . . 1104
11.11.1 ST_AsBinary/ST_AsWKB . . . 1104
11.11.2 ST_AsHexWKB . . . 1104
11.11.3 ST_AsGDALRaster . . . 1105
11.11.4 ST_AsJPEG . . . 1106
11.11.5 ST_AsPNG . . . 1108
11.11.6 ST_AsTIFF . . . 1108
11.12 Traitement des données raster : algèbre cartographique . . . 1109
11.12.1 ST_Clip . . . 1109
11.12.2 ST_ColorMap . . . 1114
11.12.3 ST_Grayscale . . . 1117
11.12.4 ST_Intersection . . . 1121
11.12.5 ST_MapAlgebra (callback function version) . . . 1124
11.12.6 ST_MapAlgebra (expression version) . . . 1131
11.12.7 ST_MapAlgebraExpr . . . 1133
11.12.8 ST_MapAlgebraExpr . . . 1137
11.12.9 ST_MapAlgebraFct . . . 1143
11.12.10 ST_MapAlgebraFct . . . 1148
11.12.11 ST_MapAlgebraFctNgb . . . 1154
11.12.12 ST_Reclass . . . 1158
11.12.13 ST_ReclassExact . . . 1159
11.12.14 ST_Union . . . 1161

11.13Fonctions de rappel intégrées d'algèbre cartographique . . . 1162
11.13.1 ST_Distinct4ma . . . 1162
11.13.2 ST_InvDistWeight4ma . . . 1163
11.13.3 ST_Max4ma . . . 1165
11.13.4 ST_Mean4ma . . . 1166
11.13.5 ST_Min4ma . . . 1167
11.13.6 ST_MinDist4ma . . . 1168
11.13.7 ST_Range4ma . . . 1169
11.13.8 ST_StdDev4ma . . . 1170
11.13.9 ST_Sum4ma . . . 1172
11.14Traitement des données raster : MNT (élévation) . . . 1173
11.14.1 ST_Aspect . . . 1173
11.14.2 ST_HillShade . . . 1176
11.14.3 ST_Roughness . . . 1177
11.14.4 ST_Slope . . . 1178
11.14.5 ST_TPI . . . 1180
11.14.6 ST_TRI . . . 1181
11.14.7 ST_InterpolateRaster . . . 1182
11.14.8 ST_Contour . . . 1183
11.15Traitement des données raster : raster vers géométrie . . . 1184
11.15.1 Box3D . . . 1184
11.15.2 ST_ConvexHull . . . 1184
11.15.3 ST_DumpAsPolygons . . . 1185
11.15.4 ST_Envelope . . . 1187
11.15.5 ST_MinConvexHull . . . 1187
11.15.6 ST_Polygon . . . 1189
11.15.7 ST_IntersectionFractions . . . 1191
11.16Opérateurs raster . . . 1192
11.16.1 && . . . 1192
11.16.2 &< . . . 1192
11.16.3 &> . . . 1193
11.16.4 = . . . 1194
11.16.5 @ . . . 1194
11.16.6 ~= . . . 1195
11.16.7 ~ . . . 1195
11.17Relations spatiales entre raster et entre bandes raster . . . 1196
11.17.1 ST_Contains . . . 1196
11.17.2 ST_ContainsProperly . . . 1197
11.17.3 ST_Covers . . . 1198

11.17.4 ST_CoveredBy . . . 1199
11.17.5 ST_Disjoint . . . 1200
11.17.6 ST_Intersects . . . 1201
11.17.7 ST_Overlaps . . . 1202
11.17.8 ST_Touches . . . 1203
11.17.9 ST_SameAlignment . . . 1204
11.17.10 ST_NotSameAlignmentReason . . . 1205
11.17.11 ST_Within . . . 1205
11.17.12 ST_DWithin . . . 1206
11.17.13 ST_DFullyWithin . . . 1207
11.18 Astuces raster . . . 1208
11.18.1 Rasters out-DB . . . 1208
11.18.1.1 Répertoire contenant de nombreux fichiers . . . 1208
11.18.1.2 Nombre maximum de fichiers ouverts . . . 1209
11.18.1.2.1 Nombre maximal de fichiers ouverts pour l'ensemble du système . . . 1209
11.18.1.2.2 Nombre maximal de fichiers ouverts par processus . . . 1210

12 PostGIS Extras 1212

13 Index des fonctions spéciales de PostGIS 1213

13.1 Fonctions d'agrégation de PostGIS . . . 1213
13.2 Fonctions Window PostGIS . . . 1214
13.3 Fonctions de PostGIS compatibles avec SQL-MM . . . 1214
13.4 Fonctions d'aide au type geography de PostGIS . . . 1217
13.5 Fonctions de support des données raster de PostGIS . . . 1218
13.6 Fonctions PostGIS de dump Geometry / Geography / Raster . . . 1222
13.7 Fonctions Box de PostGIS . . . 1223
13.8 Fonctions PostGIS supportant la 3D . . . 1224
13.9 Fonctions d'aide aux géométries courbes de PostGIS . . . 1228
13.10 Fonctions de support des surfaces polyédriques de PostGIS . . . 1230
13.11 Matrice d'aide aux fonctions de PostGIS . . . 1233
13.12 Fonctions PostGIS nouvelles, améliorées ou modifiées . . . 1251
13.12.1 Fonctions PostGIS nouvelles ou améliorées en 3.7 . . . 1251
13.12.2 Fonctions PostGIS nouvelles ou améliorées en 3.6 . . . 1253
13.12.3 Fonctions PostGIS nouvelles ou améliorées en 3.5 . . . 1254
13.12.4 Fonctions PostGIS nouvelles ou améliorées en 3.4 . . . 1255
13.12.5 Fonctions PostGIS nouvelles ou améliorées en 3.3 . . . 1256
13.12.6 Fonctions PostGIS nouvelles ou améliorées en 3.2 . . . 1257
13.12.7 Fonctions PostGIS nouvelles ou améliorées en 3.1 . . . 1258

13.12.8 Fonctions PostGIS nouvelles ou améliorées en 3.0 . . . 1259
13.12.9 Fonctions PostGIS nouvelles ou améliorées en 2.5 . . . 1261
13.12.10 Fonctions PostGIS nouvelles ou améliorées en 2.4 . . . 1262
13.12.11 Fonctions PostGIS nouvelles ou améliorées en 2.3 . . . 1263
13.12.12 Fonctions PostGIS nouvelles ou améliorées en 2.2 . . . 1264
13.12.13 Fonctions PostGIS nouvelles ou améliorées en 2.1 . . . 1266
13.12.14 Fonctions PostGIS nouvelles ou améliorées en 2.0 . . . 1271
13.12.15 Fonctions PostGIS nouvelles ou améliorées en 1.5 . . . 1279
13.12.16 Fonctions PostGIS nouvelles ou améliorées en 1.4 . . . 1280
13.12.17 Fonctions PostGIS nouvelles ou améliorées en 1.3 . . . 1280

14 Rapporter un problème **1281**

14.1 Rapporter un problème logiciel . . . 1281
14.2 Signaler les problèmes de documentation . . . 1281

15 Credits and Acknowledgments **1283**

15.1 Project Steering Committee . . . 1283
15.2 Core Contributors Present . . . 1283
15.3 Core Contributors Past . . . 1284
15.4 PostGIS Raster Origins . . . 1284
15.5 Other Contributors . . . 1284

A Notes de version **1288**

A.1 PostGIS 3.7.0 . . . 1288
A.1.1 Changements avec rupture . . . 1288
A.1.2 New Features . . . 1288
A.1.3 Améliorations . . . 1289

Résumé

PostGIS est une extension du système de base de PostgreSQL données relationnel-objet qui permet de stocker des objets SIG (Système d'Information Géographique) dans la base. PostGIS comporte un support des index spatiaux R-Tree basé sur GiST et des fonctions d'analyse et de traitement des objets SIG.





Manuel de la version 3.7.0rc1



Ce travail est soumis à une licence Creative Commons Attribution-Share Alike 3.0 License. Vous pouvez utiliser ce matériel comme bon vous semble, mais nous vous demandons de mentionner le projet PostGIS et, dans la mesure du possible, d'ajouter un lien vers https://postgis.net.

Chapitre 1

Introduction

1.1 What Is PostGIS?

PostGIS extends PostgreSQL so that geospatial data can be stored, indexed, queried, and processed in the database. It keeps location data beside ordinary relational data, so a spatial query can use the same SQL, transactions, constraints, access controls, backup tools, and application connections as the rest of a PostgreSQL database.

The main `postgis` extension adds:

— `geometry` for planar coordinate systems and `geography` for geodetic coordinates on the earth.
— Spatial indexes that let the PostgreSQL planner avoid scanning every row when objects can be filtered by location.
— Functions for measurement, relationships, overlays, geometry construction and editing, coordinate transformation, clustering, and spatial data exchange.
— Support for common geospatial encodings and formats, including WKT, WKB, GeoJSON, GML, KML, FlatGeobuf, Geobuf, and Mapbox Vector Tiles.

This makes PostgreSQL a spatial database: applications can ask both ordinary questions and location-aware questions without moving data to a separate GIS processing system. Desktop GIS, map servers, ETL tools, and application frameworks can connect through normal PostgreSQL interfaces.

1.2 What Is in the PostGIS Distribution?

This manual documents the main PostGIS source distribution. Depending on how PostGIS was built and packaged, it can provide several PostgreSQL extensions and command-line tools:

`postgis` The core geometry and geography types, spatial indexes, and most spatial functions.
`postgis_raster` Raster storage, analysis, loading, and export. Raster support uses GDAL and can be omitted from a build.
`postgis_topology` SQL/MM topology types and functions for models based on shared nodes, edges, and faces.
`postgis_sfcgal` Additional 2D and 3D processing functions backed by the optional SFCGAL library.
Loaders and utilities **shp2pgsql**, **pgsql2shp**, and **raster2pgsql** move data between common file formats and PostGIS. Other utilities support upgrades, profiling, and development.

The source tree also contains internal libraries such as `liblwgeom` and `libpgcommon`. They are implementation layers shared by PostGIS components, not separately released PostgreSQL extensions. Most users start by enabling the core extension with `CREATE EXTENSION postgis`; see Chapitre 2 for available components and build options.

1.3 The PostGIS Project Family

This manual covers the main `postgis/postgis` distribution. The official PostGIS source organization also hosts or mirrors complementary projects. These projects have their own release cycles, installation instructions, and documentation; installing the core extension does not install them.

— Address Standardizer parses and normalizes postal addresses. It moved from the main PostGIS source tree to a separate repository for the PostGIS 3.7 development cycle.
— PostGIS TIGER Geocoder provides United States address geocoding and reverse geocoding using Census TIGER data. It is also maintained separately from the main distribution.
— H3-pg provides PostgreSQL bindings for the H3 hierarchical hexagonal indexing system and interoperates with PostGIS.
— PostGIS Java provides Java bindings for PostGIS types.
— Introduction to PostGIS is the hands-on workshop maintained by the PostGIS project.

Other PostgreSQL spatial extensions can complement PostGIS while remaining independent projects. For example, pgRouting adds routing and graph analysis. Use each project's own manual for its functions and support policy.

1.4 How to Use This Manual

The manual is both a guide to the PostGIS data model and the complete reference for the SQL interface. Choose a starting point based on what you are trying to do:

If you are new to spatial SQL Start with the Introduction to PostGIS workshop for a tutorial, then use Chapitre 4 to understand spatial types, coordinate systems, validity, loading, and indexes.

If you are installing or upgrading PostGIS Read Chapitre 2 and Chapitre 3. Package-specific instructions and supported release information are maintained on the PostGIS website.

If you are writing spatial queries Read Chapitre 5 for spatial relationships and index-aware query patterns, and Chapitre 6 for common performance problems.

If you are connecting a desktop GIS Use the client's current database documentation for its user interface and supported geometry types. For example, the QGIS DB Manager lesson covers PostgreSQL connections, table layers, SQL query layers, import, and export. A query layer should return a stable unique identifier and a geometry column with a known SRID and geometry type. Expensive read-only queries can be exposed through a view or materialized view and indexed as appropriate.

If you need a particular function Use Chapitre 7 or Chapitre 13. Function pages state accepted types, dimensional support, standards conformance, dependency requirements, and version availability.

If you use an optional component See Chapitre 9, Chapitre 10, Chapitre 11, or Chapitre 8.

If something goes wrong Collect the versions reported by PostGIS_Full_Version and follow Chapitre 14. Check Annexe A when behavior differs between release lines.

1.5 Standards Support

This manual marks individual functions as compliant with spatial standards so users can tell which behavior is portable and which behavior is a PostGIS extension. PostGIS implements the OGC Simple Features for SQL model, including the geometry type, Well-Known Text (WKT), Well-Known Binary (WKB), and the common spatial accessors, constructors, predicates, and processing functions. PostGIS also implements many SQL/MM Spatial types and functions.

PostGIS supports geometry families defined by later standards, including SQL/MM curved geometry and OGC Simple Features Access 1.2 PolyhedralSurface, Triangle, and TIN. Support for these geometry families is function-specific, and each function page documents the supported input types and conformance level.

PostGIS also includes widely used extensions and integrations. Examples of PostGIS extensions include Extended WKT (EWKT), Extended WKB (EWKB), SRID metadata in those extended formats, geography, and raster. PostGIS also supports input and output exchange formats defined by other standards, such as GeoJSON, GML, and KML. These features are documented individually and should not be read as OGC Simple Features or SQL/MM requirements unless the specific function or format says so.

When the implementation intentionally differs from a standard, or when a standard leaves behavior ambiguous, the relevant function page describes the PostGIS behavior. Chapitre 13 summarizes SQL/MM compliance and geometry-type support across functions.

1.6 Project and Community

PostGIS began in 2001 as a spatial database research project at Refractions Research. It is now an OSGeo project developed by contributors and organizations around the world. The Project Steering Committee coordinates releases, project policy, and the general direction of development. The complete contributor and sponsor record is in Chapitre 15 at the end of this manual.

The manual is the authoritative reference for the released SQL interface. The project website and source tree provide the surrounding material:

— Documentation routes readers to getting-started instructions, versioned manuals, tips, training material, and the FAQ.

— Community lists mailing lists, chat, events, videos, and other support channels.

— Developer documentation covers development environments, testing, coding style, internals, maintenance, governance, releases, and website work.

Chapitre 2

Installation de PostGIS

Ce chapitre décrit les étapes nécessaires pour installer PostGIS.

2.1 Version courte

Pour compiler, en supposant que toutes les dépendances soient dans votre chemin de recherche :

Code

```
tar -xvzf postgis-3.7.0rc1.tar.gz
cd postgis-3.7.0rc1
./configure
make
make install
```

Une fois PostGIS installé, il est nécessaire de l'activer (Section 3.3) ou de le mettre à jour (Section 3.4) dans chaque base de données où vous voulez l'utiliser.

2.2 Compilation et installation depuis les sources



Note

La plupart des systèmes d'exploitation dispose de paquets pré-compilés de PostgreSQL/PostGIS. La compilation est réellement nécessaire uniquement pour disposer des toutes dernières fonctionnalités ou pour les responsables de paquets PostGIS.

This section includes general compilation instructions. If you are compiling for Windows or another operating system, see the PostGIS getting started guides and the development environment guides.

Pre-Built Packages for various OS are listed in the PostGIS getting started guides

Pour les utilisateurs Windows, des versions stables sont disponibles sur Stackbuilder ou sur le site de téléchargement PostGIS Windows. Des compilations expérimentales incluant les dernières fonctionnalités sont disponibles. Ces compilations sont généralement mises à jour une ou deux fois par semaines, ou chaque fois qu'une nouvelle fonctionnalité intéressante est ajoutée. Vous pouvez les utiliser pour suivre l'avancée des versions de PostGIS

Le module PostGIS est une extension pour le serveur backend PostgreSQL. Par conséquent, PostGIS 3.7.0rc1 *nécessite* l'ensemble des entêtes du serveur PostgreSQL pour pouvoir être compilé. PostGIS peut être compilé pour les versions PostgreSQL 14 à 19beta2. Les versions précédentes de PostgreSQL ne sont *pas* supportées.

Référez-vous aux guides d'installation de PostgreSQL si vous ne l'avez pas encore installé : https://www.postgresql.org .



Note
Pour les fonctionnalités de GEOS, quand vous installez PostgreSQL, vous aurez peut-être besoin de lier explicitement PostgreSQL avec la bibliothèque standard C++ :

Code

```
LDFLAGS=-lstdc++ ./configure [YOUR OPTIONS HERE]
```

Ceci est une solution de contournement d'exceptions C++ d'interactions bugués dans des outils de développements plus ancien. Si vous tombez sur ce genre de problèmes (backend soudainement fermé ou des choses similaires) essayez cette astuce. Cela nécessite de recompiler votre PostgreSQL du début, bien sur.

Les étapes suivantes résument la configuration et la compilation des sources PostGIS. Elles ont été rédigées pour les utilisateurs sous Linux et ne fonctionneront pas pour Windows et Mac.

2.2.1 Obtenir les Sources

Les sources de PostGIS sont disponible depuis https://download.osgeo.org/postgis/source/postgis-3.7.0rc1.tar.gz

Code

```
wget https://download.osgeo.org/postgis/source/postgis-3.7.0rc1.tar.gz
tar -xvzf postgis-3.7.0rc1.tar.gz
cd postgis-3.7.0rc1
```

Un répertoire appelé `postgis-3.7.0rc1` sera créé dans le répertoire courant.

Les sources peuvent également être obtenues depuis le dépôt git https://git.osgeo.org/gitea/postgis/postgis/ .

Code

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git postgis
cd postgis
sh autogen.sh
```

Se placer dans le nouveau répertoire créé `postgis-3.7.0rc1` pour poursuivre l'installation.

Code

```
./configure
```

2.2.2 Pré requis à l'installation

La compilation et l'utilisation de PostGIS nécessitent les éléments suivants :

Obligatoire

— PostgreSQL 14 - 19beta2. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from https://www.postgresql.org 18 .

For current release-line support and end-of-life policy, see https://postgis.net/development/versions_eol/

— Un compilateur GNUC C (`gcc`). D'autres compilateurs ANSI C peuvent être utilisés, mais la compilation avec `gcc` est source de moins de problèmes.

— GNU Make (`gmake` ou `make`). Sur beaucoup de systemes, GNU `make` est la version par défaut de make. Vous pouvez vérifier la version de make avec la commande `make -v`. D'autres versions de `make` peuvent ne pas être compatibles avec le `Makefile` de PostGIS.

— Proj, bibliothèque de reprojection. Proj 6.1 ou supérieur est nécessaire. La bibliothèque Proj est utilisée pour fournir le support de reprojection de coordonnées dans PostGIS. Proj est disponible au téléchargement depuis https://proj.org/ .
— GEOS geometry library, version 3.10.0 or greater, but GEOS 3.15+ is required to take full advantage of all the new functions and features. GEOS is available for download from https://libgeos.org .
— LibXML2, version 2.5.x ou supérieur. LibXML2 est actuellement utilisée dans certaines fonctions d'import (ST_GeomFromGML et ST_GeomFromKML). LibXML2 est disponible au téléchargement depuis https://gitlab.gnome.org/GNOME/libxml2/-/releases.
— JSON-C, version 0.9 ou supérieure. JSON-C est utilisé pour l'import GeoJSON avec la fonction ST_GeomFromGeoJson. JSON-C est disponible depuis https://github.com/json-c/json-c/releases/.
— The GNU Multiple Precision Arithmetic Library (libgmp). Used to control precision of math operations in PostGIS Topology. GMP is available for download from https://gmplib.org/.
— GDAL, version 2.4 or higher, is required for raster support. GDAL 3+ is preferred. https://gdal.org/download.html.
— Si vous compilez avec PostgreSQL+JIT, LLVM version >=6 est nécessaire https://trac.osgeo.org/postgis/ticket/4125.

Optionnel

— GDAL (pseudo optionnel) vous pouvez l'omettre seulement si vous ne voulez pas du support raster. Pensez également à activer les pilotes que vous souhaitez utiliser, comme décrit dans Section 3.2.
— GTK (GTK+2.0, 2.8+) pour compiler le chargeur de shape file shp2pgsql-gui. http://www.gtk.org/ .
— SFCGAL, 1.3.1 or higher is required, 2.3+ is needed to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Chapitre 8. https://sfcgal.gitlab.io/sfcgal/.
— Pour permettre d'utiliser ST_AsMVT, la bibliothèque protobuf-c 1.1.0 ou supérieur (pour l'utilisation) et le compilateur protoc-c (pour la compilation) sont nécessaires. Également, pkg-config est nécessaire pour vérifier la version minimale correcte de protobuf-c. Voir protobuf-c. Par défaut, PostGIS utilise Wagyu pour valider les polygones MVT plus rapidement, ce qui nécessite un compilateur c++11. Ceci utilisera les CXXFLAGS et le même compilateur que l'installation PostgreSQL. Pour désactiver cela et utiliser GEOS à la place, utilisez la variable `--without-wagyu` pendant l'étape de configuration.
— CUnit (`CUnit`). Nécessaire pour les tests de régression. http://cunit.sourceforge.net/
— DocBook (`xsltproc`) est nécessaire pour générer la documentation. Docbook est disponible depuis le site http://www.docbook.org/ .
— DBLatex (`dblatex`) est nécessaire pour générer la documentation au format PDF. DBLatex est disponible depuis http://dblatex.sour .
— PDF documentation needs an SVG rasterizer for its high-resolution figure fallbacks. The build prefers `rsvg-convert`, and can also use GraphicsMagick (`gm convert`) or ImageMagick (`magick` or `convert`).

2.2.3 Configuration de la compilation

Comme pour la plupart des installations linux, la première étape est de générer le Makefile qui sera utilisé pour compiler le code source. Ceci est réalisée en lançant le script shell

Code

```
./configure
```

Sans paramètre supplémentaire, cette commande tentera de localiser automatiquement les composants requis et les bibliothèques nécessaires à la compilation de PostGIS. Bien que cela soit l'utilisation la plus commune de la commande **./configure**, vous pouvez également ajouter différents paramètres à ce script. Par exemple, vous pouvez définir l'emplacement de bibliothèques ou de programmes si ceux-ci ne sont pas localisés à un emplacement standard.

La liste suivante présente les options les plus courantes. Pour consulter la liste complète utilisez l'option **--help** ou **--help=short**.

`--with-library-minor-version` À partir de PostGIS 3.0, les fichiers de bibliothèque générés par défaut n'auront plus la version mineur dans le nom de fichier. Ceci signifie que les libs PostGIS 3 se termineront par `postgis-3`. Ceci a été fait pour faciliter l'usage de pg_upgrade, avec la contrainte que vous ne pouvez installer qu'une seule version de PostGIS 3 sur votre serveur. Pour basculer sur l'ancien comportement et avoir des fichiers qui incluent la version mineure : e.g. `postgis-3.0`, ajoutez ce paramètre dans la commande de configuration.

`--prefix=PREFIX` This is the installation prefix for PostGIS command-line utilities. By default, utilities are installed in the executable directory reported by the detected PostgreSQL installation.



Attention
PostgreSQL extension files are installed in the directories reported by **pg_config**, so the server can load them. This includes the PostGIS shared library and SQL extension files. Use `--with-pgconfig=FILE` to choose the PostgreSQL installation that PostGIS builds and installs against.

`--with-pgconfig=FILE` PostgreSQL fournit l'utilitaire **pg_config** permettant aux extensions comme PostGIS de localiser le répertoire d'installation de PostgreSQL. Utiliser ce paramètre (**--with-pgconfig=/path/to/pg_config**) pour spécifier une installation particulière de PostgreSQL pour laquelle PostGIS doit être compilée.

`--with-gdalconfig=FILE` GDAL, une des bibliothèques requises pour le support des rasters. **gdal-config** pour permettre au logiciel de localiser le répertoire d'installation de GDAL. Utiliser ce paramètre (**--with-gdalconfig=/path/to/gdal-config**) pour spécifier un répertoire d'installation particulier de GDAL qui sera utilisé pour compiler PostGIS.

`--with-geosconfig=FILE` GEOS, une des bibliothèques requises, fournit un utilitaire appelé **geos-config** permettant aux logiciels de localiser le répertoire d'installation de GEOS. Utiliser ce paramètre (**--with-geosconfig=/path/to/geos-config**) pour spécifier le repertoire de GEOS qui sera utilisé pour la compilation de PostGIS.

`--with-xml2config=FILE` LibXML est la bibliothèque requise pour les processus GeomFromKML/GML. Elle est normalement trouvée si libxml est installé, mais si ce n'est pas le cas ou si vous souhaitez qu'une version spécifique soit utilisée, vous devrez indiquer à PostGIS un fichier confi `xml2-config` spécifique pour permettre aux installations logicielles de localiser le répertoire d'installation de LibXML. Utilisez ce paramètre (**>--with-xml2config=/path/to/xml2-config**) pour spécifier manuellement une installation LibXML particulière avec laquelle PostGIS sera construit.

`--with-projdir=DIR` Proj4 est la bibliothèque de reprojection nécessaire à PostGIS. Utilisez ce paramètre (**--with-projdir=/path/t**
pour spécifier le répertoire de Proj qui sera utilisé pour la compilation de PostGIS.

`--with-libiconv=DIR` Répertoire d'installation d'iconv.

`--with-jsondir=DIR` JSON-C est une bibliothèque sous licence MIT utilisée par PostGIS pour les traitements JSON (ST_GeomFromJSON par exemple). Utiliser ce paramètre (**--with-jsondir=/path/to/jsondir**) pour spécifier le répertoire de JSON-C qui sera utilisé pour la compilation de PostGIS.

`--with-gui` Compile l'interface graphique d'import de données (nécessite GTK+2.0). Ceci créé l'interface graphique shp2pgsql-gui à shp2pgsql.

`--without-raster` Compilation sans support des raster.

`--without-topology` Compile without topology support.

`--with-gettext=no` Par défaut PostGIS tentera de détecter la gestion de gettext et de reposer dessus pour la compilation, cependant si vous tombez sur des problèmes d'incompatibilités qui cause la cassure du chargeur, vous pouvez le désactiver entièrement avec cette commande. Référez vous au ticket http://trac.osgeo.org/postgis/ticket/748 pour un exemple de problème résolu par cette configuration. NOTE : que vous perdez beaucoup de chose en le désactivant. Cela est utilisé pour la gestion de l'aide et des labels internationaux dans le chargeur graphique qui n'est pas documenté et encore expérimental.

`--with-sfcgal=PATH` Par défaut, PostGIS ne contiendra pas le support sfcgal sans cet argument. `PATH` est un argument optionnel permettant de préciser un chemin alternatif vers sfcgal-config.

`--without-phony-revision` Désactive la mise à jour de postgis_revision.h à partir du HEAD courant du dépôt git.



Note
If you obtained PostGIS from the code repository , the first step is really to run the script

Code

```
./autogen.sh
```

Ce script générera le script **configure** qui est utilisé pour personnaliser votre installation de PostGIS.

Si vous avez obtenu PostGIS comme archive, lancer la commande **./autogen.sh** n'est pas nécessaire puisque **configure** a déjà été généré.

2.2.4 Compiler

Une fois le Makefile généré, compiler PostGIS est aussi simple que lancer

Code

```
make
```

Output

```
[last line of output]
PostGIS was built successfully. Ready to install.
```

All the functions have comments generated from the documentation. If you wish to install these comments into your spatial databases later, run the command which requires docbook. The postgis_comments.sql and other package comment files are also packaged in the tar.gz distribution in the doc folder so no need to make comments if installing from the tar ball. Comments are also included as part of the CREATE EXTENSION install.

Code

```
make comments
```

The **make cheatsheets** target generates html cheat sheets suitable for quick reference or for student handouts. This requires xsltproc to build and will generate 4 files in doc folder `topology_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html`

Vous pouvez télécharger des pré-compilations disponibles en HTML et PDF à partir de PostGIS / PostgreSQL Study Guides

Code

```
make cheatsheets
```

2.2.5 Compiler les Extensions PostGIS et les déployer

The PostGIS extensions are built and installed automatically when PostgreSQL extension support is available.

Si vous compilez à partir des dépôts des sources, vous devez compiler les descriptions de fonction d'abord. Ceci est compilé si vous avez docbook installé. Vous pouvez également compiler manuellement avec cette commande :

Code

```
make comments
```

Compiler les commentaires n'est pas nécessaire si vous avez compilé à partir d'une release d'archive puisque ceux-ci sont des pré-compilations packagés avec le tar ball.

Les extensions devraient être automatiquement compilées lors du make install. Vous pouvez, si nécessaire, compiler à partir des répertoires d'extensions ou copier les fichiers si vous en avez besoin sur un serveur différent.

Set `PGUSER` if you need to override the psql connection user, then test before installing. After installation, run the extension regression tests.

Code

```
cd extensions
cd postgis
make clean
make
export PGUSER=postgres
make check
make install
make check RUNTESTFLAGS=--extension
```



Note

`make check` utilise psql pour faire tourner les tests, et peut donc utiliser les variables d'environnement psql. Les variables classiques utiles à définir sont `PGUSER`,`PGPORT`, and `PGHOST`. Voir variables d'environnement psql

Les fichiers extensions seront toujours les mêmes pour les mêmes versions de PostGIS et PostgreSQL indépendamment de l'OS, par conséquent il n'y a pas de problème à copier les fichiers extensions d'un OS à un autre du moment que vous avez les binaires PostGIS déjà installés sur vos serveurs.

Si vous voulez installer les extensions manuellement sur un serveur différent séparé de votre développement, vous devez copier les fichiers suivants à partir du répertoire extension dans le répertoire `PostgreSQL / share / extension` de votre installation PostgreSQL ainsi que les binaires nécessaires pour une version correcte de PostGIS si vous ne les avez pas déjà sur le serveur.

— Ceux-ci sont les fichiers de contrôle qui renvoie les informations telles que la version de l'extension à installer si non spécifié. `postgis.control, postgis_topology.control`.

— Tous les fichiers dans le répertoire /sql de chaque extension. Notez que ceux-ci nécessitent d'être copiées à la racine du répertoire share/extension de PostgreSQL `extensions/postgis/sql/*.sql`, `extensions/postgis_topology/sql/*.sql`

Une fois que vous avez fait cela, vous devriez voir `postgis`, `postgis_topology` comme extensions disponibles dans PgAdmin -> extensions.

Si vous utilisez psql, vous pouvez vérifier que les extensions sont installées en lançant cette requête :

Code

```
SELECT name, default_version, installed_version
FROM pg_available_extensions
WHERE name LIKE 'postgis%'
ORDER BY name;
```

Output

```
+------------------+-----------------+-------------------+
| name             | default_version | installed_version |
+------------------+-----------------+-------------------+
| postgis          | 3.7.0rc1        | 3.7.0rc1          |
| postgis_raster   | 3.7.0rc1        | 3.7.0rc1          |
| postgis_sfcgal   | 3.7.0rc1        |                   |
| postgis_topology | 3.7.0rc1        |                   |
+------------------+-----------------+-------------------+
```

Si vous avez l'extension installée dans la base de données que vous interrogez, vous verrez la mention dans la colonne `installed_ver`
Si vous n'obtenez aucun enregistrement , cela signifie que vous n'avez pas d'extension postgis installés sur le serveur. PgAdmin III 1.14+ fournira aussi cette information dans la section `extensions` dans l'arbre de l'explorateur de la base de données et autorisera même la mise à jour ou la désinstallation par clic-droit.

Si vous avez les extensions disponibles, vous pouvez installer les extensions postgis dans votre base de données de votre choix soit en utilisant l'interface d'extension de PgAdmin ou lançant ces commandes SQL :

Code

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster;
CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION postgis_topology;
```

Avec psql, vous pouvez contrôler les versions installées ainsi que les schémas d'installation.

Code

```
\connect mygisdb
\x
\dx postgis*
```

Output

```
List of installed extensions
+-[ RECORD 1 ]-------------------------------------------------+
| Name        | postgis                                        |
| Version     | 3.7.0rc1                                         |
| Schema      | public                                         |
| Description | PostGIS geometry, geography, and raster...      |
+-[ RECORD 2 ]-------------------------------------------------+
| Name        | postgis_raster                                 |
| Version     | 3.7.0rc1                                         |
| Schema      | public                                         |
| Description | PostGIS raster types and functions             |
+-[ RECORD 3 ]-------------------------------------------------+
| Name        | postgis_topology                               |
| Version     | 3.7.0rc1                                         |
| Schema      | topology                                       |
| Description | PostGIS topology spatial types and functions   |
+-------------------------------------------------------------+
```



AVERTISSEMENT

Extension tables `spatial_ref_sys`, `layer`, `topology` can not be explicitly backed up. They can only be backed up when the respective `postgis` or `postgis_topology` extension is backed up, which only happens when you back up the whole database. Only srid records not packaged with PostGIS are captured in backups, so don't change the entries we ship and expect the modifications to persist. Put in a ticket if you find an issue. The structures of extension tables are never backed up since they are created with `CREATE EXTENSION` and assumed to be the same for a given version of an extension. These behaviors are built into the current PostgreSQL extension model.

Si vous avez installé la version 3.7.0rc1 sans utiliser le système d'extensions, il est possible de l'activer via les commandes suivantes pour packager les fonctions dans leurs extensions respectives. L'installation via `unpackaged` a été supprimé dans PostgreSQL 13, nous vous invitons donc à basculer sur une installation utilisant le système d'extensions avant de migrer à PostgreSQL 13.

Code

```
CREATE EXTENSION postgis FROM unpackaged;
CREATE EXTENSION postgis_raster FROM unpackaged;
CREATE EXTENSION postgis_topology FROM unpackaged;
```

2.2.6 Tests

Si vous désirez tester la compilation de PostGIS, lancez

Code

```
make check
```

La commande ci-dessus fonctionnera pour différentes tests de vérification et de régression en utilisant la bibliothèque générée selon la base de données actuelle.



Note

Si vous avez configuré PostGIS en utilisant un emplacement non standard de PostgreSQL, GEOS, ou Proj4, vous pourrez avoir besoin d'ajouter l'emplacement des bibliothèques à la variable d'environnement `LD_LIBRARY_PATH`.



Attention
Pour le moment, **make check** repose sur les variables d'environnement `PATH` et `PGPORT` lors de la réalisation des vérifications - il n'utilise *pas* la version de PostgreSQL qui a pu être définie en utilisant la paramètre de configuration **--with-pgconfig**. Assurez vous donc de modifier votre PATH pour correspondre l'installation de PostgreSQL détectée durant la configuration ou préparez vous à gérer des maux de tête à venir.



Note
Sandboxed build accounts that are not PostgreSQL superusers can delegate database ownership during the regression cycle by exporting `POSTGIS_REGRESS_DB_OWNER`. The harness will create the temporary regression database owned by the nominated role while continuing to connect using the less privileged account. Combine this with `POSTGIS_REGRESS_ROLE_EXT_CREATOR` when the extension creation role must differ from the database owner. These variables allow automated environments to exercise the full upgrade and extension install paths without promoting the calling account to superuser, provided the target PostgreSQL instance permits extension installation by those delegate roles.

Si réussi, make check sortira les résultats de près de 500 tests. Les résultats seront similaires à la sortie ci-dessous (avec de nombreuses lignes omises) :

Code

```
CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/

  .
  .
  .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites     44     44    n/a      0        0
               tests    300    300    300      0        0
             asserts   4215   4215   4215      0      n/a
Elapsed time =    0.229 seconds

  .
  .
  .

Running tests

  .
  .
  .

Run tests: 134
Failed: 0


-- if you build with SFCGAL

  .
  .
  .

Running tests

  .
  .
  .
```

```
Run tests: 13
Failed: 0

-- if you built with raster support

  .
  .
  .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites     12     12    n/a      0        0
               tests     65     65     65      0        0
             asserts  45896  45896  45896      0      n/a


  .
  .
  .

Running tests

  .
  .
  .

Run tests: 101
Failed: 0

-- topology regress

.
.
.

Running tests

  .
  .
  .

Run tests: 51
Failed: 0

-- if you built --with-gui, you should see this too

     CUnit - A unit testing framework for C - Version 2.1-2
     http://cunit.sourceforge.net/

  .
  .
  .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      4      4      4      0        0
             asserts      4      4      4      0      n/a
```

La sortie de la commande devrait ressembler à :

Output

```
============== dropping database "contrib_regression" ==============
DROP DATABASE
============== creating database "contrib_regression" ==============
CREATE DATABASE
ALTER DATABASE
============== installing fuzzystrmatch               ==============
CREATE EXTENSION
============== installing postgis                     ==============
CREATE EXTENSION
============== running regression test queries        ==============
test test-normalize_address   ... ok

====================
All 2 tests passed.
====================
```

2.2.7 Installation

Pour installer PostGIS, entrez

Code

```
make install
```

Ceci copiera les fichiers d'installation de PostGIS dans leur sous-répertoires appropriés définis par le paramètre de configuration **--prefix**. En particulier :

— Les binaires du chargeur et du dumper sont installés dans `[prefix]/bin`.
— Les fichiers SQL, tel que `postgis.sql`, sont installé dans `[prefix]/share/contrib`.
— Les bibliothèques PostGIS sont installées dans `[prefix]/lib`.

Si vous avez déjà lancé la commande **make comments** pour générer les fichiers `postgis_comments.sql`, `raster_comments.sql`, installer le fichier sql en lançant

Code

```
make comments-install
```



Note
`postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql` ont été séparés de la compilation initiale et des cibles de l'installation depuis qu'ils sont dépendant de **xsltproc**.

2.3 Problèmes courants pendant l'installation

Il y a plusieurs choses à vérifier quand votre installation ou mise à jour ne va pas dans la direction souhaitée.

PostgreSQL version mismatch.

Vérifiez que vous avez installé PostgreSQL 14 ou plus récent et que vous êtes en train de compiler avec la même version du code source de PostgreSQL que la version qui fonctionne. Un mélange peut arriver lorsque votre distribution (Linux) a déjà une version de PostgreSQL installée ou que vous avez oublié que vous avez déjà installée une version. PostGIS fonctionnera uniquement avec PostgreSQL 14 ou plus récent, et des messages d'erreurs étranges et inhabituelles en résultera si vous utilisez une version plus ancienne. Pour vérifier la version de PostgreSQL qui fonctionne, connectez vous à la base en utilisant psql et lancez la requête :

Code

```
SELECT version();
```

If you are running an RPM based distribution, you can check for the existence of pre-installed packages using the **rpm** command as follows:

Code

```
rpm -qa | grep postgresql
```

Restoring an upgrade into a database without PostGIS.

Si votre mise à jour plante, assurez vous de la présence de PostGIS dans la nouvelle base de données.

Code

```
SELECT PostGIS_Full_Version();
```

Vérifiez également que le script configure a correctement détecté les chemins et versions de PostgreSQL, de la bibliothèque Proj.4 et de la blibliothèque GEOS.

La sortie du configure est utilisée pour générer le fichier `postgis_config.h`. Vérifiez que les variables `POSTGIS_PGSQL_VERSIO` `POSTGIS_PROJ_VERSION` et `POSTGIS_GEOS_VERSION` ont été définies correctement.

Chapitre 3

Administration de PostGIS

3.1 Optimisation des performances

Le réglage des performances de PostGIS est similaire à celui de n'importe quelle charge de travail PostgreSQL. La seule considération supplémentaire est que les géométries et les rasters sont généralement de grande taille, donc les optimisations liées à la mémoire ont généralement plus d'impact sur PostGIS que sur d'autres types de requêtes PostgreSQL.

Pour plus de détails sur l'optimisation de PostgreSQL, reportez-vous à Tuning your PostgreSQL Server.

La configuration PostgreSQL peut être modifiée au niveau du serveur sans toucher à `postgresql.conf`, ni à `postgresql.auto.c` en utilisant la commande `ALTER SYSTEM`.

Set `work_mem`, reload the configuration so new connections use the change, and display the current value. Use **SHOW ALL** to list all settings.

Code

```
ALTER SYSTEM SET work_mem = '256MB';
SELECT pg_reload_conf();
SHOW work_mem;
```

En plus des paramètres de Postgres, PostGIS a quelques paramètres personnalisés qui sont listés dans Section 7.22.

3.1.1 Démarrage

Ces paramètres sont configurés dans `postgresql.conf` :

constraint_exclusion

— Valeur par défaut : partition

— Ceci est généralement utilisé pour le partitionnement de table. La valeur par défaut "partition" oblige le planificateur à analyser les tables en ne considérant les contraintes que si ells sont dans une hiérarchie héritée ce qui évite d'être pénaliser par le planificateur.

shared_buffers

— Défaut : ~128MB

— Réglez-le à environ 25 à 40 % de la mémoire vive disponible. Sous Windows, il se peut que vous ne puissiez pas définir une valeur aussi élevée.

max_worker_processes contrôle le nombre maximum de processus d'arrière-plan disponibles pour le serveur et limite donc également le nombre de workers pouvant participer à des requêtes parallèles.

— Valeur par défaut : 8

— Définit le nombre maximum de processus d'arrière-plan que le système peut prendre en charge. Ce paramètre ne peut être défini qu'au démarrage du serveur.

3.1.2 Temps d'exécution

work_mem - définit la taille de la mémoire utilisée pour les opérations de tri et les requêtes complexes

— Valeur par défaut : 1-4MB
— Ajustement pour les grandes bases de données, les requêtes complexes, beaucoup de RAM
— Ajuster à la baisse pour de nombreux utilisateurs simultanés ou une faible mémoire vive.
— Si vous avez beaucoup de mémoire vive et peu de développeurs :

Code

```
SET work_mem TO '256MB';
```

maintenance_work_mem - la taille de la mémoire utilisée pour VACUUM, CREATE INDEX, etc.

— Valeur par défaut : 16-64MB
— Généralement trop faible - immobilise les Entrées/Sorties, bloque les objets pendant l'échange de mémoire
— Nous recommandons 32MB à 1GB sur les serveurs de production avec beaucoup de RAM, mais cela dépend du nombre d'utilisateurs simultanés. Si vous avez beaucoup de RAM et peu de développeurs :

Code

```
SET maintenance_work_mem TO '1GB';
```

max_parallel_workers_per_gather

Parallel query support in modern PostgreSQL and PostGIS uses this setting to decide how many workers a plan may request. If set to higher than 0, then some queries such as those involving relation functions like `ST_Intersects` can use multiple processes and can run more than twice as fast when doing so. If you have a lot of processors to spare, you should change the value of this to as many processors as you have. Also make sure to bump up `max_worker_processes` to at least as high as this number.

— Valeur par défaut : 0
— Définit le nombre maximum de workers qui peuvent être démarrés par un seul nœud `Gather`. Les workers parallèles sont pris dans le pool de processus établi par `max_worker_processes`. Notez que le nombre de workers demandé peut ne pas être disponible au moment de l'exécution. Dans ce cas, le plan s'exécutera avec moins de workers que prévu, ce qui peut s'avérer inefficace. La définition de cette valeur à 0, qui est la valeur par défaut, désactive l'exécution parallèle des requêtes.

3.2 Configurer la prise en charge du raster

Si vous activez la prise en charge du raster, vous devriez lire ce qui suit afin de bien la configurer.

Les rasters de système de fichiers (out-db) ainsi que les pilotes raster sont désactivés par défaut. Pour réactiver ceux-ci, vous devez définir les variables d'environnement suivantes `POSTGIS_GDAL_ENABLED_DRIVERS` et `POSTGIS_ENABLE_OUTDB_RASTERS` dans l'environnement serveur. Vous pouvez également utiliser l'approche multiplateforme en définissant la Section 7.22 correspondant.

Si vous souhaitez activer le raster hors connexion :

Code

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

Si la variable a n'importe quelle autre valeur, ou si elle n'a pas de valeur, le support du raster hors-connexion sera désactivé.

Pour utiliser tous les pilotes GDAL disponibles dans votre installation GDAL, définissez la variable d'environnement via

Code

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

Si vous souhaitez activer une liste de pilotes spécifiques, définissez la variable d'environnement via :

Code

```
POSTGIS_GDAL_ENABLED_DRIVERS="GTiff PNG JPEG GIF XYZ"
```



Note
Si vous êtes sous Windows, ne pas mettre de guillemets autour de la liste des pilotes

La définition de variables d'environnement varie en fonction du système d'exploitation. Pour PostgreSQL installé sur Ubuntu ou Debian via apt-postgresql, la meilleure façon est d'éditer `/etc/postgresql/`*`MAJOR`*`/`*`CLUSTER`*`/environment` où les espaces réservés se réfèrent à la version majeure de PostgreSQL et au nom du cluster.

Sous Windows, si vous fonctionnez en tant que service, vous pouvez définir des variables système auxquelles vous pouvez accéder, sous Windows 7, en cliquant avec le bouton droit de la souris sur Ordinateur ->Propriétés - Paramètres système avancés ou, dans l'explorateur, en naviguant jusqu'à `Panneau de configuration - Tous les éléments du panneau de configuration - Système`. Cliquez ensuite sur *Advanced System Settings ->Advanced->Environment Variables* et ajoutez de nouvelles variables système.

Après avoir changé les variables d'environnement, vous devrez redémarrer le service PostgreSQL pour prendre en compte les changements.

3.3 Création de bases de données spatiales

3.3.1 Base de données spatiale en utilisant EXTENSION

If you have compiled and installed the extensions/postgis modules, you can turn a database into a spatial one using the EXTENSION mechanism.

L'extension cœur postgis inclus le support des types geometry et geography, la table spatial_ref_sys ainsi que toutes les fonctions et commentaires. Les supports de raster et topologie sont fournis par des extensions dédiées.

Exécutez les requêtes SQL suivantes dans la base de données où vous souhaitez activer le support spatial :

Code

```
CREATE EXTENSION IF NOT EXISTS plpgsql;
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster; -- OPTIONAL
CREATE EXTENSION postgis_topology; -- OPTIONAL
```

3.3.2 Base de données spatiale sans utiliser EXTENSION (non recommandé)



Note
Cette méthode n'est en générale nécessaire que si vous ne pouvez pas ou ne voulez pas que PostGIS soit installé dans le répertoire des extensions PostgreSQL (par exemple pour des tests, du développement, ou dans un environnement restreint).

L'ajout des objets et définitions des fonctions PostGIS dans votre base de données se fait en chargeant plusieurs fichiers sql présents dans `[prefix]/share/contrib`, cet emplacement est celui qui a été défini durant la phase de compilation.

Les objets au cœur de PostGIS (types geometry et geography, et les fonctions associées) sont dans le script `postgis.sql`. Les objets raster sont dans le script `rtpostgis.sql`. Les objets de topologie sont dans le script `topology.sql`.

Pour avoir la liste complète des définitions des systèmes de coordonnées EPSG, vous pouvez aussi charger le script `spatial_ref_sys.sql` pour remplir la table `spatial_ref_sys`. Cela permettre d'utiliser la fonction ST_Transform() pour effectuer des reprojections sur les géométries.

Si vous souhaitez ajouter les commentaires sur les fonctions PostGIS, vous pouvez les trouver dans le script `postgis_comments.sql`. Vous pouvez accéder aux commentaires d'une fonction en tapant **\dd [nom_de_la_fonction]** depuis un terminal **psql**.

Exécutez les commandes Shell suivantes dans votre terminal :

Code

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.6/

# Core objects
psql -d ${DB} -f ${SCRIPTSDIR}/postgis.sql
psql -d ${DB} -f ${SCRIPTSDIR}/spatial_ref_sys.sql
psql -d ${DB} -f ${SCRIPTSDIR}/postgis_comments.sql # OPTIONAL

# Raster support (OPTIONAL)
psql -d ${DB} -f ${SCRIPTSDIR}/rtpostgis.sql
psql -d ${DB} -f ${SCRIPTSDIR}/raster_comments.sql # OPTIONAL

# Topology support (OPTIONAL)
psql -d ${DB} -f ${SCRIPTSDIR}/topology.sql
psql -d ${DB} -f ${SCRIPTSDIR}/topology_comments.sql # OPTIONAL
```

3.4 Mise à jour des bases de données spatiales

La mise à jour des bases de données spatiales existantes peut s'avérer délicate car elle nécessite le remplacement ou l'introduction de nouvelles définitions d'objets PostGIS.

Malheureusement, toutes les définitions ne peuvent pas être facilement remplacées dans une base de données active, de sorte que la meilleure solution consiste parfois à effectuer un processus de dump/rechargement.

PostGIS propose une procédure SOFT UPGRADE pour les versions mineures ou les corrections de bugs, et une procédure HARD UPGRADE pour les versions majeures.

Avant d'essayer de mettre à jour PostGIS, il est toujours utile de sauvegarder vos données. Si vous utilisez l'option -Fc pour pg_dump, vous serez toujours en mesure de restaurer le dump lors d'un HARD UPGRADE.

3.4.1 Mise à niveau progressive (Soft upgrade)

Si vous avez installé votre base de données en utilisant des extensions, vous devrez également mettre à jour en utilisant le modèle d'extension. Si vous avez installé votre base de données à l'aide de l'ancien script SQL, il vous est conseillé de passer à l'extension car le script n'est plus pris en charge.

3.4.1.1 Mise à niveau progressive (Soft upgrade) utilisant des extensions

Si vous avez installé PostGIS à l'origine avec des extensions, vous devez également effectuer une mise à jour en utilisant des extensions. Faire une mise à jour mineure avec les extensions est assez facile.

Vous devez utiliser la fonction PostGIS_Extensions_Upgrade pour passer à la dernière version que vous avez installée.

Code

```
SELECT postgis_extensions_upgrade();
```

Si vous mettez à jour une base de données reposant sur PostGIS 2.5, exécutez la commande **ALTER EXTENSION** suivi de `postgis_extensions_upgrade()` pour s'assurer que les objets `postgis_raster` hérités sont récupérés dans l'extension principale.

Code

```
ALTER EXTENSION postgis UPDATE;
SELECT postgis_extensions_upgrade();
```

This second call is needed to rebundle the `postgis_raster` extension.

Code

```
SELECT postgis_extensions_upgrade();
```

Si plusieurs versions de PostGIS sont installées et que vous ne souhaitez pas mettre à niveau vers la dernière version, vous pouvez spécifier explicitement la version comme suit :

Code

```
ALTER EXTENSION postgis UPDATE TO "3.7.0rc1";
ALTER EXTENSION postgis_topology UPDATE TO "3.7.0rc1";
```

Si vous obtenez un message d'erreur du type :

Output

```
No migration path defined for ... to 3.7.0rc1
```

Vous devrez alors sauvegarder votre base de données, en créer une nouvelle comme décrit dans Section 3.3.1, puis restaurer votre sauvegarde sur cette nouvelle base de données.

Si vous obtenez un message du type :

Output

```
Version "3.7.0rc1" of extension "postgis" is already installed
```

Dans ce cas, tout est déjà à jour et vous pouvez l'ignorer en toute sécurité. **Sauf** vous essayez de passer d'une version de développement à la suivante (qui ne reçoit pas de nouveau numéro de version) ; dans ce cas, vous pouvez ajouter "next" à la chaîne de caractères de la version, et la prochaine fois, vous devrez à nouveau supprimer le suffixe "next" :

Code

```
ALTER EXTENSION postgis UPDATE TO "3.7.0rc1next";
        ALTER EXTENSION postgis_topology UPDATE TO "3.7.0rc1next";
```



Note

Si vous avez installé PostGIS à l'origine sans spécifier de version, vous pouvez souvent ignorer la réinstallation de l'extension postgis avant la restauration puisque la sauvegarde a juste exécuté `CREATE EXTENSION postgis` et récupère donc la dernière version la plus récente lors de la restauration.



Note

If you are upgrading PostGIS extension from a version prior to 3.0.0, you will have a new extension *postgis_raster* which you can safely drop, if you don't need raster support.

Code

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 Mise à niveau progressive (Soft Upgrade) sans extensions

Extensions have been the supported installation method for many releases. If you still rely on an unpackaged installation, perform a HARD UPGRADE into a database created with extensions and migrate your data there.

3.4.1.3 Mise à niveau complète (Hard upgrade)

Par HARD UPGRADE, nous entendons un dump/recharge complet des bases de données compatibles avec PostGIS. Vous avez besoin d'une mise à jour complète lorsque le stockage interne des objets PostGIS change ou lorsque la mise à jour progressive (Soft upgrade) n'est pas possible. L'annexe Release Notes indique pour chaque version si vous avez besoin d'un dump/recharge (HARD UPGRADE) pour effectuer la mise à jour.

The dump/reload process is assisted by the **postgis_restore** script which takes care of skipping from the dump all definitions which belong to PostGIS (including old ones), allowing you to restore your schemas and data into a database with PostGIS installed without getting duplicate symbol errors or bringing forward deprecated objects.

Supplementary instructions for windows users are available in the PostGIS Windows installation guide.

La procédure est la suivante :

1. Créer un dump au format personnalisé de la base de données que vous souhaitez mettre à jour (appelons-le `olddb`) et inclure des blobs binaires (-b) et une sortie verbose (-v). L'utilisateur peut être le propriétaire de la base de données, il n'a pas besoin d'être le super utilisateur postgres.

   Code

   ```
   pg_dump -h localhost -p 5432 -U postgres -Fc -b -v -f "/somepath/olddb.backup" olddb
   ```

2. Effectuez une nouvelle installation de PostGIS dans une nouvelle base de données - nous appellerons cette base de données `newdb`. Veuillez vous référer à Section 3.3.2 et Section 3.3.1 pour les instructions sur la façon de procéder.

   Les entrées spatial_ref_sys trouvées dans votre dump seront restaurées, mais elles ne remplaceront pas les entrées existantes dans spatial_ref_sys. Cela permet de s'assurer que les corrections du jeu officiel seront correctement propagées dans les bases de données restaurées. Si, pour une raison quelconque, vous souhaitez vraiment que vos propres entrées standard soient remplacées, ne chargez pas le fichier spatial_ref_sys.sql lors de la création de la nouvelle base de données.

   Si votre base de données est très ancienne ou si vous savez que vous avez utilisé des fonctions dépréciées depuis longtemps dans vos vues et fonctions, vous devrez peut-être charger `legacy.sql` pour que toutes vos fonctions, vues, etc. reviennent correctement. Ne le faites que si c'est _vraiment_ nécessaire. Envisagez plutôt de mettre à jour vos vues et fonctions avant de faire le dumping, si possible. Les fonctions dépréciées peuvent être supprimées plus tard en chargeant `uninstall_legacy.sql`.
3. Restaurez votre sauvegarde dans votre nouvelle base de données `newdb` à l'aide de postgis_restore. Les erreurs inattendues, s'il y en a, seront imprimées par psql dans le flux d'erreurs standard. Conservez un journal de ces erreurs.

   Code

   ```
   postgis_restore "/somepath/olddb.backup" | psql -h localhost -p 5432 -U postgres newdb ↩
       2> errors.txt
   ```

Des erreurs peuvent survenir dans les cas suivants :

1. Certaines de vos vues ou fonctions utilisent des objets PostGIS obsolètes. Pour y remédier, vous pouvez essayer de charger le script `legacy.sql` avant la restauration ou vous devrez restaurer vers une version de PostGIS qui contient encore ces objets et réessayer une migration après le portage de votre code. Si la méthode `legacy.sql` fonctionne pour vous, n'oubliez pas de corriger votre code pour arrêter d'utiliser les fonctions dépréciées et de les supprimer en chargeant `uninstall_legacy.sql`.

2. Certains enregistrements personnalisés de spatial_ref_sys dans le fichier dump ont une valeur SRID invalide. Les valeurs SRID valides sont supérieures à 0 et inférieures à 999000. Les valeurs comprises entre 999000 et 999999 sont réservées à un usage interne, tandis que les valeurs > 999999 ne peuvent pas être utilisées du tout. Tous vos enregistrements personnalisés avec des SRID invalides seront conservés, avec les valeurs > 999999 déplacées dans la plage réservée, mais la table spatial_ref_sys perdrait une contrainte de contrôle qui protège cet invariant et peut-être aussi sa clé primaire (lorsque plusieurs SRIDS invalides sont convertis en la même valeur SRID réservée).

   Pour résoudre ce problème, vous devez copier votre SRS personnalisé vers un SRID avec une valeur valide (peut-être dans la plage 910000..910999), convertir toutes vos tables vers le nouveau SRID (voir UpdateGeometrySRID), supprimer l'entrée invalide de spatial_ref_sys et reconstruire le(s) contrôle(s) avec :

   Code

   ```
   ALTER TABLE spatial_ref_sys ADD CONSTRAINT spatial_ref_sys_srid_check check (srid
   > 0 AND srid < 999000 );
   ```

   Code

   ```
   ALTER TABLE spatial_ref_sys ADD PRIMARY KEY(srid));
   ```

   Si vous mettez à jour une ancienne base de données contenant une cartographie française IGN , vous aurez probablement des SRID hors limites et vous verrez, lors de l'importation de votre base de données, des problèmes comme celui-ci :

   Code

   ```
   WARNING: SRID 310642222 converted to 999175 (in reserved zone)
   ```

   Dans ce cas, vous pouvez essayer les étapes suivantes : d'abord supprimer complètement l'IGN du sql qui résulte de postgis_restore. Ainsi, après avoir exécuté :

   Code

   ```
   postgis_restore "/somepath/olddb.backup" > olddb.sql
   ```

   Exécutez cette commande :

   Code

   ```
   grep -v IGNF olddb.sql > olddb-without-IGN.sql
   ```

   Create your `newdb`, activate the required PostGIS extensions, and insert the French IGN spatial reference definitions using this script. After these operations, import your data:

   Code

   ```
   psql -h localhost -p 5432 -U postgres -d newdb -f olddb-without-IGN.sql  2> errors.txt
   ```

Chapitre 4

Gestion des données

4.1 Modèle de données spatiales

4.1.1 Géométrie OGC

L'Open Geospatial Consortium (OGC) a développé le standard *Simple Features Access* (SFA) pour fournir un modèle de données géospatiales. Ce standard définit le type spatial **Geometry**, ainsi que les opérations pour manipuler et transformer des géométries, et permettre des tâches d'analyses spatiales. PostGIS implémente le modèle OGC Geometry sous forme de types PostgreSQL geometry et geography.

Geometry est un type *abstrait*. Les valeurs géométriques utilisent les sous-types *concrets*, qui représentent les diverses formes géométriques. Ces types incluent les types **atomiques** Point, LineString, LinearRing et Polygon, ainsi que les types de **collections** MultiPoint, MultiLineString, MultiPolygon et GeometryCollection. Le standard *Simple Features Access - Part 1: Common architecture v1.2.1* ajoute les sous-types pour les structures PolyhedralSurface, Triangle et TIN.

Geometry représente des formes dans le plan cartésien en 2 dimensions. Les types PolyhedralSurface, Triangle, et TIN peuvent également représenter des formes en 3 dimensions. La taille et la localisation des formes sont spécifiées par leurs **coordonnées**. Chaque coordonnées a une **dimension** X et une Y, qui déterminent sa position sur le plan. Les formes sont construites à partir de points ou de segments. Les points sont spécifiés par une seul coordonnée ; les segments par deux coordonnées.

Coordinates may contain optional Z and M ordinate values. The Z ordinate is often used to represent elevation. The M ordinate contains a measure value, which may represent time or distance. If Z or M values are present in a geometry value, they must be defined for each point in the geometry. If a geometry has Z or M ordinates the **coordinate dimension** is 3D; if it has both Z and M the coordinate dimension is 4D. The coordinate dimension is at least 2, because every geometry has at least X and Y coordinates.

Les valeurs géométriques sont associées à un **système de coordonnées de référence** (SCR, ou en anglais spatial reference system, SRS), qui indique dans quel système de coordonnées les valeurs sont définies. Le SCR est identifié par un identifiant appelé SRID. Les unités sur les axes X et Y sont déterminées par ce système de coordonnées de référence. Dans un système **planaire**, les coordonnées X et Y représentent les distances respectivement selon l'Est et le Nord. Dans un système **géodésique** elles représentent la longitude et la latitude. L'identifiant SRID 0 représente un plan cartésien infini, sans unité sur ses axes. Voir Section 4.5.

The geometry **dimension** is a property of geometry types. Point types have dimension 0, linear types have dimension 1, polygonal types have dimension 2, and solid types have dimension 3. Collections have the dimension of the maximum element dimension.

Une valeur géométrique peut être **vide**. Une valeur vide ne contient aucun vertex (pour les types de géométriques atomiques) ou aucun élément (pour les collections).

Une propriété important des valeurs géométriques est leur **emprise** spatiale (extent en anglais) ou leur **boîte englobante** (bounding box en anglais), que le modèle OGC nomme **enveloppe** (envelope). La boîte englobante est la boîte 2D ou 3D qui contient les coordonnées d'une géométrie. C'est une façon efficace de représenter l'emprise d'une géométrie dans un espace et de tester comment deux géométries interagissent.

The geometry model allows evaluating topological spatial relationships as described in Section 5.1.1. To support this the concepts of **interior**, **boundary** and **exterior** are defined for each geometry type. Geometries are topologically closed, so they always contain their boundary. The boundary is a geometry of dimension one less than that of the geometry itself. For example, the boundary of a Polygon is its rings (LineStrings), the boundary of a LineString is its endpoints (Points), and the boundary of a Point is empty.

The OGC geometry model defines validity rules for each geometry type. These rules ensure that geometry values represents realistic situations (e.g. it is possible to specify a polygon with a hole lying outside its shell, but this makes no sense geometrically and is thus invalid). PostGIS also allows storing and manipulating invalid geometry values. This allows detecting and fixing them if needed. See Section 4.4

4.1.1.1 Point

Un objet de type Point est une géométrie de dimension 0, qui représente un seul point dans l'espace.

Code

```
POINT (1 2)
POINT Z (1 2 3)
POINT ZM (1 2 3 4)
```

4.1.1.2 LineString

Le type LineString est une ligne, de dimension 1, formée par une séquence de segments linéaires contigus. Chaque segment est défini par deux points, l'extrémité d'un segment étant le point de départ du segment suivant. Un LineString valide au sens OGC a soit zéro, soit deux points ou plus, mais PostGIS permet d'avoir des LineStrings avec un seul point. LineStrings peuvent se croiser (auto-intersection). Un LineString peut être **fermé** si les points de début et fin sont les mêmes. Un LineString est dit **simple** s'il ne s'auto-intersecte pas.

Code

```
LINESTRING (1 2, 3 4, 5 6)
```

4.1.1.3 LinearRing

Le type LinearRing définit un LineString qui est à la fois fermé et simple. Autrement dit, le premier et dernier points doivent être égaux, et la ligne ne doit pas s'auto-intersecter.

Code

```
LINEARRING (0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0)
```

4.1.1.4 Polygon

Un polygone, de type Polygon est une région d'un plan, de dimension 2, délimité par une frontière extérieure (la coquille, shell en anglais) et zéro, une ou plusieurs frontières intérieures (trous, holes en anglais). Chaque frontière est un LinearRing.

Code

```
POLYGON ((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0), (1 1 0, 2 1 0, 2 2 0, 1 2 0, 1 1 0))
```

4.1.1.5 MultiPoint

Le type MultiPoint est une collection de Points.

Code

```
MULTIPOINT ( (0 0), (1 2) )
```

4.1.1.6 MultiLineString

Le type MultiLineString est une collection de LineStrings. Un MultiLineString est fermé si tous ses éléments sont fermés.

Code

```
MULTILINESTRING ( (0 0, 1 1, 1 2), (2 3, 3 2, 5 4) )
```

4.1.1.7 MultiPolygon

A MultiPolygon is a collection of non-overlapping, non-adjacent Polygons. Polygons in the collection may touch only at a finite number of points. (Two polygons are adjacent if they share an edge. They touch if they share only points or edges on their boundaries). For more details on MultiPolygon validity, see Section 4.4.

Code

```
MULTIPOLYGON (((1 5, 5 5, 5 1, 1 1, 1 5)), ((6 5, 9 1, 6 1, 6 5)))
```

4.1.1.8 GeometryCollection

Le type GeometryCollection représente une collection hétérogène de géométries (i.e. de types différents).

Code

```
GEOMETRYCOLLECTION ( POINT(2 3), LINESTRING(2 3, 3 4))
```

4.1.1.9 PolyhedralSurface

Le type PolyhedralSurface modélise une surface polyédrique, sous la forme d'une collection de faces qui partagent des arêtes. Chaque face est un Polygon plan. Si les coordonnées du Polygon ont une dimension Z, alors la surface est de dimension 3.

Code

```
POLYHEDRALSURFACE Z (
((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )
```

4.1.1.10 Triangle

Un Triangle est un polygone définit par trois sommets non colinéaires. Un Triangle étant un Polygon, et donc est fermé, il est définit par quatre coordonnées, la première et la dernière étant égales.

Code

```
TRIANGLE ((0 0, 0 9, 9 0, 0 0))
```

4.1.1.11 TIN

Un TIN est une collection de Triangles non superposés, représentant un réseau irrégulier triangulé (Triangulated Irregular Network).

Code

```
TIN Z ( ((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)) )
```

4.1.2 SQL/MM Part 3 - Courbes

La norme *ISO/IEC 13249-3 SQL Multimedia - Spatial* (SQL/MM) étend l'OGC SFA pour définir des sous-types de géométrie contenant des courbes avec des arcs circulaires. Les types SQL/MM prennent en charge les coordonnées 3DM, 3DZ et 4D.



Note
Toutes les comparaisons en flottant dans l'implémentation de SQL-MM sont effectuées avec une tolérance spécifiée, actuellement 1E-8.

4.1.2.1 CircularString

CircularString est le type de courbe de base, similaire à LineString dans le monde linéaire. Un segment d'arc unique est spécifié par trois points : les points de départ et d'arrivée (premier et troisième) et un autre point de l'arc. Pour spécifier un cercle fermé, les points de départ et d'arrivée sont les mêmes et le point central est le point opposé sur le diamètre du cercle (qui est le centre de l'arc). Dans une séquence d'arcs, le point final de l'arc précédent est le point de départ de l'arc suivant, tout comme les segments d'une LineString. Cela signifie qu'une CircularString doit avoir un nombre impair de points supérieur à 1.

Code

```
CIRCULARSTRING(0 0, 1 1, 1 0)

CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0)
```

4.1.2.2 CompoundCurve

Une CompoundCurve est une courbe continue unique qui peut contenir à la fois des segments d'arc de cercle et des segments linéaires. Cela signifie qu'en plus d'avoir des composantes bien formées, le point final de chaque composante (sauf la dernière) doit coïncider avec le point de départ de la composante suivante.

Code

```
COMPOUNDCURVE( CIRCULARSTRING(0 0, 1 1, 1 0), (1 0, 0 1))
```

4.1.2.3 CurvePolygon

Un CurvePolygon est semblable à un polygone, avec un anneau extérieur et zéro ou plusieurs anneaux intérieurs. La différence est qu'un anneau peut être une CircularString ou une CompoundCurve ainsi qu'une LineString.

Depuis PostGIS 1.4, PostGIS prend en charge les courbes composées dans un polygone de courbe.

Code

```
CURVEPOLYGON(
CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),
(1 1, 3 3, 3 1, 1 1) )
```

Exemple : Un PolygoneCourbe dont l'enveloppe est définie par une CompoundCurve contenant une CircularString et une LineString, et dont le trou est défini par une CircularString

Code

```
CURVEPOLYGON(
COMPOUNDCURVE( CIRCULARSTRING(0 0, 2 0, 2 1, 2 3, 4 3),
               (4 3, 4 5, 1 4, 0 0)),
CIRCULARSTRING(1.7 1, 1.4 0.4, 1.6 0.4, 1.6 0.5, 1.7 1) )
```

4.1.2.4 MultiCurve

Une MultiCurve est un ensemble de courbes qui peuvent inclure des LineStrings, des CircularStrings ou des CompoundCurves.

Code

```
MULTICURVE( (0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
```

4.1.2.5 MultiSurface

Une MultiSurface est un ensemble de surfaces, qui peuvent être des polygones (linéaires) ou des polygones courbes.

Code

```
MULTISURFACE(
CURVEPOLYGON(
  CIRCULARSTRING( 0 0, 4 0, 4 4, 0 4, 0 0),
  (1 1, 3 3, 3 1, 1 1)),
((10 10, 14 12, 11 10, 10 10), (11 11, 11.5 11, 11 11.5, 11 11)))
```

4.1.3 WKT et WKB

La spécification OGC SFA définit deux formats pour représenter des valeurs géométriques : Well-Known Text (WKT) et Well-Known Binary (WKB). Ces deux formats incluent les informations sur le type d'objet et sur les coordonnées qui le définissent.

Well-Known Text (WKT) fournit un standard pour représenter de façon textuelle des données spatiales. Voici quelques exemples de représentations WKT :

— POINT(0 0)

— POINT Z (0 0 0)

— POINT ZM (0 0 0 0)

— POINT EMPTY

— LINESTRING(0 0,1 1,1 2)

— LINESTRING EMPTY

— POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))

— MULTIPOINT((0 0),(1 2))

— MULTIPOINT Z ((0 0 0),(1 2 3))

— MULTIPOINT EMPTY

— MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))

— MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))

— GEOMETRYCOLLECTION(POINT(2 3), LINESTRING(2 3,3 4))

— GEOMETRYCOLLECTION EMPTY

Des méthodes d'entrée/sortie en WKT sont fournies via les fonctions ST_AsText et ST_GeomFromText :

Code

```
text WKT = geometry;
geometry = ST_GeomFromText(text WKT, SRID);
```

For example, a statement to create and insert a spatial object from WKT with SRID 312 is:

Code

```
INSERT INTO geotable ( geom, name )
VALUES ( ST_GeomFromText('POINT(-126.4 45.32)', 312), 'A Place');
```

Well-Known Binary (WKB) fournit un moyen portable et sans perte de précision pour représenter des données spatiales sous la forme de données binaires (tableau d'octets). Voici quelques exemples de représentations WKB :

— WKT : POINT(1 1)

WKB : 01010000000000000000F03F000000000000F03

— WKT : LINESTRING (2 2, 9 9)

WKB : 0102000000020000000000000000000040000000000000004000000000000022400000000000002240

Des méthodes d'entrée/sortie en WKB sont fournies via les fonctions ST_AsBinary et ST_GeomFromWKB :

Code

```
bytea WKB = ST_AsBinary(geometry);
geometry = ST_GeomFromWKB(bytea WKB, SRID);
```

Par exemple, une requête pour créer et insérer une objet spatial sous forme de WKB :

Code

```
INSERT INTO geotable ( geom, name )
VALUES ( ST_GeomFromWKB('\x01010000000000000000f03f000000000000f03f', 312), 'A Place');
```

4.2 Type de données Geometry

PostGIS implémente le modèle Simple Features de l'OGC via un type PostgreSQL `geometry`. Ce type représente tous les sous-types de géométries en utilisant un type interne (voir GeometryType et ST_GeometryType). Cela permet de modéliser les entités spatiales comme lignes de tables qui contiennent une colonne de type `geometry`.

The `geometry` data type is *opaque*, which means that all access is done via invoking functions on geometry values. Functions allow creating geometry objects, accessing or updating all internal fields, and computing new geometry values. PostGIS supports all the functions specified in the OGC *Simple feature access - Part 2: SQL option* (SFS) specification, as well many others. See Chapitre 7 for the full list of functions.



Note
PostGIS respecte le standard SFA en préfixant toutes les fonctions spatiales par "ST_". Initialement, cela voulait dire "Spatial and Temporal" (Spatial et Temporel), mais l'aspect temporel du standard n'a pas été développé. À la place, ceci peut être interprété comme "Spatial Type" (Type Spatial).

La standard SFA spécifie que les objets spatiaux doivent inclure un identifiant de système de coordonnées de référence (SRID). Ce SRID est obligatoire lors de la création d'objets spatiaux pour l'insertion dans la base de données (mais peut être défini par défaut à 0). Voir ST_SRID et Section 4.5

Pour optimiser les requêtes sur les géométries, PostGIS définit plusieurs types d'index spatiaux, ainsi que des opérateurs spatiaux pour les utiliser. Voir Section 4.9 et Section 5.2 pour plus d'informations.

4.2.1 PostGIS EWKB et EWKT

Les spécifications OGC SFA ne supportaient initialement que les géométries 2D, et le SRID de la géométrie n'est pas inclut dans les représentations d'entrée/sortie. La spécification OGC SFA 1.2.1 (qui est alignée avec le standard ISO 19125) ajoute le support pour la 3D (ZYZ) et les mesures (XYM et XYZM), mais n'inclut toujours pas la valeur du SRID.

À cause de ces limitations, PostGIS définit les formats étendus EWKB ((Extended Well-Known Binary) et EWKT (Extended Well-Known Text). Ils supportent la 3D (XYZ et XYM) et 4D (XYZM) et incluent l'information de SRID. Le format EWKB incluant toutes les informations de la géométrie, cela permet à PostGIS d'utiliser ce format pour les enregistrements (e.g. dans les fichiers DUMP).

EWKB et EWKT sont utilisés pour les "formes canoniques" des objets spatiaux de PostGIS. En tant qu'entrée, la forme canonique pour les données binaires est EWKB ; pour les données textuelles EWKB et EWKT sont tous deux acceptés. Cela permet de créer des valeurs géométriques en transtypant une valeur textuelle en HEXEWKB ou EWKT vers une valeur géométrique en utilisant `::geometry`. Pour la sortie, la forme canonique pour les données binaires est EWKB et HEXEWKB (hex-encoded EWKB) pour les données textuelles.

Par exemple, cette requête créé une géométrie en transtypant depuis une chaîne de caractères contenant du EKWT, et retourne sa valeur sous sa forme canonique en HEXEWKB :

Code

```
SELECT 'SRID=4;POINT(0 0)'::geometry;
```

Output

```
geometry
----------------------------------------------------
0101000020040000000000000000000000000000000000000000
```

La sortie PostGIS EWKT a quelques différences avec le OGC WKT :

— Pour les géométries 3DZ, le qualificatif Z est omis :

  OGC : POINT Z (1 2 3)

  EWKT : POINT (1 2 3)

— Pour les géométries 3DM, le qualificatif M est inclus :

  OGC : POINT M (1 2 3)

  EWKT : POINTM (1 2 3)

— Pour les géométries 4D, le qualificatif ZM est omis :

  OGC : POINT ZM (1 2 3 4)

  EWKT : POINT (1 2 3 4)

EWKT évite de sur-spécifier les dimensions et ainsi éviter les inconsistances possibles avec le format OGC/ISO, comme :

— POINT ZM (1 1)

— POINT ZM (1 1 1)

— POINT (1 1 1 1)



Attention
Les formats étendus PostGIS sont des sur-ensembles des formats OGC, donc les représentations valides WKB/WKT sont aussi des représentations EWKB/EWKT valides. Cependant, cela pourrait changer à l'avenir, si l'OGC définit un format qui rentrerait en conflit avec la définition de PosGIS. Vous ne devriez donc PAS vous appuyer sur cette compatibilité !

Voici quelques exemples de représentations d'objets spatiaux sous forme EWKT :

— POINT(0 0 0) -- XYZ

— SRID=32632;POINT(0 0) -- XY avec SRID

— POINTM(0 0 0) -- XYM

— POINT(0 0 0 0) -- XYZM

— SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM avec SRID

— MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))

— POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))

— MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))

— GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))

— MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))

— POLYHEDRALSURFACE(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))

— TRIANGLE ((0 0, 0 10, 10 0, 0 0))

— TIN(((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)))

Des méthodes d'entrée/sortie sont fournies via les fonctions suivantes :

Code

```
bytea EWKB = ST_AsEWKB(geometry);
text EWKT = geometry;
geometry = ST_GeomFromEWKB(bytea EWKB);
geometry = ST_GeomFromEWKT(text EWKT);
```

Par exemple, une requête pour créer et insérer une objet spatial sous forme de EWKT :

Code

```
INSERT INTO geotable ( geom, name )
VALUES ( ST_GeomFromEWKT('SRID=312;POINTM(-126.4 45.32 15)'), 'A Place' )
```

4.3 Type de données Geography

Le type de données PostGIS `geography` permet le support des entités spatiales utilisant un système de coordonnées géographique (parfois appelé géodésique, ou "latitude/longitude" ou "longitude/latitude"). Les coordonnées géographiques sont des coordonnées sphériques, exprimées en unités d'angle (degrés).

Le type geometry de PostGIS est lié à un plan. Ainsi, le chemin le plus court entre deux points sur un plan est la ligne droite. Les fonctions sur les types geometry (aires, distances, intersections, etc.) sont donc calculées en utilisant des lignes droites et les mathématiques cartésiennes. Cela permet des implémentations plus faciles et plus rapides à exécuter, mais cela devient imprécis lorsque la rotondité de la Terre entre en jeu.

Le type PostGIS geography repose sur un modèle sphérique. Le chemin le plus court entre deux points sur une sphère est l'arc de cercle. Les fonctions sur les types geography (aires, distances, intersections, etc.) sont donc calculées en utilisant des arcs de cercle sur une sphère. En prenant en compte la rotondité de la Terre, ces fonctions permettent d'avoir une meilleure précision.

Les mathématiques utilisées pour les calculs étant plus compliquées, moins de fonctions sont définies pour le type geography que pour le type geometry. De nouveaux algorithmes sont ajoutés au fur et à mesure des versions de PostGIS, donc le support du type geography s'étend petit à petit. Si une fonction n'est pas disponible, il est toutefois possible de convertir un type geography en geometry puis vice-versa.

Like the geometry data type, geography data is associated with a spatial reference system via a spatial reference system identifier (SRID). Any geodetic (long/lat based) spatial reference system defined in the `spatial_ref_sys` table can be used. You can add your own custom geodetic spatial reference system as described in Section 4.5.2.

Pour tous les systèmes de coordonnées de référence, les unités des valeurs retournées par les fonctions de mesure (e.g. ST_Distance, ST_Length, ST_Perimeter, ST_Area) et le paramètre de distance de ST_DWithin sont en mètres.

4.3.1 Création de tables géographiques

Vous pouvez créer une table pour stocker des données géographiques en utilisant la requête SQL CREATE TABLE, avec une colonne de type `geography`. L'exemple suivant créé une table avec une colonne géographique pour stocker des lignes 2D dans un SCR géodésique WGS84 (SRID 4326) :

Code

```
CREATE TABLE global_points (
  id SERIAL PRIMARY KEY,
  name VARCHAR(64),
  location geography(POINT, 4326)
);
```

Le type geography supporte deux modificateurs de type :

— le modificateur de type spatial restreint le type de formes et dimensions de la colonne. Les valeurs permises pour le type spatial sont : POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION. Le type geography ne supporte pas les CURVEs, TINS, ni POLYHEDRALSURFACEs. Le modificateur permet de restreindre la dimension, en ajoutant les suffixes : Z, M ou ZM. Par exemple, un modificateur 'LINESTRINGM' permet uniquement de stocker des LineStrings en 3D, et traite la troisième dimension comme une mesure. De façon similaire, 'POINTZM' limite aux données 4D (XYZM).

— le modificateur SRID restreint à un SCR spécifique. Si omis, le SRID 4326 (WGS84 géodésique) est utilisé, et tous les calculs sont effectués en utilisant WGS84.

Exemples de création de tables utilisant des colonnes géographiques :

Création d'une table avec une géographie POINT 2D avec le SRID par défaut 4326 (WGS84 longitude/latitude) :

Code

```
CREATE TABLE ptgeogwgs(gid serial PRIMARY KEY, geog geography(POINT));
```

Création d'une table avec une géographie POINT 2D dans le CRS NAD83 longlat :

Code

```
CREATE TABLE ptgeognad83(gid serial PRIMARY KEY, geog geography(POINT, 4269));
```

Création d'une table avec une géographie POINT 3D (XYZ) avec le SCR explicite 4326 :

Code

```
CREATE TABLE ptzgeogwgs84(gid serial PRIMARY KEY, geog geography(POINTZ, 4326));
```

Création d'une table avec une géographie LINESTRING 2D avec le SRID par défaut 4326 :

Code

```
CREATE TABLE lgeog(gid serial PRIMARY KEY, geog geography(LINESTRING));
```

Création d'une table avec une géographie POLYGON 2D avec le SRC 4267 (NAD 1927 long lat) :

Code

```
CREATE TABLE lgeognad27(gid serial PRIMARY KEY, geog geography(POLYGON, 4267));
```

Les colonnes géographiques sont enregistrées dans la vue système `geography_columns`. Vous pouvez effectuer un requête sur la vue `geography_columns` et vérifier que la table est bien listée :

Code

```
SELECT * FROM geography_columns;
```

La création d'index spatiaux sur les colonnes de type geography fonctionne de la même manière qu'avec le type geometry. PostGIS va prendre en compte que le type de la colonne est GEOGRAPHY et créer un index sphérique au lieu d'un index planaire utilisé pour GEOMETRY.

Code

```
CREATE INDEX global_points_gix ON global_points USING GIST ( location );
```

4.3.2 Utilisation des tables géographiques

Vous pouvez insérer des données dans des tables géographiques de la même façon qu'avec les géométries. Les données géométriques seront automatiquement transtypées en type geography si le SRID des données est 4326. Les formats EWKT et EWKB peuvent aussi être utilisés pour spécifier les valeurs géographiques.

Code

```
INSERT INTO global_points (name, location) VALUES ('Town', 'SRID=4326;POINT(-110 30)');
INSERT INTO global_points (name, location) VALUES ('Forest', 'SRID=4326;POINT(-109 29)');
INSERT INTO global_points (name, location) VALUES ('London', 'SRID=4326;POINT(0 49)');
```

Tout CRS géodésique (longitude/latitude) disponible dans la table `spatial_ref_sys` peut être utilisé comme SRID d'une géographie. L'utilisation d'un CRS non géodésique provoquera une erreur.

A NAD83 longitude/latitude geography value:

Code

```
SELECT 'SRID=4269;POINT(-123 34)'::geography;
```

Output

```
geography
----------------------------------------------------
 0101000020AD10000000000000C05EC00000000000004140
```

A NAD27 longitude/latitude geography value:

Code

```
SELECT 'SRID=4267;POINT(-123 34)'::geography;
```

Output

```
geography
----------------------------------------------------
 0101000020AB1000000000000000C05EC00000000000004140
```

NAD83 UTM zone meters fails because it is a meter-based planar projection.

Code

```
SELECT 'SRID=26910;POINT(-123 34)'::geography;
```

Output

```
ERROR:  Only lon/lat coordinate systems are supported in geography.
```

Les requêtes et les fonctions de mesure utilisent le mètre comme unité. Les paramètres de distance doivent être exprimés en mètres, et les valeurs de retours seront également en mètres (ou en mètres carré pour les surfaces).

This query finds rows within a 1000 km tolerance:

Code

```
SELECT name FROM global_points WHERE ST_DWithin(location, 'SRID=4326;POINT(-110 29)':: ↩
    geography, 1000000);
```

Vous pouvez vérifier la puissance des géographies en calculant à quel point un avion s'approche de Reykjavik (POINT(-21.96 64.15)) lors d'un trajet en arc de cercle depuis Seattle à Londres (LINESTRING(-122.33 47.606, 0.0 51.5)) (voir la route).

Le type geography donne la plus petite distance réelle de 122,235 km sur la sphère entre Reykjavik et l'arc de cercle entre Seattle et Londres.

Code

```
SELECT ST_Distance('LINESTRING(-122.33 47.606,0.0 51.5)'::geography, 'POINT(-21.96 64.15)':: ↩
    geography);
```

Output

```
st_distance
-----------------
 122235.23815667
```

Le type geometry quant à lui calcule la distance cartésienne entre Reykjavik et la ligne droite entre Seattle et Londres, telle qu'elle serait tracée sur un plan. Cette distance n'a pas de sens réel, d'autant que l'unité du résultat est techniquement en degrés, mais que le résultat ne correspond à aucune différence angulaire entre les points, donc même considérer le résultat comme des degrés serait faux.

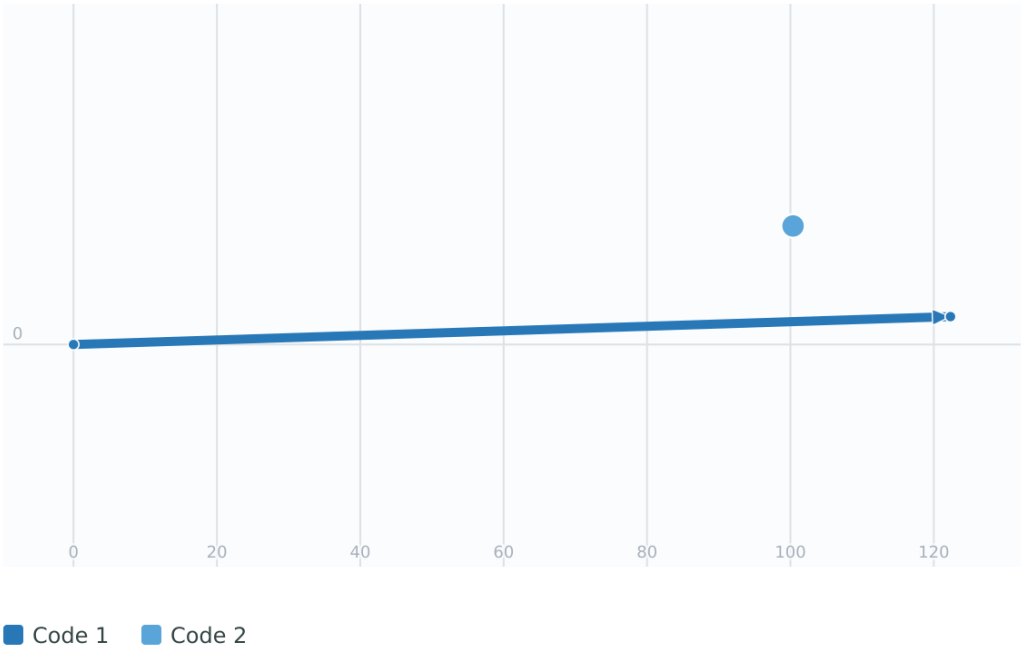
Code

```
SELECT ST_Distance('LINESTRING(-122.33 47.606,0.0 51.5)'::geometry, 'POINT(-21.96 64.15)':: ↩
    geometry);
```

Output

```
st_distance
-------------------
 13.342271221453624
```

Figure



4.3.3 Quand utiliser le type de données Geography

Le type de données geography permet de stocker des données en utilisant les coordonnées longitude/latitude, mais cela a un prix : il y a moins de fonctions disponibles sur le type GEOGRAPHY que sur le type GEOMETRY, et les fonctions disponibles prendront plus de temps CPU pour s'exécuter.

Le type de données à choisir devrait être déterminé par la zone de travail de l'application que vous construisez. Est-ce que vos données s'étendront sur l'ensemble du globe ou sur un continent, ou bien est-ce qu'elle seront locales, comme une région, un département ou une ville ?

— Si vous données sont limitées à une petite zone, utiliser un SCR adéquat et le type GEOMETRY est la meilleure solution, à la fois en terme de performances que de fonctionnalités disponibles.

— Si vos données s'étendent sur le monde entier ou couvrent un continent, le type GEOGRAPHY peut vous permettre de construire votre application sans trop vous soucier des projections. Vous pouvez stocker vos données en utilisant les coordonnées longitude/latitude, et utiliser les fonctions définies sur les GEOGRAPHY.

— Si vous n'êtes pas à l'aise avec les projections, que vous ne souhaitez pas approfondir le sujet, et que vous êtes prêts à accepter les limitations sur les fonctionnalités offertes par GEOGRAPHY, alors ce peut être plus facile d'utiliser GEOGRAPHY au lieu de GEOMETRY. Chargez vos données en longitude/latitude et partez de là.

Référez-vous à Section 13.11 pour comparer les supports Geography et Geometry. Pour avoir un résumé des fonctions géographiques disponibles, référez-vous à Section 13.4

4.3.4 FAQ Geography avancée

1. *Les calculs sont-ils faits sur la sphère ou sur la sphéroïde ?*

   By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up well with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. The difference is that the spheroid takes into account the curvature and non-spherical shape of the Earth, whereas planar calculations assume a flat surface. Toutes les fonctions géographiques ont une option pour calculer sur une sphère, en passant comme tout dernier paramètre booléen 'FALSE'. Ceci améliorera les performances des calculs, en particulier pour les géométries très simples.

2. *Qu'en est-il de la ligne de changement de date et des pôles ?*

   Tous les calculs omettent les concepts de ligne de changement de date et de pôles. Les coordonnées étant sphériques (longitude/latitude), une forme traversant la ligne de changement de date n'est, d'un point de vue calculs, pas différente de toute autre forme.

3. *Quel est l'arc le plus long que vous puissiez traiter ?*

   Nous utilisons les arcs de grand cercle comme "ligne d'interpolation" entre deux points. Cela signifie que deux points quelconques sont en fait reliés de deux manières différentes, selon la direction dans laquelle vous vous déplacez le long du grand cercle. Tout notre code suppose que les points sont reliés par le *plus court* des deux chemins le long du grand cercle. Par conséquent, les formes qui ont des arcs de plus de 180 degrés ne seront pas correctement modélisées.

4. *Pourquoi est-il si lent de calculer la superficie de l'Europe / de la Russie / insérer une grande région géographique ici ?*

   Parce que le polygone est vraiment énorme ! Les grandes zones sont néfastes pour deux raisons : leurs limites sont énormes, de sorte que l'index a tendance à tirer l'élément, quelle que soit la requête que vous exécutez ; le nombre de sommets est énorme, et les tests (distance, confinement) doivent parcourir la liste des sommets au moins une fois et parfois N fois (N étant le nombre de sommets dans l'autre élément candidat). Comme pour la GÉOMÉTRIE, nous vous recommandons, lorsque vous avez de très grands polygones, mais que vous effectuez des requêtes sur de petites zones, de "dénormaliser" vos données géométriques en morceaux plus petits, de sorte que l'index puisse effectivement interroger des parties de l'objet et que les requêtes n'aient pas à extraire l'objet entier à chaque fois. Veuillez consulter la documentation de la fonction ST_Subdivide. Ce n'est pas parce que vous *pouvez* stocker toute l'Europe dans un polygone que vous *devriez* le faire.

4.4 Validation de la géométrie

PostGIS est conforme à la spécification Simple Features de l'Open Geospatial Consortium (OGC). Cette norme définit les concepts de géométrie comme étant *simple* et *valide*. Ces définitions permettent au modèle géométrique Simple Features de représenter les objets spatiaux d'une manière cohérente et non ambiguë qui permet un calcul efficace. (Remarque : l'OGC SF et SQL/MM ont les mêmes définitions pour simple et valide.)

4.4.1 Géométrie simple

Une géométrie *simple* est une géométrie qui ne présente pas de points géométriques anormaux, tels qu'une intersection ou une tangence propre.

Un `POINT` est intrinsèquement *simple* en tant qu'objet géométrique à 0 dimension.

Les `MULTIPOINT` sont *simples* si deux coordonnées (`POINT`s) ne sont pas égales (ont des valeurs de coordonnées identiques).

Une `LINESTRING` est *simple* si elle ne passe pas deux fois par le même point, à l'exception des extrémités. Si les extrémités d'une simple LineString sont identiques, elle est dite *fermée* et appelée anneau linéaire.

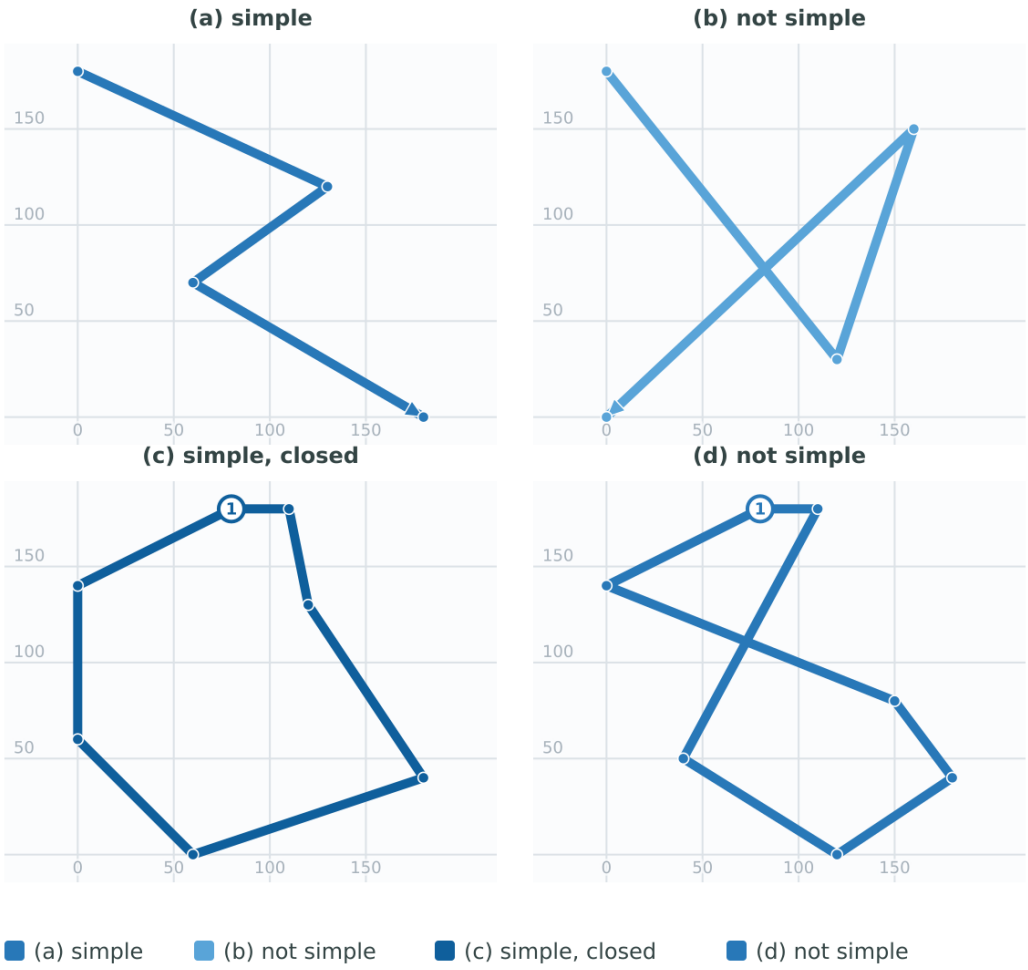
Use ST_IsSimple to classify linear geometries. Each labeled result below is returned only when the function agrees with the stated classification; example **(c)** also verifies that the simple LineString is closed.

Code

```
WITH cases(label, geom) AS (VALUES
  ('a', 'LINESTRING (10 190, 140 130, 70 80, 190 10)'::geometry),
  ('b', 'LINESTRING (10 190, 130 40, 170 160, 10 10)'::geometry),
  ('c', 'LINESTRING (90 190, 120 190, 130 140, 190 50, 70 10, 10 70, 10 150, 90 190)':: ↩
    geometry),
  ('d', 'LINESTRING (90 190, 120 190, 50 60, 130 10, 190 50, 160 90, 10 150, 90 190)':: ↩
    geometry)
)
SELECT
  (SELECT geom FROM cases WHERE label = 'a' AND ST_IsSimple(geom))
    AS "input_(a) simple",
  (SELECT geom FROM cases WHERE label = 'b' AND NOT ST_IsSimple(geom))
```

```
    AS "input_(b) not simple",
  (SELECT geom FROM cases
            WHERE label = 'c' AND ST_IsSimple(geom) AND ST_IsClosed(geom))
    AS "input_(c) simple, closed",
  (SELECT geom FROM cases WHERE label = 'd' AND NOT ST_IsSimple(geom))
    AS "input_(d) not simple";
```

Figure



Un `MULTILINESTRING` est *simple* uniquement si tous ses éléments sont simples et si la seule intersection entre deux éléments quelconques se produit en des points situés sur les limites des deux éléments.

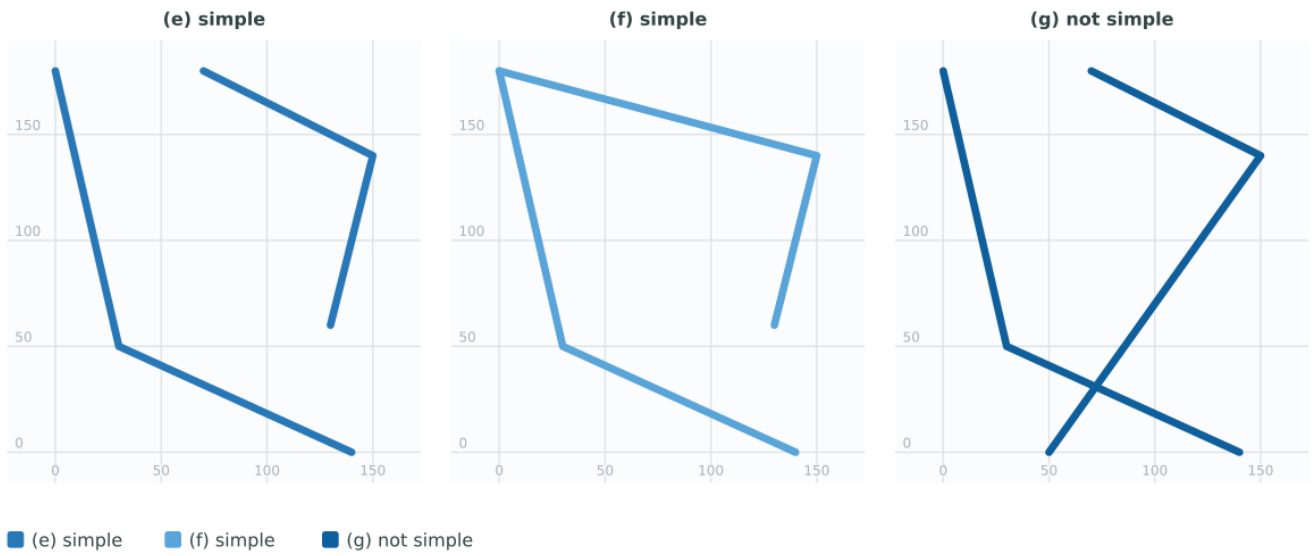
The next example applies the same classification to the three MultiLineStrings. In **(f)** the component lines meet only at a shared endpoint, while in **(g)** they cross in their interiors.

Code

```
WITH cases(label, geom) AS (VALUES
  ('e', 'MULTILINESTRING ((30 190, 60 60, 170 10), (100 190, 180 150, 160 70))'::geometry),
  ('f', 'MULTILINESTRING ((30 190, 60 60, 170 10), (30 190, 180 150, 160 70))'::geometry),
  ('g', 'MULTILINESTRING ((30 190, 60 60, 170 10), (100 190, 180 150, 80 10))'::geometry)
)
SELECT
  (SELECT geom FROM cases WHERE label = 'e' AND ST_IsSimple(geom))
    AS "input_(e) simple",
  (SELECT geom FROM cases WHERE label = 'f' AND ST_IsSimple(geom))
    AS "input_(f) simple",
  (SELECT geom FROM cases WHERE label = 'g' AND NOT ST_IsSimple(geom))
```

```
    AS "input_(g) not simple";
```

Figure



Les POLYGON sont formés à partir d'anneaux linéaires, de sorte que la géométrie polygonale valide est toujours *simple*.

En général, les fonctions PostGIS n'exigent pas que les arguments géométriques soient simples. La simplicité est principalement utilisée comme base pour définir la validité géométrique. C'est également une exigence pour certains types de modèles de données spatiales (par exemple, les réseaux linéaires interdisent souvent les lignes qui se croisent). La géométrie multipoint et linéaire peut être rendue simple en utilisant ST_UnaryUnion.

4.4.2 Géométrie valide

La validité géométrique s'applique principalement aux géométries bidimensionnelles (POLYGONs et MULTIPOLYGONs). La validité est définie par des règles qui permettent à la géométrie polygonale de modéliser des zones planes sans ambiguïté.

Un POLYGON est *valide* si :

1. les anneaux de délimitation du polygone (l'anneau extérieur de la coquille et les anneaux intérieurs des trous) sont *simples* (ils ne se croisent pas et ne se touchent pas). Pour cette raison, un polygone ne peut pas avoir de lignes de coupe, de pointes ou de boucles. Cela implique que les trous des polygones doivent être représentés par des anneaux intérieurs, plutôt que par l'anneau extérieur qui se touche (ce qu'on appelle un "trou inversé").
2. les anneaux de délimitation ne se croisent pas
3. Les anneaux de délimitation peuvent se toucher en certains points, mais uniquement sous la forme d'une tangente (c'est-à-dire pas sous la forme d'une ligne)
4. les anneaux intérieurs sont contenus dans l'anneau extérieur
5. l'intérieur du polygone est simplement connecté (c'est-à-dire que les anneaux ne doivent pas se toucher d'une manière qui divise le polygone en plus d'une partie)

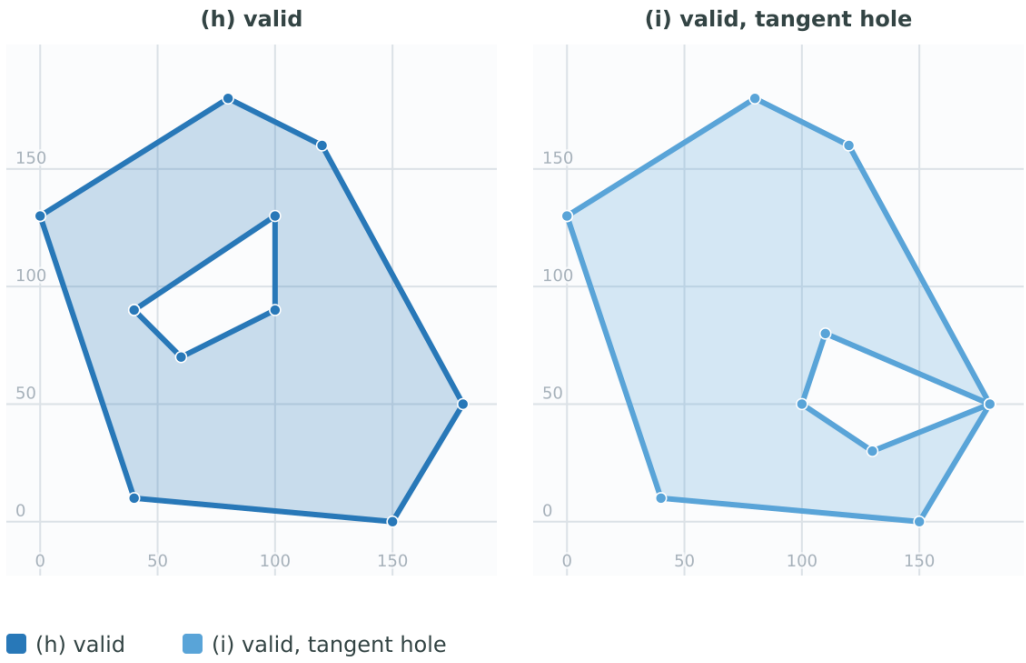
Consecutive repeated points in a polygon boundary ring do not make the ring non-simple. Under the SFS curve model, a curve is interpreted by a continuous parameterization; consecutive equal coordinates are topologically indistinguishable from a single vertex. PostGIS therefore treats them as redundant vertices, so they do not make an otherwise valid polygon invalid.

The following examples use ST_IsValid to check the validity rules. Each labeled geometry is returned only if its computed validity agrees with the label.

Code

```
WITH cases(label, geom) AS (VALUES
  ('h', 'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
                  (50 100, 70 80, 110 100, 110 140, 50 100))'::geometry),
  ('i', 'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
                  (190 60, 140 40, 110 60, 120 90, 190 60))'::geometry)
)
SELECT
  (SELECT geom FROM cases WHERE label = 'h' AND ST_IsValid(geom))
    AS "input_(h) valid",
  (SELECT geom FROM cases WHERE label = 'i' AND ST_IsValid(geom))
    AS "input_(i) valid, tangent hole";
```

Figure



Examples **(j-m)** are invalid. They show, respectively, a disconnected interior, crossing rings, a cut line, and a hole outside the shell. The invalidity reason is checked with ST_IsValidReason. The disconnected interior in **(j)** can instead be represented as a valid `MULTIPOLYGON`.

Code

```
WITH cases(label, geom, reason_pattern) AS (VALUES
  ('(j) disconnected interior',
   'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
             (130 170, 10 140, 50 120, 110 110, 130 170))'::geometry,
   'Interior is disconnected%'),
  ('(k) crossing rings',
   'POLYGON ((10 140, 90 190, 130 170, 190 60, 160 10, 50 20, 10 140),
             (90 189, 10 139, 80 110, 110 130, 90 189))'::geometry,
   'Self-intersection%'),
  ('(l) cut line',
   'POLYGON ((10 100, 60 140, 60 190, 60 140, 130 170,
              190 60, 160 10, 50 20, 10 100))'::geometry,
   'Ring Self-intersection%'),
  ('(m) hole outside shell',
   'POLYGON ((10 40, 10 100, 130 130, 190 80, 140 20, 50 10, 10 40),
             (81 143, 103 143, 112 162, 107 175, 89 183, 71 169, 81 143))'::geometry,
```

```
   'Hole lies outside shell%')
)
SELECT label, geom
FROM cases
WHERE NOT ST_IsValid(geom)
  AND ST_IsValidReason(geom) LIKE reason_pattern
ORDER BY label;
```

Output

```
label          | geom
-----------------------+-----------------------
```

Code

```
WITH cases(label, geom, valid, reason_pattern) AS (VALUES
  ('(n) point touch',
   'MULTIPOLYGON (((10 40, 29 118, 100 75, 180 88, 140 30, 70 10, 10 40)),
                  ((50 170, 100 190, 160 180, 180 140, 180 90, 120 120,
                    30 120, 50 170)))'::geometry,
   true, NULL),
  ('(o) shared edge',
   'MULTIPOLYGON (((10 40, 30 120, 120 120, 180 88, 140 30, 70 10, 10 40)),
                  ((50 170, 100 190, 160 180, 180 140, 180 90, 120 120,
                    30 120, 50 170)))'::geometry,
   false, 'Self-intersection%'),
  ('(p) overlap',
   'MULTIPOLYGON (((10 40, 42 93, 130 150, 180 86, 140 30, 70 10, 10 40)),
                  ((50 170, 100 190, 138 182, 149 156, 146 137, 100 100,
                    58 140, 50 170)))'::geometry,
   false, 'Self-intersection%')
)
SELECT label, geom
FROM cases
WHERE ST_IsValid(geom) = valid
  AND (valid OR ST_IsValidReason(geom) LIKE reason_pattern)
ORDER BY label;
```

Output

```
label       | geom
----------------+-----------------------
```

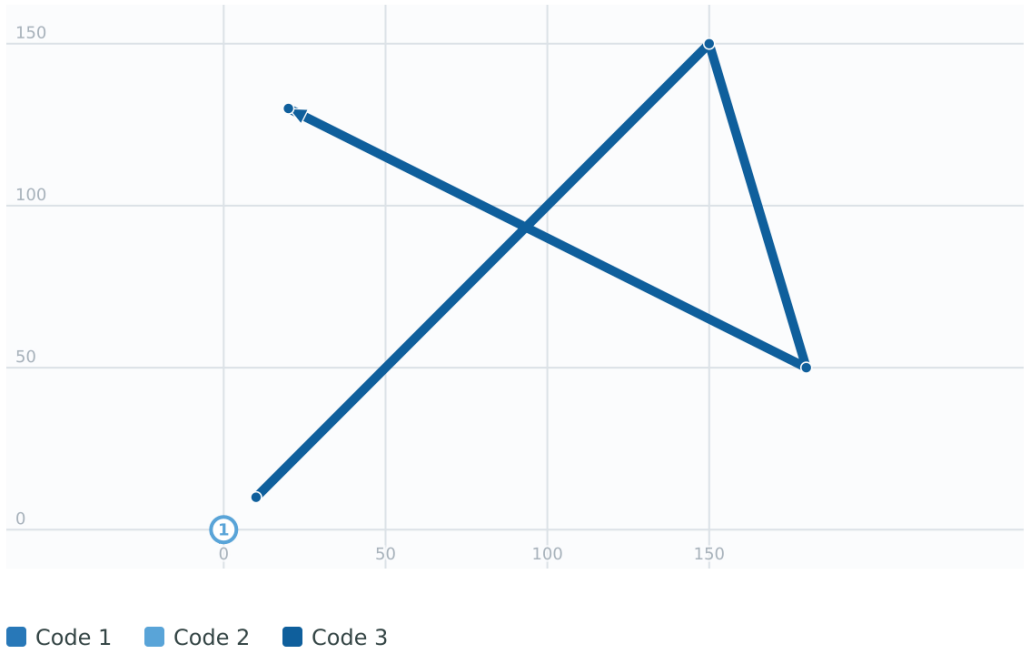
Code

```
SELECT
ST_IsValid('LINESTRING(0 0,1 1)') AS len_nonzero,
ST_IsValid('LINESTRING(0 0,0 0,0 0)') AS len_zero,
ST_IsValid('LINESTRING(10 10,150 150,180 50,20 130)') AS self_int;
```

Output

```
len_nonzero | len_zero | self_int
------------+----------+----------
 t          | f        | t
```

Figure



Les géométries `POINT` et `MULTIPOINT` n'ont pas de règles de validité.

4.4.3 Gestion de la validité

PostGIS permet de créer et de stocker des géométries valides et non valides. Cela permet de détecter les géométries non valides et de les signaler ou de les corriger. Il existe également des situations où les règles de validité de l'OGC sont plus strictes que souhaité (par exemple, les lignes de longueur nulle et les polygones avec des trous inversés.)

De nombreuses fonctions fournies par PostGIS reposent sur l'hypothèse que les arguments géométriques sont valides. Par exemple, il n'est pas logique de calculer la surface d'un polygone dont le trou est défini à l'extérieur du polygone, ou de construire un polygone à partir d'une ligne de démarcation non simple. Le fait de supposer que les entrées géométriques sont valides permet aux fonctions de fonctionner plus efficacement, puisqu'elles n'ont pas besoin de vérifier l'exactitude topologique. (Les exceptions notables sont que les lignes de longueur nulle et les polygones avec des inversions sont généralement traités correctement). En outre, la plupart des fonctions PostGIS produisent une géométrie valide si les données d'entrée sont valides. Cela permet d'enchaîner les fonctions PostGIS en toute sécurité.

Si vous rencontrez des messages d'erreur inattendus lors de l'appel de fonctions PostGIS (tels que "GEOS Intersection() threw an error !"), vous devez d'abord vous assurer que les arguments de la fonction sont valides. Si ce n'est pas le cas, envisagez d'utiliser l'une des techniques ci-dessous pour vous assurer que les données que vous traitez sont valides.



Note
Si une fonction signale une erreur avec des entrées valides, il se peut que vous ayez trouvé une erreur dans PostGIS ou dans l'une des bibliothèques qu'il utilise, et vous devez le signaler au projet PostGIS. Il en va de même si une fonction PostGIS renvoie une géométrie non valide pour des données d'entrée valides.

Pour vérifier si une géométrie est valide, utilisez la fonction ST_IsValid :

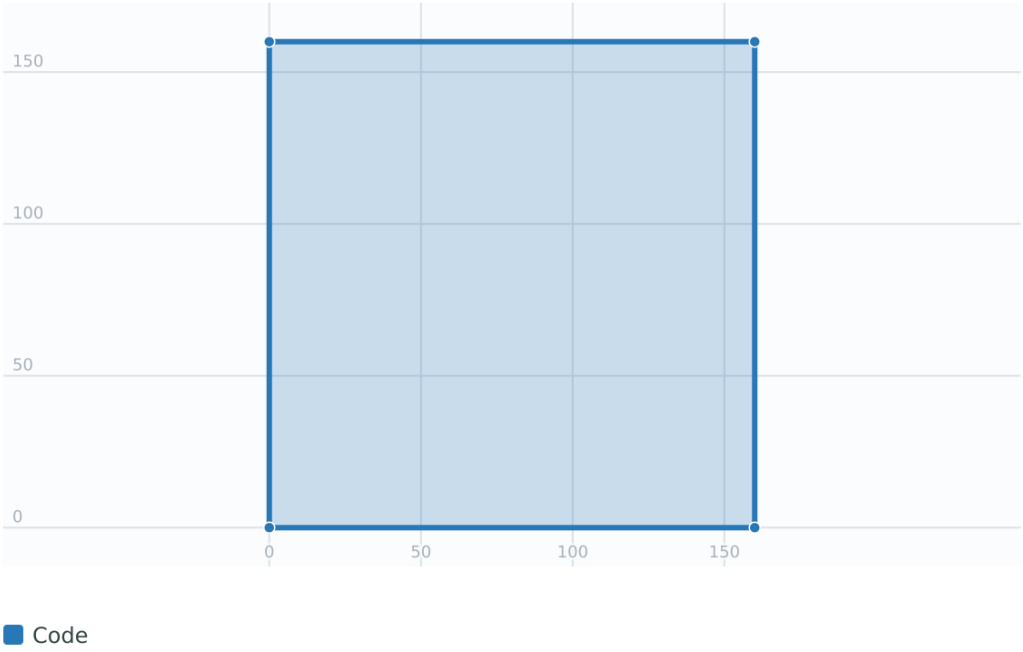
Code

```
SELECT ST_IsValid('POLYGON ((20 180,180 180,180 20,20 20,20 180))');
```

Output

```
t
```

Figure



Les informations relatives à la nature et à la localisation d'une invalidité géométrique sont fournies par la fonction ST_IsValidDetail :

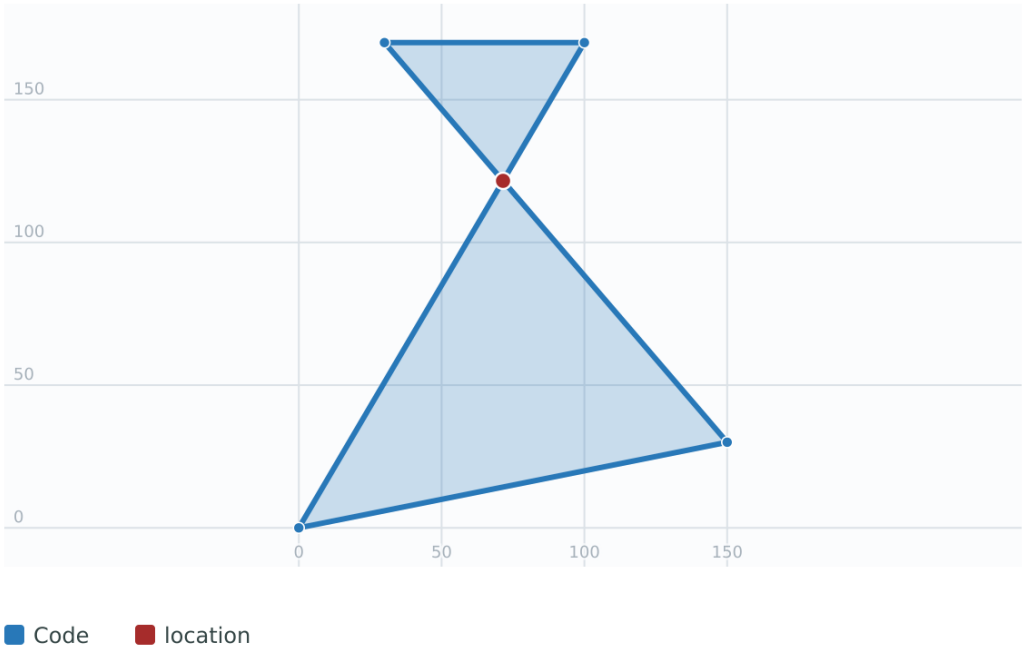
Code

```
SELECT valid, reason,location AS location
        FROM ST_IsValidDetail('POLYGON ((20 20,120 190,50 190,170 50,20 20))') AS t;
```

Output

```
valid |      reason       |                  location
-------+-------------------+---------------------------------------------
 f     | Self-intersection | POINT(91.51162790697674 141.56976744186048)
```

Figure



Dans certaines situations, il est souhaitable de corriger automatiquement la géométrie non valide. Utilisez la fonction ST_MakeValid pour cela. (`ST_MakeValid` est un exemple de fonction spatiale qui *doit* permettre une entrée non valide !)

Par défaut, PostGIS ne vérifie pas la validité lors du chargement de la géométrie, car les tests de validité peuvent prendre beaucoup de temps CPU pour les géométries complexes. Si vous ne faites pas confiance à vos sources de données, vous pouvez imposer un contrôle de validité à vos tables en ajoutant une contrainte de contrôle :

Code

```
ALTER TABLE mytable
ADD CONSTRAINT geometry_valid_check
  CHECK (ST_IsValid(geom));
```

4.5 Systèmes de référence spatiale

Un système de référence spatiale (SRS) (également appelé système de référence des coordonnées (SRC)) définit la manière dont la géométrie est référencée par rapport à des emplacements sur la surface de la Terre. Il existe trois types de SRS :

— Un SRS **géodésique** utilise des coordonnées angulaires (longitude et latitude) qui correspondent directement à la surface de la terre.

— Un SRS **projeté** utilise une transformation mathématique de projection pour "aplatir" la surface de la terre sphéroïdale sur un plan. Il attribue des coordonnées de localisation d'une manière qui permet de mesurer directement des quantités telles que la distance, la surface et l'angle. Le système de coordonnées est cartésien, ce qui signifie qu'il a un point d'origine défini et deux axes perpendiculaires (généralement orientés vers le nord et l'est). Chaque SRS projeté utilise une unité de longueur définie (généralement des mètres ou des pieds). Un SRS projeté peut être limité dans sa zone d'applicabilité afin d'éviter toute distorsion et de s'inscrire dans les limites des coordonnées définies.

— Un SRS **local** est un système de coordonnées cartésiennes qui n'est pas référencé à la surface de la terre. Dans PostGIS, ce système est spécifié par une valeur SRID de 0.

Il existe de nombreux systèmes de référence spatiale différents. Les SRS courants sont normalisés dans la base de données de l'European Petroleum Survey Group EPSG database. Par commodité, PostGIS (et de nombreux autres systèmes spatiaux) se réfère aux définitions des SRS à l'aide d'un identifiant entier appelé SRID.

Une géométrie est associée à un système de référence spatiale par sa valeur SRID, à laquelle on accède par ST_SRID. Le SRID d'une géométrie peut être assigné en utilisant ST_SetSRID. Certaines fonctions de construction de géométrie permettent

de fournir un SRID (telles que ST_Point et ST_MakeEnvelope). Le format EWKT prend en charge les SRID avec le préfixe `SRID=n;`.

Les fonctions spatiales qui traitent des paires de géométries (telles que les fonctions overlay et relationship) exigent que les géométries d'entrée soient dans le même système de référence spatiale (qu'elles aient le même SRID). Les données géométriques peuvent être transformées dans un système de référence spatiale différent en utilisant ST_Transform et ST_TransformPipeline. Les géométries renvoyées par les fonctions ont le même SRS que les géométries d'entrée.

4.5.1 Table SPATIAL_REF_SYS

La table `SPATIAL_REF_SYS` utilisée par PostGIS est une table de base de données conforme à l'OGC qui définit les systèmes de référence spatiale disponibles. Elle contient les SRID numériques et les descriptions textuelles des systèmes de coordonnées.

La définition de la table `spatial_ref_sys` est la suivante :

Code

```
CREATE TABLE spatial_ref_sys (
  srid       INTEGER NOT NULL PRIMARY KEY,
  auth_name  VARCHAR(256),
  auth_srid  INTEGER,
  srtext     VARCHAR(2048),
  proj4text  VARCHAR(2048)
)
```

Les colonnes sont :

`srid` Un code entier qui identifie de manière unique le Système de référence spatiale (SRS) dans la base de données.

`auth_name` Le nom de la norme ou de l'organisme de normalisation qui est cité pour ce système de référence. Par exemple, "EPSG" est un `auth_name` valide.

`auth_srid` L'ID du système de référence spatiale tel que défini par l'autorité citée dans le `auth_name`. Dans le cas de l'EPSG, il s'agit du code EPSG.

`srtext` Représentation Well-Known Text du système de référence spatiale. Un exemple de représentation WKT du SRS est le suivant :

Code

```
PROJCS["NAD83 / UTM Zone 10N",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980", 6378137, 298.257222101]
    ],
    PRIMEM["Greenwich", 0],
    UNIT["degree", 0.0174532925199433]
  ],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin", 0],
  PARAMETER["central_meridian", -123],
  PARAMETER["scale_factor", 0.9996],
  PARAMETER["false_easting", 500000],
  PARAMETER["false_northing", 0],
  UNIT["metre", 1]
]
```

Pour une discussion sur le SRS WKT, voir la norme OGC Well-known text representation of coordinate reference systems.

`proj4text` PostGIS utilise la bibliothèque PROJ pour fournir des capacités de transformation de coordonnées. La colonne `proj4text` contient la chaîne de caractères de définition des coordonnées PROJ pour un SRID particulier. Par exemple :

Code

```
+proj=utm +zone=10 +ellps=clrk66 +datum=NAD27 +units=m
```

Pour plus d'informations, consultez le site web de PROJ. Le fichier `spatial_ref_sys.sql` contient les définitions `srtext` et `proj4text` pour toutes les projections EPSG.

Lors de la récupération des définitions des systèmes de référence spatiale pour les utiliser dans les transformations, PostGIS utilise la stratégie suivante :

— Si `auth_name` et `auth_srid` sont présents (non NULL), utiliser le PROJ SRS basé sur ces entrées (si au moins un existe).
— Si `srtext` est présent, créez un SRS en l'utilisant, si possible.
— Si `proj4text` est présent, créez un SRS en l'utilisant, si possible.

4.5.2 Systèmes de référence spatiale définis par l'utilisateur

La table PostGIS `spatial_ref_sys` contient plus de 3000 définitions des systèmes de référence spatiale les plus courants qui sont gérés par la bibliothèque de projection PROJ. Mais il existe de nombreux systèmes de coordonnées qu'elle ne contient pas. Vous pouvez ajouter des définitions de SRS à la table si vous disposez des informations requises sur le système de référence spatiale. Vous pouvez également définir votre propre système de référence spatiale personnalisé si vous êtes familiarisé avec les constructions PROJ. Gardez à l'esprit que la plupart des systèmes de référence spatiale sont régionaux et n'ont aucune signification lorsqu'ils sont utilisés en dehors des limites pour lesquelles ils ont été conçus.

Une ressource permettant de trouver des systèmes de référence spatiale non définis dans le jeu de base est http://spatialreference.org/-

Some commonly used spatial reference systems are: 4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, and the 60 WGS84 UTM zones. UTM zones are one of the most ideal for measurement, but only cover 6-degree regions. For WGS84 longitude/latitude data, the UTM zone is `least(floor((longitude + 180) / 6) + 1, 60)`; the EPSG code is `32600 + zone` in the northern hemisphere or `32700 + zone` in the southern hemisphere.

Les États américains utilisent des systèmes de référence spatiale State Plane (basés sur le mètre ou le pied) - il en existe généralement un ou deux par État. La plupart des systèmes basés sur le mètre se trouvent dans le jeu de base, mais de nombreux systèmes basés sur le pied ou créés par ESRI devront être copiés à partir de spatialreference.org.

Vous pouvez même définir des systèmes de coordonnées non terrestres, tels que Mars 2000 Ce système de coordonnées martien n'est pas planaire (il est en degrés sphéroïdaux), mais vous pouvez l'utiliser avec le type `géographie` pour obtenir des mesures de longueur et de proximité en mètres au lieu de degrés.

Voici un exemple de chargement d'un système de coordonnées personnalisé à l'aide d'un SRID non attribué et de la définition PROJ pour une projection Lambert conforme centrée sur les États-Unis :

Code

```
INSERT INTO spatial_ref_sys (srid, proj4text)
VALUES ( 990000,
  '+proj=lcc  +lon_0=-95 +lat_0=25 +lat_1=25 +lat_2=25 +x_0=0 +y_0=0 +datum=WGS84 +units=m + ↩
    no_defs'
);
```

4.6 Tables spatiales

4.6.1 Créer une table spatiale

Vous pouvez créer une table pour stocker des données géométriques en utilisant l'instruction SQL CREATE TABLE avec une colonne de type `geometry`. L'exemple suivant crée une table avec une colonne géométrie stockant des chaînes de lignes 2D (XY) dans le système de coordonnées BC-Albers (SRID 3005) :

Code

```
CREATE TABLE roads (
  id SERIAL PRIMARY KEY,
  name VARCHAR(64),
  geom geometry(LINESTRING, 3005)
);
```

Le type `geometry` prend en charge deux **modificateurs de type** facultatifs :

— le **modificateur de type spatial** restreint le type de formes et de dimensions autorisées dans la colonne. La valeur peut être l'un des sous-types de géométrie pris en charge (par exemple POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, etc). Le modificateur prend en charge les restrictions de dimensionnalité des coordonnées en ajoutant des suffixes : Z, M et ZM. Par exemple, le modificateur "LINESTRINGM" n'autorise que les lignes à trois dimensions et traite la troisième dimension comme une mesure. De même, "POINTZM" requiert des données à quatre dimensions (XYZM).

— le **modificateur SRID** limite le système de référence spatiale SRID à un nombre particulier. S'il est omis, le SRID prend par défaut la valeur 0.

Exemples de création de tableaux avec des colonnes géométriques :

Créer une table contenant n'importe quel type de géométrie avec le SRID par défaut :

Code

```
CREATE TABLE geoms(gid serial PRIMARY KEY, geom geometry);
```

Créer une table avec une géométrie POINT 2D avec le SRID par défaut :

Code

```
CREATE TABLE pts(gid serial PRIMARY KEY, geom geometry(POINT));
```

Créer une table avec des POINTS 3D (XYZ) et un SRID explicite de 3005 :

Code

```
CREATE TABLE pts(gid serial PRIMARY KEY, geom geometry(POINTZ, 3005));
```

Créer une table avec une géométrie 4D (XYZM) LINESTRING avec le SRID par défaut :

Code

```
CREATE TABLE lines(gid serial PRIMARY KEY, geom geometry(LINESTRINGZM));
```

Créer une table avec une géométrie POLYGON 2D avec le SRID 4267 (NAD 1927 long lat) :

Code

```
CREATE TABLE polys(gid serial PRIMARY KEY, geom geometry(POLYGON, 4267));
```

Il est possible d'avoir plus d'une colonne de géométrie dans une table. Cela peut être spécifié lors de la création de la table, ou une colonne peut être ajoutée à l'aide de l'instruction SQL ALTER TABLE. Cet exemple ajoute une colonne qui peut contenir des LineStrings 3D :

Code

```
ALTER TABLE roads ADD COLUMN geom2 geometry(LINESTRINGZ, 4326);
```

4.6.2 Vue GEOMETRY_COLUMNS

L'OGC *Simple Features Specification for SQL* définit la table de métadonnées `GEOMETRY_COLUMNS` pour décrire la structure de la table géométrique. Dans PostGIS, `geometry_columns` est une vue qui lit les tables de catalogue du système de base de données. Cela garantit que les informations de métadonnées spatiales sont toujours cohérentes avec les tables et les vues actuellement définies. La structure de la vue est la suivante :

Code

```
\d geometry_columns
```

Output

```
View "public.geometry_columns"
      Column       |          Type          | Modifiers
-------------------+------------------------+-----------
 f_table_catalog   | character varying(256) |
 f_table_schema    | character varying(256) |
 f_table_name      | character varying(256) |
 f_geometry_column | character varying(256) |
 coord_dimension   | integer                |
 srid              | integer                |
 type              | character varying(30)  |
```

Les colonnes sont :

f_table_catalog, f_table_schema, f_table_name Le nom complet de la table d'entités contenant la colonne géométrie. Il n'y a pas d'analogue PostgreSQL de "catalog", cette colonne est donc laissée vide. Pour "schema", le nom du schéma PostgreSQL est utilisé (`public` est la valeur par défaut).

f_geometry_column Le nom de la colonne géométrique dans la table des caractéristiques.

coord_dimension The coordinate dimension (2, 3 or 4) of the column. The coordinate dimension is at least 2, representing X and Y coordinates.

srid L'ID du système de référence spatiale utilisé pour la géométrie des coordonnées dans cette table. Il s'agit d'une référence de clé étrangère à la table `spatial_ref_sys` (voir Section 4.5.1).

type Le type de l'objet spatial. Pour limiter la colonne spatiale à un seul type, utilisez l'une des options suivantes : POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION ou les versions XYM correspondantes POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. Pour les collections hétérogènes (de type mixte), vous pouvez utiliser "GEOMETRY" comme type.

4.6.3 Enregistrement manuel des colonnes de géométrie

Deux des cas où vous pouvez avoir besoin de cela sont les cas de vues SQL et les insertions en masse. Dans le cas des insertions en masse, vous pouvez corriger l'enregistrement dans la table geometry_columns en contraignant la colonne ou en effectuant une modification de la table. Pour les vues, vous pouvez exposer en utilisant une opération CAST. Notez que si votre colonne est basée sur un typmod, le processus de création l'enregistrera correctement, il n'est donc pas nécessaire de faire quoi que ce soit. De même, les vues qui n'ont pas de fonction spatiale appliquée à la géométrie s'enregistreront de la même manière que la colonne géométrique de la table sous-jacente.

For example, suppose you have a view created like this.

Code

```
CREATE VIEW public.vwmytablemercator AS
SELECT gid, ST_Transform(geom, 3395) As geom, f_name
FROM public.mytable;
```

To register the geometry column correctly, cast the transformed geometry.

Code

```
DROP VIEW public.vwmytablemercator;
CREATE VIEW  public.vwmytablemercator AS
    SELECT gid, ST_Transform(geom, 3395)::geometry(Geometry, 3395) As geom, f_name
    FROM public.mytable;
```

If the geometry type is known to be a 2D Polygon, specify it in the cast.

Code

```
DROP VIEW public.vwmytablemercator;
CREATE VIEW  public.vwmytablemercator AS
    SELECT gid, ST_Transform(geom, 3395)::geometry(Polygon, 3395) As geom, f_name
    FROM public.mytable;
```

For a derivative table created by a bulk insert, create the table first.

Code

```
SELECT poi.gid, poi.geom, citybounds.city_name
INTO myschema.my_special_pois
FROM poi INNER JOIN citybounds ON ST_Intersects(citybounds.geom, poi.geom);
```

Create a 2D spatial index on the new table.

Code

```
CREATE INDEX idx_myschema_myspecialpois_geom_gist
ON myschema.my_special_pois USING gist(geom);
```

If the points are 3D or measured, an n-D index may be more appropriate.

Code

```
CREATE INDEX my_special_pois_geom_gist_nd
ON my_special_pois USING gist(geom gist_geometry_ops_nd);
```

Manually register the new table's geometry column in `geometry_columns`. This also changes the underlying table structure to make the column typmod-based.

Code

```
SELECT Populate_Geometry_Columns('myschema.my_special_pois'::regclass);
```

To retain constraint-based definition behavior, such as for inherited tables where all children do not have the same type and SRID, set the optional `use_typmod` argument to false.

Code

```
SELECT Populate_Geometry_Columns('myschema.my_special_pois'::regclass, false);
```

Although the old-constraint based method is still supported, a constraint-based geometry column used directly in a view, will not register correctly in geometry_columns, as will a typmod one. In this example we define a column using typmod and another using constraints.

Code

```
CREATE TABLE pois_ny(gid SERIAL PRIMARY KEY, poi_name text, cat text, geom geometry(POINT,  ↩
    4326));
SELECT AddGeometryColumn('pois_ny', 'geom_2160', 2160, 'POINT', 2, false);
```

If we run **\d pois_ny** in psql, we observe they are defined differently -- one is typmod, one is constraint.

Code

```
\d pois_ny
```

Output

```
Table "public.pois_ny"
  Column   |         Type          |                      Modifiers

-----------+-----------------------+-----------------------------------------------------
 gid       | integer               | not null default nextval('pois_ny_gid_seq'::regclass)
 poi_name  | text                  |
 cat       | character varying(20) |
 geom      | geometry(Point,4326)  |
 geom_2160 | geometry              |
Indexes:
    "pois_ny_pkey" PRIMARY KEY, btree (gid)
Check constraints:
    "enforce_dims_geom_2160" CHECK (st_ndims(geom_2160) = 2)
    "enforce_geotype_geom_2160" CHECK (geometrytype(geom_2160) = 'POINT'::text
        OR geom_2160 IS NULL)
    "enforce_srid_geom_2160" CHECK (st_srid(geom_2160) = 2160)
```

Dans geometry_columns, les deux s'enregistrent correctement

Code

```
SELECT f_table_name, f_geometry_column, srid, type
FROM geometry_columns
WHERE f_table_name = 'pois_ny';
```

Output

```
f_table_name | f_geometry_column | srid | type
-------------+-------------------+------+-------
pois_ny      | geom              | 4326 | POINT
pois_ny      | geom_2160         | 2160 | POINT
```

Cependant, si nous devions créer une vue comme celle-ci

Code

```
CREATE VIEW vw_pois_ny_parks AS
SELECT *
  FROM pois_ny
  WHERE cat='park';

SELECT f_table_name, f_geometry_column, srid, type
    FROM geometry_columns
    WHERE f_table_name = 'vw_pois_ny_parks';
```

The typmod based geom view column registers correctly, but the constraint based one does not.

Output

```
f_table_name   | f_geometry_column | srid |   type
-----------------+-------------------+------+----------
 vw_pois_ny_parks | geom              | 4326 | POINT
 vw_pois_ny_parks | geom_2160         |    0 | GEOMETRY
```

This may change in future versions of PostGIS, but for now to force the constraint-based view column to register correctly, you need to do this:

Code

```
DROP VIEW vw_pois_ny_parks;
CREATE VIEW vw_pois_ny_parks AS
SELECT gid, poi_name, cat,
  geom,
  geom_2160::geometry(POINT,2160) As geom_2160
  FROM pois_ny
  WHERE cat = 'park';
SELECT f_table_name, f_geometry_column, srid, type
    FROM geometry_columns
    WHERE f_table_name = 'vw_pois_ny_parks';
```

Output

```
f_table_name   | f_geometry_column | srid | type
-----------------+-------------------+------+-------
 vw_pois_ny_parks | geom              | 4326 | POINT
 vw_pois_ny_parks | geom_2160         | 2160 | POINT
```

4.7 Chargement des données spatiales

Une fois que vous avez créé une table spatiale, vous êtes prêt à charger des données spatiales dans la base de données. Il existe deux façons d'intégrer des données spatiales dans une base de données PostGIS/PostgreSQL : à l'aide d'instructions SQL formatées ou à l'aide de l'utilitaire qui permet de charger des Shapefiles.

4.7.1 Utilisation de SQL pour charger des données

Si les données spatiales peuvent être converties en une représentation textuelle (WKT ou WKB), l'utilisation de SQL pourrait être le moyen le plus simple d'introduire les données dans PostGIS. Les données peuvent être chargées en masse dans PostGIS/-PostgreSQL en chargeant un fichier texte d'instructions SQL `INSERT` à l'aide de l'utilitaire SQL `psql`.

Un fichier de chargement SQL (`roads.sql` par exemple) peut ressembler à ceci :

Code

```
BEGIN;
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (1, 'LINESTRING(191232 243118,191108 243242)', 'Jeff Rd');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (2, 'LINESTRING(189141 244158,189265 244817)', 'Geordie Rd');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (3, 'LINESTRING(192783 228138,192612 229814)', 'Paul St');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (4, 'LINESTRING(189412 252431,189631 259122)', 'Graeme Ave');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (5, 'LINESTRING(190131 224148,190871 228134)', 'Phil Tce');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (6, 'LINESTRING(198231 263418,198213 268322)', 'Dave Cres');
COMMIT;
```

Le fichier SQL peut être chargé dans PostgreSQL en utilisant `psql` :

Code

```
psql -d [database] -f roads.sql
```

4.7.2 Utilisation de l'utilitaire qui permet de charger des fichiers Shapefile

Le chargeur de données `shp2pgsql` convertit les Shapefiles en SQL pour les insérer dans une base de données PostGIS/PostgreSQL au format geometry ou geography. Le chargeur a plusieurs modes de fonctionnement sélectionnés par des indicateurs de ligne de commande.

The input must be an unpacked shapefile path. The loader does not read `.zip` archives directly; unpack the `.shp`, `.shx`, and `.dbf` files before loading.

Il existe également une interface graphique `shp2pgsql-gui` qui offre la plupart des options du chargeur en ligne de commande. Cette interface peut être plus facile à utiliser pour un chargement ponctuel sans script ou si vous êtes novice en matière de PostGIS. Il peut également être configuré comme un plugin pour PgAdminIII.

(c|a|d|p) Ces options s'excluent mutuellement :

- **-c** Crée une nouvelle table et la remplit à partir du fichier Shapefile. *C'est le mode par défaut.*
- **-a** Ajoute les données du fichier Shapefile dans la table de la base de données. Notez que pour utiliser cette option afin de charger plusieurs fichiers, les fichiers doivent avoir les mêmes attributs et les mêmes types de données.
- **-d** Supprime la table de la base de données avant de créer une nouvelle table avec les données du fichier Shapefile.
- **-p** Produit uniquement le code SQL de création de la table, sans ajouter de données réelles. Cette option peut être utilisée si vous souhaitez séparer complètement les étapes de création de la table et de chargement des données.

-? Afficher l'écran d'aide.

-D Utilise le format "dump" de PostgreSQL pour les données de sortie. Ce format peut être combiné avec -a, -c et -d. Il est beaucoup plus rapide à charger que le format SQL "insert" par défaut. Utilisez-le pour les très grands ensembles de données.

-f <fid_column>, --feature-id-column <fid_column> Specify the name of the feature id column. The default is `gid`. The value must be a simple PostgreSQL identifier and cannot use a PostgreSQL system column name.

-s [<FROM_SRID>:]<SRID> Crée et remplit les tables de géométrie avec le SRID spécifié. Il est possible de spécifier que le fichier de forme d'entrée utilise le FROM_SRID donné, auquel cas les géométries seront reprojetées dans le SRID cible.

-k Conserver la casse des identifiants (colonne, schéma et attributs). Notez que les attributs dans Shapefile sont tous en majuscules.

-i Contraindre tous les entiers à des entiers 32 bits standard, ne pas créer de bigints 64 bits, même si la signature de l'en-tête DBF semble le justifier.

-I Créer un index GiST sur la colonne géométrie.

-u Create the target table as `UNLOGGED`. This can speed up loading transient staging data, but PostgreSQL does not preserve unlogged table contents after a crash or unclean shutdown.

-m Specify a file containing a set of mappings of (long) column names to 10 character DBF column names. The content of the file is one or more lines of two names separated by white space and no trailing or leading space. For example:

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

-S Génère des géométries simples au lieu de géométries MULTI. Ne réussira que si toutes les géométries sont en fait simples (c'est-à-dire un MULTIPOLYGON avec une seule enveloppe, ou un MULTIPOINT avec un seul sommet).

-t <dimensionality> Force la géométrie de sortie à avoir la dimensionnalité spécifiée. Utilisez les chaînes de caractères suivantes pour indiquer la dimensionnalité : 2D, 3DZ, 3DM, 4D.

Si l'entrée a moins de dimensions que celles spécifiées, la sortie aura ces dimensions remplies avec des zéros. Si l'entrée a plus de dimensions que celles spécifiées, les dimensions non désirées seront supprimées.

-w Sortie au format WKT, au lieu de WKB. Notez que cela peut introduire des dérives de coordonnées dues à la perte de précision.

-e Exécuter chaque instruction séparément, sans utiliser de transaction. Cela permet de charger la majorité des bonnes données lorsqu'il y a quelques mauvaises géométries qui génèrent des erreurs. Notez que cela ne peut pas être utilisé avec l'option -D car le format "dump" utilise toujours une transaction.

-W <encoding> Spécifie l'encodage des données d'entrée (fichier dbf). Lorsqu'il est utilisé, tous les attributs du fichier dbf sont convertis en UTF8 à partir de l'encodage spécifié. La sortie SQL résultante contiendra une commande `SET CLIENT_ENCODING to UTF8`, afin que le backend puisse reconvertir de UTF8 à n'importe quel encodage que la base de données est configurée pour utiliser en interne.

-N <policy> Politique de gestion des géométries NULL (insert*, skip, abort)

-n -n Importer uniquement le fichier DBF. Si vos données n'ont pas de fichier de forme correspondant, il passera automatiquement à ce mode et ne chargera que le fichier DBF. Cette option n'est donc nécessaire que si vous disposez d'un fichier de forme complet et que vous ne voulez que les données d'attributs et pas de géométrie.

-G Utiliser le type geography au lieu de geometry (nécessite des données lon/lat) dans WGS84 long lat (SRID=4326)

-T <tablespace> Spécifiez le tablespace pour la nouvelle table. Les index utiliseront toujours le tablespace par défaut à moins que le paramètre -X ne soit également utilisé. La documentation de PostgreSQL contient une bonne description de l'utilisation des tablespaces personnalisés.

-X <tablespace> Spécifier le tablespace pour les index de la nouvelle table. Ceci s'applique à l'index de clé primaire et à l'index spatial GIST si -I est également utilisé.

-Z Lorsqu'il est utilisé, cet indicateur empêche la génération des instructions `ANALYZE`. Sans l'option -Z (comportement par défaut), les instructions `ANALYZE` seront générées.

Un exemple de session utilisant le chargeur pour créer un fichier d'entrée et le charger peut ressembler à ceci :

Code

```
shp2pgsql -c -D -s 4269 -i -I shaperoads.shp myschema.roadstable > roads.sql
psql -d roadsdb -f roads.sql
```

Une conversion et un chargement peuvent être effectués en une seule étape à l'aide de pipes UNIX :

Code

```
shp2pgsql shaperoads.shp myschema.roadstable | psql -d roadsdb
```

To load a directory of Shapefiles, run the loader once per file and derive the table name from the file name. On Windows `cmd.exe`, use a single percent sign interactively and double percent signs in a batch file:

Code

```
REM In a .bat file, use %%F. At an interactive cmd.exe prompt, use %F.
for %%F in ("C:\data\shapefiles\*.shp") do (
  shp2pgsql -d -D -s 4269 -I -W UTF-8 "%%~fF" "myschema.%%~nF" | psql -d roadsdb
)
```

If every Shapefile has the same attributes and data types and should be appended into one table, create the table from the first file and use `-a` for the remaining files.

4.8 Extraction de données spatiales

Les données spatiales peuvent être extraites de la base de données à l'aide de SQL ou de Shapefile dumper. La section sur SQL présente certaines des fonctions disponibles pour effectuer des comparaisons et des requêtes sur les tables spatiales.

4.8.1 Utiliser SQL pour extraire des données

La manière la plus simple d'extraire des données spatiales de la base de données consiste à utiliser une requête SQL `SELECT` pour définir l'ensemble de données à extraire et à transférer les colonnes résultantes dans un fichier texte analysable :

Code

```
SELECT road_id,road_geom AS geom, road_name FROM roads;
```

Output

```
road_id | geom                                    | road_name
--------+-----------------------------------------+-----------
      1 | LINESTRING(191232 243118,191108 243242) | Jeff Rd
      2 | LINESTRING(189141 244158,189265 244817) | Geordie Rd
      3 | LINESTRING(192783 228138,192612 229814) | Paul St
      4 | LINESTRING(189412 252431,189631 259122) | Graeme Ave
      5 | LINESTRING(190131 224148,190871 228134) | Phil Tce
      6 | LINESTRING(198231 263418,198213 268322) | Dave Cres
      7 | LINESTRING(218421 284121,224123 241231) | Chris Way
(6 rows)
```

Il peut arriver qu'une restriction soit nécessaire pour réduire le nombre d'enregistrements renvoyés. Dans le cas de restrictions basées sur des attributs, utilisez la même syntaxe SQL que pour une table non spatiale. Dans le cas de restrictions spatiales, les fonctions suivantes sont utiles :

`ST_Intersects` Cette fonction indique si deux géométries partagent un espace.

`=` Cette fonction permet de vérifier si deux géométries sont géométriquement identiques. Par exemple, si 'POLYGON((0 0,1 1,1 0,0 0))' est identique à 'POLYGON((0 0,1 1,1 0,0 0))' (c'est le cas).

Vous pouvez ensuite utiliser ces opérateurs dans des requêtes. Notez que lorsque vous spécifiez des géométries et des box sur la ligne de commande SQL, vous devez explicitement transformer les représentations de chaînes de caractères en fonction géométrique. Le 312 est un système de référence spatiale fictif qui correspond à nos données. Ainsi, par exemple, le 312 est un système de référence spatiale fictif qui correspond à nos données :

Code

```
SELECT road_id, road_name
FROM roads
WHERE roads_geom='SRID=312;LINESTRING(191232 243118,191108 243242)'::geometry;
```

La requête ci-dessus renverrait le seul enregistrement de la table "ROADS_GEOM" dont la géométrie est égale à cette valeur.

Vérifier si certaines routes passent dans la zone définie par un polygone :

Code

```
SELECT road_id, road_name
FROM roads
WHERE ST_Intersects(roads_geom, 'SRID=312;POLYGON((...))');
```

La requête spatiale la plus courante sera probablement une requête "basée sur un cadre", utilisée par les logiciels clients, tels que les navigateurs de données et les cartographes web, pour saisir un "cadre de carte" d'une valeur de données pour l'affichage.

Lorsque vous utilisez l'opérateur "&&", vous pouvez spécifier soit une BOX3D comme élément de comparaison, soit une GEOMETRY. Toutefois, lorsque vous spécifiez une GEOMETRY, c'est sa boîte de délimitation qui sera utilisée pour la comparaison.

En utilisant un objet "BOX3D" pour le cadre, une telle requête se présente comme suit :

Code

```
SELECT roads_geom AS geom
FROM roads
WHERE
  roads_geom && ST_MakeEnvelope(191232, 243117,191232, 243119,312);
```

Notez l'utilisation du SRID 312 pour spécifier la projection de l'enveloppe.

4.8.2 Utilisation de Shapefile Dumper

Le table dumper `pgsql2shp` se connecte à la base de données et convertit une table (éventuellement définie par une requête) en un fichier shape. La syntaxe de base est la suivante :

Code

```
pgsql2shp [<options
>] <database
> [<schema
>.]<table>
```

Code

```
pgsql2shp [<options
>] <database
> <query>
```

Les options de la ligne de commande sont les suivantes :

-f <filename> Écriture de la sortie dans un fichier particulier.

-h <host> L'hôte de la base de données auquel se connecter.

-p <port> Le port auquel se connecter sur l'hôte de la base de données.

-P <password> Le mot de passe à utiliser lors de la connexion à la base de données.

-u <user> Le nom d'utilisateur à utiliser lors de la connexion à la base de données.

-g <geometry column> Dans le cas de tableaux comportant plusieurs colonnes géométriques, la colonne géométrique à utiliser lors de l'écriture du fichier de forme.

-b Utiliser un curseur binaire. L'opération sera plus rapide, mais ne fonctionnera pas si un attribut NON-geometry de la table n'est pas converti en texte.

-r Mode brut. Ne pas supprimer le champ `gid`, ni échapper les noms de colonnes.

-m filename Remapper les identifiants en noms de dix caractères. Le contenu du fichier est constitué de lignes de deux symboles séparées par un seul espace blanc, sans espace de départ ni de fin : VERYLONGSYMBOL SHORTONE ANOTHERYLONGSYMBOL SHORTER etc.

4.9 Index spatiaux

Les index spatiaux permettent d'utiliser une base de données spatiales pour de grands ensembles de données. Sans indexation, la recherche de caractéristiques nécessite un balayage séquentiel de chaque enregistrement de la base de données. L'indexation accélère la recherche en organisant les données dans une structure qui peut être rapidement parcourue pour trouver les enregistrements correspondants.

La méthode d'indexation B-tree couramment utilisée pour les données d'attributs n'est pas très utile pour les données spatiales, car elle ne permet de stocker et d'interroger les données que dans une seule dimension. Les données telles que la géométrie (qui a 2 dimensions ou plus) requièrent une méthode d'indexation qui supporte les requêtes sur toutes les dimensions des données. L'un des principaux avantages de PostgreSQL pour le traitement des données spatiales est qu'il offre plusieurs types de méthodes d'indexation qui fonctionnent bien pour les données multidimensionnelles : GiST, BRIN et SP-GiST.

— Les index **GiST (Generalized Search Tree)** décomposent les données en "choses à côté", "choses qui se chevauchent", "choses qui sont à l'intérieur" et peuvent être utilisés sur un large éventail de types de données, y compris les données SIG. PostGIS utilise un index R-Tree implémenté au-dessus de GiST pour indexer les données spatiales. GiST est la méthode d'indexation spatiale la plus couramment utilisée et la plus polyvalente, et offre de très bonnes performances en matière de requêtes.

— Les index **BRIN (Block Range Index)** fonctionnent en résumant l'étendue spatiale des plages d'enregistrements de la table. La recherche s'effectue par le biais d'un balayage des plages. L'index BRIN n'est approprié que pour certains types de données (triées dans l'espace, avec des mises à jour peu fréquentes ou inexistantes). Mais il permet une création d'index beaucoup plus rapide et une taille d'index beaucoup plus petite.

— **SP-GiST (Space-Partitioned Generalized Search Tree)** est une méthode d'indexation générique qui prend en charge les arbres de recherche partitionnés tels que les arbres quadratiques, les arbres k-d et les arbres radix (tries).

Les index spatiaux ne stockent que la boîte de délimitation des géométries. Les requêtes spatiales utilisent l'index comme **filtre primaire** pour déterminer rapidement un ensemble de géométries correspondant potentiellement à la condition de la requête. La plupart des requêtes spatiales nécessitent un **filtre secondaire** qui utilise une fonction de prédicat spatial pour tester une condition spatiale plus spécifique. Pour plus d'informations sur les requêtes avec des prédicats spatiaux, voir Section 5.2.

Voir également la section de l'atelier PostGIS sur les index spatiaux et le manuel de PostgreSQL.

4.9.1 Index GiST

GiST signifie "Generalized Search Tree" (arbre de recherche généralisé) et constitue une forme générique d'indexation pour les données multidimensionnelles. PostGIS utilise un index R-Tree implémenté au-dessus de GiST pour indexer les données spatiales. GiST est la méthode d'indexation spatiale la plus couramment utilisée et la plus polyvalente, et offre de très bonnes performances en matière de requêtes. D'autres implémentations de GiST sont utilisées pour accélérer les recherches sur toutes sortes de structures de données irrégulières (tableaux d'entiers, données spectrales, etc.) qui ne se prêtent pas à l'indexation B-Tree normale. Pour plus d'informations, voir le manuel PostgreSQL.

Dès qu'une table de données spatiales dépasse quelques milliers de lignes, vous devez créer un index pour accélérer les recherches spatiales dans les données (sauf si toutes vos recherches sont basées sur des attributs, auquel cas vous devez créer un index normal sur les champs d'attributs).

La syntaxe pour construire un index GiST sur une colonne "geometry" est la suivante :

Code

```
CREATE INDEX [indexname] ON [tablename] USING GIST ( [geometryfield]);
```

La syntaxe ci-dessus permet toujours de créer un index 2D. Pour obtenir un index à n dimensions pour le type de géométrie, vous pouvez en créer un à l'aide de cette syntaxe :

Code

```
CREATE INDEX [indexname] ON [tablename] USING GIST ([geometryfield] gist_geometry_ops_nd);
```

La construction d'un index spatial est un exercice de calcul intensif. Il bloque également l'accès en écriture à votre table pendant le temps de sa création, donc sur un système de production, vous voudrez peut-être le faire d'une manière plus lente, en tenant compte de la notion de CONCURRENCE :

Code

```
CREATE INDEX CONCURRENTLY [indexname] ON [tablename] USING GIST ( [geometryfield]);
```

Après avoir construit un index, il est parfois utile de forcer PostgreSQL à collecter les statistiques de la table, qui sont utilisées pour optimiser les plans de requête :

Code

```
VACUUM ANALYZE [table_name] [(column_name)];
```

4.9.2 Index BRIN

BRIN stands for "Block Range Index". It is a general-purpose index method provided by PostgreSQL. BRIN is a *lossy* index method, meaning that a secondary check is required to confirm that a record matches a given search condition (which is the case for all provided spatial indexes). It provides much faster index creation and much smaller index size, with reasonable read performance. Its primary purpose is to support indexing very large tables on columns which have a correlation with their physical location within the table. In addition to spatial indexing, BRIN can speed up searches on various kinds of attribute data structures (integer, arrays etc). For more information see the PostgreSQL manual.

Dès qu'une table spatiale dépasse quelques milliers de lignes, vous voudrez construire un index pour accélérer les recherches spatiales dans les données. Les index GiST sont très performants tant que leur taille ne dépasse pas la quantité de mémoire vive disponible pour la base de données, et tant que vous pouvez vous permettre la taille de stockage de l'index et le coût de la mise à jour de l'index en écriture. Sinon, pour les très grandes tables, l'index BRIN peut être considéré comme une alternative.

Un index BRIN stocke la boîte englobant toutes les géométries contenues dans les lignes d'un ensemble contigu de blocs de table, appelé *plage de blocs*. Lors de l'exécution d'une requête à l'aide de l'index, les plages de blocs sont examinées pour trouver celles qui recoupent l'étendue de la requête. Cette méthode n'est efficace que si les données sont physiquement ordonnées de manière à ce que les zones de délimitation des plages de blocs se chevauchent le moins possible (et, dans l'idéal, s'excluent mutuellement). L'index qui en résulte est de très petite taille, mais il est généralement moins performant en lecture qu'un index GiST sur les mêmes données.

La construction d'un index BRIN est beaucoup moins gourmande en ressources CPU que la construction d'un index GiST. Il est courant de constater qu'un index BRIN est dix fois plus rapide à construire qu'un index GiST pour les mêmes données. Et comme un index BRIN ne stocke qu'une seule boîte englobante pour chaque plage de blocs de table, il est courant d'utiliser jusqu'à mille fois moins d'espace disque qu'un index GiST.

Vous pouvez choisir le nombre de blocs à résumer dans une plage. Si vous diminuez ce nombre, l'index sera plus volumineux mais offrira probablement de meilleures performances.

Pour que le BRIN soit efficace, les données de la table doivent être stockées dans un ordre physique qui minimise le chevauchement de l'étendue du bloc. Il se peut que les données soient déjà triées de manière appropriée (par exemple, si elles sont chargées à partir d'un autre jeu de données déjà trié dans l'ordre spatial). Dans le cas contraire, il est possible de trier les données en fonction d'une clé spatiale unidimensionnelle. Une façon de procéder consiste à créer une nouvelle table triée par les valeurs géométriques (qui, dans les versions récentes de PostGIS, utilisent un classement efficace par courbe de Hilbert) :

Code

```
CREATE TABLE table_sorted AS
SELECT * FROM table  ORDER BY geom;
```

Il est également possible de trier les données en place en utilisant un GeoHash comme index (temporaire) et en effectuant un regroupement sur cet index :

Code

```
CREATE INDEX idx_temp_geohash ON table
    USING btree (ST_GeoHash(ST_Transform(geom, 4326 ), 20));
CLUSTER table USING idx_temp_geohash;
```

La syntaxe pour construire un index BRIN sur une colonne `geometry` est la suivante :

Code

```
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geome_col]);
```

La syntaxe ci-dessus permet de créer un index en 2D. Pour construire un index à trois dimensions, utilisez la syntaxe suivante :

Code

```
CREATE INDEX [indexname] ON [tablename]
USING BRIN ([geome_col] brin_geometry_inclusion_ops_3d);
```

Vous pouvez également obtenir un index quadridimensionnel à l'aide de la classe d'opérateurs 4D :

Code

```
CREATE INDEX [indexname] ON [tablename]
USING BRIN ([geome_col] brin_geometry_inclusion_ops_4d);
```

Les commandes ci-dessus utilisent le nombre par défaut de blocs dans une plage, qui est de 128. Pour spécifier le nombre de blocs à résumer dans une plage, utilisez la syntaxe suivante

Code

```
CREATE INDEX [indexname] ON [tablename]
USING BRIN ( [geome_col] ) WITH (pages_per_range = [number]);
```

Gardez à l'esprit qu'un index BRIN ne stocke qu'une entrée d'index pour un grand nombre de lignes. Si votre table stocke des géométries avec un nombre variable de dimensions, il est probable que l'index qui en résultera sera peu performant. Vous pouvez éviter cette pénalité de performance en choisissant la classe d'opérateur avec le plus petit nombre de dimensions des géométries stockées

Le type de données `geography` est pris en charge pour l'indexation BRIN. La syntaxe pour construire un index BRIN sur une colonne géographique est la suivante :

Code

```
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geog_col]);
```

La syntaxe ci-dessus permet de créer un index 2D pour les objets géospatiaux sur le sphéroïde.

Actuellement, seul le "support d'inclusion" est fourni, ce qui signifie que seuls les opérateurs `&&`, `~` et `@` peuvent être utilisés pour les cas 2D (à la fois pour `geometry` et `geography`), et seulement l'opérateur `&&&` pour les géométries 3D. Il n'y a actuellement pas de support pour les recherches kNN.

Une différence importante entre BRIN et les autres types d'index est que la base de données ne maintient pas l'index de manière dynamique. Les modifications apportées aux données spatiales de la table sont simplement ajoutées à la fin de l'index. Cela entraîne une dégradation des performances de recherche de l'index au fil du temps. L'index peut être mis à jour en effectuant

un VACUUM, ou en utilisant une fonction spéciale brin_summarize_new_values(regclass). C'est la raison pour laquelle BRIN peut être plus approprié pour les données qui sont en lecture seule ou qui ne changent que rarement. Pour plus d'informations, consultez le manuel.

En résumé, l'utilisation de BRIN pour les données spatiales :

— Le temps de construction de l'index est très rapide et la taille de l'index est très réduite.
— Le temps d'interrogation de l'index est plus lent que celui de GiST, mais reste très acceptable.
— Nécessite que les données du tableau soient triées dans un ordre spatial.
— Nécessite une maintenance manuelle de l'index.
— Plus approprié pour les tables de très grande taille, avec peu ou pas de chevauchement (par exemple des points), qui sont statiques ou qui changent peu souvent.
— Plus efficace pour les requêtes qui renvoient un nombre relativement important d'enregistrements de données.

4.9.3 Index SP-GiST

SP-GiST signifie "Space-Partitioned Generalized Search Tree" et est une forme générique d'indexation pour les types de données multidimensionnelles qui prend en charge les arbres de recherche partitionnés, tels que les arbres quadratiques, les arbres k-d et les arbres radix (tries). La caractéristique commune de ces structures de données est qu'elles divisent de manière répétée l'espace de recherche en partitions qui ne sont pas nécessairement de taille égale. Outre l'indexation spatiale, SP-GiST est utilisé pour accélérer les recherches sur de nombreux types de données, telles que le routage téléphonique, le routage ip, la recherche de substrats, etc. Pour plus d'informations, voir le manuel PostgreSQL.

Comme pour les index GiST, les index SP-GiST sont avec perte, dans le sens où ils stockent la boîte englobante qui entoure les objets spatiaux. Les index SP-GiST peuvent être considérés comme une alternative aux index GiST.

Lorsqu'une table de données SIG dépasse quelques milliers de lignes, un index SP-GiST peut être utilisé pour accélérer les recherches spatiales dans les données. La syntaxe pour construire un index SP-GiST sur une colonne "géométrie" est la suivante :

Code

```
CREATE INDEX [indexname] ON [tablename] USING SPGIST ( [geometryfield]);
```

La syntaxe ci-dessus permet de créer un index bidimensionnel. Un index tridimensionnel pour le type de géométrie peut être créé à l'aide de la classe d'opérateurs 3D :

Code

```
CREATE INDEX [indexname] ON [tablename] USING SPGIST ([geometryfield] spgist_geometry_ops_3d ↩
    );
```

La création d'un index spatial est une opération très gourmande en ressources informatiques. Elle bloque également l'accès en écriture à votre table pendant la durée de la création, donc sur un système de production, vous voudrez peut-être le faire d'une manière plus lente, en tenant compte de la notion de CONCURRENCE :

Code

```
CREATE INDEX CONCURRENTLY [indexname] ON [tablename] USING SPGIST ( [geometryfield]);
```

Après avoir construit un index, il est parfois utile de forcer PostgreSQL à collecter les statistiques de la table, qui sont utilisées pour optimiser les plans de requête :

Code

```
VACUUM ANALYZE [table_name] [(column_name)];
```

Un index SP-GiST peut accélérer les requêtes impliquant les opérateurs suivants :

— <<, &<, &>, >>, <<|, &<|, |&>, |>>, &&, @>, <@, et ~=, pour les index à 2 dimensions,
— &/&, ~==, @>>, et <<@, pour les index à 3 dimensions.

Il n'y a pas de support pour les recherches kNN pour le moment.

4.9.4 Optimisation de l'utilisation de l'index

Normalement, les index accélèrent de manière invisible l'accès aux données : une fois qu'un index est construit, le planificateur de requêtes de PostgreSQL décide automatiquement quand l'utiliser pour améliorer les performances de la requête. Mais dans certaines situations, le planificateur ne choisit pas d'utiliser les index existants, et les requêtes finissent par utiliser des balayages séquentiels lents au lieu d'un index spatial.

Si vous constatez que vos index spatiaux ne sont pas utilisés, il y a plusieurs choses que vous pouvez faire :

— Examinez le plan de requête et vérifiez que votre requête calcule effectivement ce dont vous avez besoin. Un JOIN erroné, oublié ou vers la mauvaise table, peut récupérer plusieurs fois des enregistrements de la table de manière inattendue. Pour obtenir le plan de la requête, exécutez avec `EXPLAIN` devant la requête.

— Assurez-vous que des statistiques sont collectées sur le nombre et la distribution des valeurs dans une table, afin de fournir au planificateur de requêtes de meilleures informations pour prendre des décisions concernant l'utilisation de l'index. La **VACUUM ANALYZE** calculera les deux.

  Vous devriez de toute façon vacuum régulièrement vos bases de données. De nombreux administrateurs de bases de données PostgreSQL exécutent **VACUUM** en tant que tâche cron en dehors des heures de pointe sur une base régulière.

— Si vacuum ne suffit pas, vous pouvez temporairement forcer le planificateur à utiliser les informations d'index en utilisant la commande **SET ENABLE_SEQSCAN TO OFF;**. De cette façon, vous pouvez vérifier si le planificateur est capable de générer un plan de requête accéléré par l'index pour votre requête. Vous ne devez utiliser cette commande qu'à des fins de débogage ; en règle générale, le planificateur sait mieux que vous quand utiliser les index. Une fois que vous avez exécuté votre requête, n'oubliez pas de lancer la commande **SET ENABLE_SEQSCAN TO ON;** afin que le planificateur fonctionne normalement pour les autres requêtes.

— Si **SET ENABLE_SEQSCAN TO OFF;** aide votre requête à s'exécuter plus rapidement, votre Postgres n'est probablement pas adapté à votre matériel. Si vous trouvez que le planificateur se trompe sur le coût des balayages séquentiels par rapport aux balayages d'index, essayez de réduire la valeur de `RANDOM_PAGE_COST` dans `postgresql.conf`, ou utilisez **SET RANDOM_PAGE_COST TO 1.1;**. La valeur par défaut de `RANDOM_PAGE_COST` est 4.0. Essayez de la fixer à 1.1 (pour les disques SSD) ou à 2.0 (pour les disques magnétiques rapides). En diminuant la valeur, le planificateur est plus enclin à utiliser les balayages d'index.

— Si la **SET ENABLE_SEQSCAN TO OFF;** n'aide pas votre requête, il se peut qu'elle utilise une construction SQL que le planificateur Postgres n'est pas encore en mesure d'optimiser. Il peut être possible de réécrire la requête de manière à ce que le planificateur soit en mesure de la traiter. Par exemple, une sous-requête avec un SELECT en ligne peut ne pas produire un plan efficace, mais peut être réécrite en utilisant un JOIN LATERAL.

Pour plus d'informations, voir la section du manuel Postgres sur Query Planning.

Chapitre 5

Requêtes spatiales

La *raison d'être* des bases de données spatiales est de réaliser à l'intérieur de la base de données des requêtes qui normalement nécessiteraient des fonctionnalités d'un SIG Desktop. Utiliser PostGIS requiert en effet la connaissance de quelles fonctions spatiales sont disponibles, comment les utiliser dans une requête, et de s'assurer de la mise en place adéquate des index, pour une bonne performance.

5.1 Déterminer les relations spatiales

Les relations spatiales indiquent comment deux géométries interagissent l'une avec l'autre. Elles constituent une fonctionnalité fondamentale pour effectuer des requêtes sur des géométries.

5.1.1 Modèle à 9 intersections dimensionnellement étendu

Selon le OpenGIS Simple Features Implementation Specification for SQL, "l'approche de base pour comparer deux géométries consiste à effectuer des tests par paire des intersections entre les intérieurs, les limites et les extérieurs des deux géométries et à classer la relation entre les deux géométries sur la base des entrées de la matrice "intersection" résultante."

Dans la théorie de la topologie des ensembles de points, les points d'une géométrie intégrée dans un espace à deux dimensions sont classés en trois ensembles :

Limites

La limite d'une géométrie est l'ensemble des géométries de la dimension immédiatement inférieure. Pour les `POINT`, qui ont une dimension de 0, la limite est l'ensemble vide. La limite d'une `LINESTRING` est constituée par ses deux extrémités. Pour les `POLYGON`, la limite est le réseau de lignes des anneaux extérieur et intérieur.

Intérieur

L'intérieur d'une géométrie est constitué des points de la géométrie qui ne se trouvent pas dans la limite. Pour les `POINT`, l'intérieur est le point lui-même. L'intérieur d'une `LINESTRING` est l'ensemble des points situés entre les extrémités. Pour les `POLYGON`, l'intérieur est la surface aréale à l'intérieur du polygone.

Extérieur

L'extérieur d'une géométrie est le reste de l'espace dans lequel la géométrie est intégrée ; en d'autres termes, tous les points qui ne se trouvent pas à l'intérieur ou sur la limite de la géométrie. Il s'agit d'une surface non fermée à 2 dimensions.

Le Dimensionally Extended 9-Intersection Model (DE-9IM) décrit la relation spatiale entre deux géométries en spécifiant les dimensions des 9 intersections entre les ensembles ci-dessus pour chaque géométrie. Les dimensions des intersections peuvent être représentées formellement dans une **matrice d'intersections** de 3x3.

Pour une géométrie *g*, les *Limites*, *Intérieures*, et *Extérieures* sont désignés par la notation *I(g)*, *B(g)*, et *E(g)*. En outre, *dim(s)* désigne la dimension d'un ensemble *s* dont le domaine est `{0,1,2,F}` :

— `0` => point

— `1` => ligne

— `2` => area

— `F` => ensemble vide

En utilisant cette notation, la matrice d'intersection pour deux géométries *a* et *b* est :

	Intérieur	**Limite**	**Extérieur**
Intérieur	*dim(I(a) ∩ I(b))*	*dim(I(a) ∩ B(b))*	*dim(I(a) ∩ E(b))*
Limite	*dim(B(a) ∩ I(b))*	*dim(B(a) ∩ B(b))*	*dim(B(a) ∩ E(b))*
Extérieur	*dim(E(a) ∩ I(b))*	*dim(E(a) ∩ B(b))*	*dim(E(a) ∩ E(b))*

The following overlapping polygons provide a concrete example. ST_Relate computes the matrix `212101212`; it matches the overlap pattern `T*T***T**` used by ST_Overlaps.

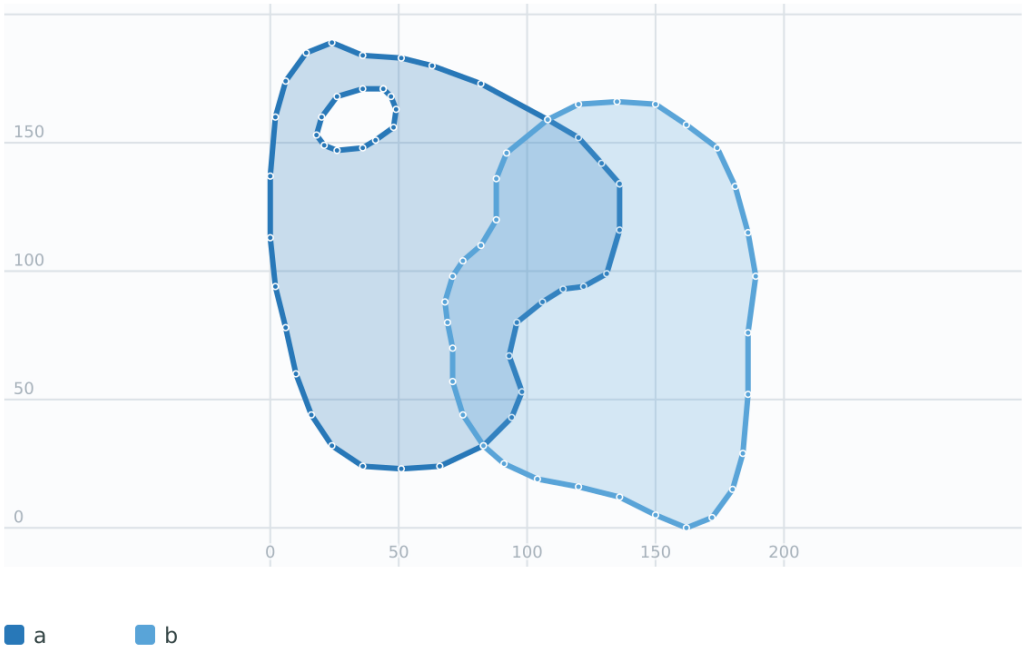
Code

```
WITH example AS (
  SELECT
    'POLYGON ((140 140,140 122,135 105,126 100,118 99,110 94,100 86,97 73,102 59,98 49,87 ↩
    38,70 30,55 29,40 30,28 38,20 50,14 66,10 84,6 100,4 119,4 143,6 166,10 180,18 191,28 ↩
    195,40 190,55 189,67 186,86 179,112 165,124 158,133 148,140 140),(48 177,40 177,30 ↩
    174,24 166,22 159,25 155,30 153,40 154,45 157,52 162,53 169,51 174,48 177))'::geometry ↩
    AS a,
    'POLYGON ((75 63,79 50,87 38,95 31,108 25,124 22,140 18,154 11,166 6,176 10,184 21,188 ↩
    35,190 58,190 82,193 104,190 121,185 139,178 154,166 163,154 171,139 172,124 171,112 ↩
    165,96 152,92 142,92 126,86 116,79 110,75 104,72 94,73 86,75 76,75 63))'::geometry AS b
), relation AS (
  SELECT a, b, ST_Relate(a, b) AS matrix
  FROM example
)
SELECT matrix,
       ST_RelateMatch(matrix, 'T*T***T**') AS matches_overlap_pattern,
       ST_Overlaps(a, b) AS overlaps
FROM relation;
```

Output

```
matrix   | matches_overlap_pattern | overlaps
-----------+-------------------------+----------
 212101212 | t                       | t
(1 row)
```

Figure



The executable query and generated overlay above are the visual source for the relation. The query below renders the nine cells of the same kind of matrix as a 3-by-3 set of examples. The final exterior/exterior cell is clipped to a finite viewport, since the true exterior/exterior intersection is unbounded.

Code

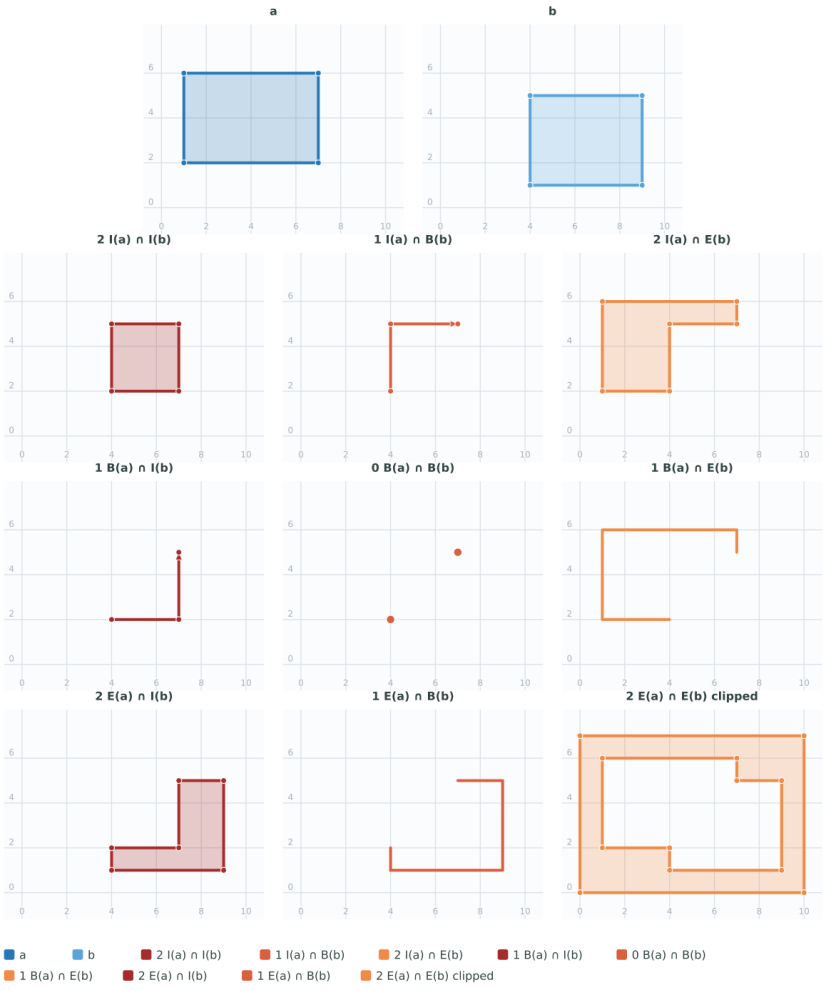
```
WITH example AS (
  SELECT
    ST_MakeEnvelope(0, 0, 6, 4) AS a,
    ST_MakeEnvelope(3, -1, 8, 3) AS b,
    ST_MakeEnvelope(-1, -2, 9, 5) AS viewport
), parts AS (
  SELECT
    a,
    b,
    ST_Boundary(a) AS ba,
    ST_Boundary(b) AS bb,
    viewport
  FROM example
)
SELECT
  ST_Relate(a, b) AS matrix,
  ST_AsText(ST_Normalize(a)) AS input_a,
  ST_AsText(ST_Normalize(b)) AS input_b,
  ST_AsText(ST_Normalize(ST_Intersection(a, b))) AS "2 I(a) ∩ I(b)",
  ST_AsText(ST_Normalize(ST_Intersection(a, bb))) AS "1 I(a) ∩ B(b)",
  ST_AsText(ST_Normalize(ST_Difference(a, b))) AS "2 I(a) ∩ E(b)",
  ST_AsText(ST_Normalize(ST_Intersection(ba, b))) AS "1 B(a) ∩ I(b)",
  ST_AsText(ST_Normalize(ST_Intersection(ba, bb))) AS "0 B(a) ∩ B(b)",
  ST_AsText(ST_Normalize(ST_Collect(
      'LINESTRING(0 0,0 4,6 4,6 3)'::geometry,
      'LINESTRING(0 0,3 0)'::geometry
  ))) AS "1 B(a) ∩ E(b)",
  ST_AsText(ST_Normalize(ST_Difference(b, a))) AS "2 E(a) ∩ I(b)",
  ST_AsText(ST_Normalize(ST_Collect(
      'LINESTRING(3 -1,8 -1,8 3,6 3)'::geometry,
      'LINESTRING(3 -1,3 0)'::geometry
```

```
  ))) AS "1 E(a) ∩ B(b)",
  ST_AsText(ST_Normalize(ST_Difference(viewport, ST_Union(a, b))))
    AS "2 E(a) ∩ E(b) clipped"
FROM parts;
```

Output

```
-[ RECORD 1 ]----------+--------------------------------------------------------------------------------
matrix                 | 212101212
input_a                | POLYGON((0 0,0 4,6 4,6 0,0 0))
input_b                | POLYGON((3 -1,3 3,8 3,8 -1,3 -1))
2 I(a) ∩ I(b)          | POLYGON((3 0,3 3,6 3,6 0,3 0))
1 I(a) ∩ B(b)          | LINESTRING(3 0,3 3,6 3)
2 I(a) ∩ E(b)          | POLYGON((0 0,0 4,6 4,6 3,3 3,3 0,0 0))
1 B(a) ∩ I(b)          | LINESTRING(3 0,6 0,6 3)
0 B(a) ∩ B(b)          | MULTIPOINT((6 3),(3 0))
1 B(a) ∩ E(b)          | MULTILINESTRING((0 0,0 4,6 4,6 3),(0 0,3 0))
2 E(a) ∩ I(b)          | POLYGON((3 -1,3 0,6 0,6 3,8 3,8 -1,3 -1))
1 E(a) ∩ B(b)          | MULTILINESTRING((3 -1,8 -1,8 3,6 3),(3 -1,3 0))
2 E(a) ∩ E(b) clipped  | POLYGON((-1 -2,-1 5,9 5,9 -2,-1 -2),(0 0,3 0,3 -1,8 -1,8 3,6 3,6 4,0 4,0 0))
```

Figure



En lisant de gauche à droite et de haut en bas, la matrice d'intersection est représentée par la chaîne de texte "**212101212**".

Pour plus d'informations, voir :

— OpenGIS Simple Features Implementation Specification for SQL (version 1.1, section 2.1.13.2)

— Wikipedia : Dimensionally Extended Nine-Intersection Model (DE-9IM)

— GeoTools : Théorie des ensembles de points et matrice DE-9IM

5.1.2 Relations spatiales nommées

Pour faciliter la détermination des relations spatiales communes, l'OGC SFS définit un ensemble de *prédicats de relations spatiales*. PostGIS fournit ces prédicats sous la forme des fonctions ST_Contains, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within. Il définit également les prédicats de relation non standard ST_Covers, ST_CoveredBy, et ST_ContainsProperly.

Les prédicats spatiaux sont généralement utilisés comme conditions dans les clauses SQL `WHERE` ou `JOIN`. Les prédicats spatiaux nommés utilisent automatiquement un index spatial s'il existe, de sorte qu'il n'est pas nécessaire d'utiliser également l'opérateur de boîte de délimitation `&&`. Par exemple :

Code

```
SELECT city.name, state.name, city.geom
FROM city JOIN state ON ST_Intersects(city.geom, state.geom);
```

Pour plus de détails et d'illustrations, voir l' Atelier PostGIS.

5.1.3 Relations spatiales générales

In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition. These requirements can be expressed by computing the full DE-9IM intersection matrix with ST_Relate.

Pour tester une relation spatiale particulière, un **modèle de matrice d'intersection** est utilisé. Il s'agit de la représentation matricielle augmentée des symboles supplémentaires `{T,*}` :

— `T` => la dimension d'intersection est non vide ; c'est-à-dire qu'elle se trouve dans `{0,1,2}`

— `*` => indifférent

Using intersection matrix patterns, specific spatial relationships can be evaluated succinctly. The ST_Relate and ST_RelateMatch functions can both test these patterns.

These road segments share a line segment. ST_Crosses does not find this relationship for linear features, because it only returns `true` when their interiors intersect at a point. The pattern `1*1***1**` tests for a one-dimensional intersection instead.

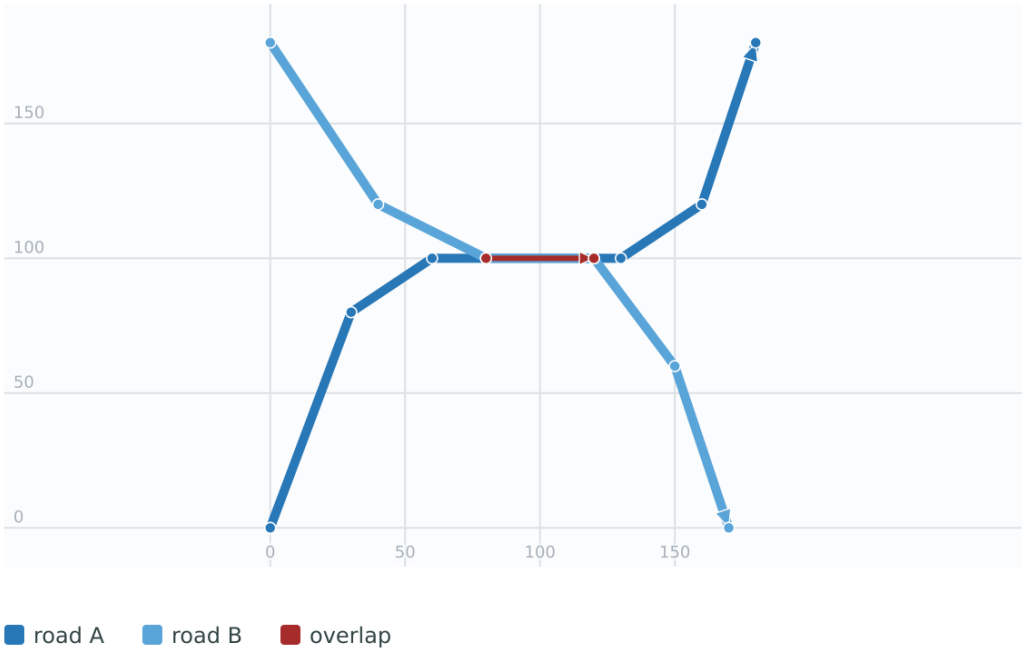
Code

```
WITH roads AS (
  SELECT
    'LINESTRING(10 10,40 90,70 110,140 110,170 130,190 190)'::geometry AS "road A",
    'LINESTRING(10 190,50 130,90 110,130 110,160 70,180 10)'::geometry AS "road B"
), relation AS (
  SELECT "road A", "road B", ST_Relate("road A", "road B") AS matrix
  FROM roads
)
SELECT
  ST_Intersection("road A", "road B") AS overlap,
  matrix,
  ST_Crosses("road A", "road B") AS "ST_Crosses",
  ST_RelateMatch(matrix, '1*1***1**') AS "matches 1*1***1**"
FROM relation;
```

Output

```
overlap           |  matrix   | ST_Crosses | matches 1*1***1**
--------------------------+-----------+------------+-------------------
 LINESTRING(90 110,130 110) | 1F1FF0102 | f          | t
(1 row)
```

Figure



The next example finds a wharf which runs from the lake interior onto its shoreline, with one endpoint on the shoreline. The other wharves provide nearby counterexamples in the figure. The pattern `102101FF2` expresses the complete required relationship in one test.

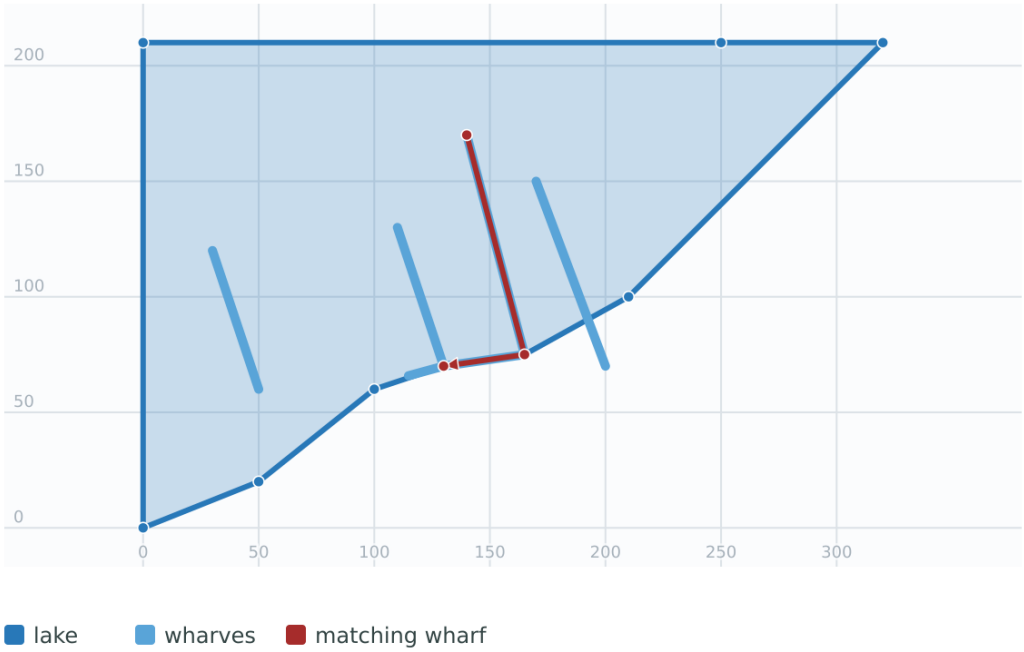
Code

```
WITH data AS (
  SELECT
    'POLYGON((-20 10,30 30,80 70,110 80,145 85,190 110,300 220,230 220,-20 220,-20 10))':: ↩
   geometry AS lake,
    'MULTILINESTRING((120 180,145 85,110 80),(10 130,30 70),(90 140,110 80,94.81118881118876 ↩
     75.83496503496498),(150 160,180 80))'::geometry AS wharves
), wharf AS (
  SELECT lake, (item).path[1] AS id, (item).geom
  FROM data CROSS JOIN LATERAL ST_Dump(wharves) AS item
), relation AS (
  SELECT id, geom, ST_Relate(lake, geom) AS matrix
  FROM wharf
)
SELECT
  id AS wharf,
  geom AS "matching wharf",
  matrix
FROM relation
WHERE ST_RelateMatch(matrix, '102101FF2')
ORDER BY id;
```

Output

```
wharf |           matching wharf           |  matrix
-------+------------------------------------+-----------
     1 | LINESTRING(120 180,145 85,110 80) | 102101FF2
(1 row)
```

Figure



5.2 Utilisation des index spatiaux

Lors de la construction de requêtes utilisant des conditions spatiales, il est important, pour de meilleures performances, de s'assurer qu'un index spatial est utilisé, s'il existe (voir Section 4.9). Pour ce faire, un opérateur spatial ou une fonction sensible à l'index doit être utilisé dans une clause `WHERE` ou `ON` de la requête.

Les opérateurs spatiaux comprennent les opérateurs de boîte de délimitation (dont le plus couramment utilisé est &&; voir Section 7.10.1 pour la liste complète) et les opérateurs de distance utilisés dans les requêtes sur les plus proches voisins (le plus courant étant <->; voir Section 7.10.2 pour la liste complète.)

Les fonctions tenant compte de l'index ajoutent automatiquement un opérateur de boîte de délimitation à la condition spatiale. Les fonctions tenant compte de l'index comprennent les prédicats de relation spatiale nommés ST_Contains, ST_ContainsProperly, ST_CoveredBy, ST_Covers, ST_Crosses, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within, ST_Within, et ST_3DIntersects, et les prédicats de distance ST_DWithin, ST_DFullyWithin, ST_3DDFullyWithin, et ST_3DDWithin .)

Les fonctions telles que ST_Distance n'utilisent *pas* les index pour optimiser leur fonctionnement. Par exemple, la requête suivante serait assez lente sur une grande table :

Code

```
SELECT geom
FROM geom_table
WHERE ST_Distance(geom, 'SRID=312;POINT(100000 200000)') < 100
```

Cette requête sélectionne toutes les géométries de la `geom_table` qui se trouvent à moins de 100 unités du point (100000, 200000). Elle sera lente car elle calcule la distance entre chaque point du tableau et le point spécifié, c'est-à-dire qu'un calcul `ST_Distance()` est effectué pour **chaque** ligne du tableau.

Le nombre de lignes traitées peut être considérablement réduit en utilisant la fonction d'indexation ST_DWithin :

Code

```
SELECT geom
FROM geom_table
WHERE ST_DWithin(geom, 'SRID=312;POINT(100000 200000)', 100)
```

Cette requête sélectionne les mêmes géométries, mais de manière plus efficace. Ceci est possible en `ST_DWithin()` utilisant l'opérateur `&&` en interne sur une boîte de délimitation élargie de la géométrie de la requête. S'il existe un index spatial sur `geom`, le planificateur de requêtes reconnaîtra qu'il peut utiliser l'index pour réduire le nombre de lignes analysées avant de calculer la distance. L'index spatial permet d'extraire uniquement les enregistrements dont les boîtes de délimitation chevauchent l'étendue élargie et qui, par conséquent, *pourraient* se trouver à l'intérieur de la distance requise. La distance réelle est ensuite calculée pour confirmer l'inclusion de l'enregistrement dans l'ensemble des résultats.

Pour plus d'informations et d'exemples, voir l' atelier de travail PostGIS.

5.3 Exemples de SQL spatial

Les exemples de cette section utilisent une table de routes linéaires et une table de limites de communes polygonales. La définition de la table `bc_roads` est la suivante :

Output

```
Column    | Type              | Description
----------+-------------------+-------------------
gid       | integer           | Unique ID
name      | character varying | Road Name
geom      | geometry          | Location Geometry (Linestring)
```

La définition de la table `bc_municipality` est la suivante :

Output

```
Column   | Type              | Description
---------+-------------------+-------------------
gid      | integer           | Unique ID
code     | integer           | Unique ID
name     | character varying | City / Town Name
geom     | geometry          | Location Geometry (Polygon)
```

1. *Quelle est la longueur totale de toutes les routes, exprimée en kilomètres ?*

   Vous pouvez répondre à cette question à l'aide d'un code SQL très simple :

   Code

   ```
   SELECT sum(ST_Length(geom))/1000 AS km_roads FROM bc_roads;
   ```

   Output

   ```
   km_roads
   ------------------
   70842.1243039643
   ```

2. *Quelle est la superficie de la ville de Prince George, en hectares ?*

   Cette requête combine une condition d'attribut (sur le nom de la municipalité) avec un calcul spatial (de la surface du polygone) :

Code

```
SELECT
  ST_Area(geom)/10000 AS hectares
FROM bc_municipality
WHERE name = 'PRINCE GEORGE';
```

Output

```
hectares
------------------
32657.9103824927
```

3. *Quelle est la plus grande municipalité de la province, en termes de superficie ?*

   Cette requête utilise une mesure spatiale comme valeur de référence. Il existe plusieurs façons d'aborder ce problème, mais la plus efficace est la suivante :

   Code

   ```
   SELECT
     name,
     ST_Area(geom)/10000 AS hectares
   FROM bc_municipality
   ORDER BY hectares DESC
   LIMIT 1;
   ```

   Output

   ```
   name           | hectares
   ---------------+-----------------
   TUMBLER RIDGE  | 155020.02556131
   ```

   Notez que pour répondre à cette question, nous devons calculer la surface de chaque polygone. Si nous faisions cela souvent, il serait logique d'ajouter une colonne de surface à la table qui pourrait être indexée pour des raisons de performance. En ordonnant les résultats dans une direction décroissante et en utilisant la commande "LIMIT" de PostgreSQL, nous pouvons facilement sélectionner la plus grande valeur sans utiliser une fonction d'agrégation comme MAX().

4. *Quelle est la longueur des routes entièrement comprises dans chaque municipalité ?*

   Il s'agit d'un exemple de "jointure spatiale", qui rassemble les données de deux tables (avec une jointure) en utilisant une interaction spatiale ("contenu") comme condition de jointure (plutôt que l'approche relationnelle habituelle de jointure sur une clé commune) :

   Code

   ```
   SELECT
     m.name,
     sum(ST_Length(r.geom))/1000 as roads_km
   FROM bc_roads AS r
   JOIN bc_municipality AS m
     ON ST_Contains(m.geom, r.geom)
   GROUP BY m.name
   ORDER BY roads_km;
   ```

   Output

   ```
   name                        | roads_km
   ----------------------------+----------
   SURREY                      | 1539.476
   VANCOUVER                   | 1450.331
   LANGLEY DISTRICT            |  833.793
   BURNABY                     |  773.769
   ```

```
PRINCE GEORGE               |  694.376
...
```

Cette requête prend un certain temps, car chaque route du tableau est résumée dans le résultat final (environ 250 000 routes pour le tableau de l'exemple). Pour des ensembles de données plus petits (plusieurs milliers d'enregistrements sur plusieurs centaines), la réponse peut être très rapide.

5. *Créez un nouveau tableau avec toutes les routes de la ville de Prince George.*

   Il s'agit d'un exemple de "superposition", qui prend en compte deux tableaux et produit un nouveau tableau composé de résultats coupés dans l'espace. Contrairement à la "jointure spatiale" présentée ci-dessus, cette requête crée de nouvelles géométries. Une superposition est comme une jointure spatiale turbocompressée, et est utile pour des travaux d'analyse plus précis :

   Code

   ```
   CREATE TABLE pg_roads as
   SELECT
     ST_Intersection(r.geom, m.geom) AS intersection_geom,
     ST_Length(r.geom) AS rd_orig_length,
     r.*
   FROM bc_roads AS r
   JOIN bc_municipality AS m
     ON ST_Intersects(r.geom, m.geom)
   WHERE
     m.name = 'PRINCE GEORGE';
   ```

6. *Quelle est la longueur en kilomètres de "Douglas St" à Victoria ?*

   Code

   ```
   SELECT
     sum(ST_Length(r.geom))/1000 AS kilometers
   FROM bc_roads r
   JOIN bc_municipality m
     ON ST_Intersects(m.geom, r.geom
   WHERE
     r.name = 'Douglas St'
     AND m.name = 'VICTORIA';
   ```

   Output

   ```
   kilometers
   ------------------
   4.89151904172838
   ```

7. *Quel est le plus grand polygone municipal comportant un trou ?*

   Code

   ```
   SELECT gid, name, ST_Area(geom) AS area
   FROM bc_municipality
   WHERE ST_NRings(geom)
   > 1
   ORDER BY area DESC LIMIT 1;
   ```

   Output

   ```
   gid  | name         | area
   -----+--------------+------------------
   12   | SPALLUMCHEEN | 257374619.430216
   ```

Chapitre 6

Conseils sur les performances

6.1 Petites tables de grandes géométries

6.1.1 Description du problème

Current PostgreSQL versions suffer from a query optimizer weakness regarding TOAST tables. TOAST tables are a kind of "extension room" used to store large (in the sense of data size) values that do not fit into normal data pages (like long texts, images or complex geometries with lots of vertices), see the PostgreSQL Documentation for TOAST for more information).

Le problème apparaît s'il vous arrive d'avoir une table avec d'assez grandes géométries, mais pas beaucoup de lignes d'entre elles (comme un tableau contenant les frontières de tous les pays européens en haute résolution). Ensuite, le tableau lui-même est petit, mais il utilise beaucoup d'espace TOAST. Dans notre exemple, le cas, la table elle-même avait environ 80 lignes et seulement 3 pages de données utilisées, mais la table TOAST 8225 pages utilisé.

Maintenant émettre une requête en utilisant l'opérateur de géométrie && pour rechercher une boîte englobante qui correspond que très peu de ces lignes. Maintenant l'optimiseur de requêtes voit que la table n'a que 3 pages et 80 lignes. Il estime qu'une analyse séquentielle sur une telle petite table est beaucoup plus rapide que d'utiliser un index. Et alors il décide d'ignorer l'index GIST. Habituellement, cette estimation est correcte. Mais dans notre cas, l'opérateur && doit aller chercher chaque géométrie à partir du disque pour comparer les boîtes englobantes, et par conséquent la lecture de toutes les pages TOAST également.

Pour voir si vous souffrez de ce problème, utilisez la commande postgresql "EXPLAIN ANALYZE". Pour plus d'informations et les détails techniques, vous pouvez lire le fil de discussion sur la liste de diffusion sur les performances de PostgreSQL : http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php

et un fil d'actualités plus récent sur PostGIS https://lists.osgeo.org/pipermail/postgis-devel/2017-June/026209.html

6.1.2 Solutions de contournement

Les personnes de PostgreSQL essayent de résoudre ce problème en faisant l'estimation de la requête TOAST-courant. Pour l'instant, voici deux solutions :

La première solution consiste à forcer le planificateur de requêtes à utiliser l'index. Envoyer "SET enable_seqscan TO off;" au serveur avant d'émettre la requête. Cela force le planificateur de requêtes à éviter balayages séquentiels lorsque cela est possible. Donc, il utilise l'index GIST comme d'habitude. Mais cet indicateur doit être fixé à chaque connexion, et il provoque le planificateur de requêtes à faire des erreurs d'estimation dans les autres cas, vous devrez donc faire "SET POUR enable_seqscan sur;" après la requête.

La deuxième solution consiste à faire le balayage séquentielle aussi vite que le planificateur de requêtes pense. Ceci peut être réalisé en créant une colonne supplémentaire qui "cache" la bbox, et contre cette correspondance. Dans notre exemple, les commandes sont comme :

Code

```
SELECT AddGeometryColumn('myschema', 'mytable', 'bbox', '4326', 'GEOMETRY', '2');
UPDATE mytable SET bbox = ST_Envelope(ST_Force2D(geom));
```

Modifiez maintenant votre requête pour utiliser l'opérateur && contre bbox au lieu de geom_column, comme suit :

Code

```
SELECT geom_column
FROM mytable
WHERE bbox && ST_SetSRID('BOX3D(0 0,1 1)'::box3d,4326);
```

Bien sûr, si vous changez ou ajoutez des lignes à mytable, vous devez garder la bbox "synchro". La façon la plus transparente pour ce faire serait des déclencheurs, mais vous pouvez également modifier votre application afin de maintenir la colonne bbox courante ou exécuter la requête UPDATE ci-dessus après chaque modification.

6.2 CLUSTER d'index géométriques

The PostgreSQL CLUSTER command physically reorders table rows according to an index. For a mostly read-only table whose queries frequently use the same spatial index, this can reduce I/O and improve cache locality.

PostgreSQL supports clustering on GiST indices, including indices on nullable geometry columns. To cluster a table using a geometry index:

Code

```
CLUSTER my_table USING my_geom_index;
```

Output

```
CLUSTER
```

6.3 Eviter les conversions de dimension

Il arrive parfois que vous ayez des données 3D ou 4D dans votre table, mais que vous y accédiez toujours à l'aide des fonctions ST_AsText() ou ST_AsBinary() conformes à OpenGIS qui ne produisent que des géométries 2D. Pour ce faire, elles appellent en interne la fonction ST_Force2D(), qui introduit une surcharge importante pour les géométries de grande taille. Pour éviter cette surcharge, il peut être possible de pré-déposer ces dimensions supplémentaires une fois pour toutes :

Code

```
UPDATE mytable SET geom = ST_Force2D(geom);
VACUUM FULL ANALYZE mytable;
```

Notez que si vous avez ajouté votre colonne de géométrie à l'aide AddGeometryColumn (), il y aura une contrainte sur la dimension de la géométrie. Pour contourner vous devrez supprimer la contrainte. N'oubliez pas de mettre à jour l'entrée dans la table geometry_columns et recréer la contrainte par la suite.

En cas de grandes tables, il peut être judicieux de diviser cette mise à jour en petites portions en restreignant l'UPDATE à une partie de la table via une clause WHERE et votre clé primaire ou d'un autre critère, et exécutant un simple «VACUUM» ; entre votre mises à jour. Cela réduit considérablement le besoin d'espace disque temporaire. En outre, si vous avez des données géométriques de dimension mixte, restreindre la mise à jour en "WHERE dimension(the_geom)>2" saute la ré-écriture des géométries qui sont déjà en 2D.

Chapitre 7

Référence PostGIS

Les fonctions suivantes sont celles dont un utilisateur normal de PostGIS peut avoir besoin. Il existe d'autres fonctions nécessaires au fonctionnement de PostGIS. Elles ne sont normalement pas destinées à un utilisateur standard.



Note
PostGIS a entamé une phase de transition concernant le nom des fonctions pour les faire correspondre à la norme SQL-MM. La plupart des fonctions ont été renommées en utilisant le préfixe des types spatiaux (ST, pour Spatial Type). Les anciennes fonctions, toujours disponibles, ne sont pas listées ci-après si leur équivalent est disponible. Les fonctions non préfixées par ST, qui ne sont pas listées dans cette documentation, sont dépréciées et seront supprimées des prochaines version de PostGIS. IL NE FAUT PLUS LES UTILISER.

7.1 Types de données PostGIS Geometry/Geography/Box

7.1.1 box2d

box2d — Le type représentant une boîte de délimitation bidimensionnelle.

Description

`box2d` est un type de données spatiales utilisé pour représenter la boîte de délimitation bidimensionnelle entourant une géométrie ou une collection de géométries. Par exemple, la fonction d'agrégation ST_Extent renvoie un objet `box2d`.

La représentation contient les valeurs `xmin, ymin, xmax, ymax`. Ce sont les valeurs minimales et maximales des étendues X et Y.

Les objets `box2d` ont une représentation textuelle qui ressemble à `BOX(1 2,5 6)`.

Transtypages

Ce tableau énumère les casts automatiques et explicites autorisés pour ce type de données :

Transtypage vers	Comportement
box3d	automatique
geometry	automatique

Voir aussi

Section 13.7

7.1.2 box3d

box3d — Le type représentant une boîte de délimitation tridimensionnelle.

Description

`box3d` est un type de données spatiales PostGIS utilisé pour représenter la boîte de délimitation tridimensionnelle entourant une géométrie ou une collection de géométries. Par exemple, la fonction d'agrégation ST_3DExtent renvoie un objet `box3d`.

La représentation contient les valeurs `xmin, ymin, zmin, xmax, ymax, zmax`. Il s'agit des valeurs minimales et maximales des étendues X, Y et Z.

Les objets `box3d` ont une représentation textuelle qui ressemble à `BOX3D(1 2 3,5 6 5)`.

Transtypages

Ce tableau énumère les casts automatiques et explicites autorisés pour ce type de données :

Transtypage vers	Comportement
box	automatique
box2d	automatique
geometry	automatique

Voir aussi

Section 13.7

7.1.3 geometry

geometry — Le type représentant les caractéristiques spatiales avec des systèmes de coordonnées planaires.

Description

`géométrie` est un type de données spatiales fondamental de PostGIS utilisé pour représenter une caractéristique dans des systèmes de coordonnées planes (euclidiennes).

Toutes les opérations spatiales sur la géométrie utilisent les unités du système de référence spatiale dans lequel se trouve la géométrie.

Transtypages

Ce tableau énumère les casts automatiques et explicites autorisés pour ce type de données :

Transtypage vers	Comportement
box	automatique
box2d	automatique
box3d	automatique
bytea	automatique
geography	automatique
text	automatique

Voir aussi

Section 4.1, Section 13.3

7.1.4 geometry_dump

geometry_dump — Un type composite utilisé pour décrire les parties d'une géométrie complexe.

Description

`geometry_dump` est un type de données composite contenant les champs :

— `geom` - une géométrie représentant un composant de la géométrie explosée. Le type de géométrie dépend de la fonction d'origine.

— `path[]` - un tableau d'entiers qui définit le chemin de navigation dans la géométrie explosée vers le composant `geom`. Le tableau de chemins est basé sur 1 (c'est-à-dire que `path[1]` est le premier élément)

Il est utilisé par la famille de fonctions `ST_Dump*` comme type de sortie pour exploser une géométrie complexe en ses parties constitutives.

Voir aussi

Section 13.6

7.1.5 geography

geography — Le type représentant les entités spatiales avec des systèmes de coordonnées géodésiques (ellipsoïdales).

Description

`geography` est un type de données spatiales utilisé pour représenter une entité dans les systèmes de coordonnées géodésiques. Les systèmes de coordonnées géodésiques modélisent la terre à l'aide d'un ellipsoïde.

Les opérations spatiales sur le type geography fournissent des résultats plus précis en prenant en compte le modèle ellipsoïdal.

Transtypages

Ce tableau énumère les casts automatiques et explicites autorisés pour ce type de données :

Transtypage vers	Comportement
geometry	explicite

Voir aussi

Section 4.3, Section 13.4

7.2 Fonctions de gestion des tables

7.2.1 AddGeometryColumn

AddGeometryColumn — Ajoute une colonne de géométrie à une table existante.

Synopsis

text **AddGeometryColumn**(varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);
text **AddGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);
text **AddGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension, boolean use_typmod=true);

Description

Ajoute une colonne géométrique à une table attributaire existante. `schema_name` est le nom du schéma de la table. `srid` est un entier positif présent dans la table SPATIAL_REF_SYS. `type` est le type de géométrie en texte, par exemple 'POLYGON' ou 'MULTILINESTRING'. Une erreur est renvoyée si le schéma n'existe pas (ou n'est pas visible dans le search_path courant) ou si le SRID, type de géométrie ou dimension est invalide.

Changement : 2.0.0 Cette fonction ne met plus à jour geometry_columns maintenant que geometry_columns est une vue basée sur le catalogue système. Par défaut, elle ne créée plus de contraintes mais utilise le modificateur de type de PostgreSQL. Ainsi, par exemple, créer une colonne de type POINT WGS84 est désormais équivalent à : `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326) ;`

Changement : 2.0.0 Si l'ancien mécanisme basé sur les contraintes est nécessaire, utiliser le paramètre `use_typmod` avec la valeur false.

Changement : 2.0.0 Les vues ne peuvent plus être enregistrées dans geometry_columns. Cependant, les vues construites à partir de tables contenant des géométries définies avec le modificateur de type et n'utilisant pas de fonctions d'encapsulation seront enregistrées dans la vue geometry_columns car elles héritent du mécanisme des tables dont elles sont issues. Les vues utilisant des fonctions renvoyant d'autres géométries doivent être transtypées vers des géométries avec modificateur de type pour pouvoir être correctement référencées dans la vue geometry_columns. Cf. Section 4.6.3.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Amélioration : 2.0.0 introduction du paramètre use_typmod. Le comportement par défaut est de créer une colonne géométrique avec modificateur de type au lieu de contraintes sur la colonne.

Exemples

This example creates a schema to hold data.

Code

```
CREATE SCHEMA my_schema;
```

Create a simple PostgreSQL table in the schema.

Code

```
CREATE TABLE my_schema.my_spatial_table (id serial);
```

Describing the table shows a single `id` column.

Code

```
\d my_schema.my_spatial_table
```

Output

```
Table "my_schema.my_spatial_table"
 Column |  Type   |                                 Modifiers
--------+---------+-------------------------------------------------------------------------
 id     | integer | not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)
```

Add a spatial column to the table.

Code

```
SELECT AddGeometryColumn('my_schema', 'my_spatial_table', 'geom', 4326, 'POINT', 2);
```

Add a point column using the old constraint-based behavior.

Code

```
SELECT AddGeometryColumn('my_schema', 'my_spatial_table', 'geom_c', 4326, 'POINT', 2, false) ↩
    ;
```

Add a CurvePolygon column using the old constraint-based behavior.

Code

```
SELECT AddGeometryColumn('my_schema', 'my_spatial_table', 'geomcp_c', 4326, 'CURVEPOLYGON',  ↩
    2, false);
```

Output

```
my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
```

Describe the table again to see the new geometry columns.

Code

```
\d my_schema.my_spatial_table
```

Output

```
Table "my_schema.my_spatial_table"

Columns:
  id       integer              not null default nextval('my_schema.my_spatial_table_id_seq'::regclass)
  geom     geometry(Point,4326)
  geom_c   geometry
  geomcp_c geometry

Check constraints:
    "enforce_dims_geom_c" CHECK (st_ndims(geom_c) = 2)
    "enforce_dims_geomcp_c" CHECK (st_ndims(geomcp_c) = 2)
    "enforce_geotype_geom_c" CHECK (geometrytype(geom_c) = 'POINT'::text OR geom_c IS NULL)
    "enforce_geotype_geomcp_c" CHECK (geometrytype(geomcp_c) = 'CURVEPOLYGON'::text OR geomcp_c IS NULL)
    "enforce_srid_geom_c" CHECK (st_srid(geom_c) = 4326)
    "enforce_srid_geomcp_c" CHECK (st_srid(geomcp_c) = 4326)
```

The `geometry_columns` view also registers the new columns.

Code

```
SELECT f_geometry_column As col_name, type, srid, coord_dimension As ndims
FROM geometry_columns
WHERE f_table_name = 'my_spatial_table' AND f_table_schema = 'my_schema';
```

Output

```
col_name |     type     | srid | ndims
----------+--------------+------+-------
 geom     | Point        | 4326 |     2
 geom_c   | Point        | 4326 |     2
 geomcp_c | CurvePolygon | 4326 |     2
```

Voir aussi

DropGeometryColumn, DropGeometryTable, Section 4.6.2, Section 4.6.3

7.2.2 DropGeometryColumn

DropGeometryColumn — Supprime une colonne géométrique d'une table spatiale.

Synopsis

text **DropGeometryColumn**(varchar table_name, varchar column_name);
text **DropGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);
text **DropGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name);

Description

Removes a geometry column from a spatial table. Note that schema_name will need to match the f_table_schema field shown by the `geometry_columns` view.

AVERTISSEMENT
`DropGeometryColumn` is retained for backward compatibility and is not recommended for new applications. Use the standard **ALTER TABLE ... DROP COLUMN** command instead. The `geometry_columns` view updates automatically from the system catalogs.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Changed: 2.0.0. Retained for backward compatibility after `geometry_columns` became a view of the system catalogs.

Exemples

Use standard SQL to drop the geometry column. The `geometry_columns` view updates automatically.

Code

```
ALTER TABLE my_schema.my_spatial_table DROP COLUMN geom;
```

The legacy compatibility function performs the equivalent operation.

Code

```
SELECT DropGeometryColumn ('my_schema', 'my_spatial_table', 'geom');
```

Output

```
my_schema.my_spatial_table.geom effectively removed.
```

Voir aussi

AddGeometryColumn, DropGeometryTable, Section 4.6.2

7.2.3 DropGeometryTable

DropGeometryTable — Supprime une table et toutes ces références dans geometry_columns.

Synopsis

boolean **DropGeometryTable**(varchar table_name);
boolean **DropGeometryTable**(varchar schema_name, varchar table_name);
boolean **DropGeometryTable**(varchar catalog_name, varchar schema_name, varchar table_name);

Description

Supprime une table et toutes ces références dans geometry_columns. Note : utilise la fonction current_schema() sur les installations PostgreSQL le supportant, si le schéma n'est pas fourni.



AVERTISSEMENT
`DropGeometryTable` is retained for backward compatibility and is not recommended for new applications. Use the standard **DROP TABLE** command instead. The `geometry_columns` view updates automatically from the system catalogs.

Changed: 2.0.0. Retained for backward compatibility after `geometry_columns` became a view of the system catalogs.

Exemples

Use standard SQL to drop a table containing geometry columns. The `geometry_columns` view updates automatically.

Code

```
DROP TABLE my_schema.my_spatial_table;
```

The legacy compatibility function performs the equivalent operation.

Code

```
SELECT DropGeometryTable ('my_schema', 'my_spatial_table');
```

Output

```
my_schema.my_spatial_table dropped.
```

Voir aussi

AddGeometryColumn, DropGeometryColumn, Section 4.6.2

7.2.4 Find_SRID

Find_SRID — Renvoie le SRID défini pour une colonne géométrique.

Synopsis

integer **Find_SRID**(varchar a_schema_name, varchar a_table_name, varchar a_geomfield_name);

Description

Renvoie le SRID entier de la colonne géométrique spécifiée en effectuant une recherche dans la table GEOMETRY_COLUMNS. Si la colonne géométrique n'a pas été correctement ajoutée (par exemple avec la fonction AddGeometryColumn), cette fonction ne fonctionnera pas.

Exemples

Code

```
SELECT Find_SRID('public', 'tiger_us_state_2007', 'geom_4269');
```

Output

```
4269
```

Voir aussi

ST_SRID

7.2.5 Populate_Geometry_Columns

Populate_Geometry_Columns — Assure que les colonnes géométriques sont définies avec des modificateurs de type ou qu'elles sont soumises à des contraintes spatiales appropriées.

Synopsis

text **Populate_Geometry_Columns**(boolean use_typmod=true);
int **Populate_Geometry_Columns**(oid relation_oid, boolean use_typmod=true);

Description

S'assure que les colonnes géométriques ont des modificateurs de type ou des contraintes spatiales appropriés pour garantir qu'elles sont enregistrées correctement dans la vue `geometry_columns`. Par défaut, convertira toutes les colonnes géométriques sans modificateur de type en colonnes avec modificateur de type.

Pour conserver la rétro compatibilité et pour des besoins particuliers comme par exemple des tables héritées ayant des types géométriques différents, l'ancien mécanisme est toujours supporté. Si ce mécanisme est nécessaire, le nouveau paramètre optionnel doit être mis à false : `use_typmod=false`. Avec cette valeur, la colonne géométrique sera créée sans modificateur de type mais 3 contraintes seront définies. Cela signifie concrètement que chaque colonne géométrique de la table aura au moins 3 contraintes :

— `enforce_dims_geom` - garantit que chaque géométrie a la même dimension (voir ST_NDims)
— `enforce_geotype_geom` - garantit que chaque géométrie est du même type (voir GeometryType)
— `enforce_srid_the_geom` - s'assure que toutes les géométries sont dans la même projection (see ST_SRID)

Si un identifiant de table `oid` est fourni, cette fonction tente de déterminer le SRID, la dimension et le type géométrique de toutes les colonnes géométriques de la table, ajoutant des contraintes si nécessaire. En cas de succès, une ligne est insérée dans la table geometry_columns, sinon, une erreur est affichée indiquant le problème.

Si le `oid` d'une vue est fourni, comme avec un oid de table, cette fonction essaie de déterminer le srid, la dimension et le type de toutes les géométries de la vue, en insérant les entrées appropriées dans la table `geometry_columns`, mais rien n'est fait pour appliquer les contraintes.

La version sans paramètre est un raccourci pour la version avec paramètres. Elle vide puis remplit la table geometry_columns pour chaque table ou vue spatiale de la base, ajoutant les contraintes aux tables si besoin. Retourne un résumé montrant le nombre de colonnes géométriques identifiées dans la base et le nombre inséré dans la table `geometry_columns`. La version avec paramètres renvoie juste le nombre de lignes insérées dans la table `geometry_columns` .

Disponibilité : 1.4.0

Changement : 2.0.0 Par défaut, utilise les modificateurs de type au lieu de contraintes de vérification pour contraindre les types géométriques. Le comportement basé sur les contraintes peut être activé en mettant le nouveau paramètre `use_typmod` à false.

Amélioration : 2.0.0 L'argument optionnel `use_typmod` a été introduit pour controler si les colonnes sont créés avec des modificateurs de type ou des contraintes de vérification.

Exemples

This example populates geometry column metadata using typmod behavior. For this to work, there must be existing data.

Code

```
CREATE TABLE public.myspatial_table(gid serial, geom geometry);
INSERT INTO myspatial_table(geom) VALUES(ST_GeomFromText('LINESTRING(1 2,3 4)', 4326) );
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);
```

Output

```
1
```

Describe the table to see the typmod-based geometry column.

Code

```
\d myspatial_table
```

Output

```
Table "public.myspatial_table"

Columns:
  gid  integer                    not null default nextval('myspatial_table_gid_seq':: ↩
    regclass)
  geom geometry(LineString,4326)
```

This example changes the geometry columns to use constraints if they are not typmod or already constrained. For this to work, there must be existing data.

Code

```
CREATE TABLE public.myspatial_table_cs(gid serial, geom geometry);
INSERT INTO myspatial_table_cs(geom) VALUES(ST_GeomFromText('LINESTRING(1 2,3 4)', 4326) );
SELECT Populate_Geometry_Columns('public.myspatial_table_cs'::regclass, false);
```

Output

```
1
```

Describe the table to see the constraint-based geometry column.

Code

```
\d myspatial_table_cs
```

Output

```
Table "public.myspatial_table_cs"
 Column |   Type   |                            Modifiers
--------+----------+-------------------------------------------------------------------
 gid    | integer  | not null default nextval('myspatial_table_cs_gid_seq'::regclass)
 geom   | geometry |
Check constraints:
    "enforce_dims_geom" CHECK (st_ndims(geom) = 2)
    "enforce_geotype_geom" CHECK (geometrytype(geom) = 'LINESTRING'::text OR geom IS NULL)
    "enforce_srid_geom" CHECK (st_srid(geom) = 4326)
```

7.2.6 UpdateGeometrySRID

UpdateGeometrySRID — Met à jour le SRID de tous les éléments d'une colonne géométrique et les métadonnées de la table.

Synopsis

text **UpdateGeometrySRID**(varchar table_name, varchar column_name, integer srid);
text **UpdateGeometrySRID**(varchar schema_name, varchar table_name, varchar column_name, integer srid);
text **UpdateGeometrySRID**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid);

Description

Met à jour le SRID de tous les objets d'une colonne géométrique et met à jour les métadonnées de geometry_columns et la contrainte sur le SRID. Note : utilise la fonction current_schema() sur les installations PostgreSQL le supportant, si le schéma n'est pas fourni.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Insérer des géométries dans une table roads avec un jeu de SRID utilisant déjà le format EWKT :

Code

```
COPY roads (geom) FROM STDIN;
SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

Cela va changer le srid de la table roads à 4326 quelle que soit sa valeur avant :

Code

```
SELECT UpdateGeometrySRID('roads', 'geom', 4326);
```

L'exemple précédent est équivalent à cette requête DDL :

Code

```
ALTER TABLE roads
ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 4326)
  USING ST_SetSRID(geom, 4326);
```

Si vous vous êtes trompé de projection (ou si vous l'avez introduite en tant qu'inconnue) dans le chargement et que vous voulez transformer en mercator web en une seule fois, vous pouvez le faire avec DDL mais il n'y a pas de fonction de gestion PostGIS équivalente pour le faire en une seule fois.

Code

```
ALTER TABLE roads
ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 3857) USING ST_Transform(ST_SetSRID(geom,  ↩
    4326), 3857) ;
```

Voir aussi

UpdateRasterSRID, ST_SetSRID, ST_Transform, ST_GeomFromEWKT

7.3 Constructeurs de géométries

7.3.1 ST_Collect

ST_Collect — Crée une géométrie GeometryCollection ou Multi à partir d'un ensemble de géométries.

Synopsis

geometry **ST_Collect**(geometry g1, geometry g2);
geometry **ST_Collect**(geometry[] g1_array);
geometry **ST_Collect**(geometry set g1field);

Description

Rassemble les géométries dans une collection de géométries. Le résultat est soit un Multi*, soit une GeometryCollection, selon que les géométries d'entrée ont des types identiques ou différents (homogènes ou hétérogènes). Les géométries d'entrée restent inchangées dans la collection.

Variante 1: accepte géométries en entrée

Variante 2: accepte un tableau de géométries

Variante 3: fonction agrégée acceptant un ensemble de géométries.



AVERTISSEMENT
`ST_Collect` does not dissolve boundaries or resolve overlaps. Collecting overlapping polygons can produce an invalid MultiPolygon. If the result must be a valid polygonal geometry with overlaps merged, use ST_Union instead. `ST_Union` also splits linestrings at intersections and may return a single geometry after dissolving boundaries.

If any input geometry is a collection (Multi* or GeometryCollection), `ST_Collect` returns a GeometryCollection, since that is the only type which can contain nested collections. To prevent this, use ST_Dump in a subquery to expand input collections to their atomic elements (see example below).

Disponibilité : 1.4.0 - ST_Collect(geomarray) a été introduit. ST_Collect a été amélioré pour gérer plus de géométries plus rapidement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Two-input variant.

Collectez les points 2D.

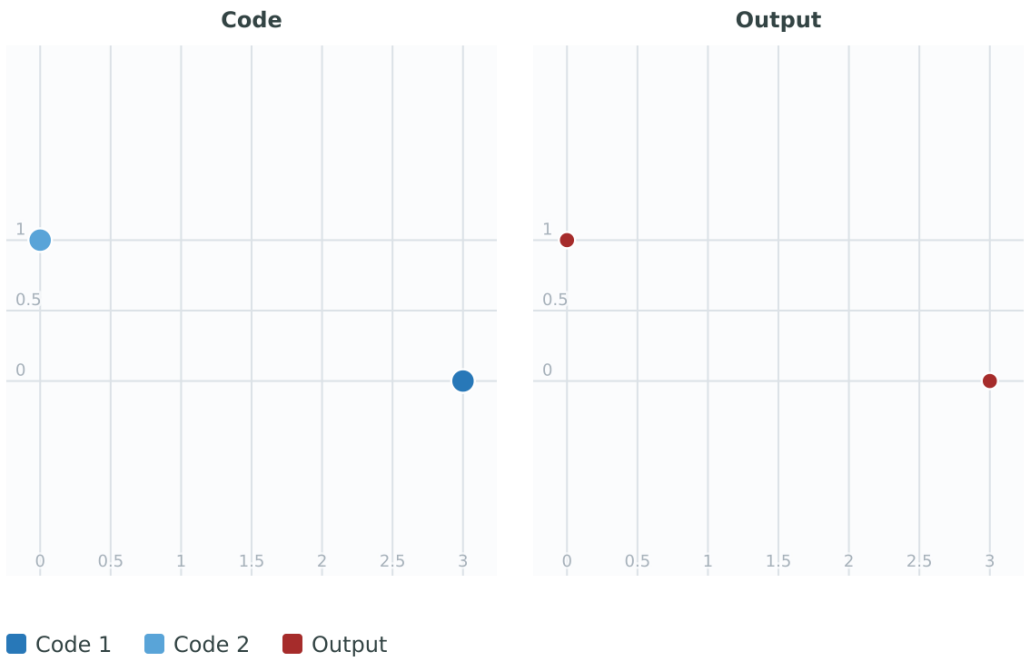
Code

```
SELECT ST_Collect(
    'POINT(1 2)',
    'POINT(-2 3)' );
```

Output

```
MULTIPOINT((1 2),(-2 3))
```

Figure



Collectez les points 3D.

Code

```
SELECT ST_Collect(
        'POINT(1 2 3)',
        'POINT(1 2 4)' );
```

Output

```
MULTIPOINT(1 2 3,1 2 4)
```

Figure

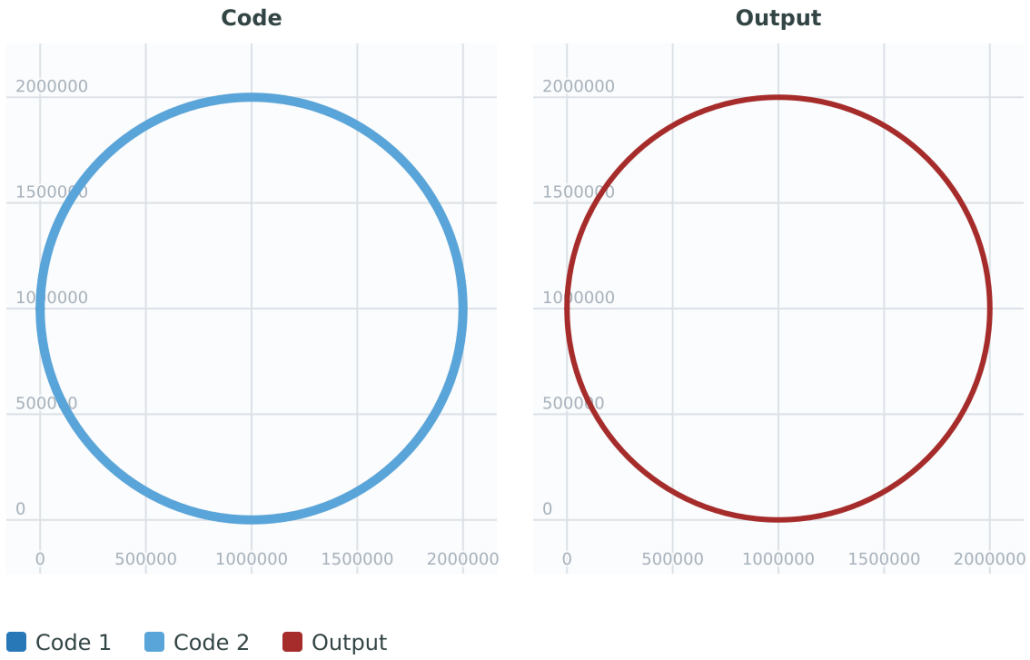


Collectez les courbes.

Code

```
SELECT ST_Collect(
        'CIRCULARSTRING(220268 150415,220227 150505,220227 150406)',
        'CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)');
```

Figure



Array variant.

Utilisation d'un constructeur de tableau pour une sous-requête.

Code

```
SELECT ST_Collect(ARRAY(SELECT geom FROM sometable));
```

Utilisation d'un constructeur de tableau pour les valeurs.

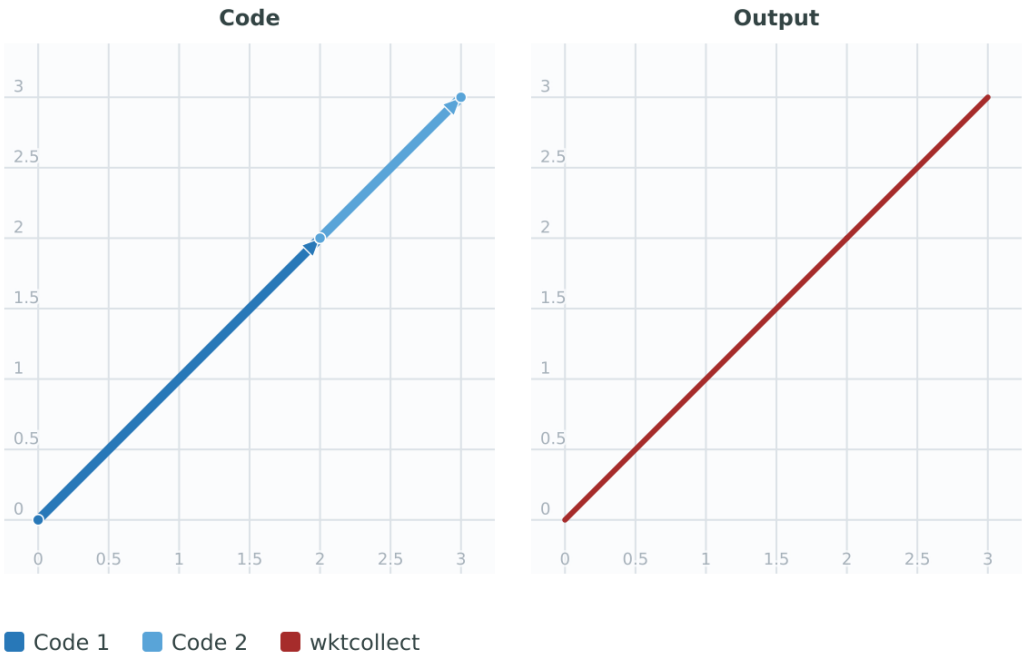
Code

```
SELECT ST_Collect(
        ARRAY[ 'LINESTRING(1 2,3 4)'::geometry,
            'LINESTRING(3 4,4 5)'::geometry ] ) As wktcollect;
```

Output

```
MULTILINESTRING((1 2,3 4),(3 4,4 5))
```

Figure



Aggregate variant.

Création de collections multiples en regroupant les géométries dans une table.

Code

```
SELECT stusps, ST_Collect(geom) AS geom
FROM (
  SELECT stusps, (ST_Dump(geom)).geom AS geom
  FROM somestatetable
) AS f
GROUP BY stusps;
```

Voir aussi

ST_Dump, ST_Union

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — Crée une LineString à partir d'une géométrie MultiPoint.

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

Description

Crée une LineString à partir d'une géométrie MultiPoint.

Utilisez ST_MakeLine pour créer des lignes à partir des entrées Point ou LineString.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Créer une LineString 3D à partir d'un MultiPoint 3D

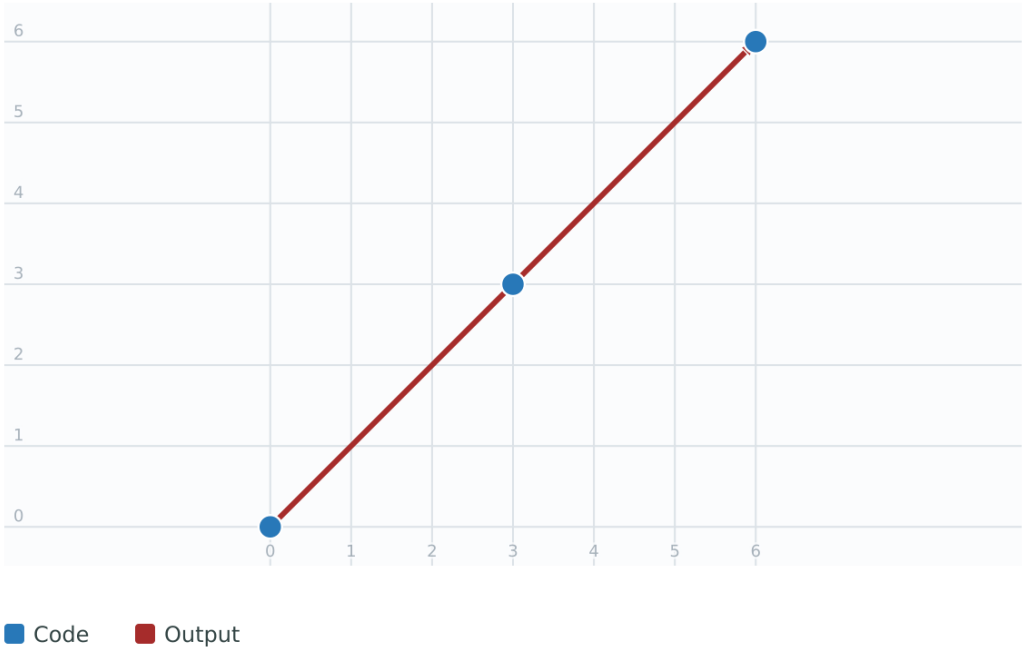
Code

```
SELECT ST_LineFromMultiPoint('MULTIPOINT(1 2 3,4 5 6,7 8 9)');
```

Output

```
LINESTRING(1 2 3,4 5 6,7 8 9)
```

Figure



Voir aussi

ST_AsEWKT, ST_MakeLine

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — Crée un polygone rectangulaire à partir des coordonnées minimales et maximales.

Synopsis

geometry **ST_MakeEnvelope**(float xmin, float ymin, float xmax, float ymax, integer srid=unknown);

Description

Crée un polygone rectangulaire à partir des valeurs minimales et maximales de X et Y. Les valeurs entrées doivent être dans le système de référence spatiale spécifié par le SRID. Si aucun SRID n'est spécifié, le système de référence spatiale inconnu (SRID 0) est utilisé.

Disponibilité : 1.5

Amélioré : 2.0 : La possibilité de spécifier une enveloppe sans spécifier un SRID a été introduite.

Exemples

Building a bounding box polygon.

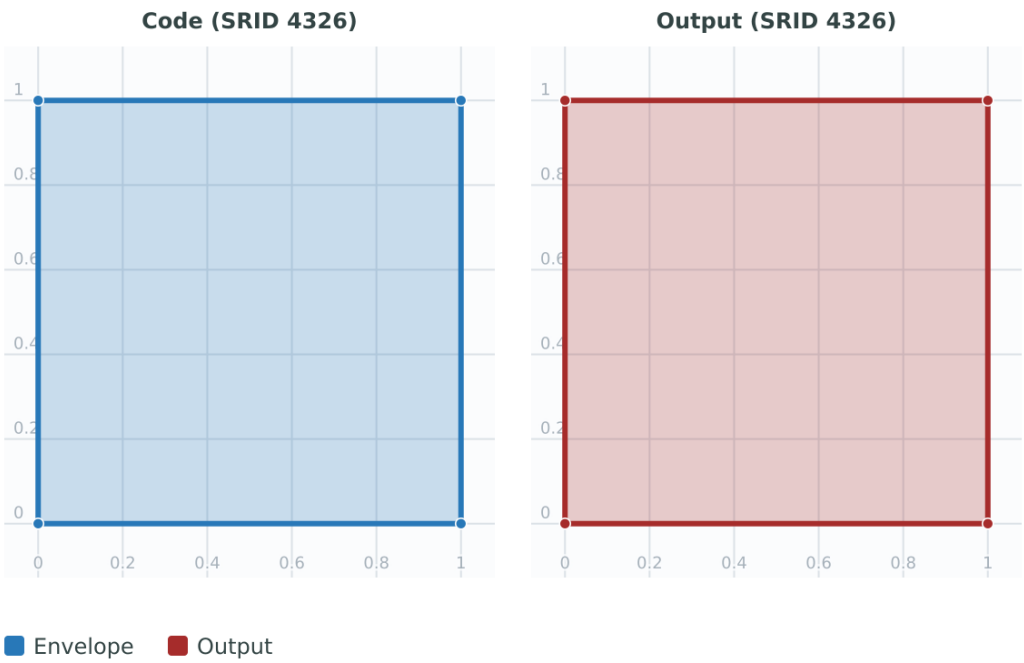
Code

```
SELECT ST_MakeEnvelope(10, 10, 11, 11, 4326);
```

Output

```
POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))
```

Figure



Voir aussi

ST_MakePoint, ST_MakeLine, ST_MakePolygon, ST_TileEnvelope

7.3.4 ST_MakeLine

ST_MakeLine — Crée une LineString à partir de géométries Point, MultiPoint ou LineString.

Synopsis

geometry **ST_MakeLine**(geometry geom1, geometry geom2);
geometry **ST_MakeLine**(geometry[] geoms_array);
geometry **ST_MakeLine**(geometry set geoms);

Description

Crée une LineString contenant les points des géométries Point, MultiPoint ou LineString. Les autres types de géométrie provoquent une erreur.

Variante 1: accepte géométries en entrée

Variante 2: accepte un tableau de géométries

Variante 3: fonction agrégée acceptant un ensemble de géométries. Pour garantir l'ordre des géométries d'entrée, utilisez `ORDER BY` dans l'appel de fonction, ou une sous-requête avec une clause `ORDER BY`.

Repeated nodes at the beginning of input LineStrings are collapsed to a single point. Repeated points in Point and MultiPoint inputs are not collapsed. Components of MultiLineString are handled in the order they appear in the collection. ST_RemoveRepeatedPoints can be used to collapse repeated points from the output LineString.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Availability: 3.7.0 - Support for MultiLineString input elements was introduced

Disponibilité : 2.3.0 - La prise en charge des éléments d'entrée MultiPoint a été introduite

Disponibilité : 2.0.0 - La prise en charge des éléments d'entrée LineString a été introduite

Disponibilité : 1.4.0 - création de ST_MakeLine(geomarray). L'agrégat spatial ST_MakeLine amélioré pour supporter plus de points plus rapidement.

Exemples

Two-input variant.

Créez une ligne composée de deux points.

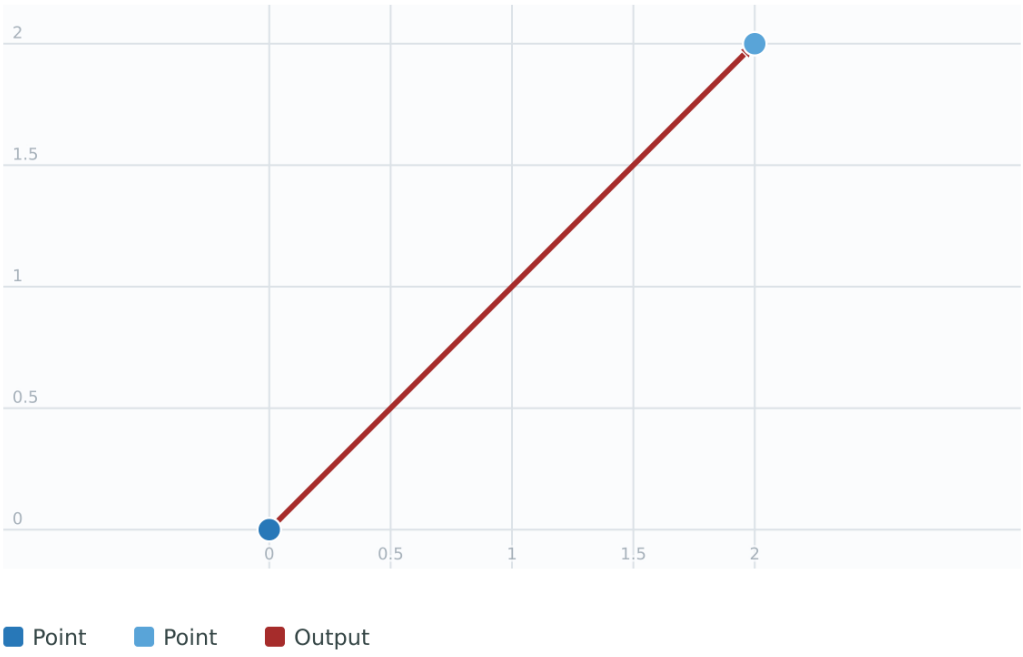
Code

```
SELECT ST_MakeLine(ST_Point(1, 2), ST_Point(3, 4));
```

Output

```
LINESTRING(1 2,3 4)
```

Figure



Créer une ligne 3D à partir de deux points 3D.

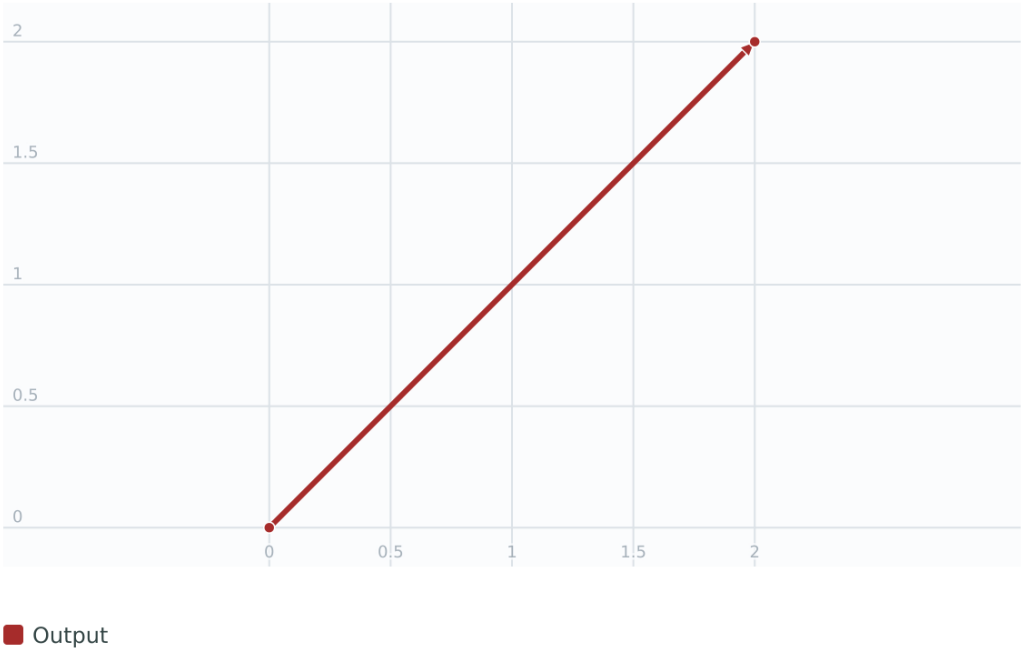
Code

```
SELECT ST_MakeLine(ST_MakePoint(1, 2, 3), ST_MakePoint(3, 4, 5));
```

Output

```
LINESTRING(1 2 3,3 4 5)
```

Figure



Crée une ligne à partir de deux LineStrings disjointes.

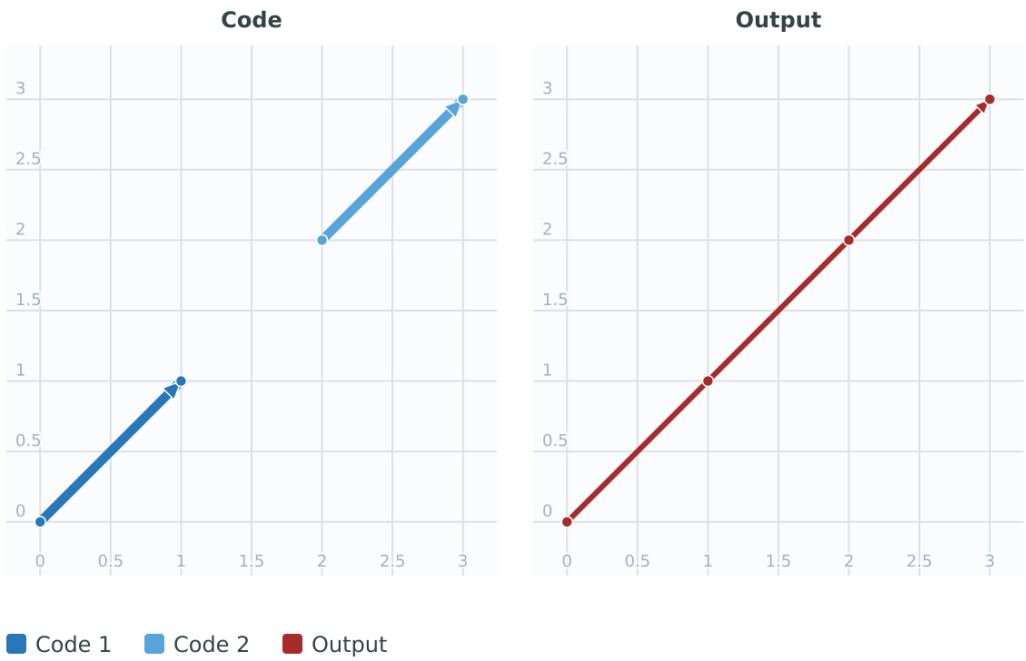
Code

```
SELECT ST_MakeLine('LINESTRING(0 0,1 1)', 'LINESTRING(2 2,3 3)');
```

Output

```
LINESTRING(0 0,1 1,2 2,3 3)
```

Figure



Array variant.

Créer une ligne à partir d'un tableau formé par une sous-requête avec ordonnancement.

Code

```
SELECT ST_MakeLine(ARRAY(SELECT ST_Centroid(geom) FROM visit_locations ORDER BY visit_time)) ↩
    ;
```

Créer une ligne 3D à partir d'un tableau de points 3D

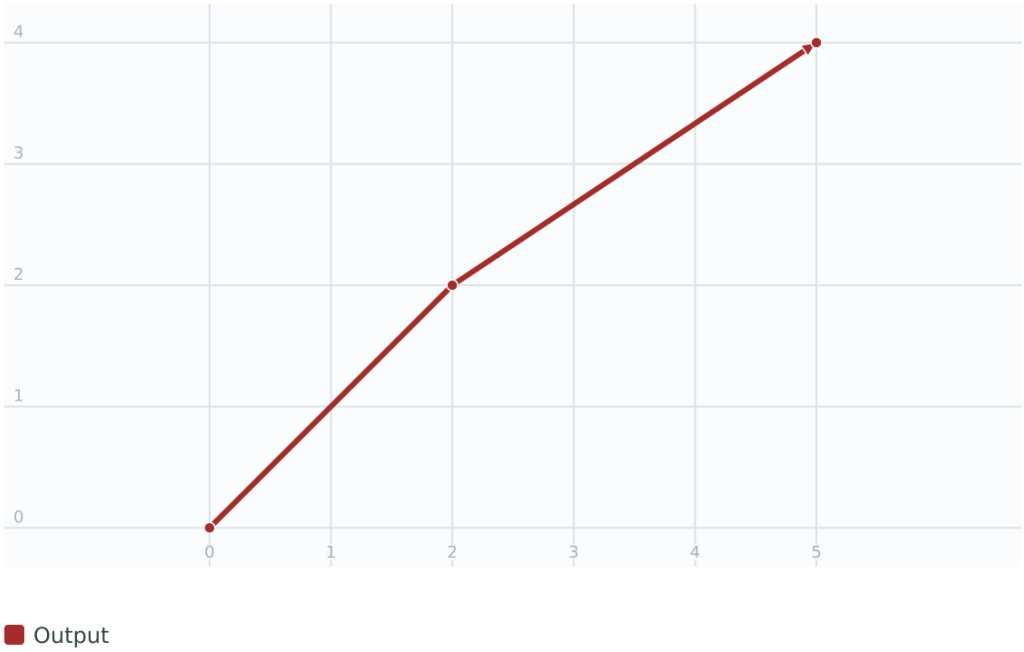
Code

```
SELECT ST_MakeLine(ARRAY[ST_MakePoint(1, 2, 3), ST_MakePoint(3, 4, 5), ST_MakePoint(6, 6, 6) ↩
    ]);
```

Output

```
LINESTRING(1 2 3,3 4 5,6 6 6)
```

Figure



Aggregate variant.

Cet exemple interroge des séquences temporelles de points GPS à partir d'un ensemble de pistes et crée un enregistrement pour chaque piste. Les géométries résultantes sont des LineStrings composées des points GPS dans l'ordre de leur déplacement.

L'utilisation de l'agrégat `ORDER BY` fournit une LineString correctement ordonnée.

Code

```
SELECT gps.track_id, ST_MakeLine(gps.geom ORDER BY gps_time) AS geom
FROM gps_points AS gps
GROUP BY gps.track_id;
```

When ordering within the aggregate is not possible, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

Code

```
SELECT gps.track_id, ST_MakeLine(gps.geom) AS geom
FROM (
  SELECT track_id, gps_time, geom
  FROM gps_points
  ORDER BY track_id, gps_time
) AS gps
GROUP BY gps.track_id;
```

Voir aussi

ST_RemoveRepeatedPoints, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_MakePoint, ST_Point

7.3.5 ST_MakePoint

ST_MakePoint — Crée un point 2D, 3DZ ou 4D.

Synopsis

geometry **ST_MakePoint**(float x, float y);

geometry **ST_MakePoint**(float x, float y, float z);

geometry **ST_MakePoint**(float x, float y, float z, float m);

Description

Crée une géométrie de point 2D XY, 3D XYZ ou 4D XYZM. Utilisez ST_MakePointM pour créer des points avec des coordonnées XYM.

Utilisez ST_SetSRID pour spécifier un SRID pour le point créé.

Bien que non conforme à l'OGC, `ST_MakePoint` est plus rapide que ST_GeomFromText et ST_PointFromText. Il est également plus facile à utiliser pour les valeurs de coordonnées numériques.



Note
Pour les coordonnées géodésiques, `X` est la longitude et `Y` la latitude



Note
Les fonctions ST_Point, ST_PointZ, ST_PointM et ST_PointZM peuvent être utilisées pour créer des points avec un SRID donné.



Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

This example creates a point with an unknown SRID.

Code

```
SELECT ST_MakePoint(-71.1043443253471, 42.3150676015829);
```

This example creates a point in the WGS 84 geodetic CRS.

Code

```
SELECT ST_SetSRID(ST_MakePoint(-71.1043443253471, 42.3150676015829), 4326);
```

This example creates a 3D point with altitude.

Code

```
SELECT ST_MakePoint(1, 2, 1.5);
```

This example returns the Z coordinate of a point.

Code

```
SELECT ST_Z(ST_MakePoint(1, 2, 1.5));
```

Output

```
1.5
```

Voir aussi

ST_GeomFromText, ST_PointFromText, ST_SetSRID, ST_MakePointM, ST_Point, ST_PointZ, ST_PointM, ST_PointZM

7.3.6 ST_MakePointM

ST_MakePointM — Crée un point à partir des valeurs X, Y et M.

Synopsis

geometry **ST_MakePointM**(float x, float y, float m);

Description

Crée un point avec des coordonnées X, Y et M (mesure). Utilisez ST_MakePoint pour créer des points avec des coordonnées XY, XYZ ou XYZM.

Utilisez ST_SetSRID pour spécifier un SRID pour le point créé.



Note
Pour les coordonnées géodésiques, `X` est la longitude et `Y` la latitude



Note
Les fonctions ST_PointM et ST_PointZM peuvent être utilisées pour créer des points avec une valeur M et un SRID donné.

Exemples



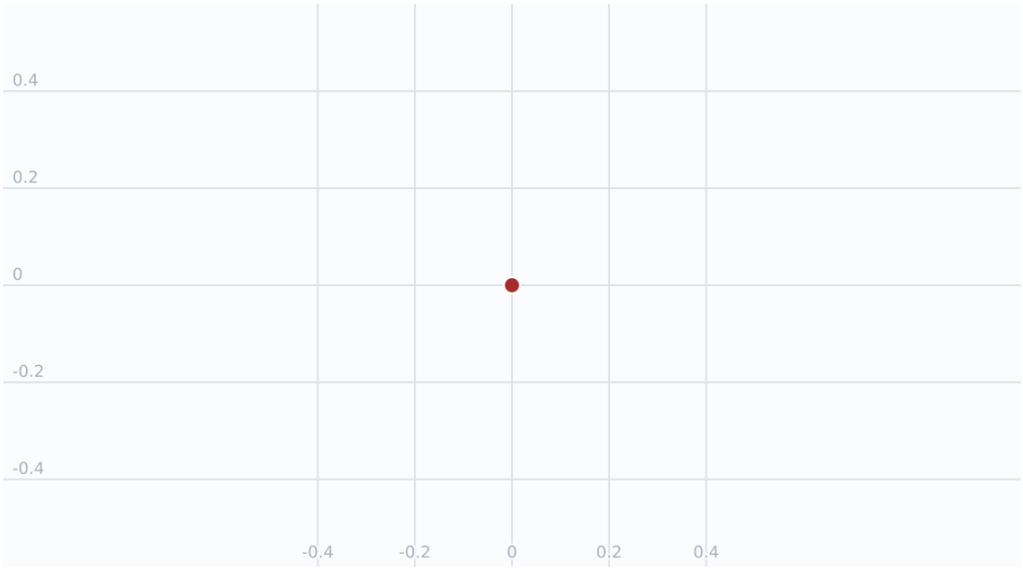
Note
ST_AsEWKT est utilisé pour la sortie texte car ST_AsText ne prend pas en charge les valeurs M.

Créer un point avec un SRID inconnu.

Code

```
SELECT ST_MakePointM(-71.1043443253471, 42.3150676015829, 10);
```

Figure

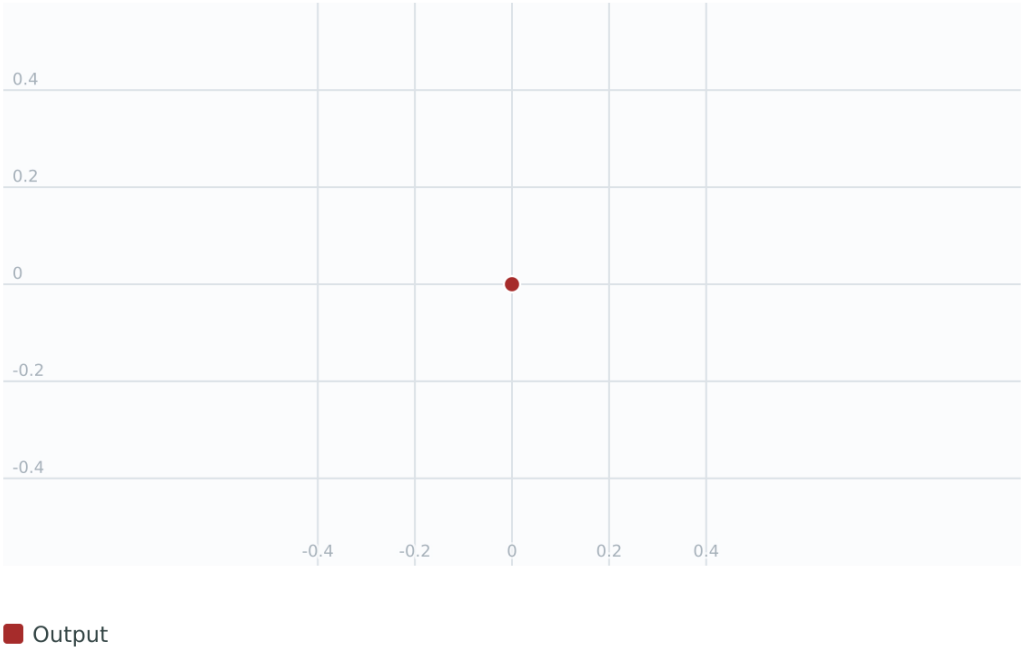


Créer un point avec une mesure dans le système de coordonnées géodésiques WGS 84.

Code

```
SELECT ST_SetSRID(ST_MakePointM(-71.104, 42.315, 10), 4326);
```

Figure



Obtenir la mesure du point créé.

Code

```
SELECT ST_M(ST_MakePointM(-71.104, 42.315, 10));
```

Output

```
10
```

Voir aussi

ST_MakePoint, ST_SetSRID, ST_PointM, ST_PointZM

7.3.7 ST_MakePolygon

ST_MakePolygon — Creates a Polygon or CurvePolygon from a shell and optional list of holes.

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

geometry **ST_MakePolygon**(geometry outerlinestring, geometry[] interiorlinestrings);

Description

Creates a Polygon or CurvePolygon formed by the given shell and optional array of holes. Input geometries must be closed LineStrings, CircularStrings, CompoundCurves, or NURBSCurves (rings). If any input ring is curved, the result is a CurvePolygon.

Variant 1: Accepts one shell ring.

Variant 2: Accepts a shell ring and an array of inner (hole) rings. A geometry array can be constructed using the PostgreSQL array_agg(), ARRAY[] or ARRAY() constructs.



Note
Cette fonction n'accepte pas les MultiLineStrings. Utilisez ST_LineMerge pour générer une LineString, ou ST_Dump pour extraire les LineStrings.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Enhanced: 3.7.0 - Support for curved input rings was introduced.

Exemples

Single input variant.

Créez un polygone à partir d'une LineString 2D.

Code

```
SELECT ST_MakePolygon('LINESTRING(75 29,77 29,77 29,75 29)');
```

Créez un polygone à partir d'une LineString ouverte, en utilisant ST_StartPoint et ST_AddPoint pour la fermer.

Code

```
SELECT ST_MakePolygon(ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)) )
FROM (
  SELECT 'LINESTRING(75 29,77 29,77 29,75 29)'::geometry As open_line) As foo;
```

Créer un polygone à partir d'une LineString 3D

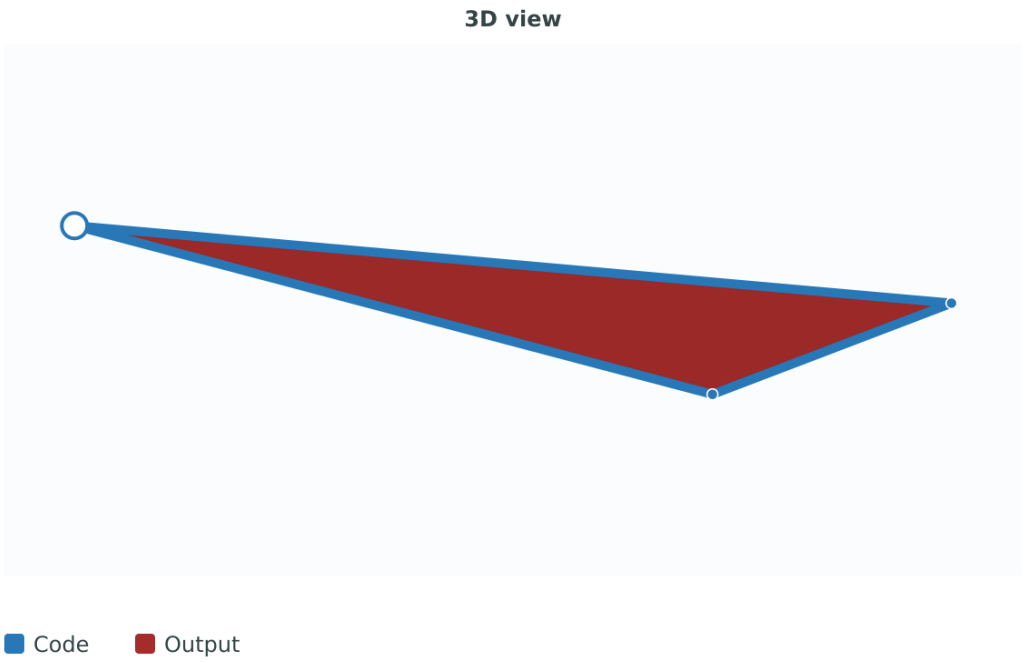
Code

```
SELECT ST_MakePolygon('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1)');
```

Output

```
POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))
```

Figure

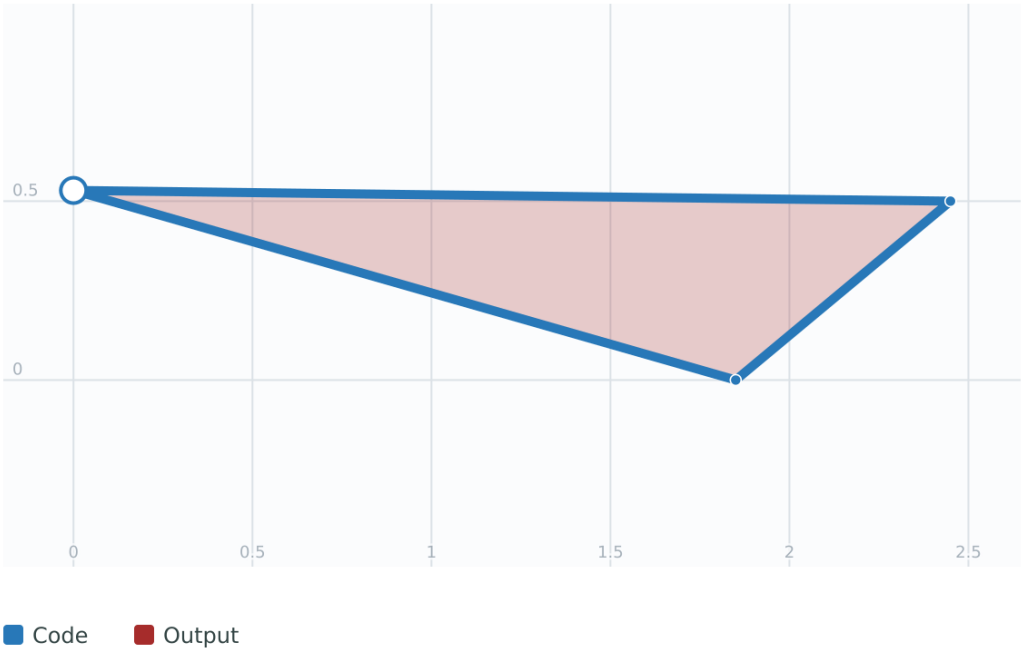


Créer un polygone à partir d'une LineString avec des mesures

Code

```
SELECT ST_MakePolygon('LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15 29.53 2)');
```

Figure



Create a CurvePolygon from a CircularString shell.

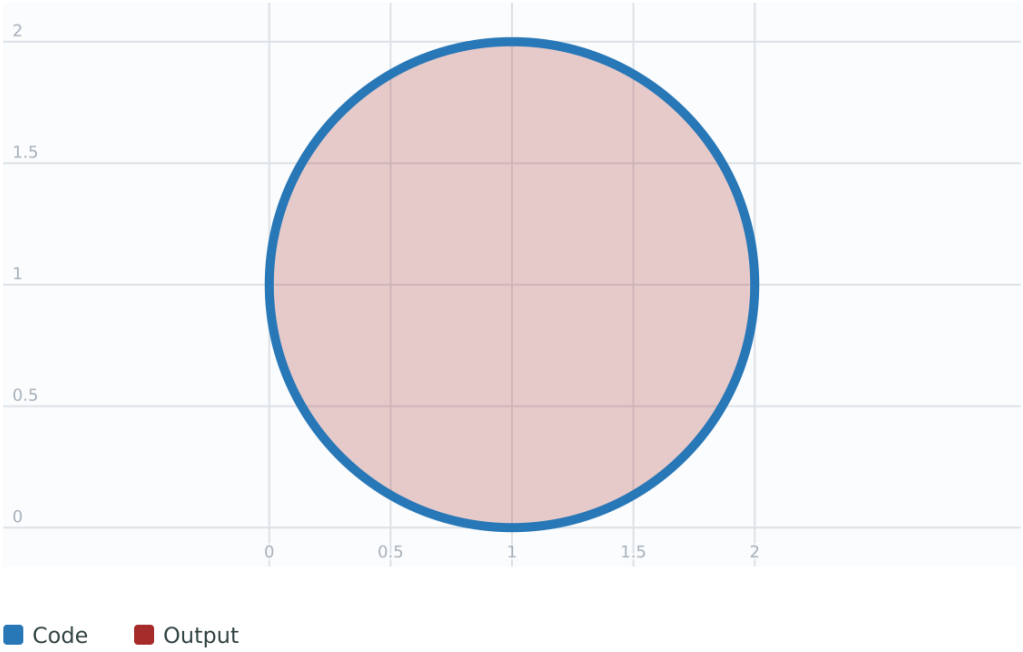
Code

```
SELECT ST_MakePolygon('CIRCULARSTRING(0 0,1 1,2 0,1 -1,0 0)');
```

Output

```
CURVEPOLYGON(CIRCULARSTRING(0 0,1 1,2 0,1 -1,0 0))
```

Figure



Outer shell with inner holes variant.

Créer un polygone en forme de donut avec un trou supplémentaire

Code

```
SELECT ST_MakePolygon(
    ST_ExteriorRing(ST_Buffer(ring.line, 10)),
    ARRAY[  ST_Translate(ring.line, 1, 1),
        ST_ExteriorRing(ST_Buffer(ST_Point(20, 20), 1)) ]
    )
FROM (SELECT ST_ExteriorRing(ST_Buffer(ST_Point(10, 10), 10, 10)) AS line ) AS ring;
```

Créez un ensemble de frontières de province avec des trous représentant des lacs. L'entrée est un tableau de polygones/multi-polygones de province et un tableau de lignes d'eau. Les lignes formant des lacs sont déterminées en utilisant ST_IsClosed. Le réseau de la province est extrait en utilisant ST_Boundary. Comme requis par `ST_MakePolygon`, la frontière est forcée à être une seule LineString en utilisant ST_LineMerge. (Cependant, notez que si une province a plus d'une région ou a des îles, cela produira un polygone invalide). L'utilisation d'un LEFT JOIN garantit que toutes les provinces sont incluses, même si elles n'ont pas de lacs.



Note
La construction CASE est utilisée car le passage d'un tableau nul dans ST_MakePolygon entraîne une valeur de retour NULL.

Code

```
SELECT p.gid, p.province_name,
  CASE
    WHEN array_agg(w.geom) IS NULL THEN p.geom
    ELSE ST_MakePolygon(
      ST_LineMerge(ST_Boundary(p.geom)),
      array_agg(w.geom)
    )
  END AS geom
FROM provinces p
LEFT JOIN waterlines w
  ON ST_Within(w.geom, p.geom)
  AND ST_IsClosed(w.geom)
GROUP BY p.gid, p.province_name, p.geom;
```

Une autre technique consiste à utiliser une sous-requête corrélée et le constructeur ARRAY() qui convertit un ensemble de lignes en un tableau.

Code

```
SELECT p.gid, p.province_name,
  CASE
    WHEN EXISTS (
      SELECT w.geom
      FROM waterlines w
      WHERE ST_Within(w.geom, p.geom)
      AND ST_IsClosed(w.geom)
    )
    THEN ST_MakePolygon(
      ST_LineMerge(ST_Boundary(p.geom)),
      ARRAY(
        SELECT w.geom
        FROM waterlines w
        WHERE ST_Within(w.geom, p.geom)
```

```
        AND ST_IsClosed(w.geom)
      )
    )
    ELSE p.geom
  END AS geom
FROM provinces p;
```

Voir aussi

ST_BuildArea ST_Polygon

7.3.8 ST_Point

ST_Point — Crée un point avec des valeurs X, Y et SRID.

Synopsis

geometry **ST_Point**(float x, float y);

geometry **ST_Point**(float x, float y, integer srid=unknown);

Description

Renvoie un point avec les valeurs de coordonnées X et Y données. C'est l'équivalent SQL-MM de ST_MakePoint qui ne prend que X et Y.



Note
Pour les coordonnées géodésiques, `X` est la longitude et `Y` la latitude

Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie.


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 6.1.2

Exemples

Geometry.

Code

```
SELECT ST_Point(-71.104, 42.315);
```

Création d'un point avec le SRID spécifié :

Code

```
SELECT ST_Point(-71.104, 42.315, 4326);
```

Autre façon de spécifier le SRID :

Code

```
SELECT ST_SetSRID(ST_Point(-71.104, 42.315), 4326);
```

Créez des points geography en utilisant la syntaxe `::` :

Code

```
SELECT ST_Point(-71.104, 42.315, 4326)::geography;
```

Code Pre-PostGIS 3.2, utilisant `CAST` :

Code

```
SELECT CAST(ST_SetSRID(ST_Point(-71.104, 42.315), 4326) AS geography);
```

Si les coordonnées du point ne sont pas dans un système de coordonnées géodésiques (tel que WGS84), elles doivent être reprojetées avant d'être projetées dans une géographie. Dans cet exemple, un point en pieds du plan de l'État de Pennsylvanie (SRID 2273) est projeté en WGS84 (SRID 4326).

Code

```
SELECT ST_Transform(ST_Point(3637510, 3014852, 2273), 4326)::geography;
```

Voir aussi

ST_MakePoint, ST_PointZ, ST_PointM, ST_PointZM, ST_SetSRID, ST_Transform

7.3.9 ST_PointZ

ST_PointZ — Crée un point avec des valeurs X, Y, Z et SRID.

Synopsis

geometry **ST_PointZ**(float x, float y, float z, integer srid=unknown);

Description

Renvoie un point avec les valeurs de coordonnées X, Y et Z données, et éventuellement un numéro SRID.

Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie.

Exemples

Code

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, 4326)
```

Code

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, srid => 4326)
```

Code

```
SELECT ST_PointZ(-71.104, 42.315, 3.4)
```

Voir aussi

ST_MakePoint, ST_Point, ST_PointM, ST_PointZM

7.3.10 ST_PointM

ST_PointM — Crée un point avec des valeurs X, Y, M et SRID.

Synopsis

geometry **ST_PointM**(float x, float y, float m, integer srid=unknown);

Description

Renvoie un point avec les valeurs de coordonnées X, Y et M données, et éventuellement un numéro SRID.

Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie.

Exemples

Code

```
SELECT ST_PointM(-71.104, 42.315, 3.4, 4326)
```

Code

```
SELECT ST_PointM(-71.104, 42.315, 3.4, srid => 4326)
```

Code

```
SELECT ST_PointM(-71.104, 42.315, 3.4)
```

Voir aussi

ST_MakePoint, ST_Point, ST_PointZ, ST_PointZM

7.3.11 ST_PointZM

ST_PointZM — Crée un point avec des valeurs X, Y, Z, M et SRID.

Synopsis

geometry **ST_PointZM**(float x, float y, float z, float m, integer srid=unknown);

Description

Renvoie un point avec les valeurs de coordonnées X, Y, Z et M données, et éventuellement un numéro SRID.

Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie.

Exemples

Code

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, 4326)
```

Code

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, srid => 4326)
```

Code

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5)
```

Voir aussi

ST_MakePoint, ST_Point, ST_PointM, ST_PointZ, ST_SetSRID

7.3.12 ST_Polygon

ST_Polygon — Crée un polygone à partir d'une LineString avec un SRID spécifié.

Synopsis

geometry **ST_Polygon**(geometry lineString, integer srid);

Description

Renvoie un polygone construit à partir de la LineString donnée et définit le système de référence spatiale à partir du `srid`.

ST_Polygon est similaire à la variante 1 de ST_MakePolygon avec l'ajout de la définition du SRID.

Pour créer des polygones avec des trous, utilisez ST_MakePolygon la variante 2 et ensuite ST_SetSRID.



Note
Cette fonction n'accepte pas les MultiLineStrings. Utilisez ST_LineMerge pour générer une LineString, ou ST_Dump pour extraire les LineStrings.


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.3.2


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Créer un polygone 2D.

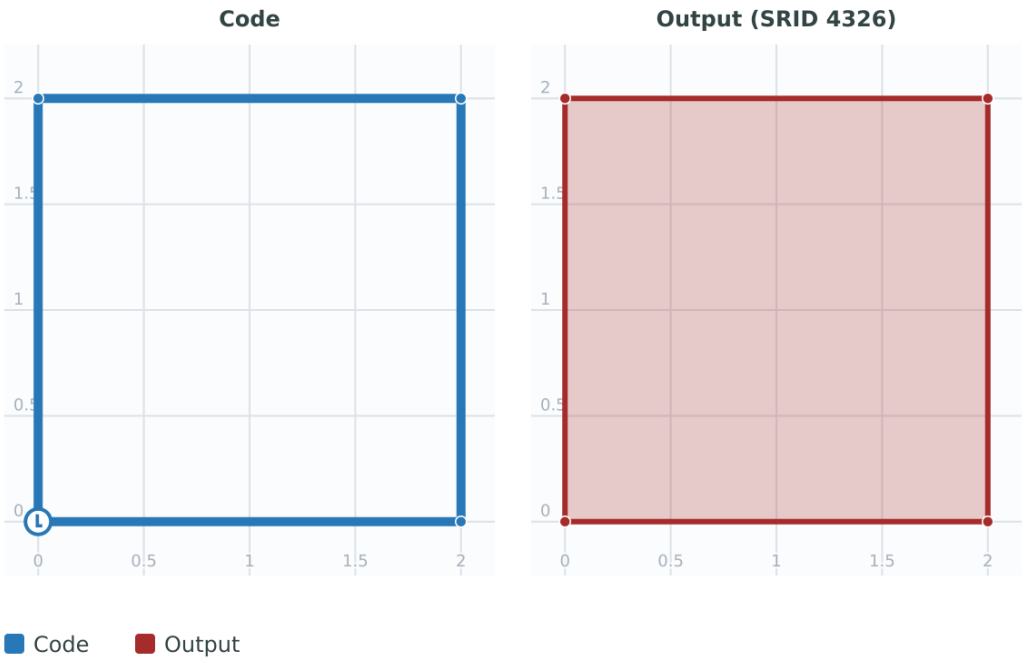
Code

```
SELECT ST_Polygon('LINESTRING(75 29,77 29,77 31,75 31,75 29)'::geometry, 4326);
```

Output

```
POLYGON((75 29,77 29,77 31,75 31,75 29))
```

Figure



Créer un polygone 3D.

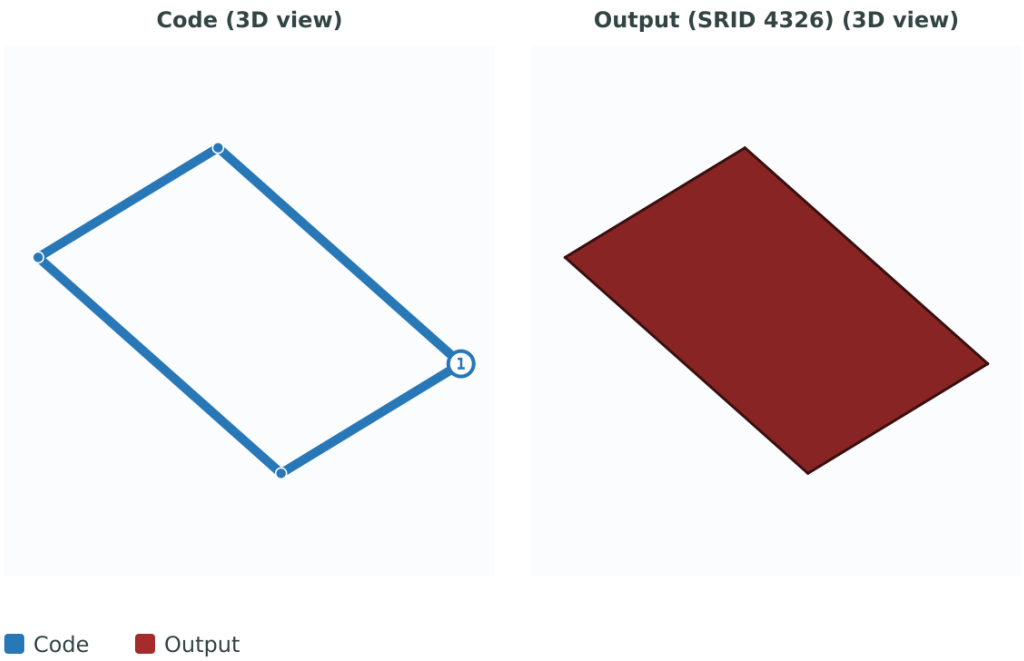
Code

```
SELECT ST_Polygon('LINESTRING(75 29 1,77 29 2,77 31 2,75 31 1,75 29 1)'::geometry, 4326);
```

Output

```
SRID=4326;POLYGON((75 29 1,77 29 2,77 31 2,75 31 1,75 29 1))
```

Figure



Voir aussi

ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_GeomFromText, ST_LineMerge, ST_MakePolygon

7.3.13 ST_TileEnvelope

ST_TileEnvelope — Crée un polygone rectangulaire dans Web Mercator (SRID:3857) en utilisant le système de tuiles XYZ.

Synopsis

geometry **ST_TileEnvelope**(integer tileZoom, integer tileX, integer tileY, geometry bounds=SRID=3857;LINESTRING(-20037508.342 -20037508.342789,20037508.342789 20037508.342789), float margin=0.0);

Description

Crée un polygone rectangulaire donnant l'étendue d'une tuile dans le système de tuiles XYZ. La tuile est spécifiée par le niveau de zoom Z et l'indice XY de la tuile dans la grille à ce niveau. Peut être utilisé pour définir les limites de la tuile requises par ST_AsMVTGeom pour convertir la géométrie dans l'espace de coordonnées de la tuile MVT.

Par défaut, l'enveloppe de la tuile est dans le système de coordonnées Web Mercator (SRID:3857) utilisant la plage standard du système Web Mercator (-20037508.342789, 20037508.342789). Il s'agit du système de coordonnées le plus couramment utilisé pour les tuiles MVT. Le paramètre facultatif `bounds` peut être utilisé pour générer des tuiles dans n'importe quel système de coordonnées. Il s'agit d'une géométrie qui possède le SRID et l'étendue du carré "Niveau de zoom zéro" dans lequel le système de tuiles XYZ est inscrit.

Le paramètre facultatif `margin` peut être utilisé pour étendre une tuile du pourcentage donné. Par exemple, `margin=0.125` étend la tuile de 12,5%, ce qui équivaut à buffer=512 lorsque la taille de l'étendue de la tuile est de 4096, comme utilisé dans ST_AsMVTGeom. Ceci est utile pour créer un tampon de tuile afin d'inclure des données situées en dehors de la zone visible de la tuile, mais dont l'existence affecte le rendu de la tuile. Par exemple, le nom d'une ville (un point) peut se trouver près du bord d'une tuile, son étiquette doit donc être rendue sur deux tuiles, même si le point se trouve dans la zone visible d'une seule tuile. L'utilisation de tuiles étendues dans une requête inclura le point de la ville dans les deux tuiles. Utilisez une valeur négative pour

réduire la tuile à la place. Les valeurs inférieures à -0,5 sont interdites, car cela éliminerait complètement la tuile. Ne spécifiez pas de marge lors de l'utilisation avec `ST_AsMVTGeom`. Voir l'exemple pour ST_AsMVT.

Amélioré : 3.1.0 Ajout d'un paramètre de marge.

Disponibilité : 3.0.0

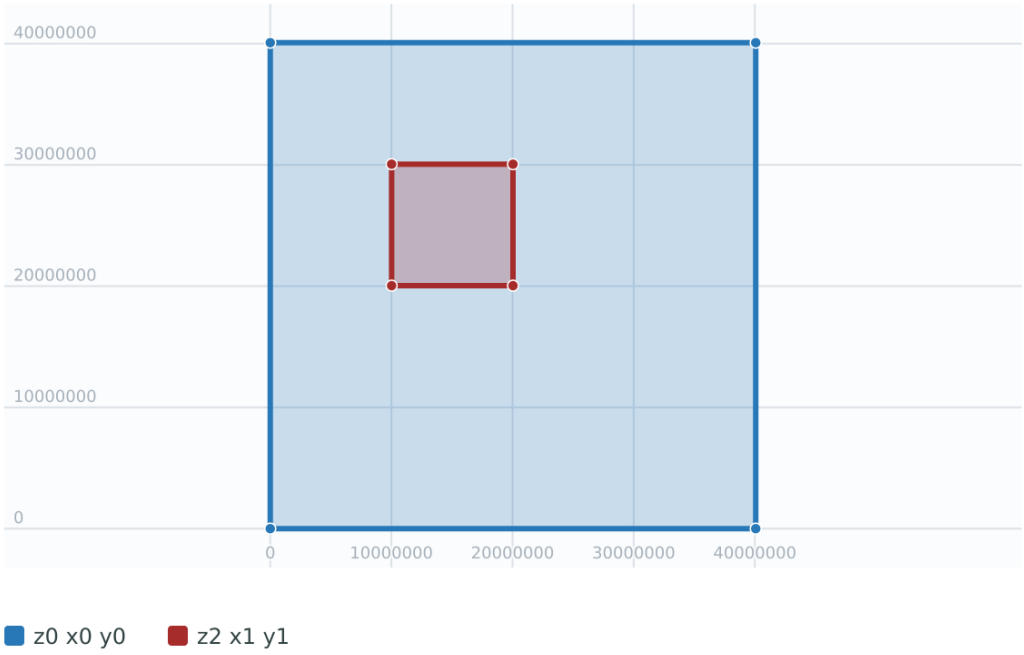
Exemples

Show tile (2, 1, 1) within the root tile (0, 0, 0).

Code

```
SELECT ST_TileEnvelope(0, 0, 0) AS input_z0_x0_y0, ST_TileEnvelope(2, 1, 1) AS z2_x1_y1;
```

Figure



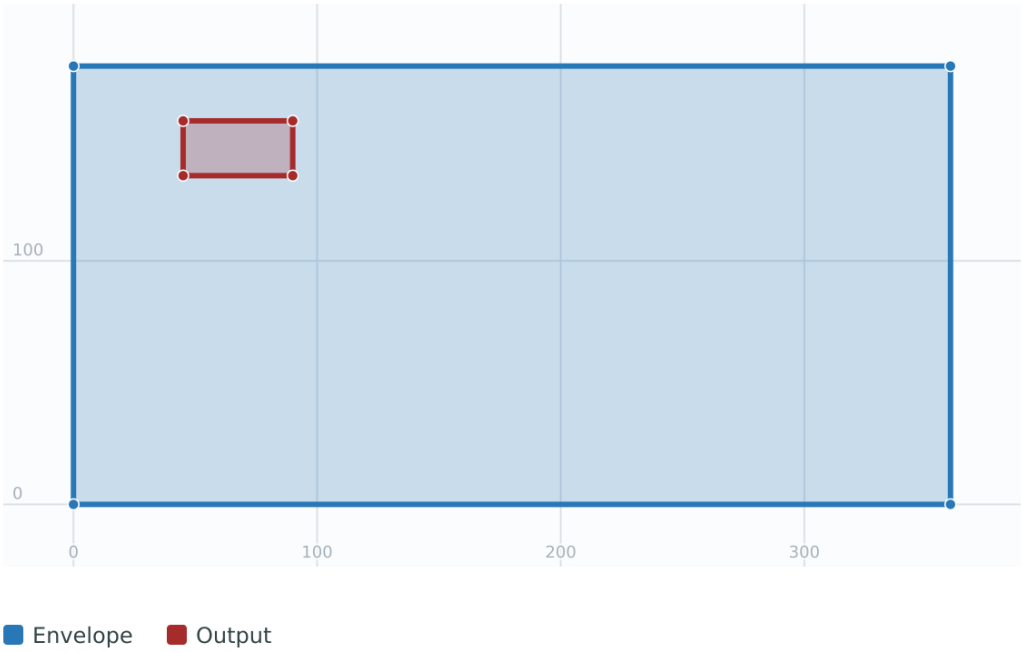
Code

```
SELECT ST_TileEnvelope(3, 1, 1, ST_MakeEnvelope(-180, -90, 180, 90, 4326));
```

Output

```
POLYGON((-135 45,-135 67.5,-90 67.5,-90 45,-135 45))
```

Figure



Voir aussi

ST_MakeEnvelope

7.3.14 ST_HexagonGrid

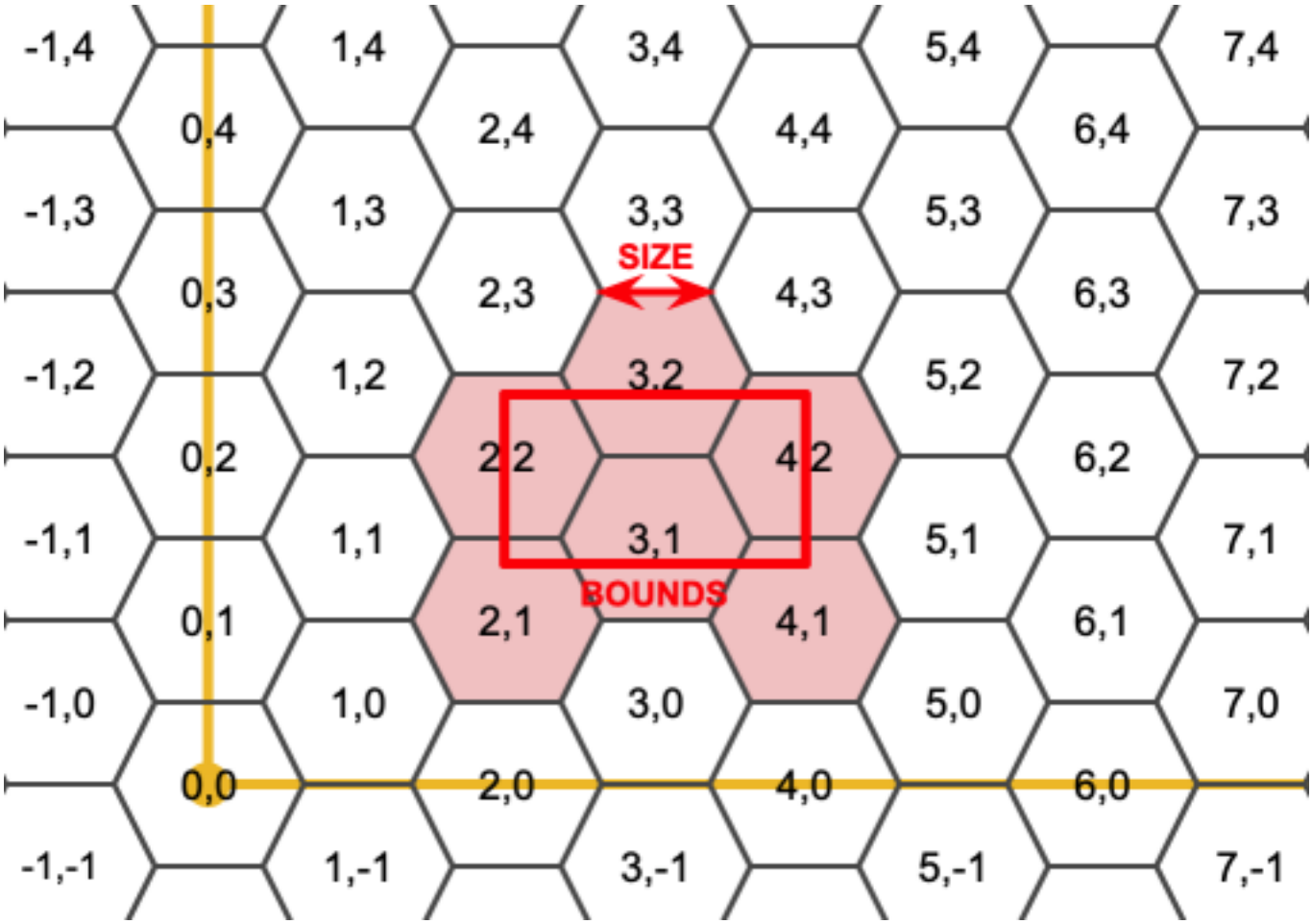
ST_HexagonGrid — Renvoie un ensemble d'hexagones et d'indices de cellules qui couvrent complètement les limites de l'argument géométrie.

Synopsis

setof record **ST_HexagonGrid**(float8 size, geometry bounds);

Description

Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe; for H3 use the h3-pg extension.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, Tiling(SRS, Size). This function answers the question: what hexagons in a given Tiling(SRS, Size) overlap with a given bounds.



Le SRS pour les hexagones de sortie est le SRS fourni par la géométrie des limites.

Doubler ou tripler la taille des bords de l'hexagone génère un nouveau pavage parent qui s'adapte au pavage d'origine. Malheureusement, il n'est pas possible de générer des tuiles d'hexagones parents dans lesquelles les tuiles enfants s'insèrent parfaitement.

Disponibilité : 3.1.0

Exemples

Unit hexagons covering rectangular bounds.

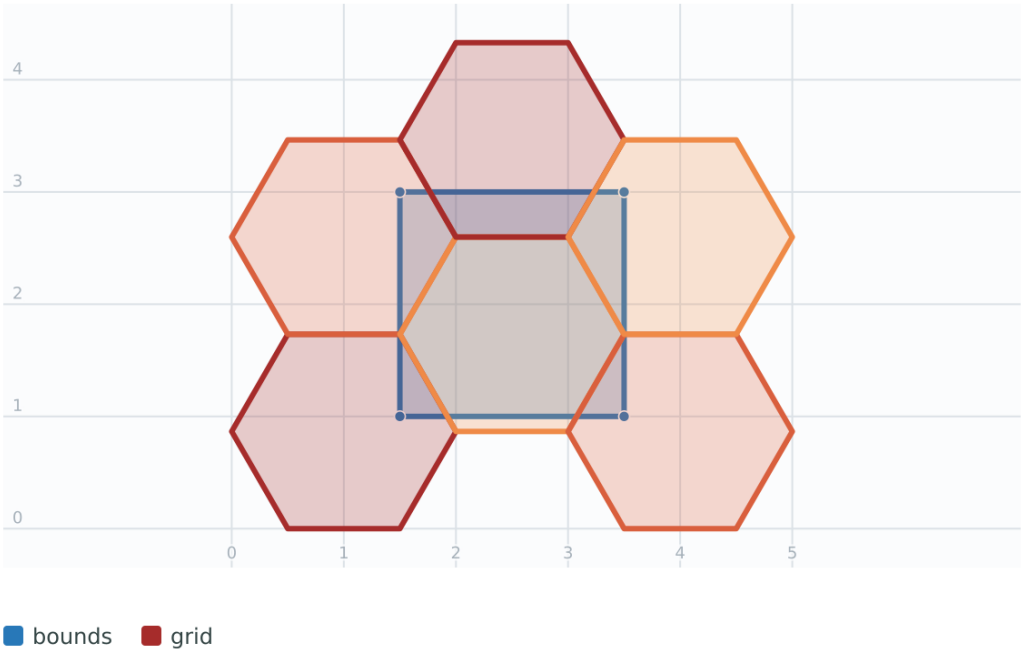
Code

```
WITH bounds AS (
    SELECT ST_MakeEnvelope(2, 1, 4, 3, 3857) AS geom
), grid AS (
    SELECT hex.*
    FROM bounds
    CROSS JOIN LATERAL ST_HexagonGrid(1, geom) AS hex
)
SELECT (SELECT geom FROM bounds) AS input_bounds,
    ST_Collect(geom ORDER BY i, j) AS grid
FROM grid;
```

Output

```
POLYGON((2 1,2 3,4 3,4 1,2 1)) | MULTIPOLYGON(((0.5 0.866,1 0,2 0,2.5 0.866,2 1.732,1 1.732,0.5 0.866)),((0.5 2.598,1 ↩
    1.732,2 1.732,2.5 2.598,2 3.464,1 3.464,0.5 2.598)),((2 1.732,2.5 0.866,3.5 0.866,4 1.732,3.5 2.598,2.5 2.598,2 ↩
    1.732)),((2 3.464,2.5 2.598,3.5 2.598,4 3.464,3.5 4.33,2.5 4.33,2 3.464)),((3.5 0.866,4 0,5 0,5.5 0.866,5 1.732,4 ↩
    1.732,3.5 0.866)),((3.5 2.598,4 1.732,5 1.732,5.5 2.598,5 3.464,4 3.464,3.5 2.598)))
```

Figure

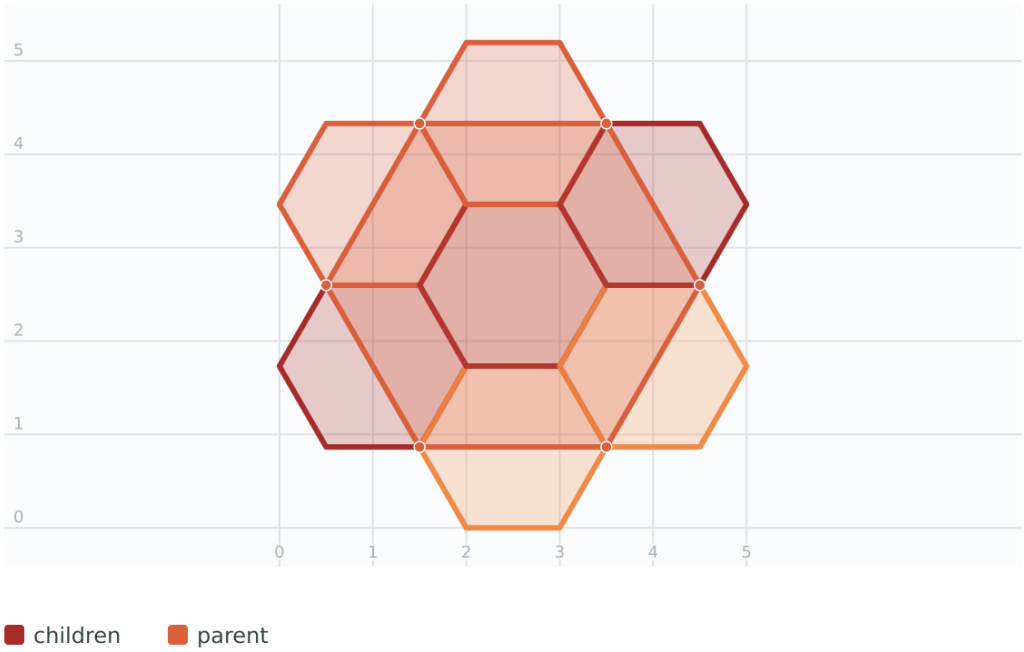


Child tiles that intersect a doubled-size parent hexagon share the tiling origin, but do not nest perfectly inside the parent.

Code

```
WITH parent AS (
    SELECT ST_Hexagon(2, 0, 0) AS geom
), children AS (
    SELECT child.*
    FROM parent
    CROSS JOIN LATERAL ST_HexagonGrid(1, geom) AS child
    WHERE ST_Area(ST_Intersection(child.geom, parent.geom)) > 0
)
SELECT ST_Collect(geom ORDER BY i, j) AS children,
    (SELECT geom FROM parent) AS parent
FROM children;
```

Figure



Counting points in hexagons.

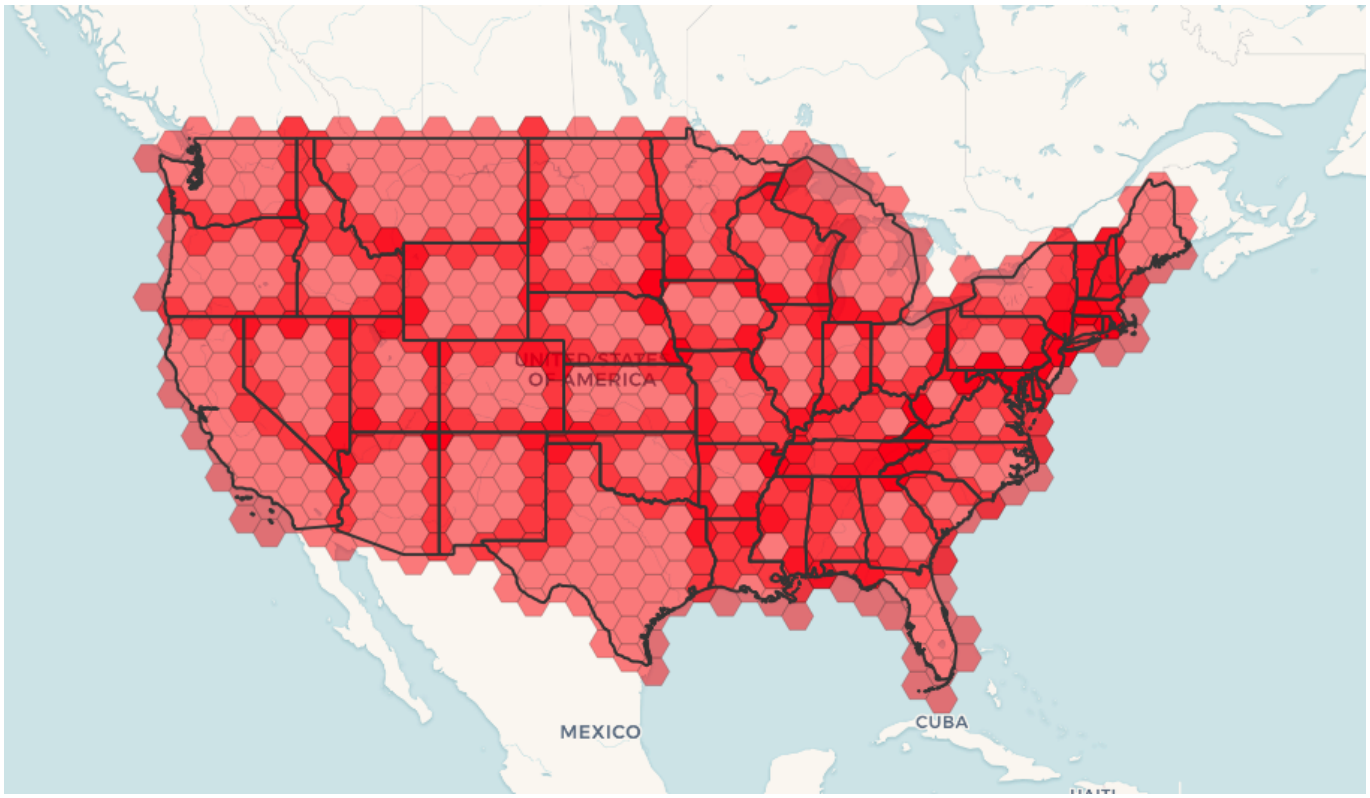
Pour faire un résumé des points par rapport à un pavage hexagonal, générez une grille hexagonale en utilisant l'étendue des points comme limites, puis joignez spatialement cette grille.

Code

```
SELECT COUNT(*), hexes.geom
FROM ST_HexagonGrid(
  10000,
  ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
) AS hexes
INNER JOIN pointtable AS pts
  ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

Generating hex coverage of polygons.

If a grid is generated for each polygon boundary and filtered to intersecting cells, adjacent regions receive overlapping hexagons along their shared border. The build-time figure below uses two simple regions in place of an external administrative-boundary table, so the example is self-contained.



Code

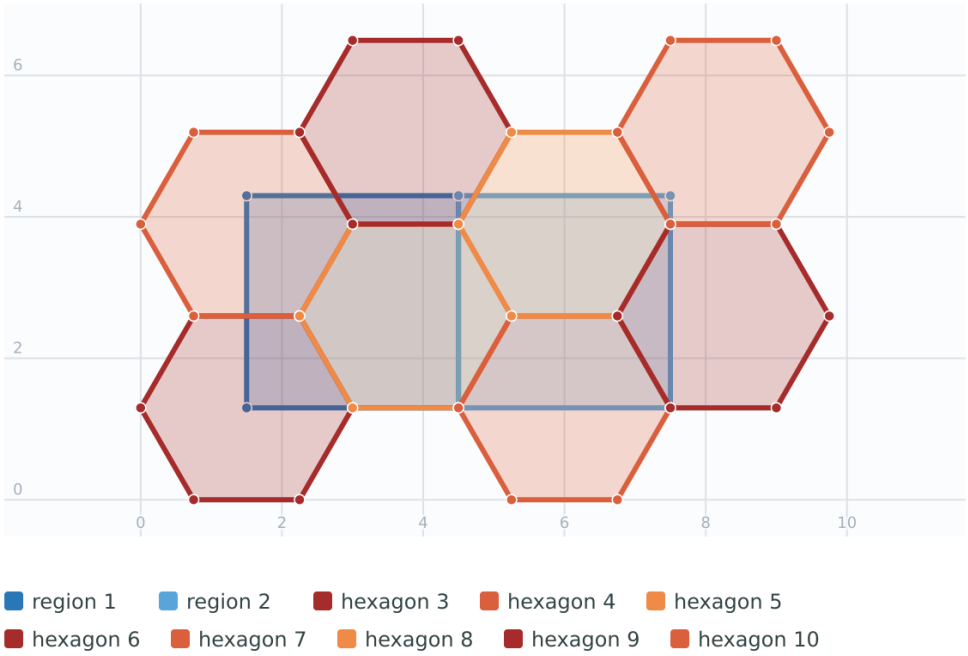
```
WITH admin1(gid, geom) AS (
    VALUES
        (1, ST_MakeEnvelope(0, 0, 3, 3, 3857)),
        (2, ST_MakeEnvelope(3, 0, 6, 3, 3857))
), coverage AS (
    SELECT hex.i, hex.j, hex.geom
    FROM admin1
    CROSS JOIN LATERAL ST_HexagonGrid(1.5, admin1.geom) AS hex
    WHERE ST_Intersects(admin1.geom, hex.geom)
    GROUP BY hex.i, hex.j, hex.geom
), layers AS (
    SELECT 'region' AS kind, gid AS position, geom
    FROM admin1
    UNION ALL
    SELECT
        'hexagon',
        1000 + row_number() OVER (ORDER BY i, j),
        geom
    FROM coverage
)
SELECT
    CASE kind WHEN 'region' THEN geom END AS input_region,
    CASE kind WHEN 'hexagon' THEN geom END AS hexagon
FROM layers
ORDER BY position;
```

Output

```
POLYGON((0 0,0 3,3 3,3 0,0 0)) | null
POLYGON((3 0,3 3,6 3,6 0,3 0)) | null
null | POLYGON((-1.5 0,-0.75 -1.3,0.75 -1.3,1.5 0,0.75 1.3,-0.75 1.3,-1.5 0))
null | POLYGON((-1.5 2.6,-0.75 1.3,0.75 1.3,1.5 2.6,0.75 3.9,-0.75 3.9,-1.5 2.6))
null | POLYGON((0.75 1.3,1.5 0,3 0,3.75 1.3,3 2.6,1.5 2.6,0.75 1.3))
null | POLYGON((0.75 3.9,1.5 2.6,3 2.6,3.75 3.9,3 5.2,1.5 5.2,0.75 3.9))
```

```
null | POLYGON((3 0,3.75 -1.3,5.25 -1.3,6 0,5.25 1.3,3.75 1.3,3 0))
null | POLYGON((3 2.6,3.75 1.3,5.25 1.3,6 2.6,5.25 3.9,3.75 3.9,3 2.6))
null | POLYGON((5.25 1.3,6 0,7.5 0,8.25 1.3,7.5 2.6,6 2.6,5.25 1.3))
null | POLYGON((5.25 3.9,6 2.6,7.5 2.6,8.25 3.9,7.5 5.2,6 5.2,5.25 3.9))
```

Figure



The same pattern applies to a real administrative-boundary table:



Note
Le mot-clé LATERAL est implicite pour les fonctions de retour d'ensemble lorsqu'elles font référence à une table antérieure dans la liste FROM. Ainsi, CROSS JOIN LATERAL, CROSS JOIN, ou tout simplement , sont des constructions équivalentes pour cet exemple.

Code

```
SELECT admin1.gid, hex.geom
FROM
    admin1
    CROSS JOIN
    ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
    adm0_a3 = 'USA'
    AND
    ST_Intersects(admin1.geom, hex.geom)
```

Voir aussi

ST_EstimatedExtent, ST_SetSRID, ST_SquareGrid, ST_TileEnvelope

7.3.15 ST_Hexagon

ST_Hexagon — Renvoie un seul hexagone, en utilisant la taille du bord et les coordonnées de la cellule fournies dans l'espace de la grille de l'hexagone.

Synopsis

geometry **ST_Hexagon**(float8 size, integer cell_i, integer cell_j, geometry origin);

Description

Utilise le même concept de tuilage d'hexagones que ST_HexagonGrid, mais génère un seul hexagone à la coordonnée de cellule souhaitée. En option, peut ajuster la coordonnée d'origine du tuilage, l'origine par défaut est à 0,0.

Les hexagones sont générés sans SRID défini, utilisez donc ST_SetSRID pour définir le SRID à celui que vous attendez.

Disponibilité : 3.1.0

Exemples

Creating a hexagon at the origin.

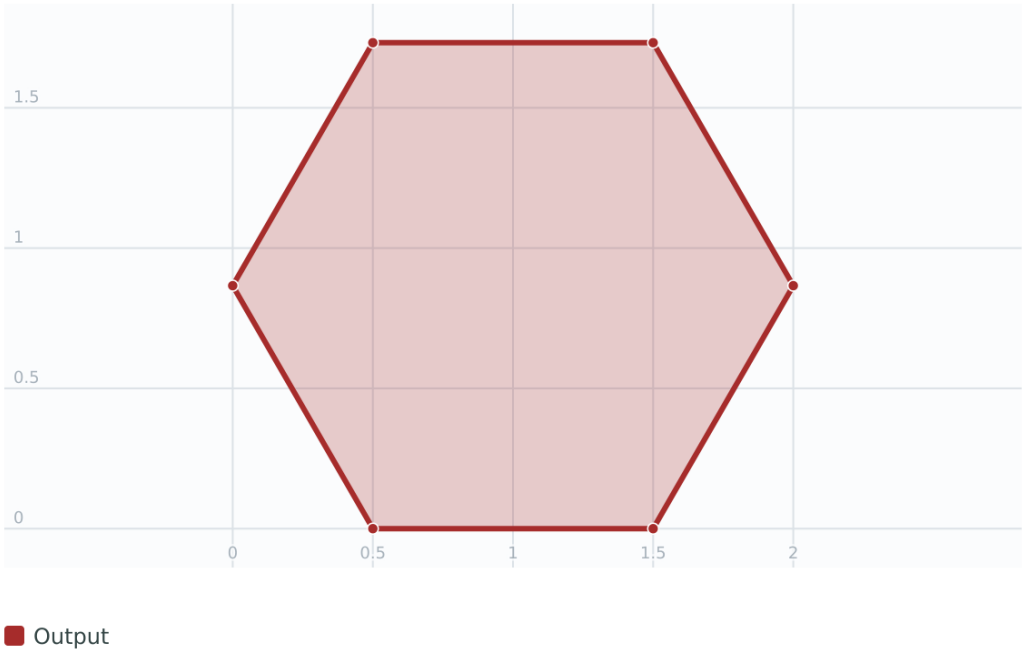
Code

```
SELECT ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857);
```

Output

```
POLYGON((-1 0,-0.5 -0.8660254037844387,0.5 -0.8660254037844387,1 0,0.5 0.8660254037844387,-0.5 0.8660254037844387,-1 0) ↩
    )
```

Figure



Voir aussi

ST_TileEnvelope, ST_HexagonGrid, ST_Square

7.3.16 ST_SquareGrid

ST_SquareGrid — Renvoie un ensemble de carrés de grille et d'indices de cellules qui couvrent complètement les limites de l'argument géométrie.

Synopsis

setof record **ST_SquareGrid**(float8 size, geometry bounds);

Description

Commence par le concept de tuilage carré du plan. Pour un SRS plan donné, et une taille d'arête donnée, en commençant à l'origine du SRS, il existe un unique pavage carré du plan, Tiling(SRS, Size). Cette fonction répond à la question : quelles grilles dans un Tiling(SRS, Size) donné se chevauchent avec une limite donnée.

Le SRS des carrés de sortie est le SRS fourni par la géométrie des limites.

Le doublement de la taille du carré ou de son bord génère un nouveau pavage parent qui s'adapte parfaitement au pavage d'origine. Les carrelages standard des cartes Web dans mercator ne sont que des puissances de deux grilles carrées dans le plan mercator.

Disponibilité : 3.1.0

Exemples

Generating a 1 degree grid for a country.

La grille remplira toutes les limites du pays, donc si vous voulez seulement des carrés qui touchent le pays, vous devrez filtrer ensuite avec ST_Intersects.

Code

```
WITH grid AS (
SELECT (ST_SquareGrid(1, ST_Transform(geom, 4326))).*
FROM admin0 WHERE name = 'Canada'
)
  SELEcT geom
  FROM grid
```

Counting points in squares (using single chopped grid).

Pour faire un résumé des points par rapport à un tuilage carré, générez une grille carrée en utilisant l'étendue des points comme limites, puis joignez spatialement à cette grille. Notez que l'étendue estimée peut être différente de l'étendue réelle, soyez donc prudent et assurez-vous au moins d'avoir analysé votre tableau.

Code

```
SELECT COUNT(*), squares.geom
FROM pointtable AS pts
INNER JOIN ST_SquareGrid(
  1000,
  ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
) AS squares
  ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom;
```

Counting points in squares using set of grid per point.

Cette méthode donne le même résultat que le premier exemple mais sera plus lente pour un grand nombre de points

Code

```
SELECT COUNT(*), squares.geom
FROM pointtable AS pts
INNER JOIN ST_SquareGrid(
  1000,
  pts.geom
) AS squares
  ON ST_Intersects(pts.geom, squares.geom)
GROUP BY squares.geom;
```

Voir aussi

ST_TileEnvelope, ST_HexagonGrid , ST_EstimatedExtent , ST_SetSRID

7.3.17 ST_Square

ST_Square — Renvoie un seul carré, en utilisant la taille du bord et la coordonnée de la cellule fournies dans l'espace de la grille du carré.

Synopsis

geometry **ST_Square**(float8 size, integer cell_i, integer cell_j, geometry origin='POINT(0 0)');

Description

Utilise le même concept de tuilage carré que ST_SquareGrid, mais génère un seul carré à la coordonnée de cellule souhaitée. En option, peut ajuster la coordonnée d'origine du tuilage, l'origine par défaut est à 0,0.

Les carrés sont générés avec le SRID de l'origine donnée. Utilisez ST_SetSRID pour définir le SRID si l'origine donnée a un SRID inconnu (comme c'est le cas par défaut).

Disponibilité : 3.1.0

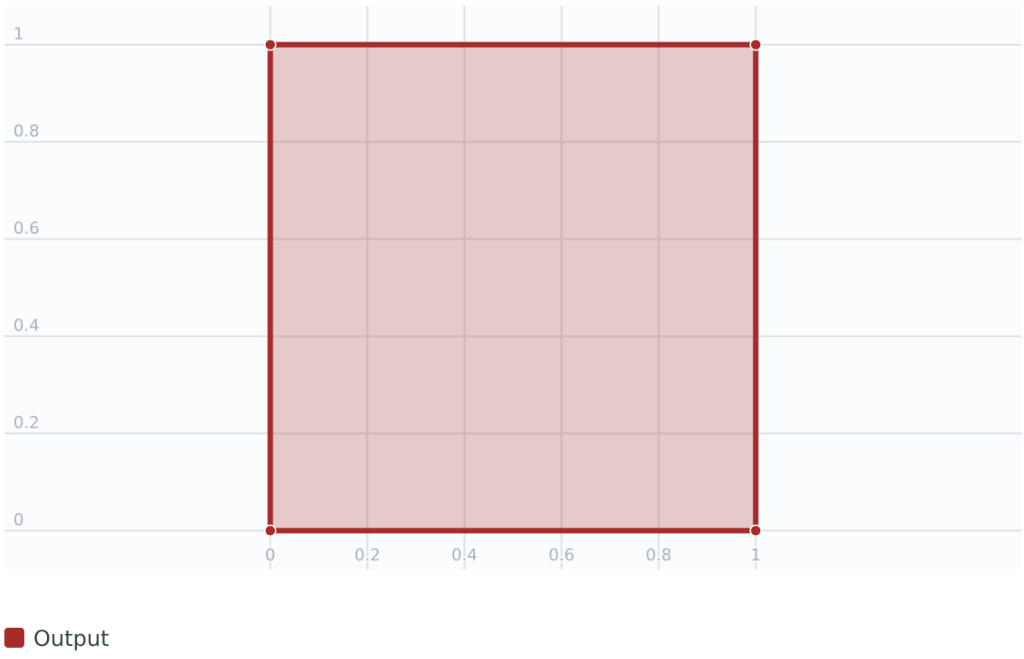
Exemples

Creating a square at the origin.

Code

```
SELECT ST_SetSRID(ST_Square(1.0, 0, 0), 3857);
```

Figure



Voir aussi

ST_TileEnvelope, ST_SquareGrid, ST_Hexagon

7.3.18 ST_Letters

ST_Letters — Renvoie les lettres d'entrée rendues sous forme de géométrie avec une position de départ par défaut à l'origine et une hauteur de texte par défaut de 100.

Synopsis

geometry **ST_Letters**(text letters, json font);

Description

Utilise une police intégrée pour rendre une chaîne de caractères sous forme de géométrie multipolygonale. La hauteur de texte par défaut est de 100,0, soit la distance entre le bas d'une descendante et le haut d'une capitale. La position de départ par défaut place le début de la ligne de base à l'origine. Pour remplacer la police, il faut passer une carte json, avec un caractère comme clé, et des TWKB encodés en base64 pour la forme de la police, les polices ayant une hauteur de 1000 unités du bas des descendantes au haut des capitales.

Le texte est généré à l'origine par défaut, donc pour repositionner et redimensionner le texte, appliquez d'abord la fonction `ST_Scale` et ensuite appliquez la fonction `ST_Translate`.

Disponibilité : 3.3.0

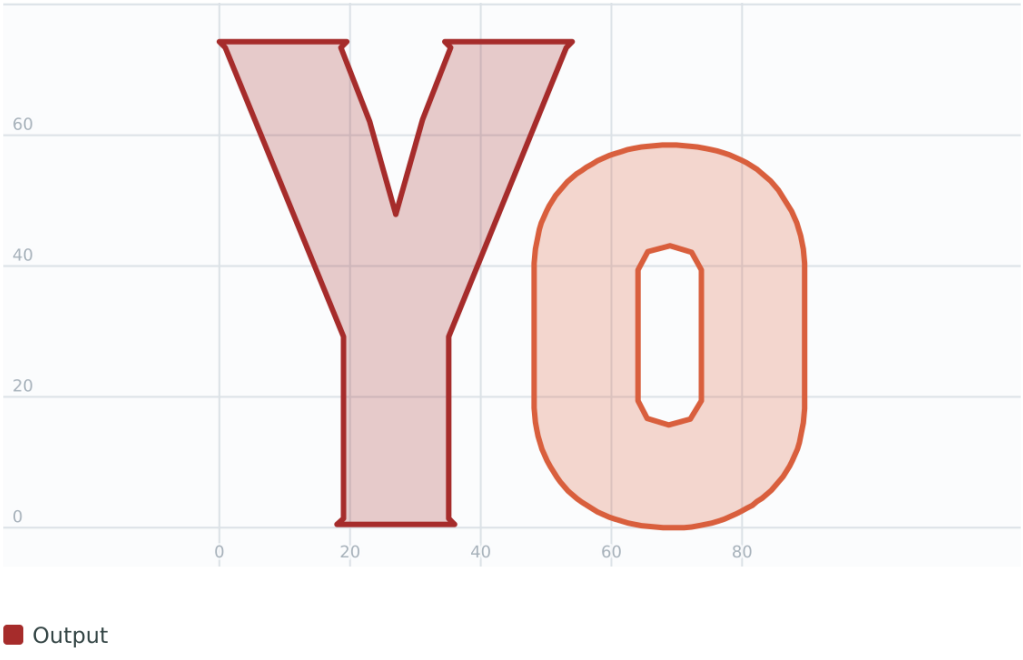
Exemples

Generating the word 'Yo'.

Code

```
SELECT ST_Letters('Yo');
```

Figure



Generating the word 'PostGIS'.

Code

```
SELECT ST_Letters('PostGIS') AS letters;
```

Figure

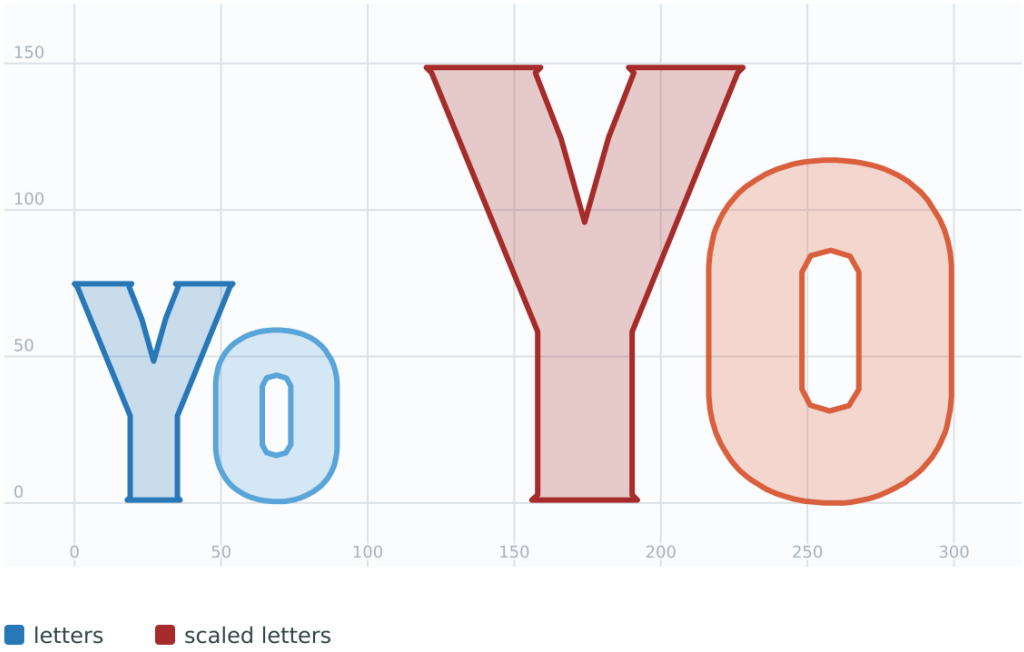


Scaling and moving words.

Code

```
WITH letters AS (
    SELECT ST_Letters('Yo') AS geom
)
SELECT geom AS input_letters,
       ST_Translate(ST_Scale(geom, 2, 2), 120, 0)
           AS scaled_letters
FROM letters;
```

Figure



Voir aussi

ST_AsTWKB, ST_Scale, ST_Translate

7.3.19 ST_MakeNurbsCurve

ST_MakeNurbsCurve — Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.

Synopsis

geometry **ST_MakeNurbsCurve**(integer degree, geometry control_points, float8[] weights, float8[] knots);

Description

Creates a NURBS (Non-Uniform Rational B-Spline) curve geometry from the specified parameters. NURBS curves are smooth parametric curves that provide precise control over curve shape through control points, weights, and knot values.

The `degree` parameter specifies the polynomial degree of the curve (must be between 1 and 10). Higher degrees produce smoother curves.

The `control_points` must be a LINESTRING geometry defining the control polygon. The number of control points must be at least `degree` + 1.

The optional `weights` array assigns a weight to each control point. All weights must be positive. If NULL or omitted, uniform weights (1.0) are used, resulting in a non-rational B-spline.

The optional `knots` array defines the parameter space. If NULL or omitted, a uniform clamped knot vector is automatically generated. If provided, the knot vector must have exactly (`npoints` + `degree` + 1) elements and must be non-decreasing.

Availability: 3.7.0

Exemples

This example creates a degree-2 NURBS curve with uniform weights.

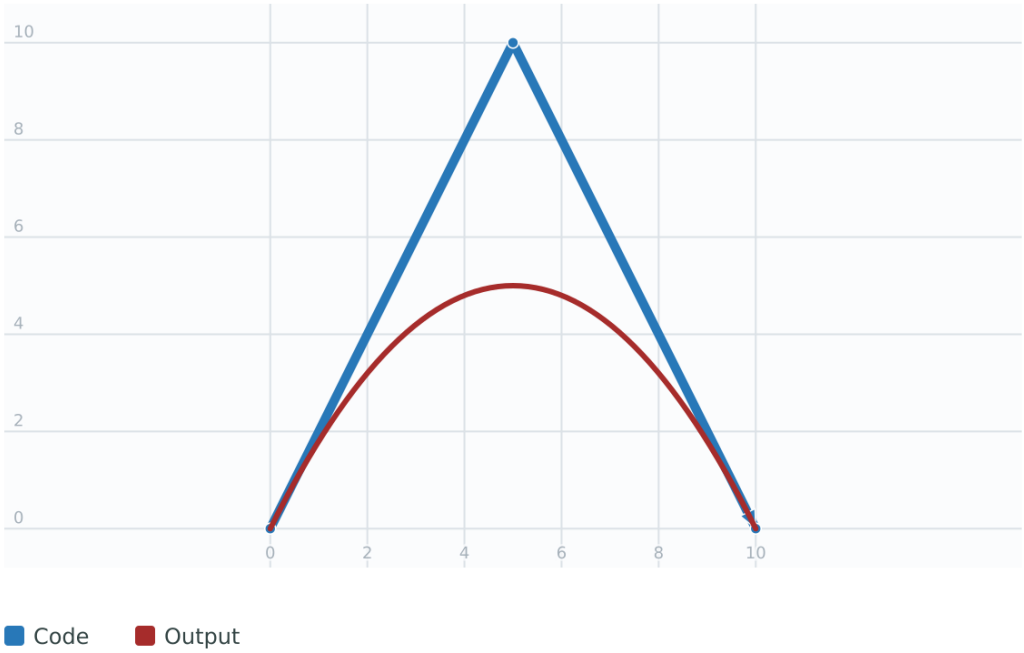
Code

```
SELECT ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0,5 10,10 0)'::geometry
);
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 1),
    NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



This example creates a rational NURBS curve with custom weights.

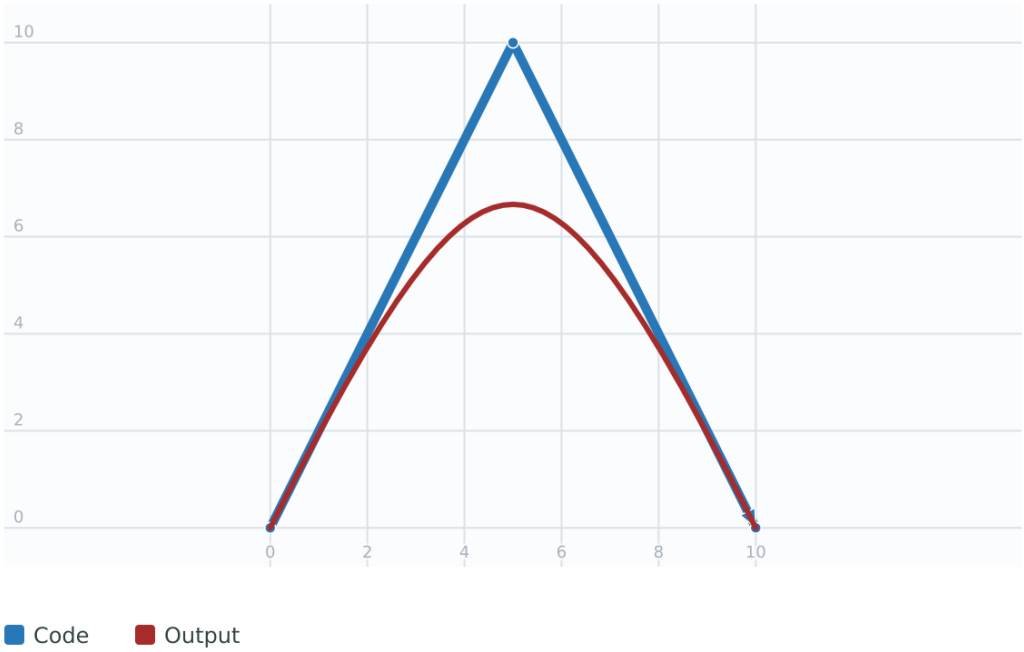
Code

```
SELECT ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0,5 10,10 0)'::geometry,
    ARRAY[1.0, 2.0, 1.0]
);
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 2),
    NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



This example creates a NURBS curve with a non-uniform knot vector.

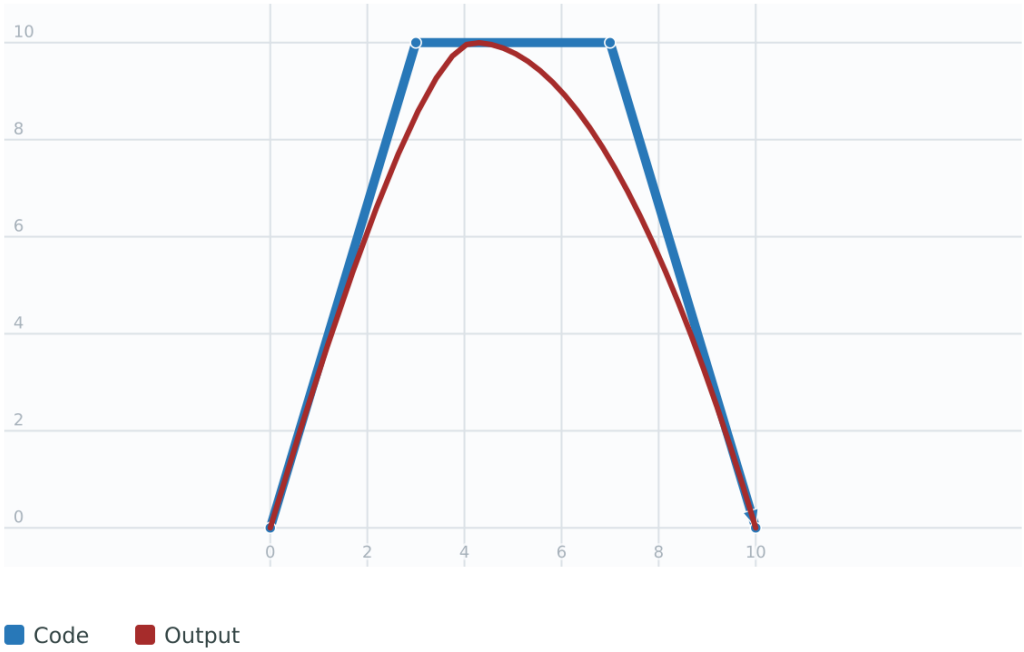
Code

```
SELECT ST_MakeNurbsCurve(
    2,
    'LINESTRING(0 0,3 10,7 10,10 0)'::geometry,
    ARRAY[1.0, 1.0, 1.0, 1.0],
    ARRAY[0.0, 0.0, 0.0, 0.3, 1.0, 1.0, 1.0]
);
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(3 10),WEIGHT 1),
    NURBSPOINT(WEIGHTEDPOINT(7 10),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(0.3,1),
     KNOT(1,3)))
```

Figure



Voir aussi

ST_NurbsToLineString, ST_Evaluate, ST_ControlPoints

7.4 Fonctions d'accès aux géométries

7.4.1 GeometryType

GeometryType — Renvoie le type d'une géométrie sous forme de texte.

Synopsis

text **GeometryType**(geometry geomA);

Description

Retourne le type de la géométrie, par exemple : 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Retourne le nom du sous type instanciable de la géométrie. Le nom est retourné sous forme de texte.



Note
Cette fonction indique si la géométrie comporte une dimension de type mesure, en retournant un texte de la forme 'POINTM'.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

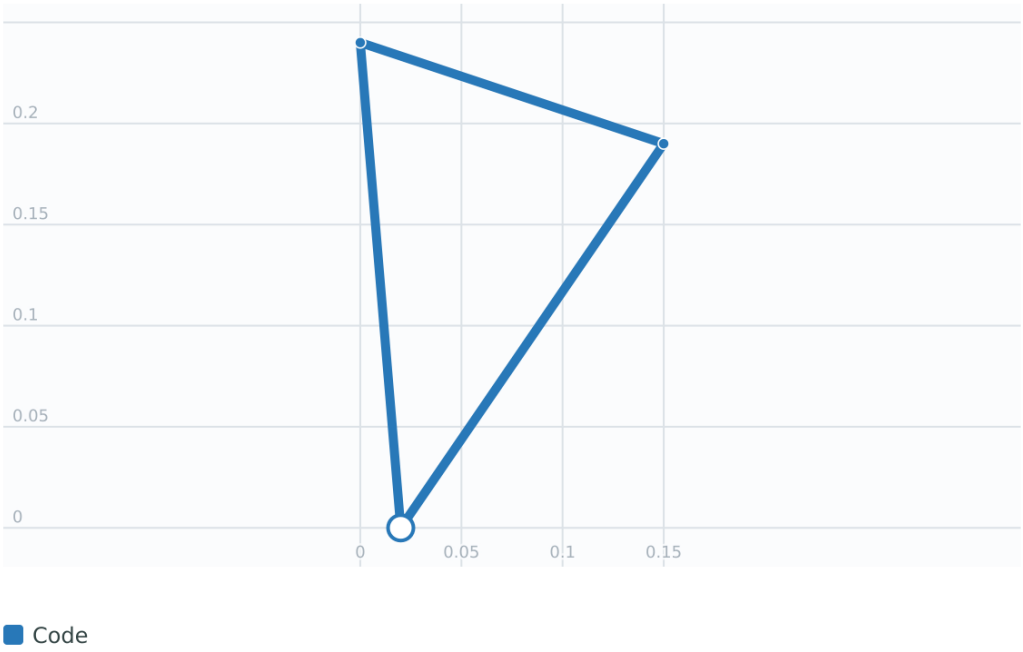
Code

```
SELECT GeometryType('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'::geometry) ↩
    ;
```

Output

```
LINESTRING
```

Figure



Code

```
SELECT ST_GeometryType('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )');
    POLYHEDRALSURFACE
```

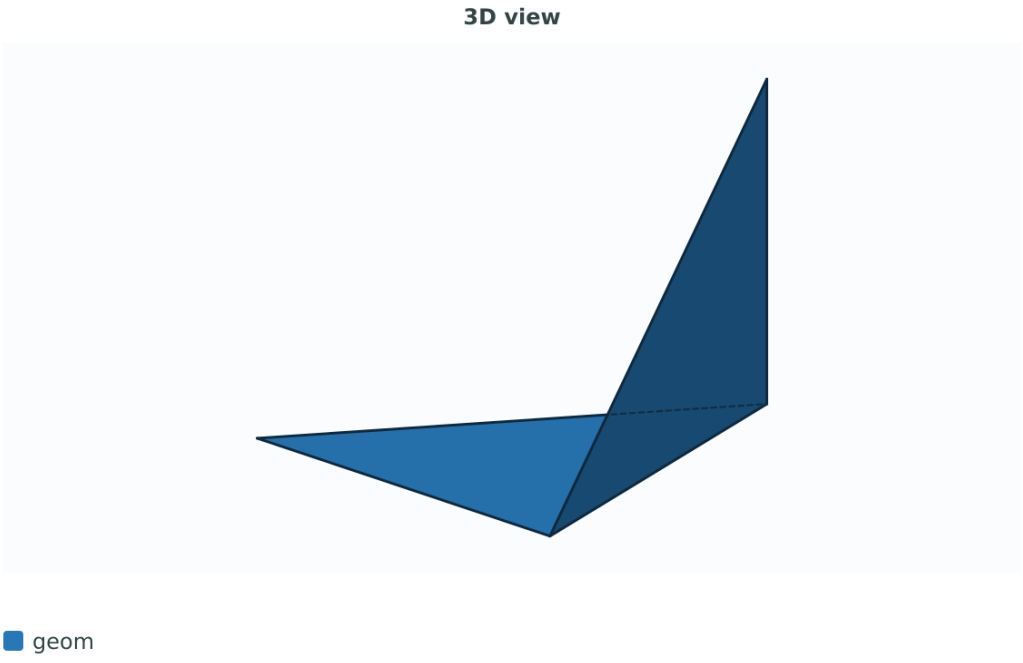
Code

```
SELECT GeometryType(geom) as result
FROM
  (SELECT
     'TIN (((
               0 0 0,
               0 0 1,
               0 1 0,
               0 0 0
           )),((
               0 0 0,
               0 1 0,
               1 1 0,
               0 0 0
           ))
           )'::geometry  AS geom
  ) AS g;
```

Output

```
TIN
```

Figure



Voir aussi

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — Renvoie la limite d'une géométrie.

Synopsis

geometry **ST_Boundary**(geometry geomA);

Description

Renvoie l'ensemble formant la frontière finie de cette géométrie. La notion de frontière est définie dans la section 3.12.3.2 des spécifications OGC. Le résultat de cette fonction est un ensemble topologiquement fermé, représentable avec les types de base, comme décrit dans la section 3.12.2 des spécifications OGC.

Effectué par le module GEOS



Note
Avant la version 2.0.0, cette fonction renvoie une exception si une `GEOMETRYCOLLECTION` est passée en paramètre. A partir de la 2.0.0, la fonction renvoie null (paramètre non supporté).

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1

 Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1.17

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Amélioration : 2.1.0 introduction du support pour Triangle

Modifié : 3.2.0 support pour TIN, n'utilise pas geos, ne linéarise pas les courbes

Exemples

LineString with boundary points overlaid.

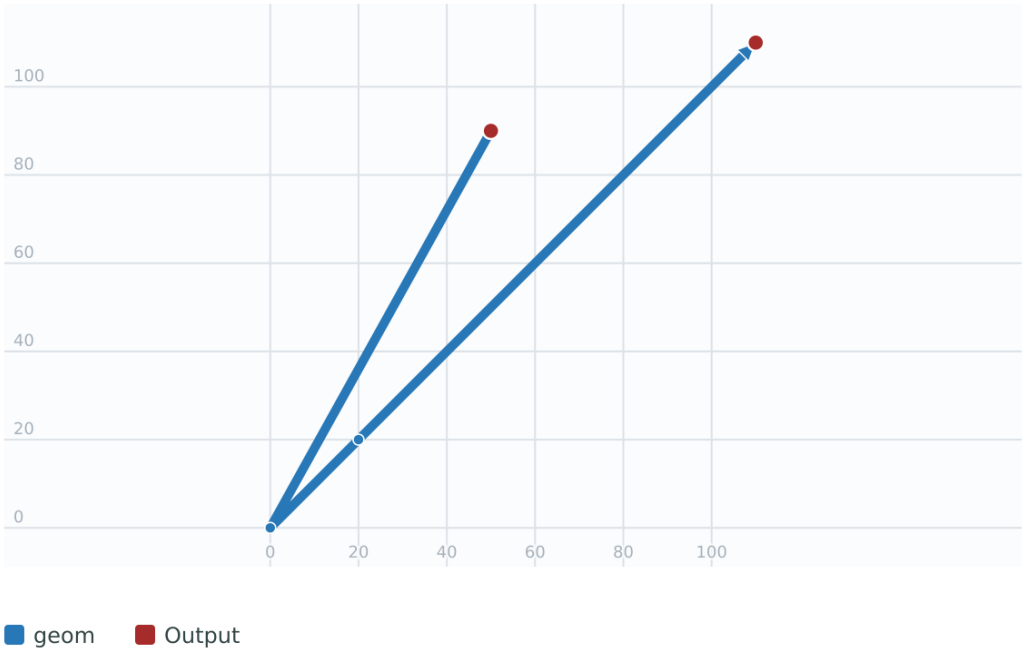
Code

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60,70 80,160 170)'::geometry As geom) As f;
```

Output

```
MULTIPOINT((100 150),(160 170))
```

Figure



Polygon holes with the boundary MultiLineString overlaid.

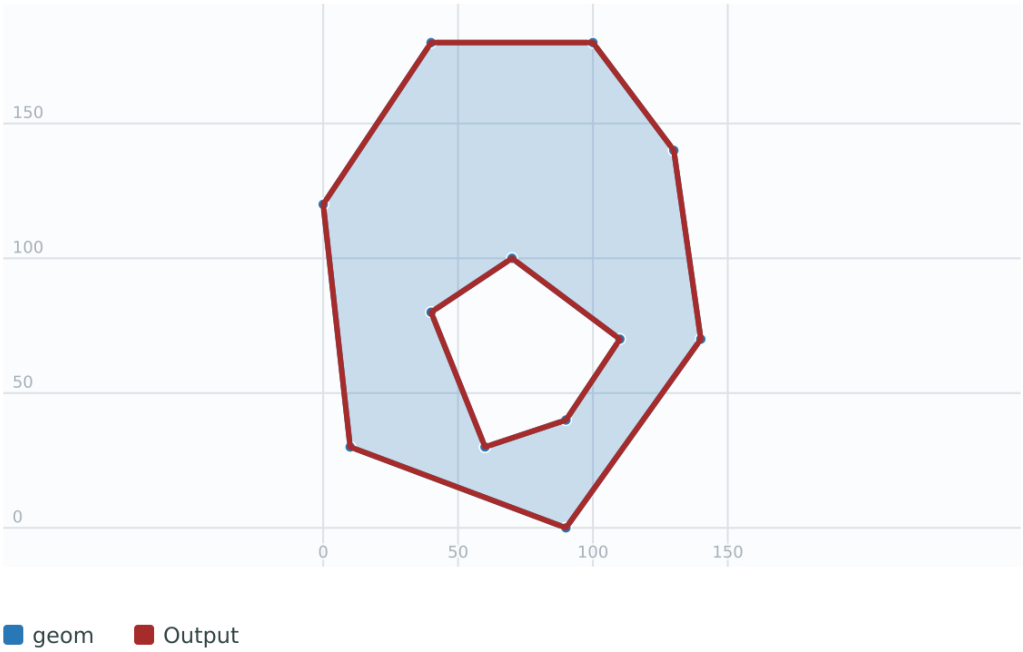
Code

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130 ),
    ( 70 40,100 50,120 80,80 110,50 90,70 40 ))'::geometry As geom) As f;
```

Output

```
MULTILINESTRING((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130),
        (70 40,100 50,120 80,80 110,50 90,70 40))
```

Figure



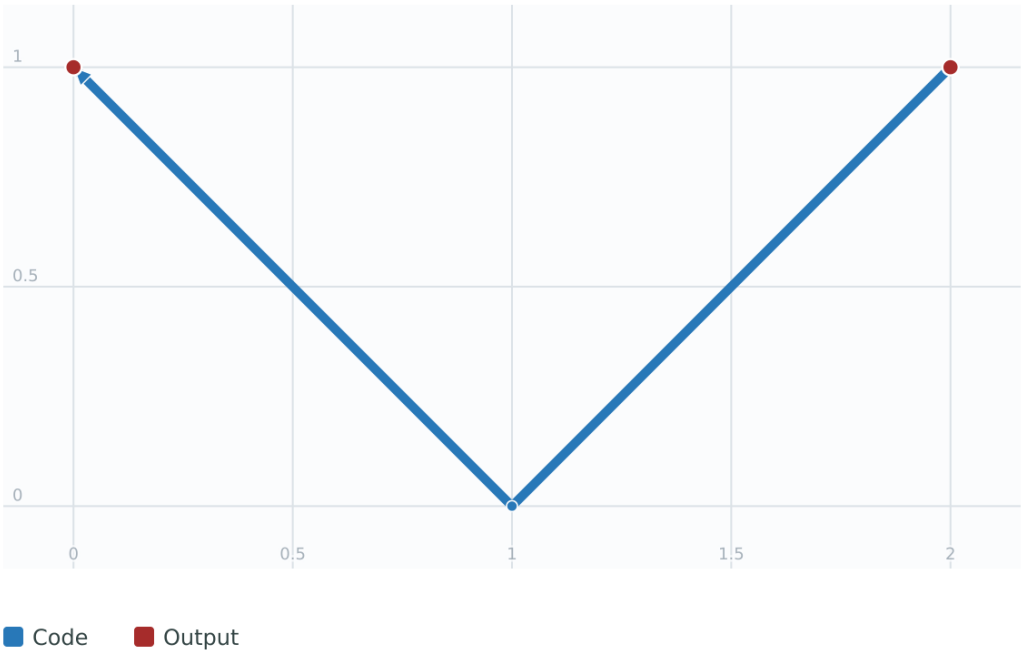
Code

```
SELECT ST_Boundary('LINESTRING(1 1,0 0,-1 1)');
```

Output

```
MULTIPOINT((1 1),(-1 1))
```

Figure



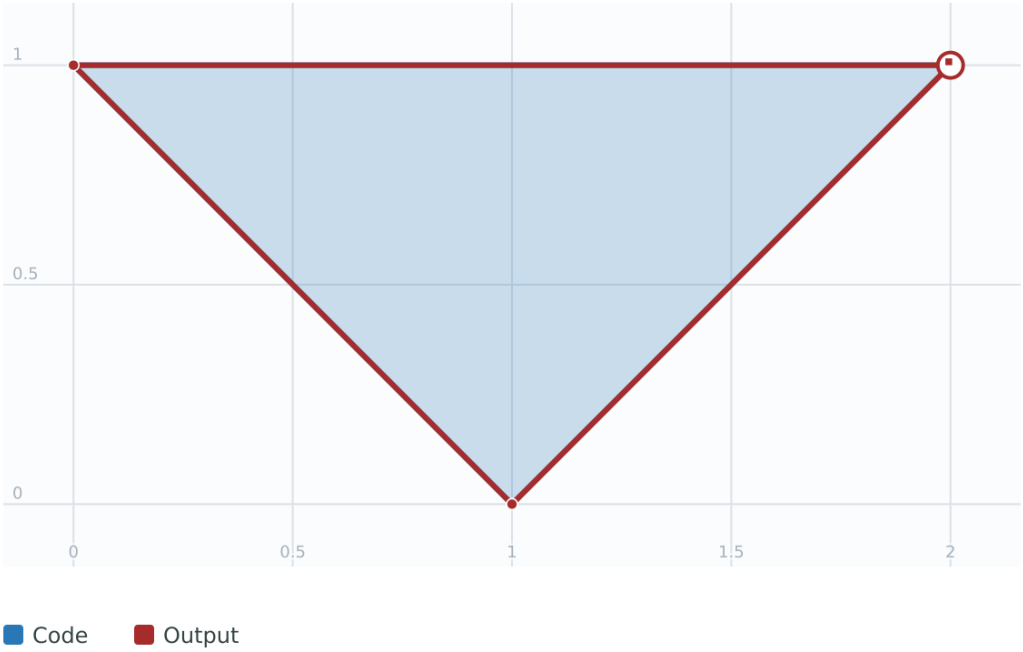
Code

```
SELECT ST_Boundary('POLYGON((1 1,0 0,-1 1,1 1))');
```

Output

```
LINESTRING(1 1,0 0,-1 1,1 1)
```

Figure



This example uses a 3D polygon.

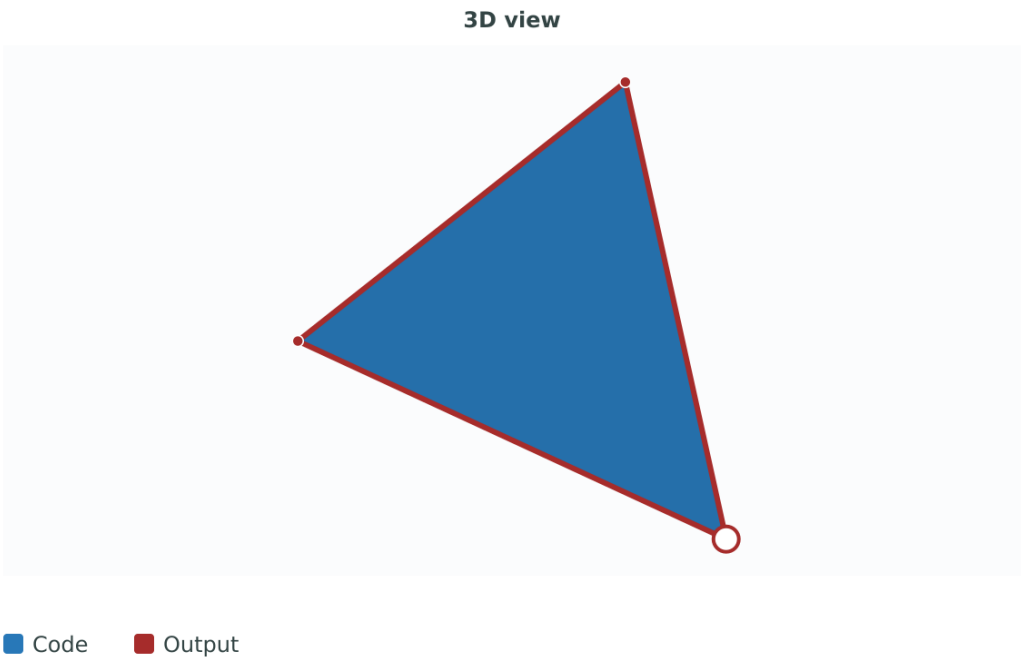
Code

```
SELECT ST_Boundary('POLYGON((1 1 1,0 0 1,-1 1 1,1 1 1))');
```

Output

```
LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)
```

Figure



This example uses a 3D MultiLineString.

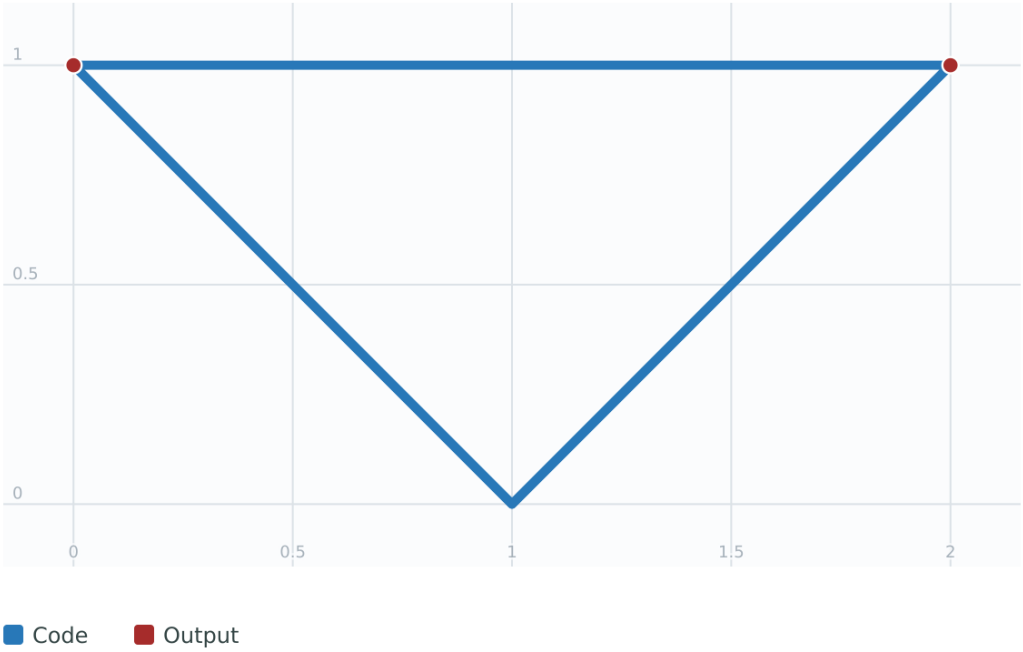
Code

```
SELECT ST_Boundary('MULTILINESTRING((1 1 1,0 0 0.5,-1 1 1),(1 1 0.5,0 0 0.5,-1 1 0.5,1 1 ↩
    0.5) )');
```

Output

```
MULTIPOINT(1 1 1,-1 1 1)
```

Figure



Voir aussi

ST_AsText, ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — Retourne la diagonale de la boîte englobante pour la géométrie en argument.

Synopsis

geometry **ST_BoundingDiagonal**(geometry geom, boolean fits=false);

Description

Renvoie la diagonale de la boîte de délimitation de la géométrie fournie sous la forme d'une LineString. La diagonale est une LineString à 2 points avec les valeurs minimales de chaque dimension dans son point de départ et les valeurs maximales dans son point d'arrivée. Si la géométrie d'entrée est vide, la diagonale est une LINESTRING EMPTY.

Le paramètre facultatif `fits` spécifie si le meilleur ajustement est nécessaire. S'il est faux, la diagonale d'une boîte de délimitation un peu plus grande peut être acceptée (ce qui est plus rapide à calculer pour les géométries comportant de nombreux sommets). Dans les deux cas, la boîte de délimitation de la ligne diagonale renvoyée couvre toujours la géométrie d'entrée.

La géométrie renvoyée conserve le SRID et la dimensionnalité (présence Z et M) de la géométrie en entrée.



Note
Dans les cas dégénérés (c'est-à-dire un seul sommet dans l'entrée), la ligne retournée sera formellement invalide (pas d'intérieur). Le résultat est toujours topologiquement valide.

Disponibilité : 2.2.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette fonction prend en charge les coordonnées M.

Exemples

This example shows a polygon together with the diagonal of its bounding box.

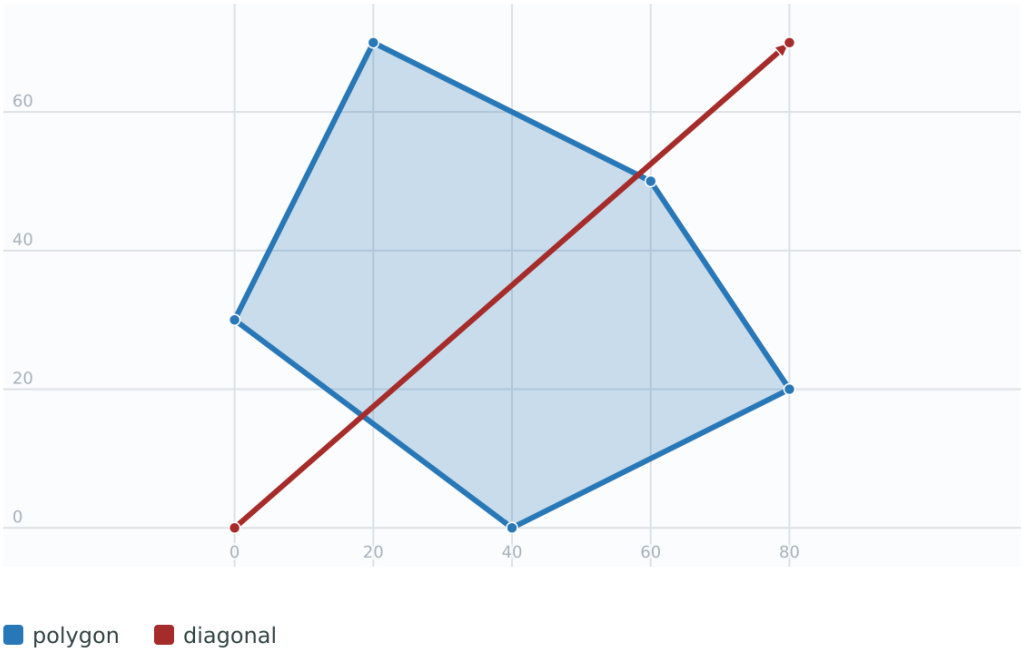
Code

```
WITH input AS (
  SELECT 'POLYGON((10 20,30 60,70 40,90 10,50 -10,10 20))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_BoundingDiagonal(geom) AS diagonal
FROM input;
```

Output

```
POLYGON((10 20,30 60,70 40,90 10,50 -10,10 20)) | LINESTRING(10 -10,90 60)
```

Figure



This example returns the minimum X coordinate of a buffer around a point.

Code

```
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_Point(0, 0), 10)
)));
```

Output

```
-10
```

Voir aussi

ST_StartPoint, ST_EndPoint, ST_X, ST_Y, ST_Z, ST_M, ST_Envelope

7.4.4 ST_CoordDim

ST_CoordDim — Renvoie la dimension des coordonnées d'une géométrie.

Synopsis

integer **ST_CoordDim**(geometry geomA);

Description

Retourne la dimension des coordonnées d'une valeur ST_Geometry.

Alias SQL/MM pour la fonction ST_NDims

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.3

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Code

```
SELECT ST_CoordDim('CIRCULARSTRING(1 2 3,1 3 4,5 6 7,8 9 10,11 12 13)');
```

Output

```
3
```

Code

```
SELECT ST_CoordDim(ST_Point(1, 2));
```

Output

```
2
```

Voir aussi

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — Renvoie la dimension topologique d'une géométrie.

Synopsis

integer **ST_Dimension**(geometry g);

Description

Renvoie la dimension topologique de cet objet Geometry, qui doit être inférieure ou égale à la dimension des coordonnées. OGC SPEC s2.1.1.1 - renvoie 0 pour `POINT`, 1 pour `LINESTRING`, 2 pour `POLYGON`, et la plus grande dimension des composants d'une `GEOMETRYCOLLECTION`. Si la dimension est inconnue (par exemple, pour une `GEOMETRYCOLLECTION` vide), 0 est renvoyé.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.2



Note
A `TIN`, including a closed TIN surface, is a surface geometry and has dimension 2. Use CG_MakeSolid to make the solid representation explicit; the resulting 3D solid has dimension 3. Solid geometry semantics, such as volume, are handled separately by the SQL/MM 3D solid functions (SQL-MM IEC 13249-3: 9.1).

Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Ne renvoie plus une exception si une GEOMETRY EMPTY est passée.



Note
Avant la version 2.0.0, cette fonction lève une exception si elle est utilisée avec une géométrie vide.


Cette fonction prend en charge les surfaces Polyhedral.


Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

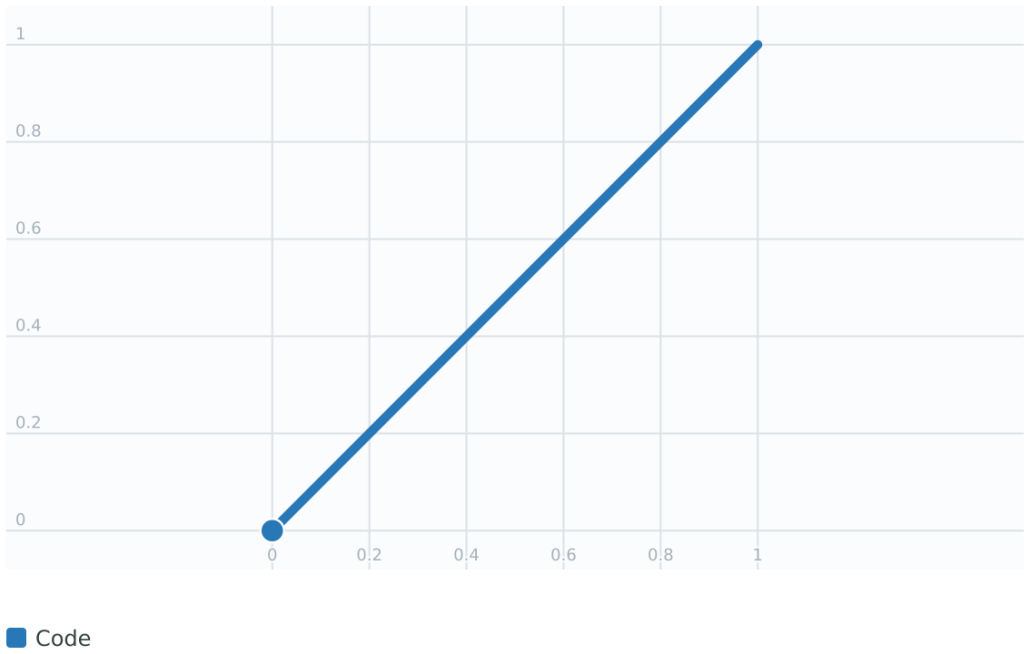
Code

```
SELECT ST_Dimension('GEOMETRYCOLLECTION(LINESTRING(1 1,0 0), POINT(0 0))');
```

Output

```
1
```

Figure



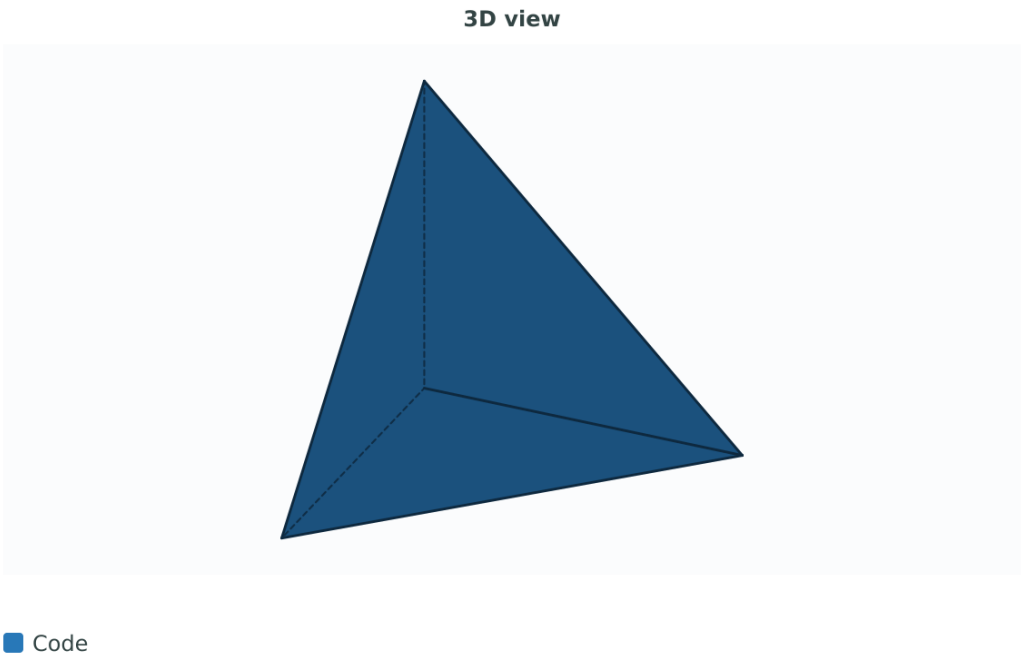
Code

```
SELECT ST_Dimension('TIN Z (((0 0 0,0 0 1,0 1 0,0 0 0)),((0 0 0,0 1 0,1 0 0,0 0 0)),((0 0 ↩
    0,1 0 0,0 0 1,0 0 0)),((1 0 0,0 1 0,0 0 1,1 0 0)))');
```

Output

```
2
```

Figure



Voir aussi

ST_NDims, CG_MakeSolid, CG_Volume

7.4.6 ST_Dump

ST_Dump — Renvoie un ensemble de lignes `geometry_dump` pour les composants d'une géométrie.

Synopsis

geometry_dump[] **ST_Dump**(geometry g1);

Description

Une fonction de retour d'ensemble (SRF) qui extrait les composants d'une géométrie. Elle renvoie un ensemble de geometry_dump lignes, chacune contenant une géométrie (champ *`geom`*) et un tableau d'entiers (champ *`path`*).

Pour un type de géométrie atomique (POINT, LINESTRING, POLYGONE), un seul enregistrement est renvoyé avec un tableau *`path`* vide et la géométrie en entrée en tant que *`geom`*. Pour une collection ou une géométrie multiple, un enregistrement est renvoyé pour chacun des composants de la collection, et le *`path`* indique la position du composant à l'intérieur de la collection.

ST_Dump est utile pour développer les géométries. C'est l'inverse d'un ST_Collect / GROUP BY, en ce sens qu'il crée de nouvelles lignes. Par exemple, il peut être utilisé pour développer les MULTIPOLYGONES en POLYGONES.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Disponibilité : PostGIS 1.0.0RC1. Nécessite PostgreSQL 7.3 ou plus.

Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Dump the components of geometries stored in a table.

Code

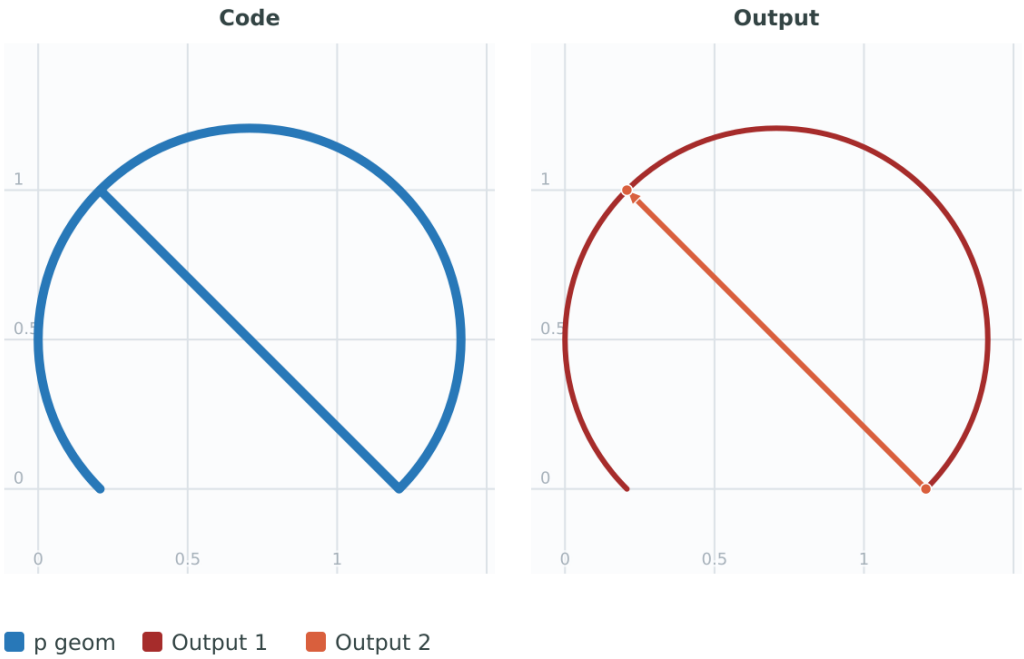
```
SELECT sometable.field1,
       (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;
```

Break a CompoundCurve into its constituent LineStrings and CircularStrings.

Code

```
SELECT a.geom, ST_HasArc(a.geom)
  FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
         FROM (SELECT 'COMPOUNDCURVE(CIRCULARSTRING(0 0,1 1,1 0),(1 0,0 1))'::geometry AS ↩
    p_geom) AS b
        ) AS a;
```

Figure

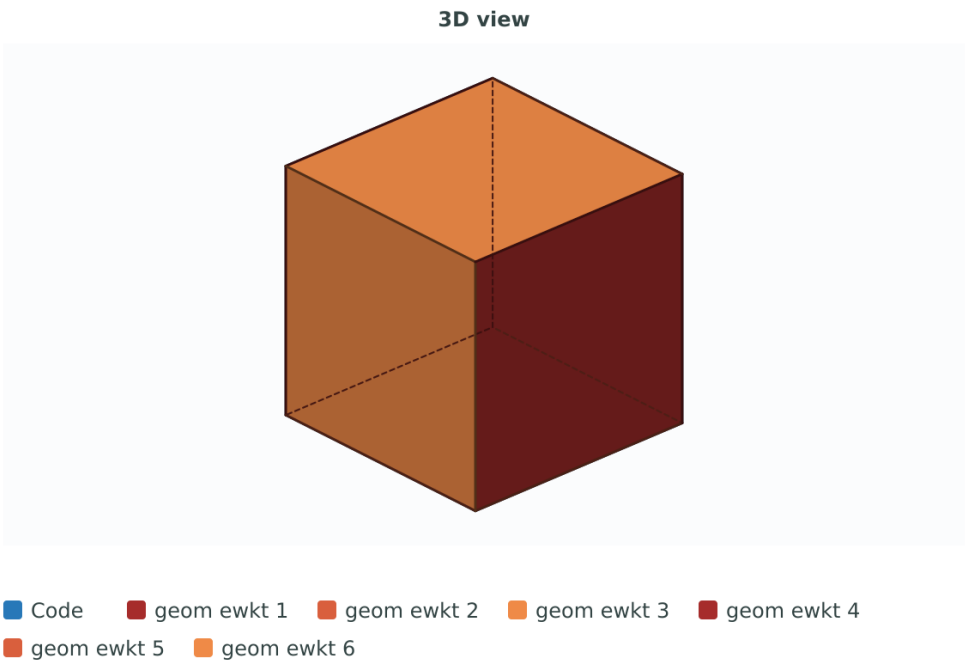


This example breaks a PolyhedralSurface into its faces.

Code

```
SELECT (a.p_geom).path[1] As path,(a.p_geom).geom As geom_ewkt
  FROM (SELECT ST_Dump('POLYHEDRALSURFACE(
((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)' ) AS p_geom )  AS a;
```

Figure

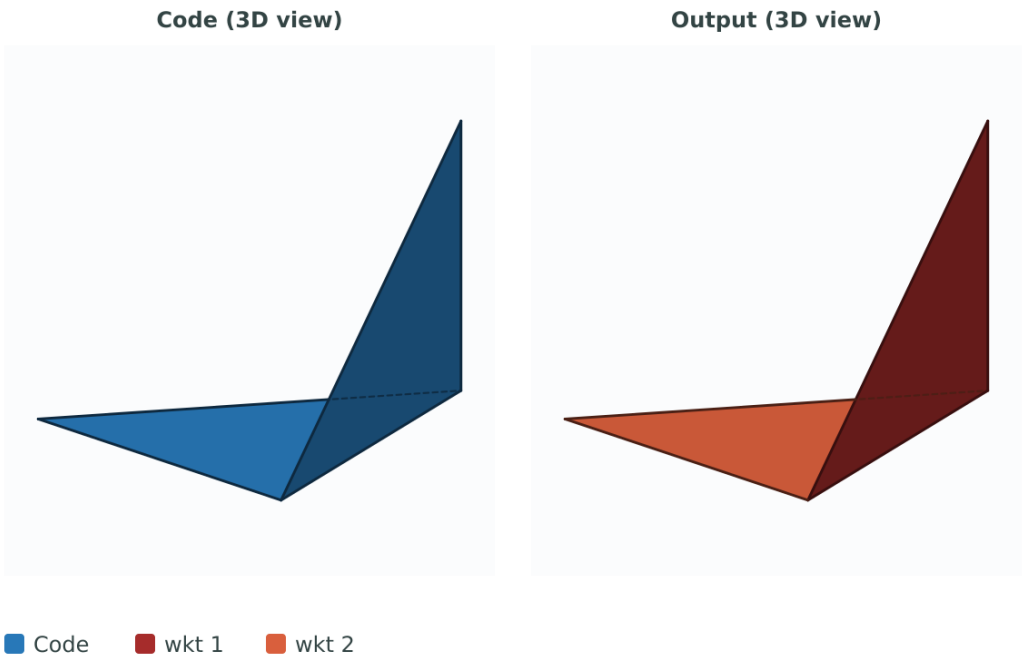


This example uses a TIN geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_Dump('TIN (((
                  0 0 0,
                  0 0 1,
                  0 1 0,
                  0 0 0
            )),((
                  0 0 0,
                  0 1 0,
                  1 1 0,
                  0 0 0
            ))
            )' ) AS gdump
    ) AS g;
```

Figure



Voir aussi

geometry_dump, Section 13.6, ST_Collect, ST_GeometryN

7.4.7 ST_DumpPoints

ST_DumpPoints — Renvoie un ensemble de lignes `geometry_dump` pour les coordonnées dans une géométrie.

Synopsis

geometry_dump[] **ST_DumpPoints**(geometry geom);

Description

Une fonction de retour d'ensemble (SRF) qui extrait les coordonnées (sommets) d'une géométrie. Elle renvoie un ensemble de geometry_dump lignes, chacune contenant une géométrie (champ *`geom`*) et un tableau d'entiers (champ *`path`*).

— le champ *`geom`* `POINT` représente les coordonnées de la géométrie fournie.

— le champ *`path`* (un `integer[]`) est un index énumérant les positions des coordonnées dans les éléments de la géométrie fournie. Les indices sont basés sur 1. Par exemple, pour un `LINESTRING`, les chemins sont `{i}` où `i` est la `nième` coordonnée dans le `LINESTRING`. Pour un `POLYGONE`, les chemins sont `{i, j}` où `i` est le numéro de l'anneau (1 est l'anneau extérieur ; les anneaux intérieurs suivent) et `j` est la position de la coordonnée dans l'anneau.

Pour obtenir une géométrie unique contenant les coordonnées, utilisez ST_Points.

Amélioré : 2.1.0 Vitesse plus rapide. Réimplémentation en C natif.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Disponibilité : 1.5.0

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

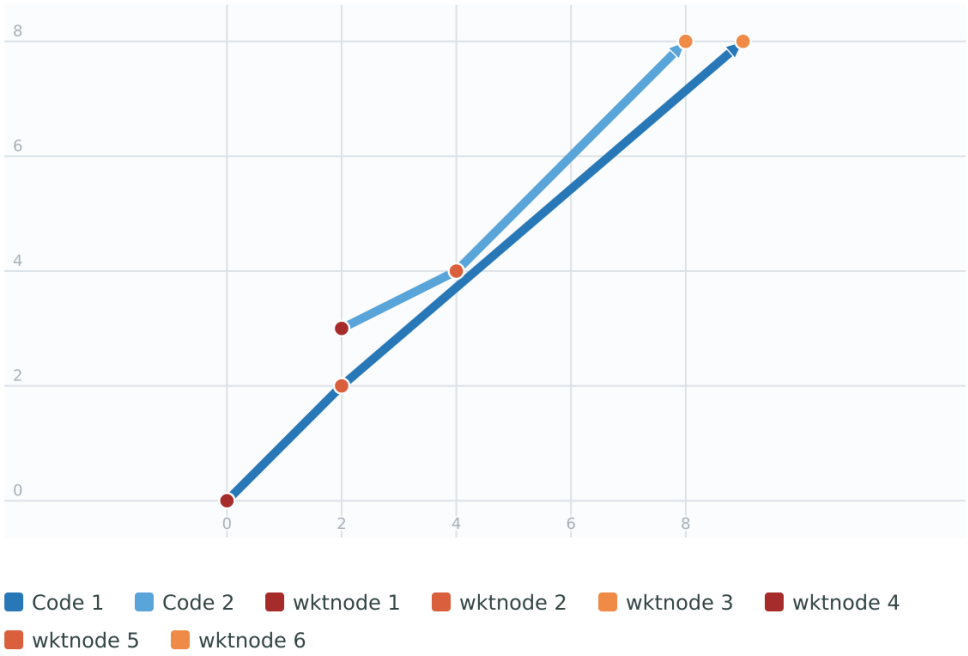
Exemples

Explode a table of LineStrings into their vertices.

Code

```
SELECT edge_id, (dp).path[1] As index,(dp).geom As wktnode
FROM (SELECT 1 As edge_id,
    ST_DumpPoints(ST_GeomFromText('LINESTRING(1 2,3 4,10 10)')) AS dp
     UNION ALL
     SELECT 2 As edge_id,
    ST_DumpPoints(ST_GeomFromText('LINESTRING(3 5,5 6,9 10)')) AS dp
   ) As foo;
```

Figure



Dump every vertex from a mixed GeometryCollection.

Code

```
SELECT path,geom
FROM (
  SELECT (ST_DumpPoints(g.geom)).*
  FROM
    (SELECT
       'GEOMETRYCOLLECTION(
          POINT ( 0 1 ),
          LINESTRING ( 0 3,3 4 ),
          POLYGON (( 2 0,2 3,0 2,2 0 )),
          POLYGON (( 3 0,3 3,6 3,6 0,3 0 ),
```

```
                    ( 5 1,4 2,5 2,5 1 )),
          MULTIPOLYGON (
                  (( 0 5,0 8,4 8,4 5,0 5 ),
                   ( 1 6,3 6,2 7,1 6 )),
                  (( 5 4,5 8,6 7,5 4 ))
          )
        )'::geometry AS geom
    ) AS g
  ) j;
```

Output

```
path    | st_astext
--------+-----------
 {1,1}     | POINT(0 1)
 {2,1}     | POINT(0 3)
 {2,2}     | POINT(3 4)
 {3,1,1}   | POINT(2 0)
 {3,1,2}   | POINT(2 3)
 {3,1,3}   | POINT(0 2)
 {3,1,4}   | POINT(2 0)
 {4,1,1}   | POINT(3 0)
 {4,1,2}   | POINT(3 3)
 {4,1,3}   | POINT(6 3)
 {4,1,4}   | POINT(6 0)
 {4,1,5}   | POINT(3 0)
 {4,2,1}   | POINT(5 1)
 {4,2,2}   | POINT(4 2)
 {4,2,3}   | POINT(5 2)
 {4,2,4}   | POINT(5 1)
 {5,1,1,1} | POINT(0 5)
 {5,1,1,2} | POINT(0 8)
 {5,1,1,3} | POINT(4 8)
 {5,1,1,4} | POINT(4 5)
 {5,1,1,5} | POINT(0 5)
 {5,1,2,1} | POINT(1 6)
 {5,1,2,2} | POINT(3 6)
 {5,1,2,3} | POINT(2 7)
 {5,1,2,4} | POINT(1 6)
 {5,2,1,1} | POINT(5 4)
 {5,2,1,2} | POINT(5 8)
 {5,2,1,3} | POINT(6 7)
 {5,2,1,4} | POINT(5 4)
(29 rows)
```

Collect the dumped vertices to visualize them over the source geometry.

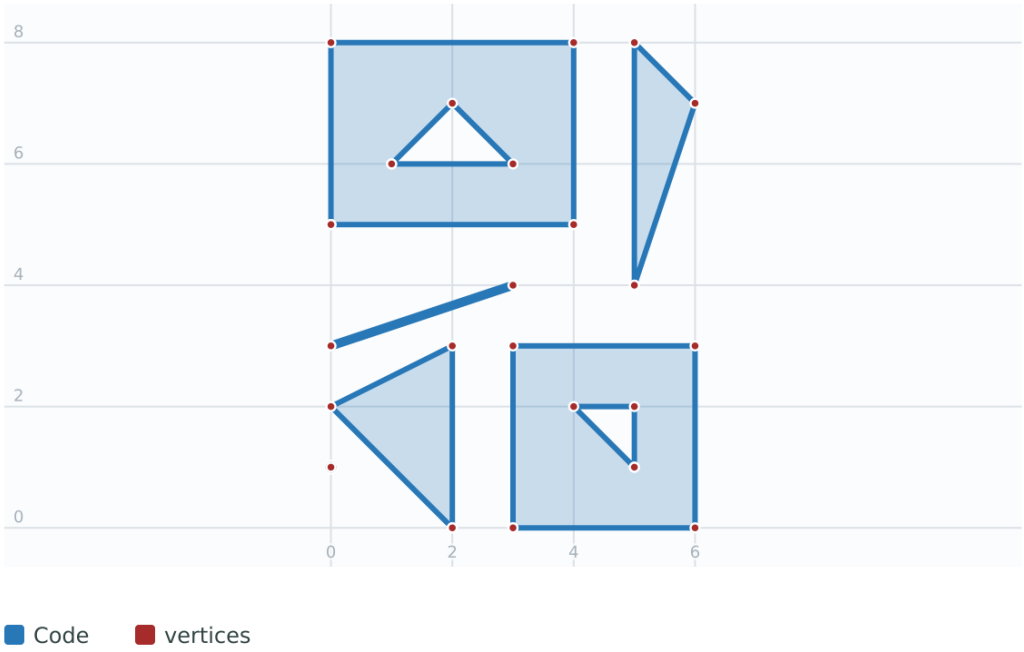
Code

```
WITH source(geom) AS (VALUES (
  'GEOMETRYCOLLECTION(
     POINT (0 1),
     LINESTRING (0 3,3 4),
     POLYGON ((2 0,2 3,0 2,2 0)),
     POLYGON ((3 0,3 3,6 3,6 0,3 0),(5 1,4 2,5 2,5 1)),
     MULTIPOLYGON (((0 5,0 8,4 8,4 5,0 5),(1 6,3 6,2 7,1 6)),
                   ((5 4,5 8,6 7,5 4))))'::geometry
))
SELECT ST_Collect((dp).geom ORDER BY (dp).path) AS vertices
FROM source
CROSS JOIN LATERAL ST_DumpPoints(source.geom) AS dp;
```

Output

```
MULTIPOINT((0 1),(0 3),(3 4),(2 0),(2 3),(0 2),(2 0),(3 0),(3 3),(6 3),(6 0),(3 0),(5 1),(4 2),(5 2),(5 1),(0 5),(0 8) ↩
    ,(4 8),(4 5),(0 5),(1 6),(3 6),(2 7),(1 6),(5 4),(5 8),(6 7),(5 4))
```

Figure



This example uses a cube represented as a PolyhedralSurface.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_DumpPoints('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )' ) AS gdump
    ) AS g;
```

Output

```
path   |     wkt
---------+--------------
 {1,1,1} | POINT(0 0 0)
 {1,1,2} | POINT(0 0 1)
 {1,1,3} | POINT(0 1 1)
 {1,1,4} | POINT(0 1 0)
 {1,1,5} | POINT(0 0 0)
 {2,1,1} | POINT(0 0 0)
 {2,1,2} | POINT(0 1 0)
 {2,1,3} | POINT(1 1 0)
 {2,1,4} | POINT(1 0 0)
 {2,1,5} | POINT(0 0 0)
 {3,1,1} | POINT(0 0 0)
 {3,1,2} | POINT(1 0 0)
 {3,1,3} | POINT(1 0 1)
 {3,1,4} | POINT(0 0 1)
 {3,1,5} | POINT(0 0 0)
 {4,1,1} | POINT(1 1 0)
 {4,1,2} | POINT(1 1 1)
 {4,1,3} | POINT(1 0 1)
 {4,1,4} | POINT(1 0 0)
 {4,1,5} | POINT(1 1 0)
 {5,1,1} | POINT(0 1 0)
 {5,1,2} | POINT(0 1 1)
 {5,1,3} | POINT(1 1 1)
 {5,1,4} | POINT(1 1 0)
 {5,1,5} | POINT(0 1 0)
 {6,1,1} | POINT(0 0 1)
 {6,1,2} | POINT(1 0 1)
```

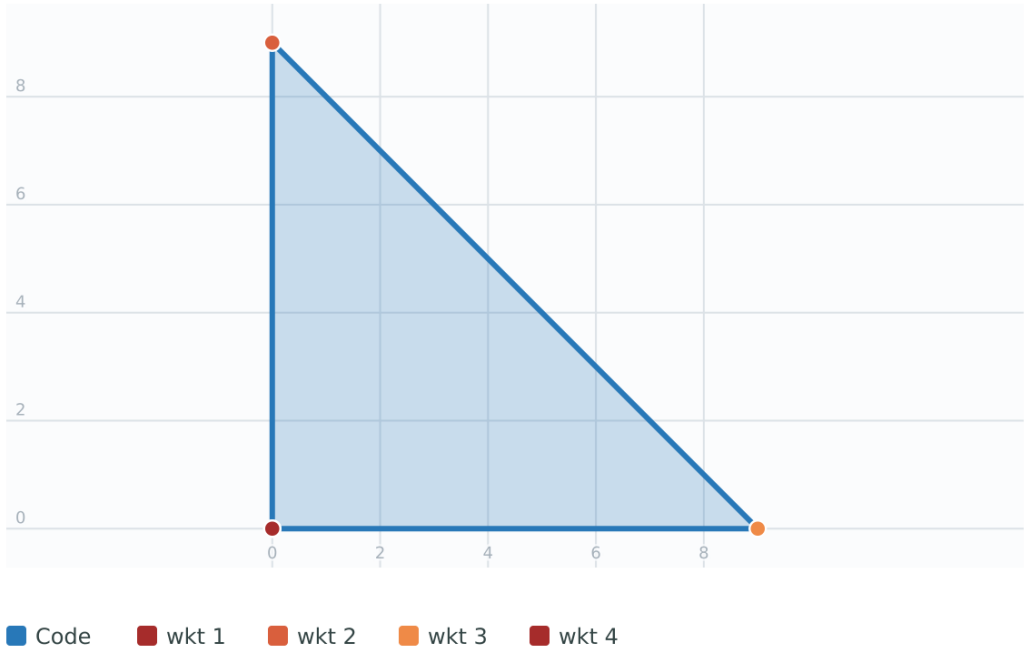
```
 {6,1,3} | POINT(1 1 1)
 {6,1,4} | POINT(0 1 1)
 {6,1,5} | POINT(0 0 1)
(30 rows)
```

This example uses a Triangle geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_DumpPoints('TRIANGLE ((
                  0 0,
                  0 9,
                  9 0,
                  0 0
              ))' ) AS gdump
    ) AS g;
```

Figure



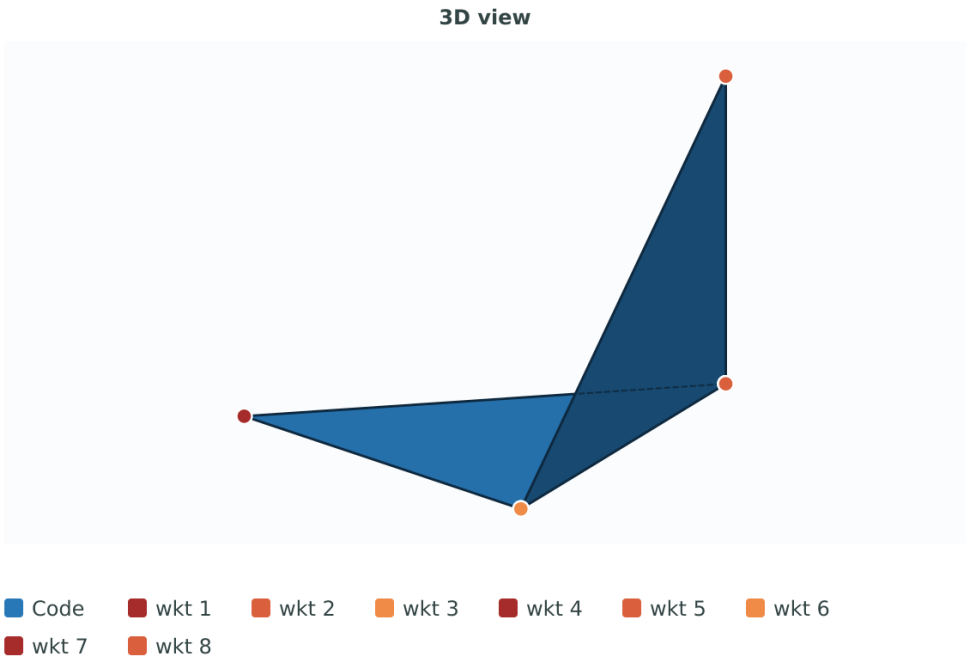
This example uses a TIN geometry.

Code

```
SELECT (g.gdump).path,(g.gdump).geom as wkt
  FROM
    (SELECT
       ST_DumpPoints('TIN (((
                  0 0 0,
                  0 0 1,
                  0 1 0,
                  0 0 0
              )),((
                  0 0 0,
                  0 1 0,
                  1 1 0,
                  0 0 0
```

```
            ))
            )' ) AS gdump
    ) AS g;
```

Figure



Voir aussi

geometry_dump, Section 13.6, ST_Dump, ST_DumpRings, ST_Points

7.4.8 ST_DumpSegments

ST_DumpSegments — Renvoie un ensemble de lignes `geometry_dump` pour les segments d'une géométrie.

Synopsis

geometry_dump[] **ST_DumpSegments**(geometry geom);

Description

Une fonction de retour d'ensemble (SRF) qui extrait les segments d'une géométrie. Elle renvoie un ensemble de geometry_dump lignes, chacune contenant une géométrie (champ *`geom`*) et un tableau d'entiers (champ *`path`*).

— le champ *`geom`* `LINESTRING`s représente les segments linéaires de la géométrie fournie, tandis que `CIRCULARSTRING`s représente les segments d'arc.

— le champ *`path`* (un `integer[]`) est un index énumérant les positions des points de départ des segments dans les éléments de la géométrie fournie. Les indices sont basés sur 1. Par exemple, pour un `LINESTRING`, les chemins sont `{i}` où `i` est le `nième` point de départ du segment dans le `LINESTRING`. Pour un `POLYGONE`, les chemins sont `{i, j}` où `i` est le numéro de l'anneau (1 est l'anneau extérieur ; les anneaux intérieurs suivent) et `j` est la position du point de départ du segment dans l'anneau.

Disponibilité : 3.2.0

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

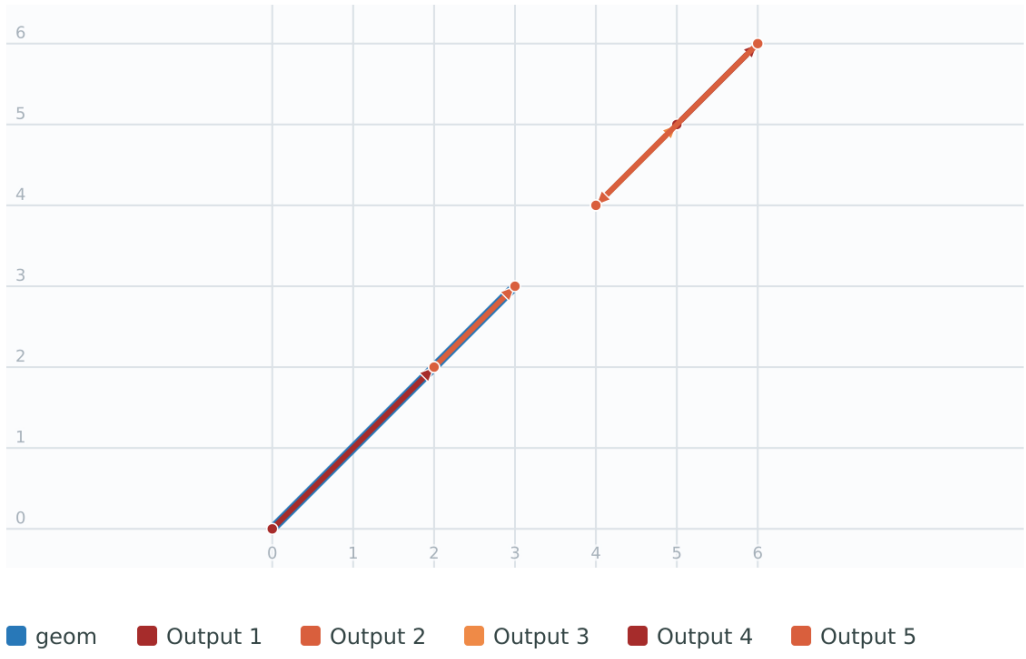
Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples de géométrie standard

Code

```
SELECT path,geom
FROM (
    SELECT (ST_DumpSegments(g.geom)).*
    FROM (SELECT 'GEOMETRYCOLLECTION(
    LINESTRING(1 1,3 3,4 4),
    POLYGON((5 5,6 6,7 7,5 5))
)'::geometry AS geom
        ) AS g
) j;
```

Figure



Exemples de TIN et de triangles

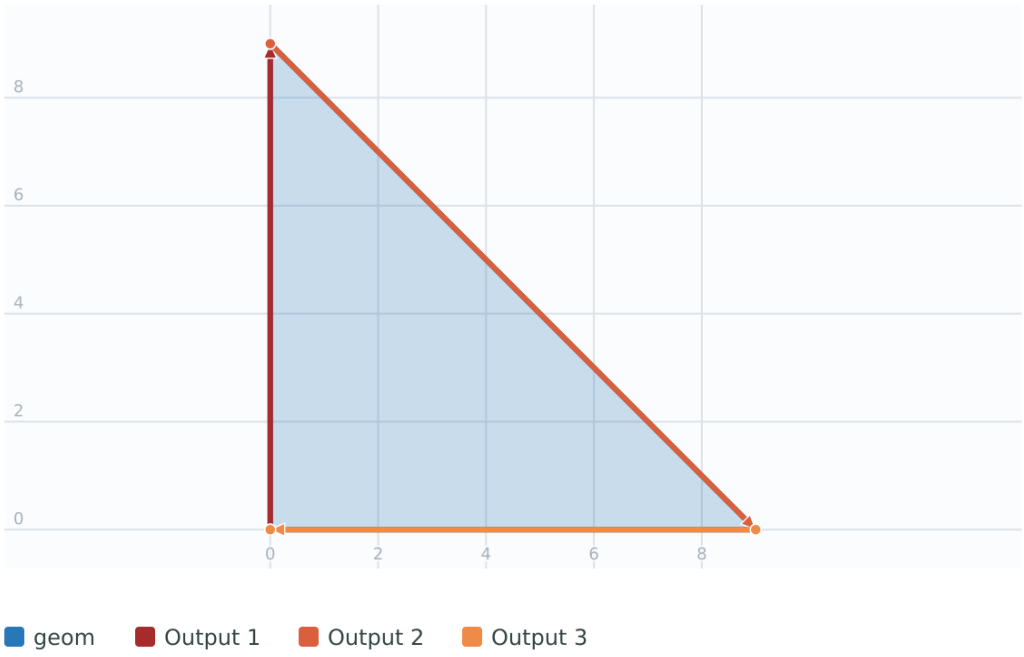
This example uses a Triangle geometry.

Code

```
SELECT path,geom
FROM (
    SELECT (ST_DumpSegments(g.geom)).*
    FROM (SELECT 'TRIANGLE((
        0 0,
        0 9,
        9 0,
```

```
        0 0
    ))'::geometry AS geom
        ) AS g
) j;
```

Figure

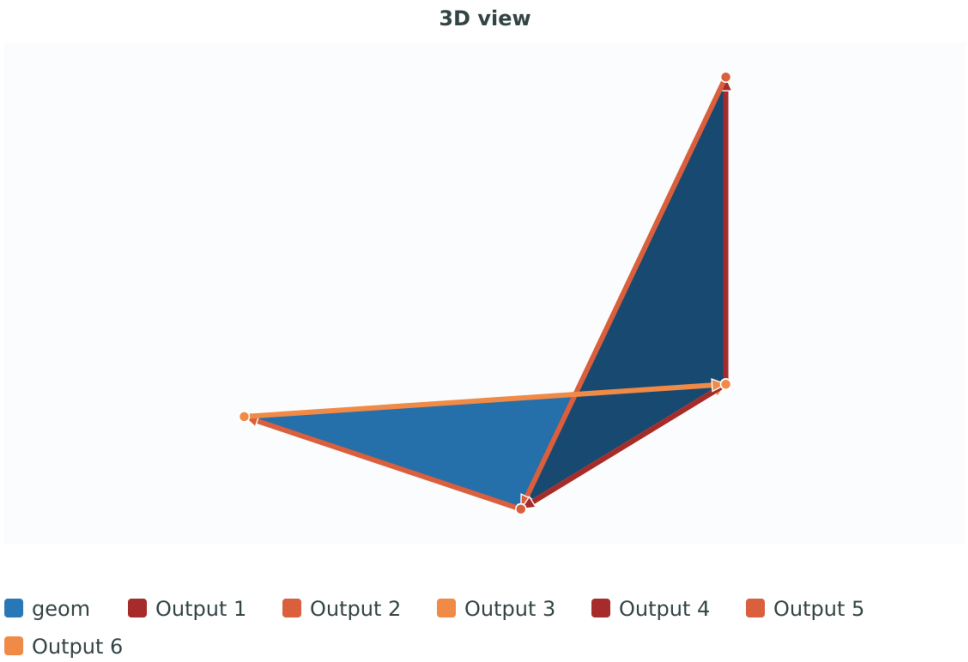


This example uses a TIN geometry.

Code

```
SELECT path,geom
FROM (
    SELECT (ST_DumpSegments(g.geom)).*
    FROM (SELECT 'TIN(((
        0 0 0,
        0 0 1,
        0 1 0,
        0 0 0
    )),((
        0 0 0,
        0 1 0,
        1 1 0,
        0 0 0
    ))
    )'::geometry AS geom
        ) AS g
) j;
```

Figure



Voir aussi

geometry_dump, Section 13.6, ST_Dump, ST_DumpRings

7.4.9 ST_DumpRings

ST_DumpRings — Renvoie un ensemble de lignes `geometry_dump` pour les anneaux extérieurs et intérieurs d'un polygone.

Synopsis

geometry_dump[] **ST_DumpRings**(geometry a_polygon);

Description

Une fonction de retour d'ensemble (SRF) qui extrait les anneaux d'un polygone. Elle renvoie un ensemble de geometry_dump lignes, chacune contenant une géométrie (champ *`geom`*) et un tableau d'entiers (champ *`path`*).

Le champ *`geom`* contient chaque anneau sous forme de POLYGONE. Le champ *`path`* est un tableau d'entiers de longueur 1 contenant l'indice de l'anneau du polygone. L'anneau extérieur (coquille) a l'indice 0. Les anneaux intérieurs (trous) ont des indices de 1 et plus.



Note
This only works for POLYGON geometries, not MULTIPOLYGONs. Use ST_Dump to extract polygon elements from polygonal geometries: `ST_DumpRings((ST_Dump(geom)).geom )`

Disponibilité : PostGIS 1.1.3. Nécessite PostgreSQL 7.3 ou plus.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Extracting all rings as polygons.

Code

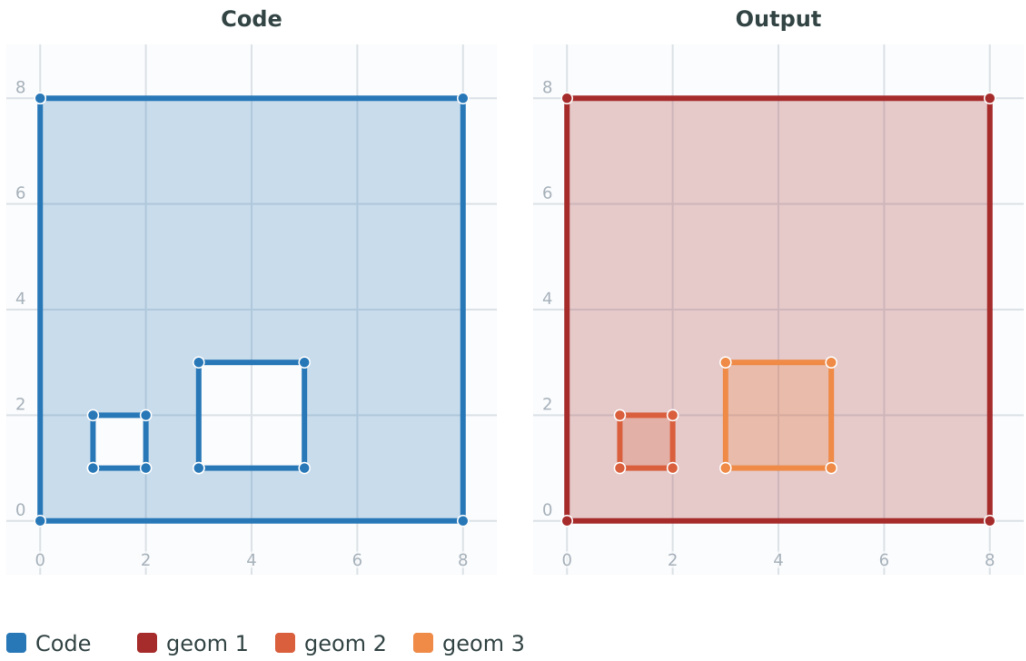
```
SELECT polyTable.field1, polyTable.field1,
      (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable
```

Extracting shell and holes from a polygon.

Code

```
SELECT path,geom As geom
FROM ST_DumpRings('POLYGON ((1 9,9 9,9 1,1 1,1 9),(2 2,2 3,3 3,3 2,2 2),(4 2,4 4,6 4,6 2,4 ↩
    2))');
```

Figure



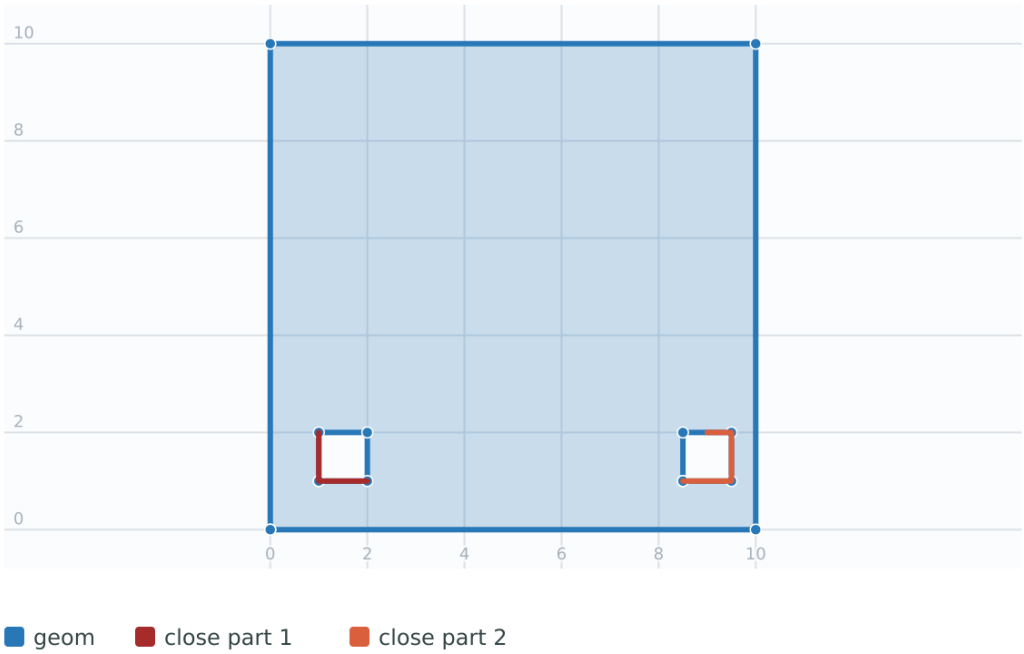
Finding interior rings which lie close to the exterior ring.

Code

```
WITH poly AS (
  SELECT 'POLYGON ((
      0 0, 10 0, 10 10, 0 10, 0 0
    ), (
      1 1, 1 2, 2 2, 2 1, 1 1
    ), (
      8.5 1, 8.5 2, 9.5 2, 9.5 1, 8.5 1
    ))'::geometry AS geom
),
rings AS (
  SELECT d.path[1] AS ring_no,
         ST_Boundary(d.geom) AS ring_geom
  FROM poly
  CROSS JOIN LATERAL ST_DumpRings(poly.geom) AS d
```

```
),
shell AS (
  SELECT ring_geom
  FROM rings
  WHERE ring_no = 0
),
holes AS (
  SELECT ring_no, ring_geom
  FROM rings
  WHERE ring_no > 0
)
SELECT h.ring_no,
       ST_Normalize(ST_Multi(ST_LineMerge(
         ST_Intersection(ST_Buffer(s.ring_geom, 1.0), h.ring_geom)
       ))) AS close_part
FROM shell AS s
JOIN holes AS h
  ON ST_DWithin(s.ring_geom, h.ring_geom, 1.0);
```

Figure



Voir aussi

geometry_dump, Section 13.6, ST_Boundary, ST_Buffer, ST_Dump, ST_DWithin, ST_ExteriorRing, ST_InteriorRingN, ST_Intersectio

7.4.10 ST_EndPoint

ST_EndPoint — Returns the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_EndPoint**(geometry g);

Description

Returns the last point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry as a POINT. Returns NULL if the input is not a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.1.4

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Modifié : 2.0.0 ne fonctionne plus avec les MultiLineStrings à géométrie unique. Dans les anciennes versions de PostGIS, une MultiLineString à géométrie unique fonctionnait avec cette fonction et renvoyait le point final. Dans la version 2.0.0, elle renvoie NULL comme toute autre MultiLineString. L'ancien comportement était une fonctionnalité non documentée, mais les personnes qui supposaient que leurs données étaient stockées en tant que LINESTRING peuvent voir ces dernières retourner NULL dans la version 2.0.0.

Exemples

Point final d'une ligne

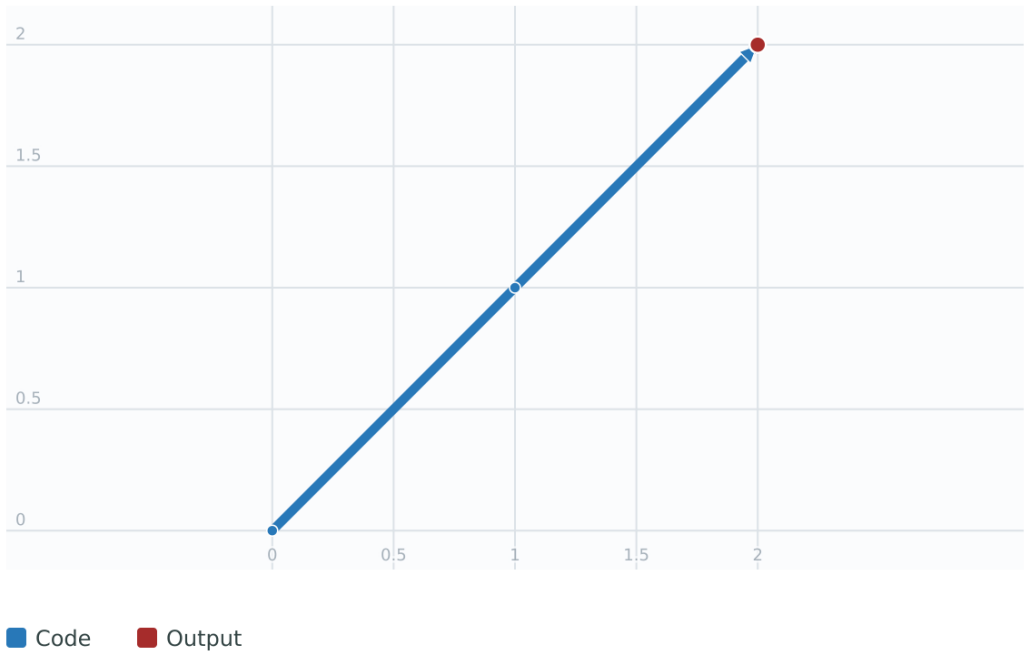
Code

```
SELECT ST_EndPoint('LINESTRING(1 1,2 2,3 3)'::geometry);
```

Output

```
POINT(3 3)
```

Figure



Le point final d'une ligne qui n'en est pas une est NULL

Code

```
SELECT ST_EndPoint('POINT(1 1)'::geometry) IS NULL AS is_null;
```

Output

```
t
```

Point final d'une ligne 3D

3D endpoint.

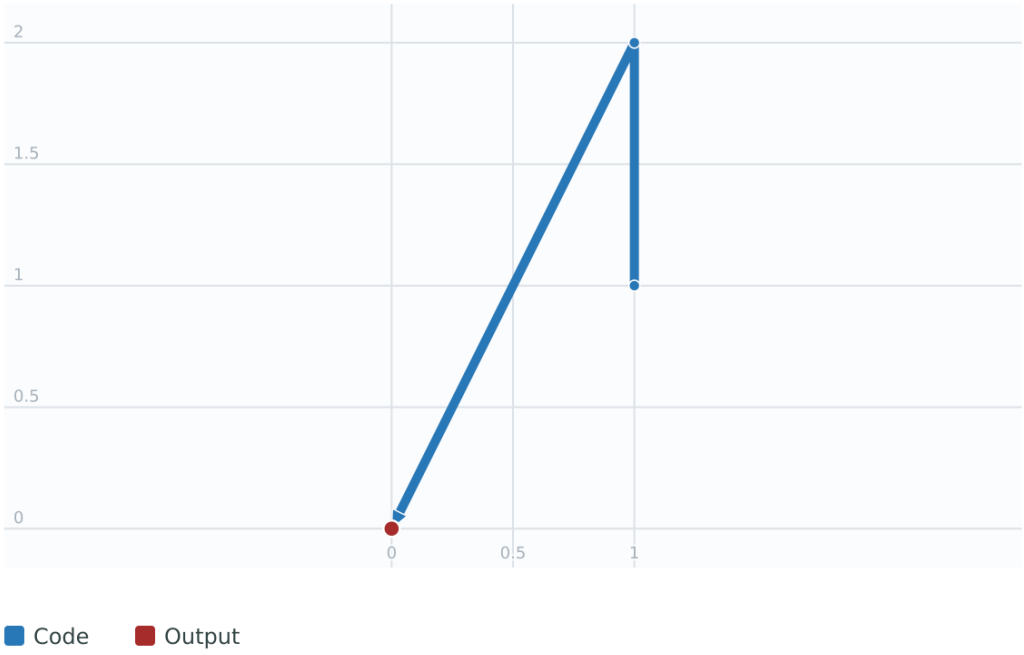
Code

```
SELECT ST_EndPoint('LINESTRING(1 1 2,1 2 3,0 0 5)');
```

Output

```
POINT(0 0 5)
```

Figure



Point d'arrivée d'une CircularString

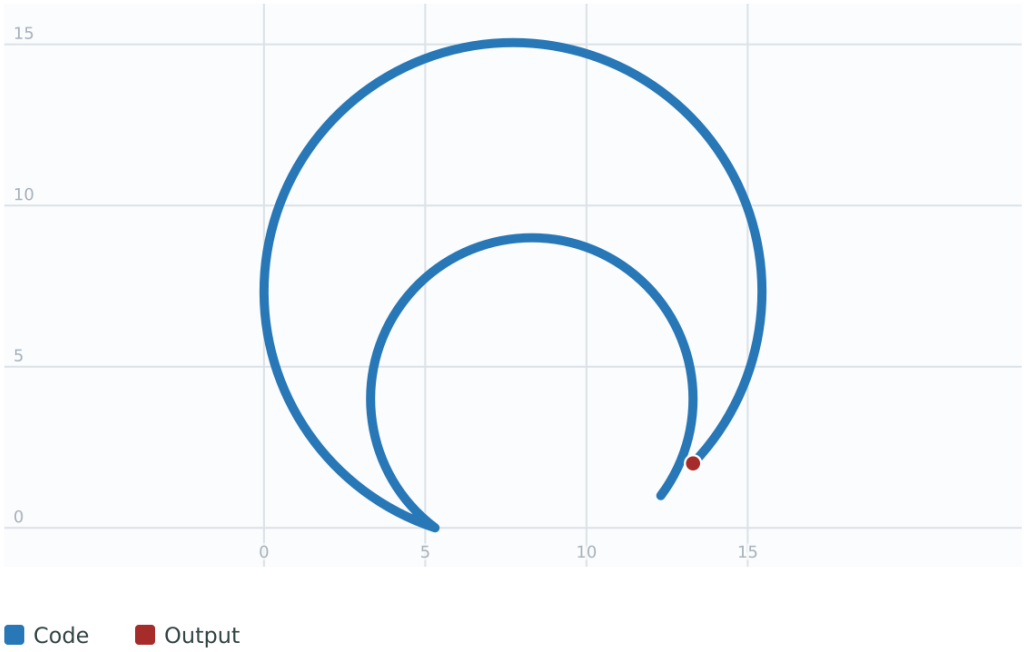
Code

```
SELECT ST_EndPoint('CIRCULARSTRING(5 2,-3 2,-2 1,-4 2,6 3)'::geometry);
```

Output

```
POINT(6 3)
```

Figure



End point of a NURBSCurve

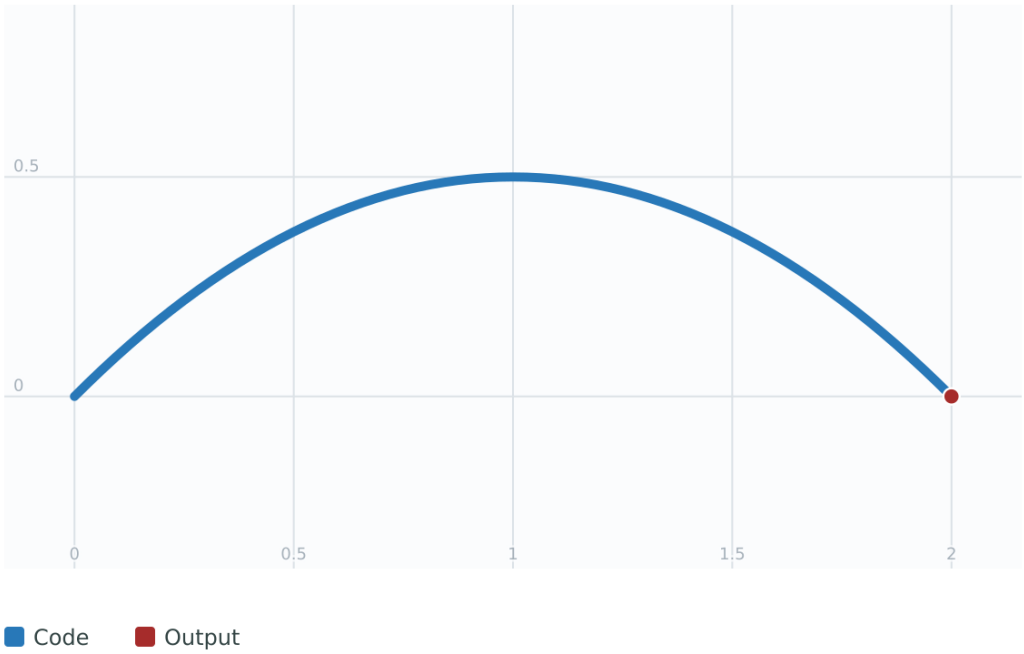
Code

```
SELECT ST_EndPoint('NURBSCURVE(2, (0 0, 1 1, 2 0))'::geometry);
```

Output

```
POINT(2 0)
```

Figure



Voir aussi

ST_PointN, ST_StartPoint

7.4.11 ST_Envelope

ST_Envelope — Renvoie une géométrie représentant la boîte de délimitation d'une géométrie.

Synopsis

geometry **ST_Envelope**(geometry g1);

Description

Renvoie la boîte de délimitation minimale en double précision (float8) pour la géométrie fournie, en tant que géométrie. Le polygone est défini par les points d'angle de la boîte de délimitation ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`)). (PostGIS ajoutera également une coordonnée `ZMIN`/`ZMAX`).

D'autres cas (lignes verticales, points) retourneront une géométrie de dimension inférieure à `POLYGON`, c'est-à-dire `POINT` ou `LINESTRING`.

Disponibilité : 1.5.0 changement pour renvoyer un type double précision à la place de float4

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.19

Exemples

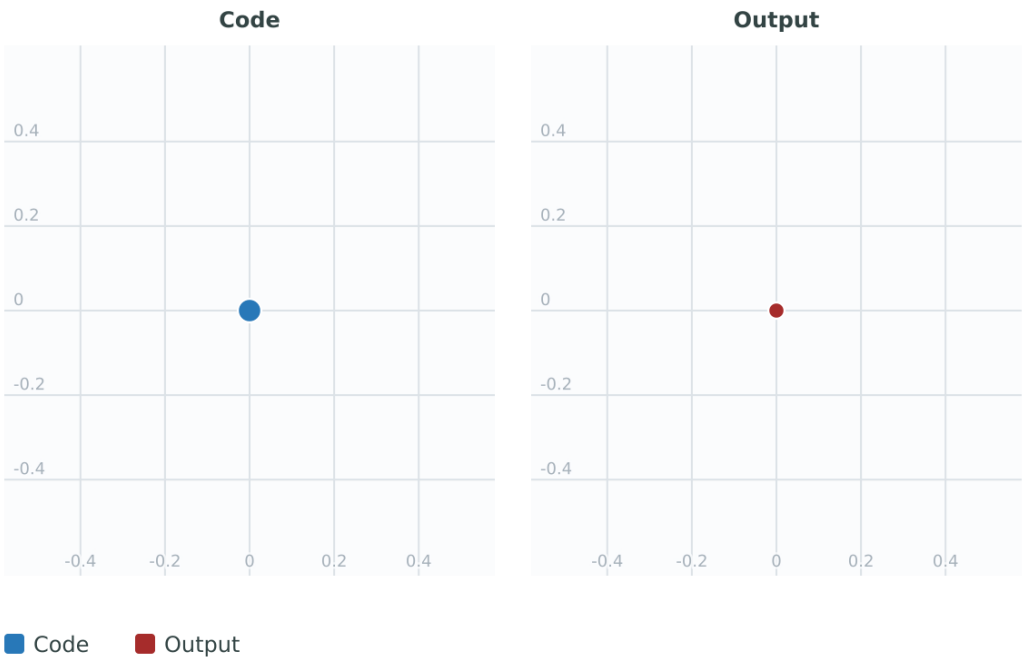
Code

```
SELECT ST_Envelope('POINT(1 3)'::geometry);
```

Output

```
POINT(1 3)
```

Figure



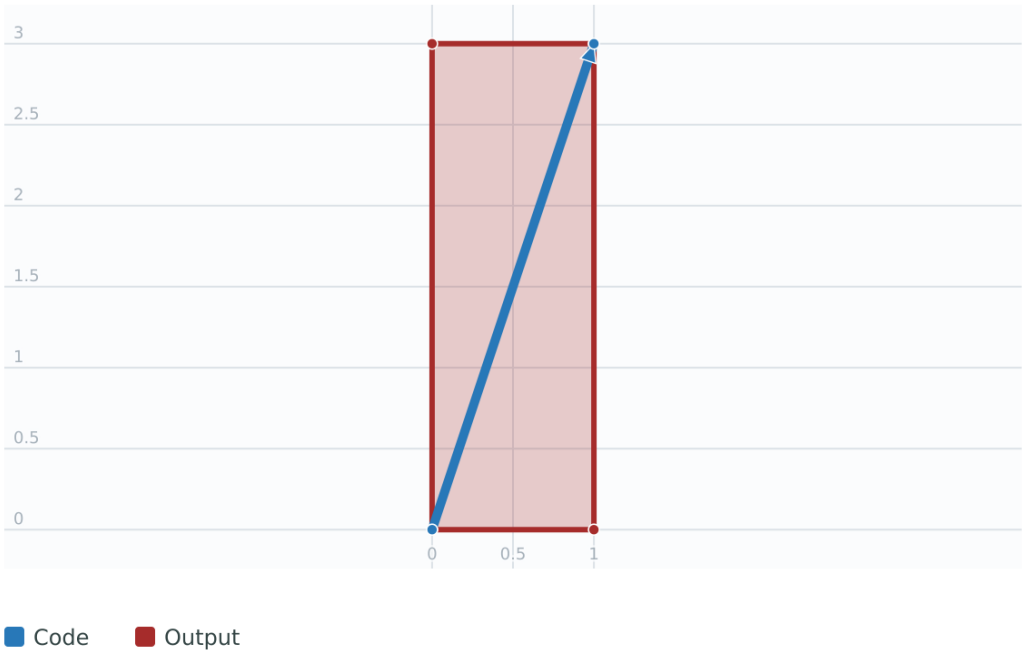
Code

```
SELECT ST_Envelope('LINESTRING(0 0,1 3)'::geometry);
```

Output

```
POLYGON((0 0,0 3,1 3,1 0,0 0))
```

Figure



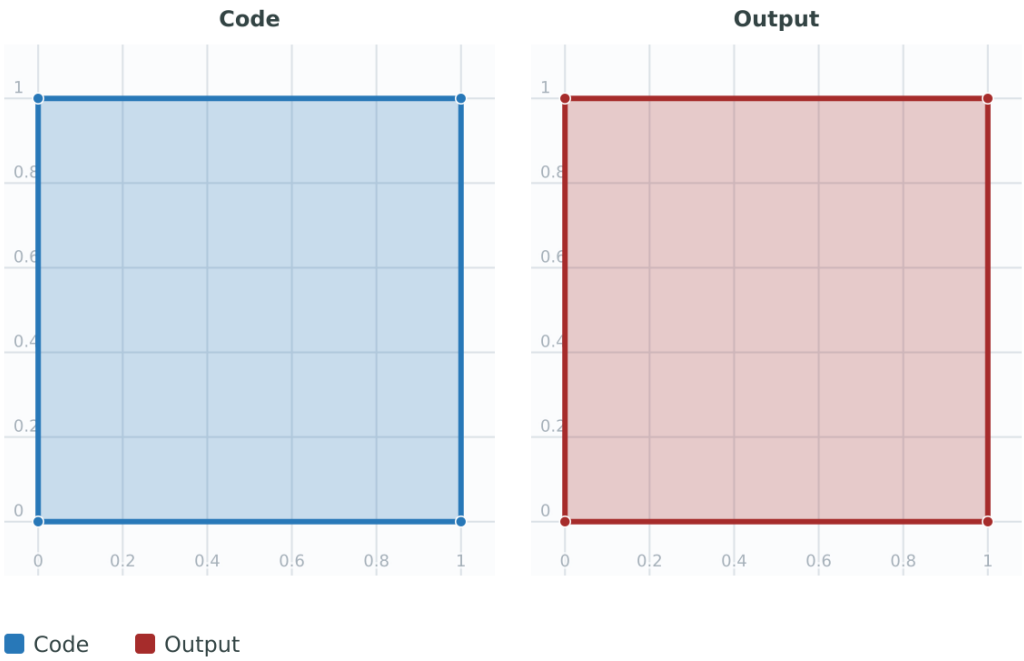
Code

```
SELECT ST_Envelope('POLYGON((0 0,0 1,1.0000001 1,1.0000001 0,0 0))'::geometry);
```

Output

```
POLYGON((0 0,0 1,1.0000001 1,1.0000001 0,0 0))
```

Figure



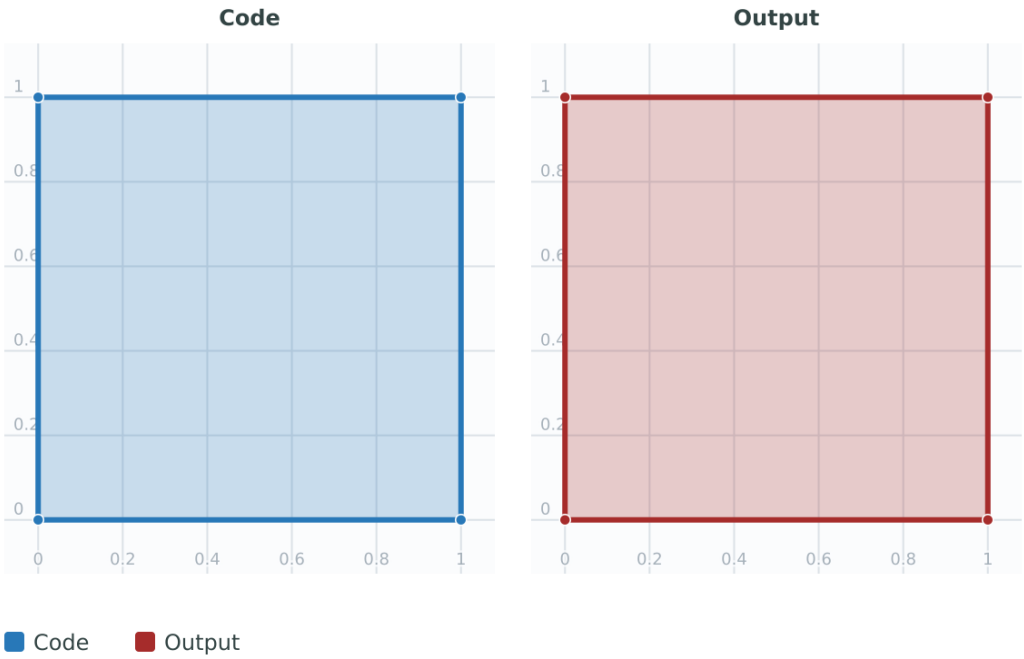
Code

```
SELECT ST_Envelope('POLYGON((0 0,0 1,1.0000000001 1,1.0000000001 0,0 0))'::geometry);
```

Output

```
POLYGON((0 0,0 1,1.0000000001 1,1.0000000001 0,0 0))
```

Figure



Code

```
SELECT Box3D(geom), Box2D(geom), ST_Envelope(geom) As envelopewkt
    FROM (SELECT 'POLYGON((0 0,0 1000012333334.34545678,1.0000001 1,1.0000001 0,0 0))':: ↩
    geometry As geom) As foo;
```

Envelope of a point and LineString.

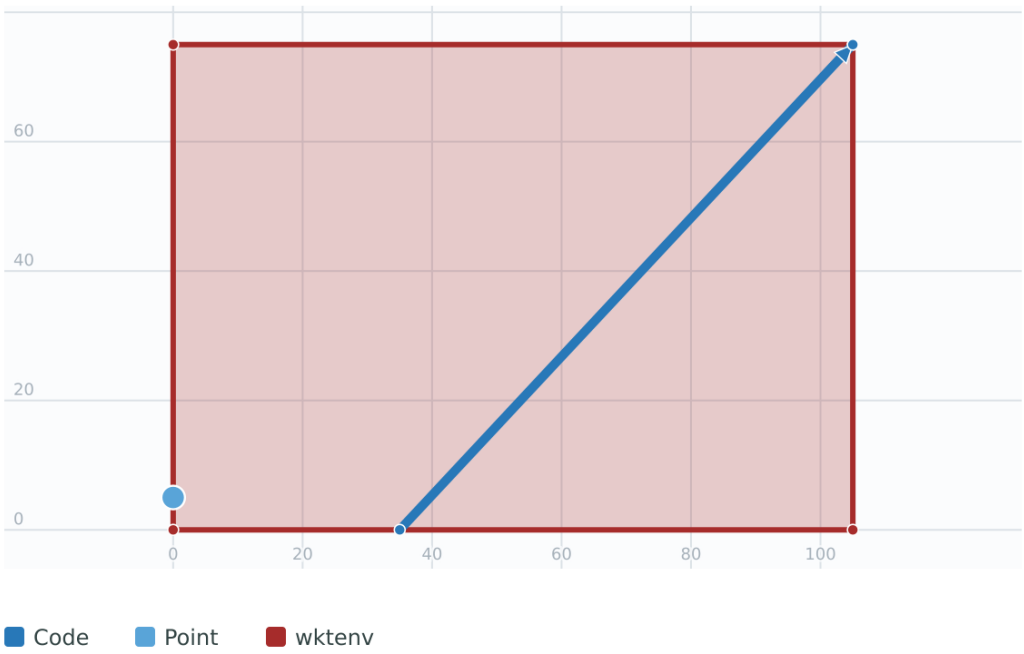
Code

```
SELECT ST_Envelope(ST_Collect(
            'LINESTRING(55 75,125 150)',
            ST_Point(20, 80)
            )) As wktenv;
```

Output

```
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

Figure



Voir aussi

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — Renvoie une ligne représentant l'anneau extérieur d'un polygone.

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

Description

Renvoie une LINESTRING représentant l'anneau extérieur (coquille) d'un POLYGONE. Renvoie NULL si la géométrie n'est pas un polygone.

Note
Cette fonction ne prend pas en charge les MULTIPOLYGONES. Pour les MULTIPOLYGONs, utilisez conjointement avec ST_GeometryN ou ST_Dump

---

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.2.3, 8.3.3

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Extract the exterior ring from a Polygon with a hole.

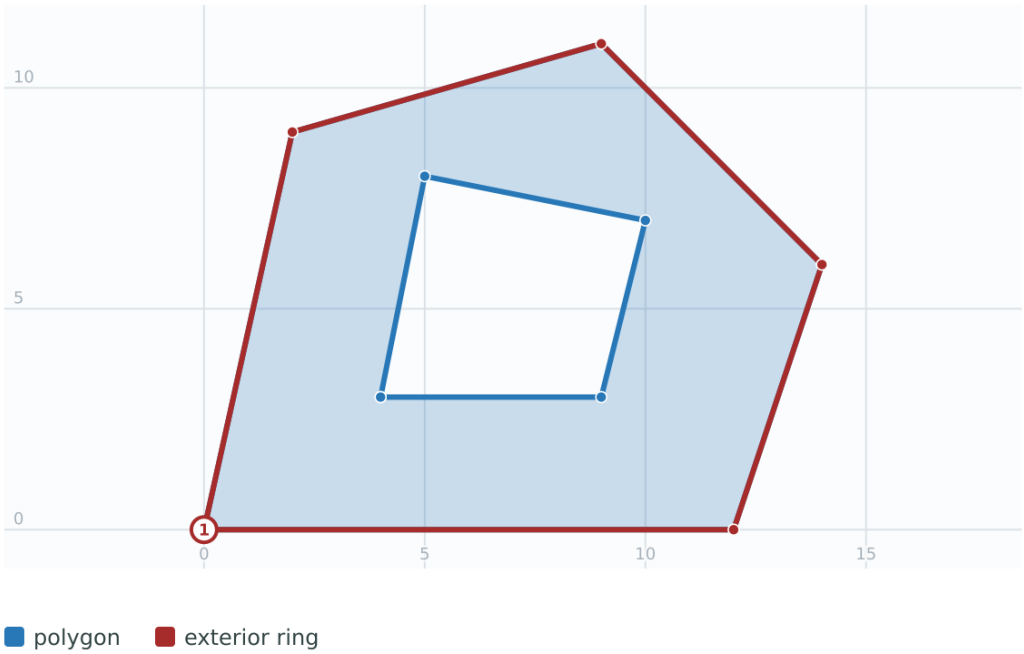
Code

```
WITH input(geom) AS (VALUES (
  'POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3))'::geometry
))
SELECT geom AS input_polygon,
       ST_ExteriorRing(geom) AS exterior_ring
FROM input;
```

Output

```
POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)) | LINESTRING(0 0,2 9,9 11,14 6,12 0,0 0)
```

Figure



For a MultiPolygon, dump the component Polygons before extracting their exterior rings. Both component Polygons in this example have holes.

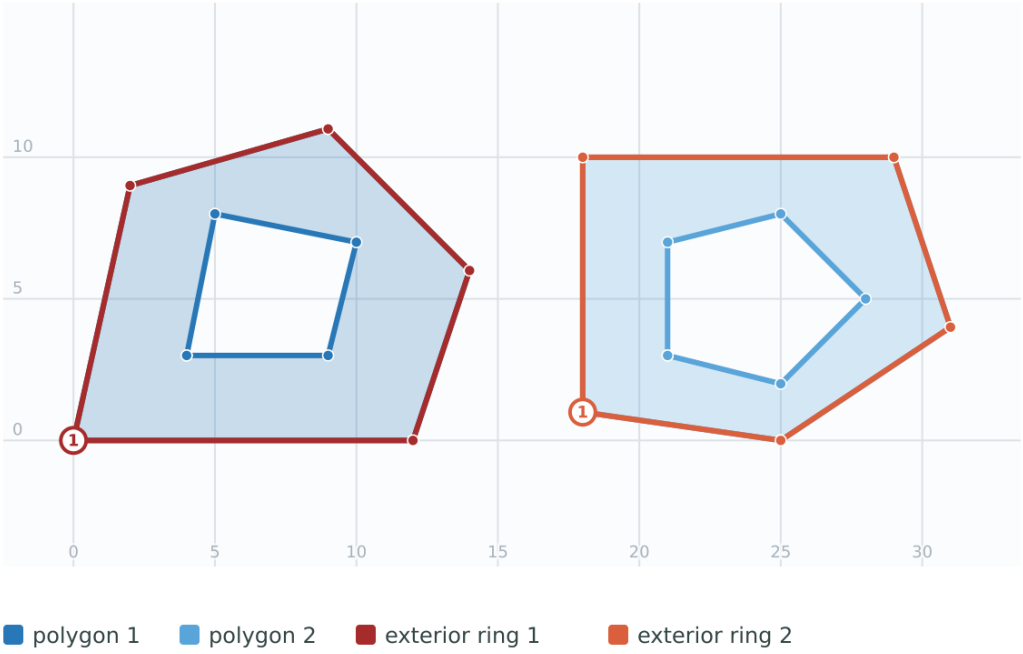
Code

```
WITH input(geom) AS (VALUES (
  'MULTIPOLYGON(
    ((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)),
    ((18 1,18 10,29 10,31 4,25 0,18 1),(21 3,25 2,28 5,25 8,21 7,21 3))
  )'::geometry
))
SELECT part.geom AS input_polygon,
       ST_ExteriorRing(part.geom) AS exterior_ring
FROM input
CROSS JOIN LATERAL ST_Dump(geom) AS part
ORDER BY part.path;
```

Output

```
POLYGON((0 0,2 9,9 11,14 6,12 0,0 0),(4 3,9 3,10 7,5 8,4 3)) | LINESTRING(0 0,2 9,9 11,14 6,12 0,0 0)
POLYGON((18 1,18 10,29 10,31 4,25 0,18 1),(21 3,25 2,28 5,25 8,21 7,21 3)) | LINESTRING(18 1,18 10,29 10,31 4,25 0,18 ↩
    1)
```

Figure



Extract exterior rings from a table of Polygons.

Code

```
SELECT gid, ST_ExteriorRing(geom) AS ering
FROM sometable;
```

For a table of MultiPolygons, dump the component Polygons and collect their exterior rings into a MultiLineString.

Code

```
SELECT gid, ST_Collect(ST_ExteriorRing(geom)) AS erings
FROM (
  SELECT gid, (ST_Dump(geom)).geom AS geom
  FROM sometable
) AS foo
GROUP BY gid;
```

This example returns the 3D exterior ring of a Polygon.

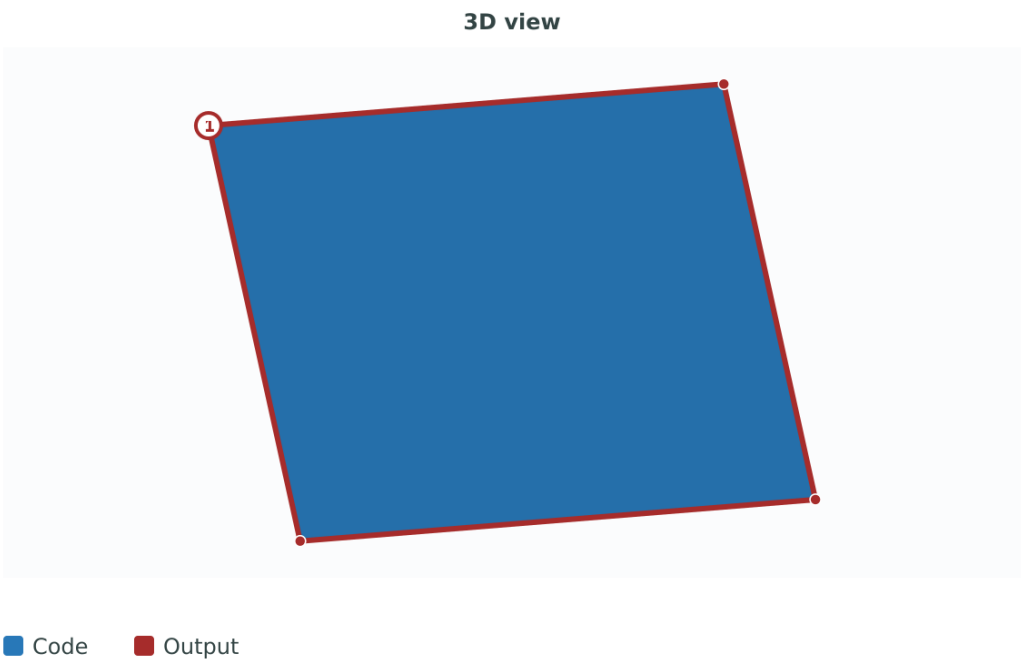
Code

```
SELECT ST_ExteriorRing(
    'POLYGON Z ((0 0 5,4 0 5,4 3 5,0 3 5,0 0 5))'
    );
```

Output

```
LINESTRING(0 0 5,4 0 5,4 3 5,0 3 5,0 0 5)
```

Figure



Voir aussi

ST_InteriorRingN, ST_Boundary, ST_NumInteriorRings

7.4.13 ST_GeometryN

ST_GeometryN — Renvoie un élément d'une collection de géométries.

Synopsis

geometry **ST_GeometryN**(geometry geomA, integer n);

Description

Returns the 1-based Nth element of a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE, or MULTIPOLYGON. For a unitary geometry, including a TIN or PolyhedralSurface, an index of 1 returns the input geometry and any other index returns NULL. Use ST_PatchN and ST_NumPatches to access and count the faces of a TIN or PolyhedralSurface.



Note
L'index commence à 1 pour respecter les spécificarions OGC depuis la version 0.8.0. Dans les versions antérieures, l'index commençait à 0.



Note
Pour extraire tous les éléments d'une géométrie, ST_Dump est plus efficace et fonctionne pour les géométries atomiques.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Changement : 2.0.0. Les versions antérieures renvoient NULL pour les géometries simples (un seul objet). Renvoie désormais la géométrie pour le cas ST_GeometryN(..,1).

Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and are not decomposed into patches; use ST_PatchN to access faces.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 9.1.5

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

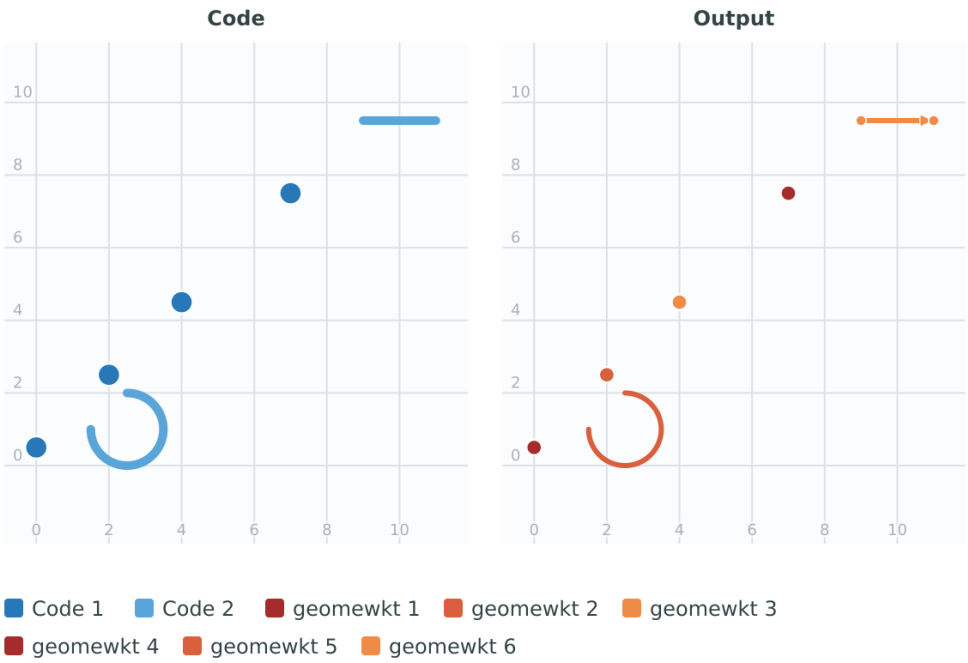
Exemples standard

Extracting a subset of points from a 3D multipoint.

Code

```
SELECT n, ST_GeometryN(geom, n) As geomewkt
FROM (
VALUES
    (1, 'MULTIPOINT((1 2 7), (3 4 7), (5 6 7), (8 9 10))'::geometry ),
    (2, 'MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))'::geometry )
    ) As foo(id, geom)
    CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom)
ORDER BY id, n;
```

Figure



Extracting all geometries (useful when you want to assign an id).

Code

```
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

Exemples TIN, Triangle et Surfaces Polyédriques

A PolyhedralSurface or TIN is treated as a single geometry, not as a collection of faces. `ST_GeometryN` does not decompose the surface into patches, so an index greater than 1 returns `NULL`. Use ST_PatchN and ST_NumPatches to access individual faces.

This example requests the third element of a PolyhedralSurface.

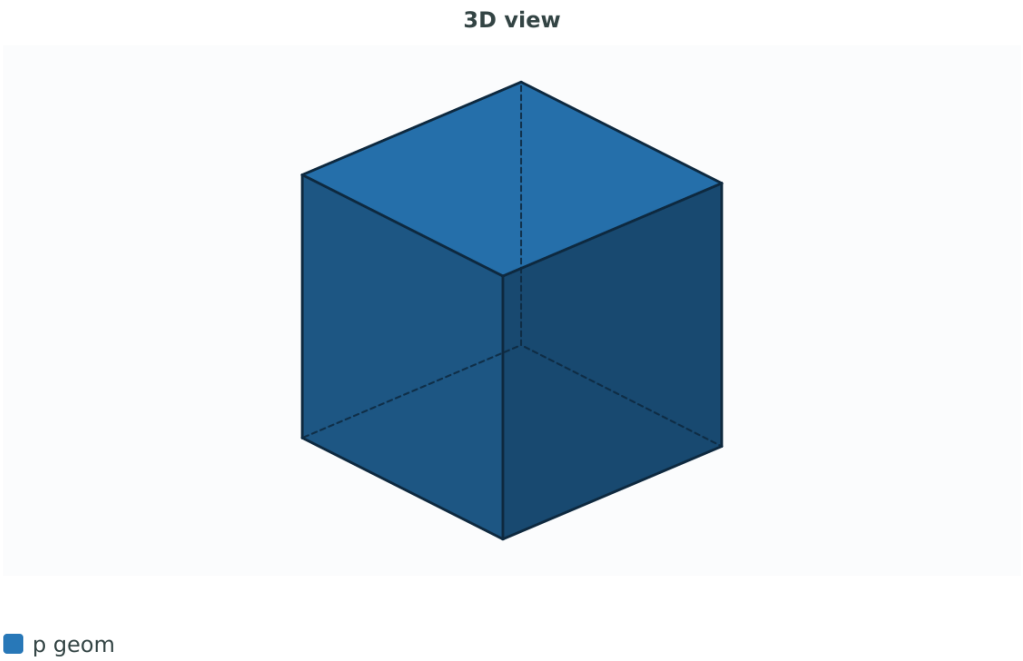
Code

```
SELECT ST_GeometryN(p_geom, 3) As geom_ewkt
  FROM (SELECT 'POLYHEDRALSURFACE(
((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)'::geometry  AS p_geom )  AS a;
```

Output

```
NULL
```

Figure



This example requests the second element of a TIN.

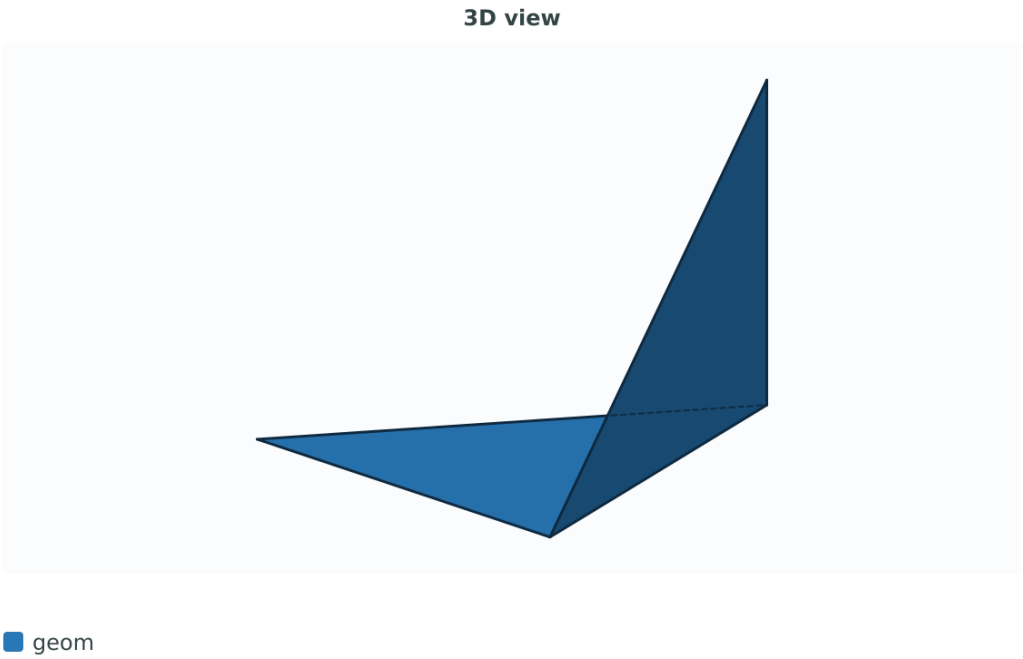
Code

```
SELECT ST_GeometryN(geom, 2) as wkt
  FROM
    (SELECT
       'TIN (((
                0 0 0,
                0 0 1,
                0 1 0,
                0 0 0
            )),((
                0 0 0,
                0 1 0,
                1 1 0,
                0 0 0
            ))
            )'::geometry  AS geom
    ) AS g;
```

Output

```
NULL
```

Figure



Voir aussi

ST_Dump, ST_NumGeometries, ST_PatchN, ST_NumPatches

7.4.14 ST_GeometryType

ST_GeometryType — Renvoie le type SQL-MM d'une géométrie sous forme de texte.

Synopsis

text **ST_GeometryType**(geometry g1);

Description

Renvoie le type de la géométrie sous forme de texte, par exemple : 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. Cette fonction diffère de GeometryType(geometry) par la casse du texte renvoyé et par le préfixe ST_ en début de texte. N'indique pas si la géométrie comporte une dimension MESURE.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.4

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_GeometryType('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)':: ↩
    geometry);
```

Output

```
ST_LineString
```

Code

```
SELECT ST_GeometryType('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry);
```

Output

```
ST_PolyhedralSurface
```

Code

```
SELECT ST_GeometryType(geom) AS result
FROM
  (SELECT
     'TIN (((
               0 0 0,
               0 0 1,
               0 1 0,
               0 0 0
           )),((
               0 0 0,
               0 1 0,
               1 1 0,
               0 0 0
           ))
           )'::geometry AS geom
  ) AS g;
```

Output

```
ST_Tin
```

Voir aussi

GeometryType

7.4.15 ST_HasArc

ST_HasArc — Teste si une géométrie contient un arc de cercle

Synopsis

boolean **ST_HasArc**(geometry geomA);

Description

Renvoie true si une géométrie ou une collection de géométries contient une circular string

Disponibilité : 1.2.3 ?

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

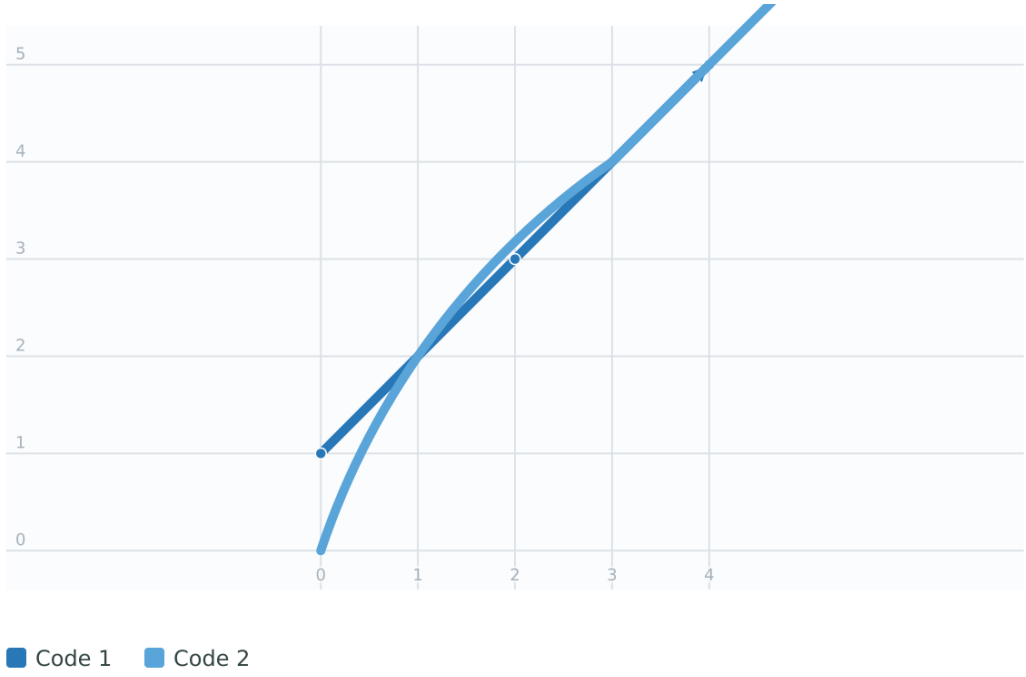
Code

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2,3 4,5 6)', 'CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6) ↩
    '));
```

Output

```
t
```

Figure



Voir aussi

ST_CurveToLine, ST_LineToCurve

7.4.16 ST_InteriorRingN

ST_InteriorRingN — Renvoie le Nième anneau intérieur (trou) d'un polygone.

Synopsis

geometry **ST_InteriorRingN**(geometry a_polygon, integer n);

Description

Renvoie le Nième anneau intérieur (trou) d'une géométrie POLYGONE sous forme de LINESTRING. L'indice commence à 1. Renvoie NULL si la géométrie n'est pas un polygone ou si l'indice est hors de la plage.



Note
Cette fonction ne prend pas en charge les MULTIPOLYGONES. Pour les MULTIPOLYGONs, utilisez conjointement avec ST_GeometryN ou ST_Dump

---

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.2.6, 8.3.5

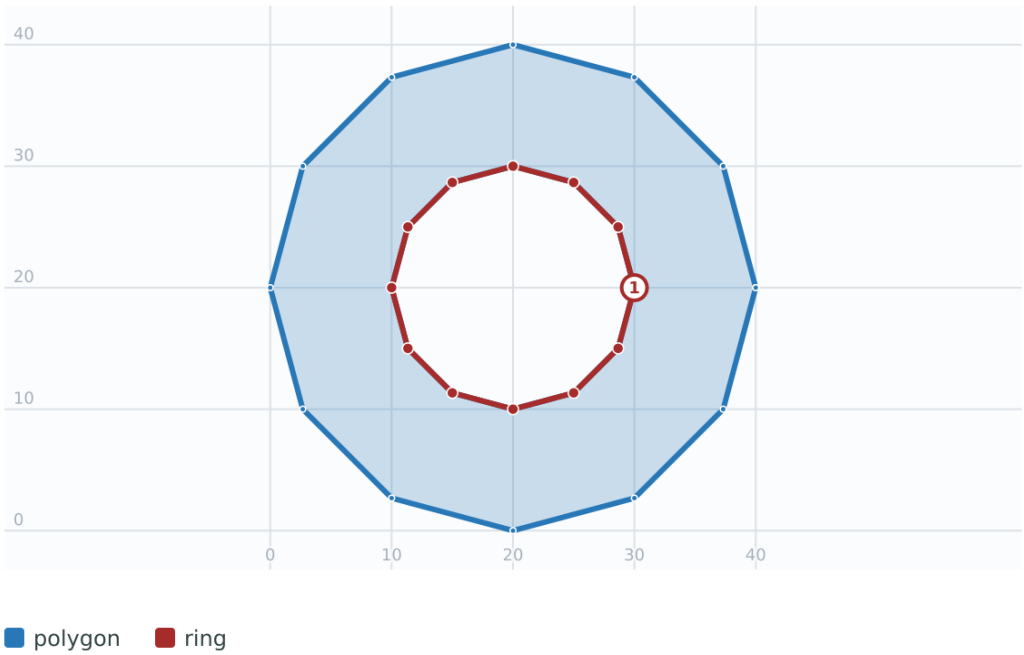
 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
WITH data AS (
  SELECT ST_BuildArea(ST_Collect(
    ST_Buffer(ST_Point(1, 2), 20, 3),
    ST_Buffer(ST_Point(1, 2), 10, 3)
  )) AS geom
)
SELECT geom AS input_polygon,
       ST_InteriorRingN(geom, 1) AS ring
FROM data;
```

Figure



Voir aussi

ST_ExteriorRing, ST_BuildArea, ST_Collect, ST_Dump, ST_NumInteriorRing, ST_NumInteriorRings

7.4.17 ST_NumCurves

ST_NumCurves — Renvoie le nombre de courbes composantes d'une CompoundCurve.

Synopsis

integer **ST_NumCurves**(geometry a_compoundcurve);

Description

Renvoie le nombre de courbes composantes d'une CompoundCurve, zéro pour une CompoundCurve vide, ou NULL pour une entrée qui n'est pas une CompoundCurve.

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.2.6, 8.3.5

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

This CompoundCurve contains three component curves.

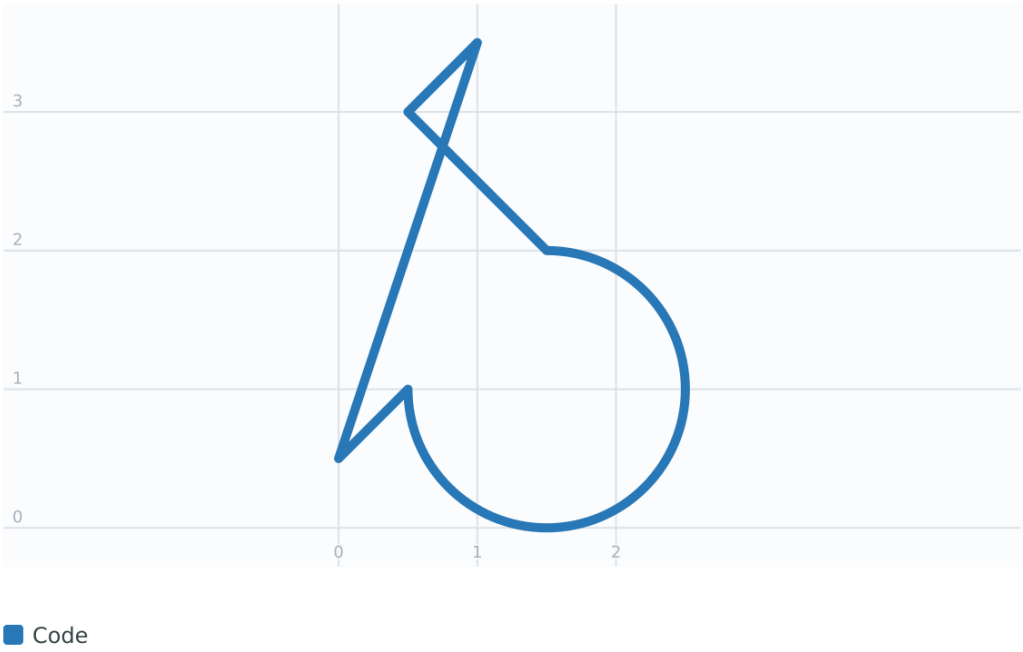
Code

```
SELECT ST_NumCurves('COMPOUNDCURVE(
    (2 2,2.5 2.5),
    CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5),
    (3.5 3.5,2.5 4.5,3 5,2 2)
  )');
```

Output

```
3
```

Figure



An empty CompoundCurve has no component curves.

Code

```
SELECT ST_NumCurves('COMPOUNDCURVE EMPTY');
```

Output

```
0
```

Voir aussi

ST_CurveN, ST_Dump, ST_ExteriorRing, ST_NumInteriorRings, ST_NumGeometries

7.4.18 ST_CurveN

ST_CurveN — Renvoie la Nième courbe composante d'une CompoundCurve.

Synopsis

geometry **ST_CurveN**(geometry a_compoundcurve, integer index);

Description

Renvoie la Nièmecourbe composante d'une CompoundCurve. L'index commence à 1. Renvoie NULL si la géométrie n'est pas une CompoundCurve ou si l'index est hors de la plage.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.2.6, 8.3.5

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

This example extracts the second component curve, a CircularString, from a CompoundCurve.

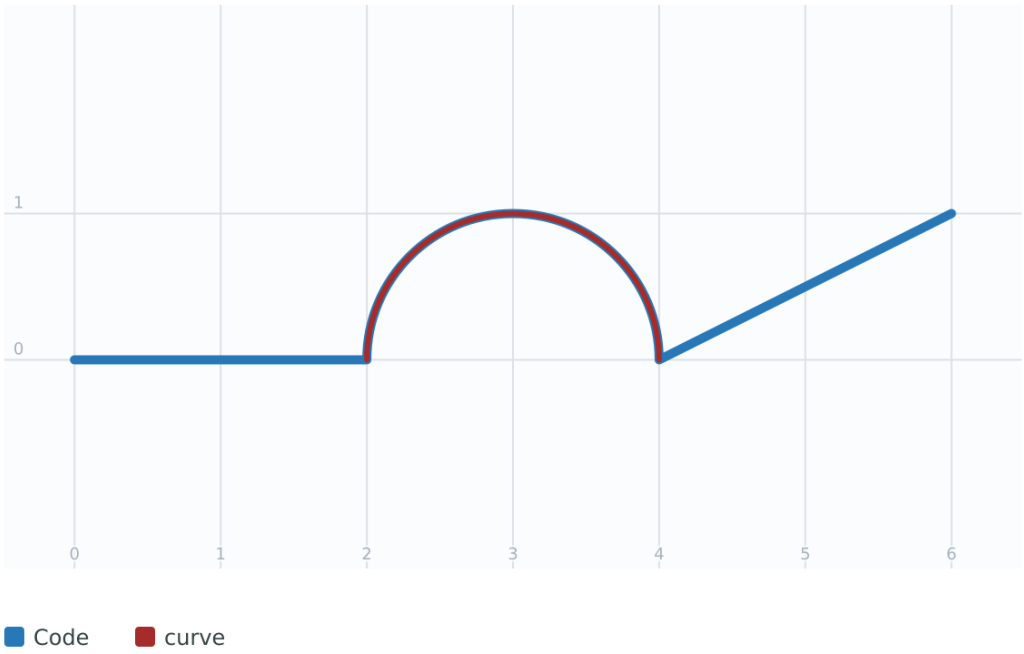
Code

```
SELECT ST_CurveN(
  'COMPOUNDCURVE(
    (0 0,2 0),
    CIRCULARSTRING(2 0,3 1,4 0),
    (4 0,6 1)
  )',
  2) AS curve;
```

Output

```
CIRCULARSTRING(2 0,3 1,4 0)
```

Figure



Voir aussi

ST_NumCurves, ST_Dump, ST_ExteriorRing, ST_NumInteriorRings, ST_NumGeometries

7.4.19 ST_IsClosed

ST_IsClosed — Teste si les points de départ et d'arrivée d'une LineString coïncident. Pour une PolyhedralSurface, teste si elle est fermée (volumétrique).

Synopsis

boolean **ST_IsClosed**(geometry g);

Description

Renvoie `TRUE` si les premier et dernier points de la `LINESTRING` sont identiques. Pour les surface polyhédriques, indique si la surface est surfacique (ouverte) ou volumétrique (fermée).

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.1.5, 9.3.3

Note
SQL-MM définit le résultat de `ST_IsClosed(NULL)` comme étant 0, alors que PostGIS renvoie `NULL`.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples de lignes et de points

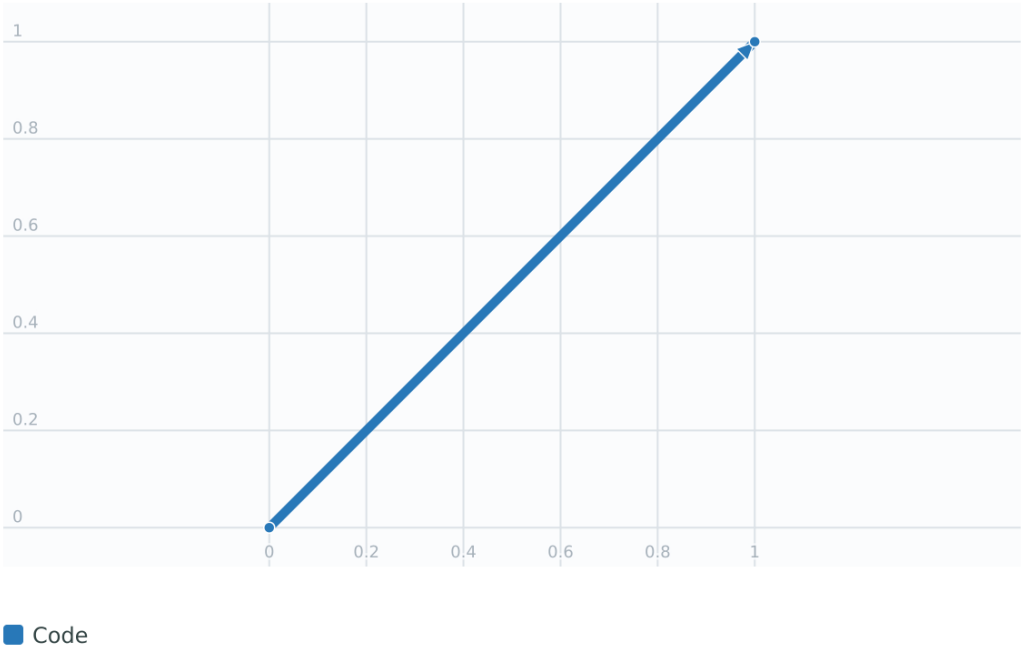
Code

```
SELECT ST_IsClosed('LINESTRING(0 0,1 1)'::geometry);
```

Output

```
f
```

Figure



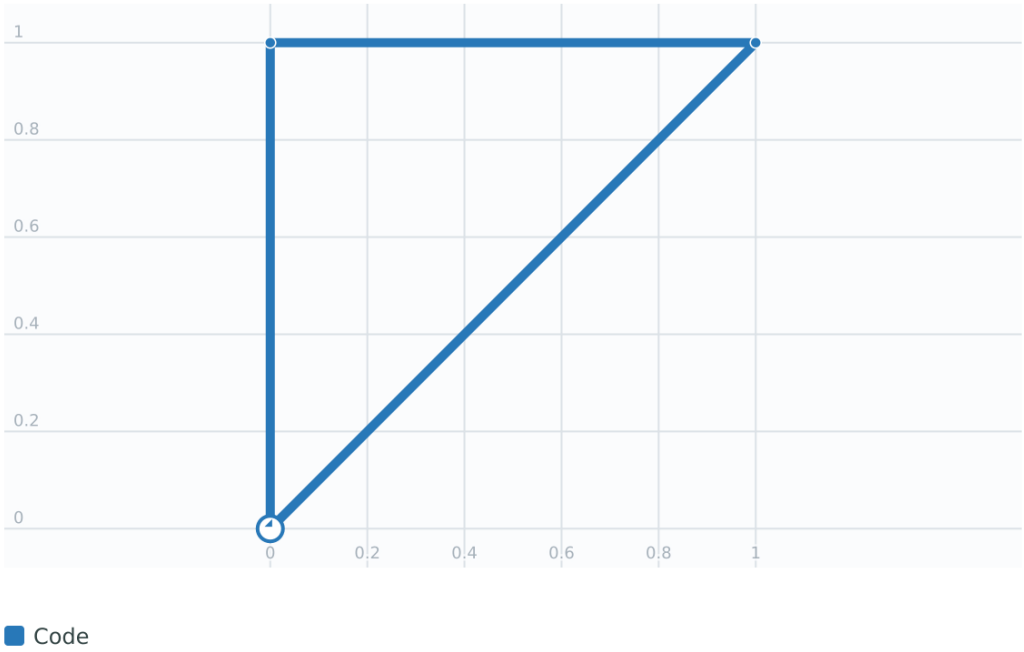
Code

```
SELECT ST_IsClosed('LINESTRING(0 0,0 1,1 1,0 0)'::geometry);
```

Output

```
t
```

Figure



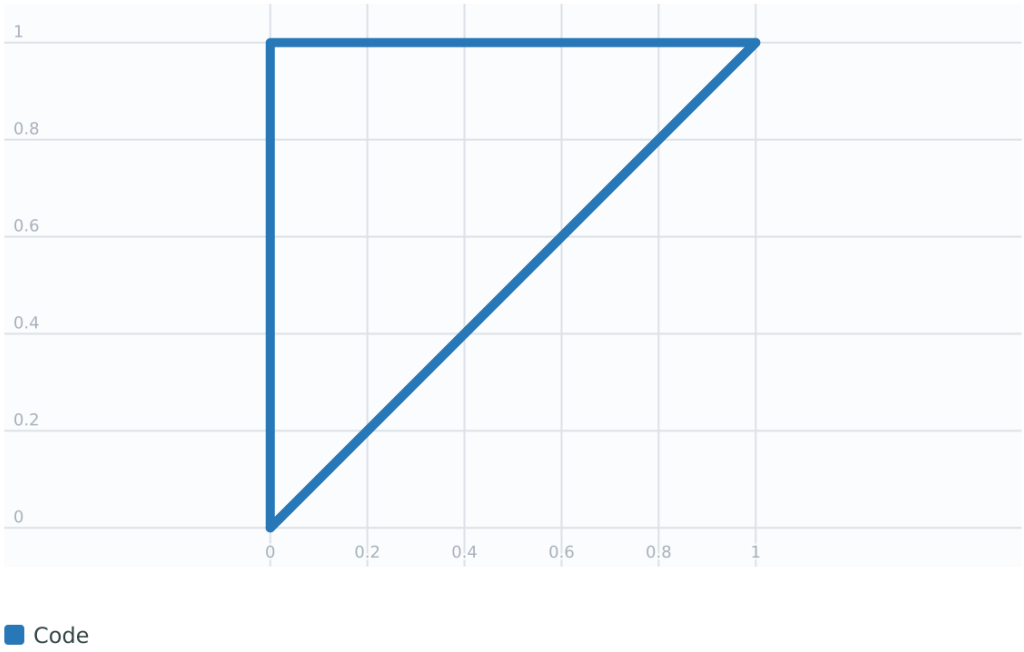
Code

```
SELECT ST_IsClosed('MULTILINESTRING((0 0,0 1,1 1,0 0),(0 0,1 1))'::geometry);
```

Output

```
f
```

Figure



Code

```
SELECT ST_IsClosed('POINT(0 0)'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsClosed('MULTIPOINT((0 0),(1 1))'::geometry);
```

Output

```
t
```

Exemples : surfaces polyédriques

This example uses a cube.

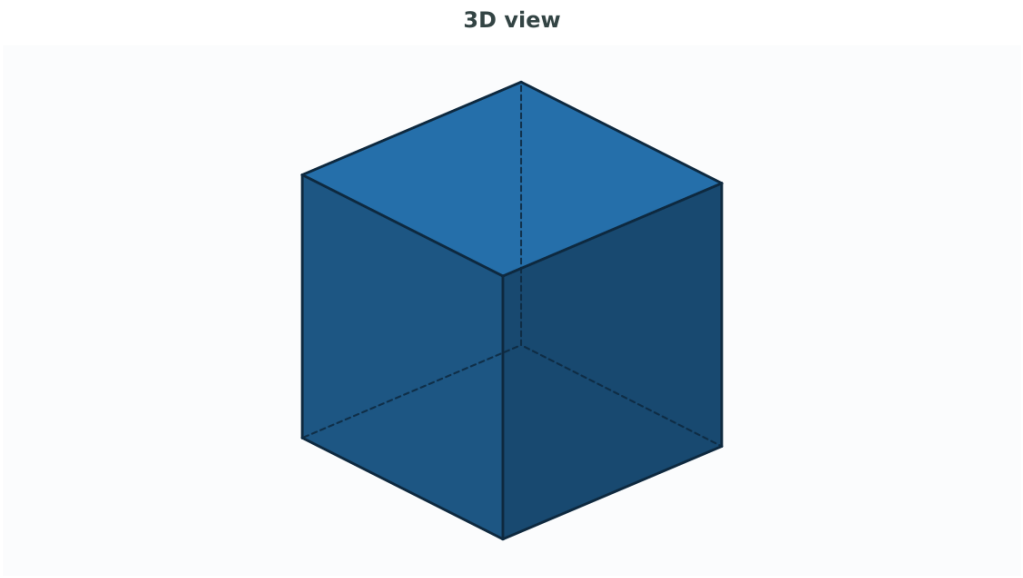
Code

```
SELECT ST_IsClosed('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )');
```

Output

```
t
```

Figure



Code

This example uses the same cube with one side missing.

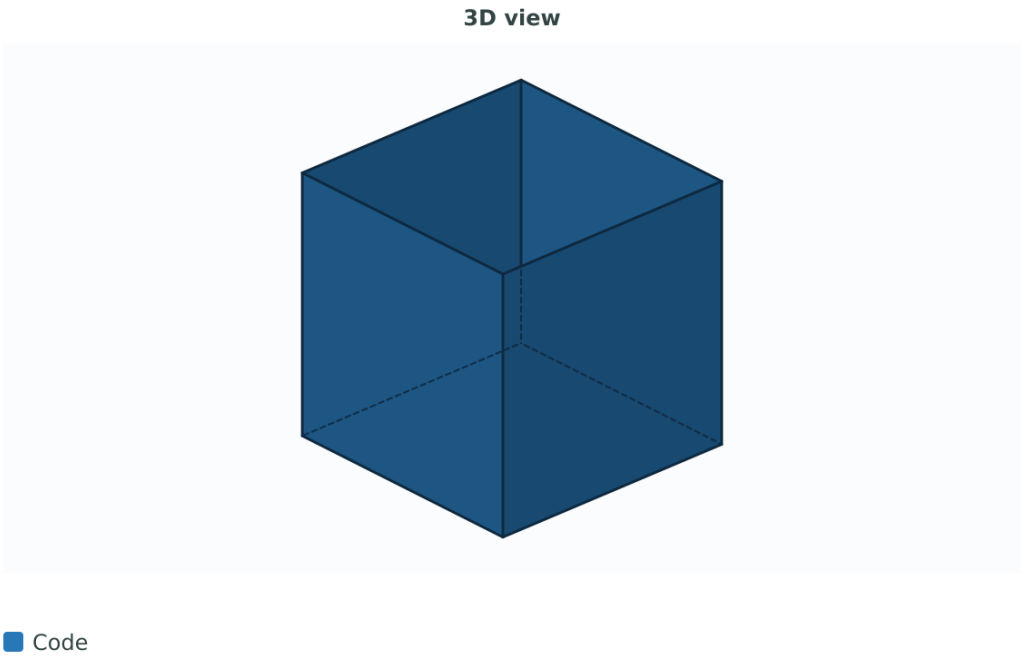
Code

```
SELECT ST_IsClosed('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
       ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
       ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
       ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)) )');
```

Output

```
f
```

Figure



Voir aussi

ST_IsRing

7.4.20 ST_IsCollection

ST_IsCollection — Teste si une géométrie est un type de collection de géométrie.

Synopsis

boolean **ST_IsCollection**(geometry g);

Description

Renvoie `TRUE` si le type de géométrie de l'argument est un type de collection de géométrie. Les types de collection sont les suivants :

— GEOMETRYCOLLECTION
— MULTI{POINT,POLYGON,LINESTRING,CURVE,SURFACE}
— COMPOUNDCURVE

Note
Cette fonction analyse le type de la géométrie. Renvoie `TRUE` pour les collections vides ou ne contenant qu'un seul élément.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

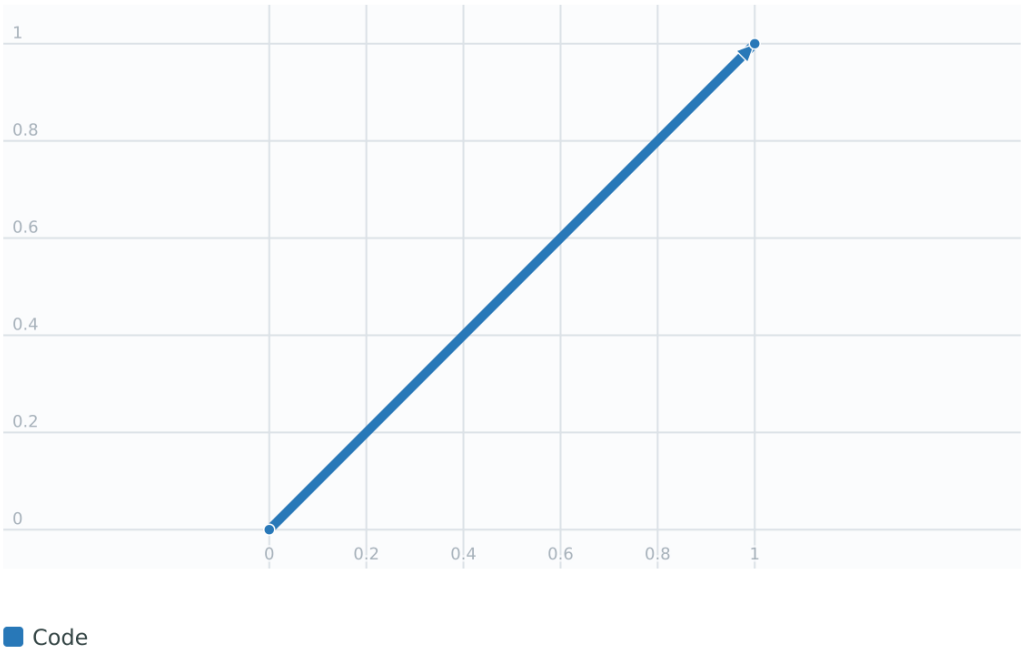
Code

```
SELECT ST_IsCollection('LINESTRING(0 0,1 1)'::geometry);
```

Output

```
f
```

Figure



Code

```
SELECT ST_IsCollection('MULTIPOINT EMPTY'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsCollection('MULTIPOINT((0 0))'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsCollection('MULTIPOINT((0 0),(42 42))'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsCollection('GEOMETRYCOLLECTION(POINT(0 0))'::geometry);
```

Output

```
t
```

Voir aussi

ST_NumGeometries

7.4.21 ST_IsEmpty

ST_IsEmpty — Teste si une géométrie est vide.

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Description

Renvoie true si cette géométrie est une géométrie vide. Si vrai, alors cette géométrie représente une collection de géométrie vide, un polygone, un point, etc.



Note
La norme SQL-MM stipule que ST_IsEmpty(NULL) doit renvoyer 0. PostGIS renvoie NULL.


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.7


Cette méthode prend en charge les types Circular String et Curve.

Modifié : 2.0.0 Dans les versions précédentes de PostGIS, ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') était autorisé. Ceci est maintenant illégal dans PostGIS 2.0.0 pour mieux se conformer aux normes SQL/MM

Exemples

Code

```
SELECT ST_IsEmpty('GEOMETRYCOLLECTION EMPTY'::geometry);
```

Output

```
t
```

Code

```
SELECT ST_IsEmpty('POLYGON EMPTY'::geometry);
```

Output

```
t
```

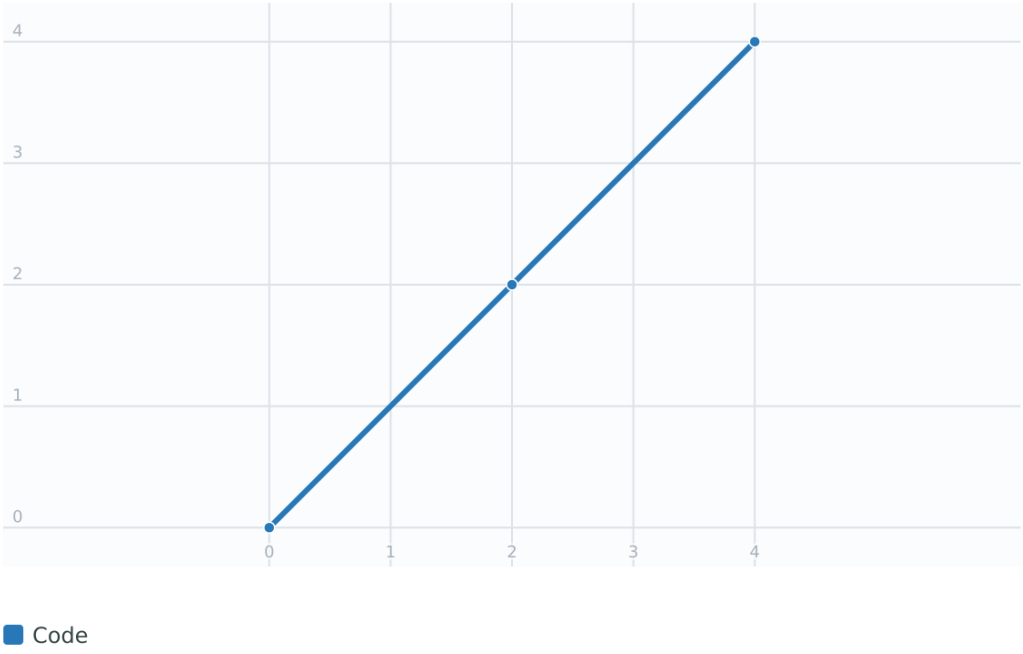
Code

```
SELECT ST_IsEmpty('POLYGON((1 2,3 4,5 6,1 2))'::geometry);
```

Output

```
f
```

Figure



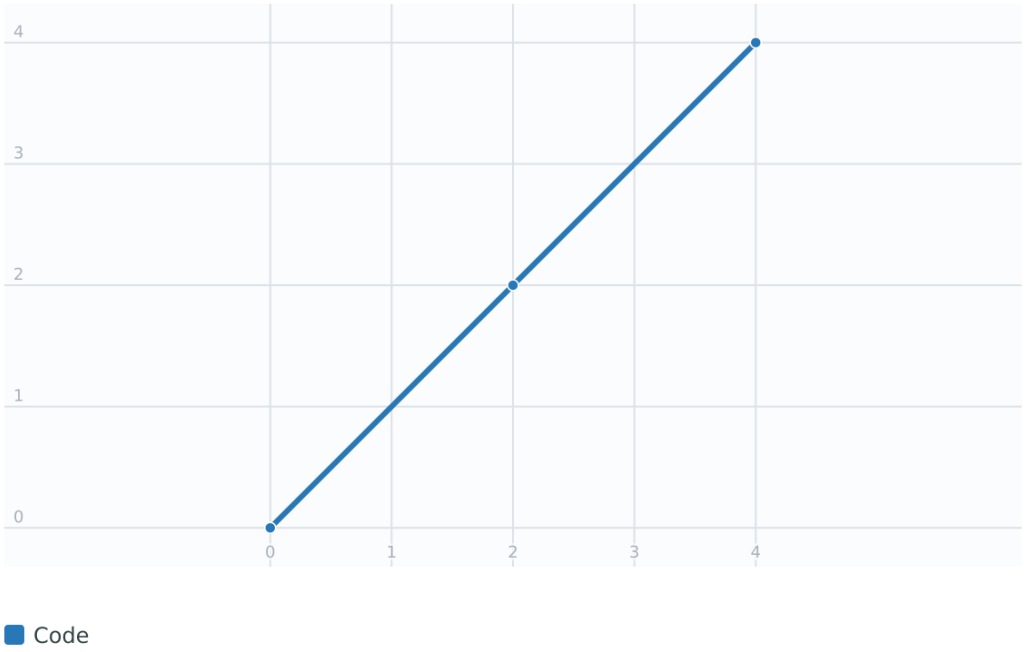
Code

```
SELECT ST_IsEmpty('POLYGON((1 2,3 4,5 6,1 2))'::geometry) = false;
```

Output

```
t
(1 row)
```

Figure



Code

```
SELECT ST_IsEmpty('CIRCULARSTRING EMPTY'::geometry);
```

Output

```
t
```

7.4.22 ST_IsPolygonCCW

ST_IsPolygonCCW — Teste si les polygones ont des anneaux extérieurs orientés dans le sens inverse des aiguilles d'une montre et des anneaux intérieurs orientés dans le sens des aiguilles d'une montre.

Synopsis

boolean **ST_IsPolygonCCW**(geometry geom);

Description

Renvoie un résultat positif si tous les composants polygonaux de la géométrie en entrée utilisent une orientation anti-horaire pour leur anneau extérieur et une orientation horaire pour tous les anneaux intérieurs.

Retourne vrai si la géométrie n'a pas de composants polygonaux.



Note
Les lignes fermées ne sont pas considérées comme des composants polygonaux, de sorte que vous obtiendrez toujours un retour vrai en passant une seule ligne fermée, quelle que soit son orientation.



Note
Si une géométrie polygonale n'utilise pas l'orientation inversée pour les anneaux intérieurs (c'est-à-dire si un ou plusieurs anneaux intérieurs sont orientés dans la même direction qu'un anneau extérieur), ST_IsPolygonCW et ST_IsPolygonCCW renvoient tous deux la valeur false.

Disponibilité : 2.4.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette fonction prend en charge les coordonnées M.

Voir aussi

ST_ForcePolygonCW, ST_ForcePolygonCCW, ST_IsPolygonCW

7.4.23 ST_IsPolygonCW

ST_IsPolygonCW — Teste si les polygones ont des anneaux extérieurs orientés dans le sens des aiguilles d'une montre et des anneaux intérieurs orientés dans le sens inverse des aiguilles d'une montre.

Synopsis

boolean **ST_IsPolygonCW**(geometry geom);

Description

Renvoie un résultat positif si tous les composants polygonaux de la géométrie d'entrée utilisent une orientation dans le sens des aiguilles d'une montre pour leur anneau extérieur, et dans le sens inverse des aiguilles d'une montre pour tous les anneaux intérieurs.

Retourne vrai si la géométrie n'a pas de composants polygonaux.



Note
Les lignes fermées ne sont pas considérées comme des composants polygonaux, de sorte que vous obtiendrez toujours un retour vrai en passant une seule ligne fermée, quelle que soit son orientation.



Note
Si une géométrie polygonale n'utilise pas l'orientation inversée pour les anneaux intérieurs (c'est-à-dire si un ou plusieurs anneaux intérieurs sont orientés dans la même direction qu'un anneau extérieur), ST_IsPolygonCW et ST_IsPolygonCCW renvoient tous deux la valeur false.

Disponibilité : 2.4.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette fonction prend en charge les coordonnées M.

Voir aussi

ST_ForcePolygonCW, ST_ForcePolygonCCW, ST_IsPolygonCW

7.4.24 ST_IsRing

ST_IsRing — Teste si une ligne est fermée et simple.

Synopsis

boolean **ST_IsRing**(geometry g);

Description

Returns `TRUE` if this `LINESTRING` is both ST_IsClosed (`ST_StartPoint(g) ~= ST_EndPoint(g)`) and ST_IsSimple (does not self intersect).

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.1.6



Note
SQL-MM définit le résultat de `ST_IsRing(NULL)` comme étant 0, alors que PostGIS renvoie `NULL`.

Exemples

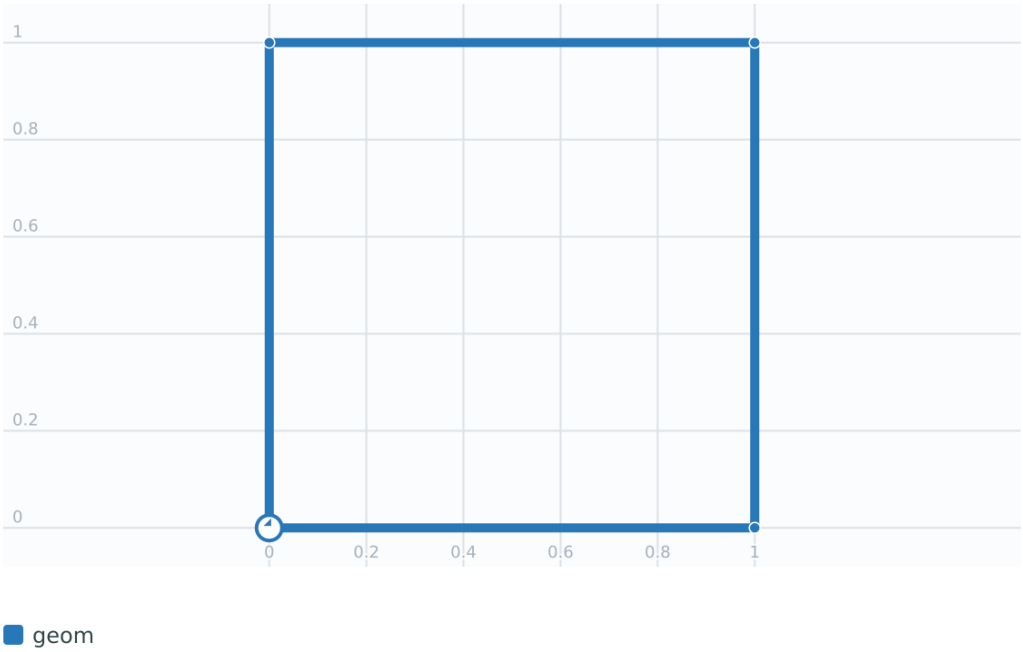
Code

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0,0 1,1 1,1 0,0 0)'::geometry AS geom) AS foo;
```

Output

```
st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 t         | t           | t
(1 row)
```

Figure



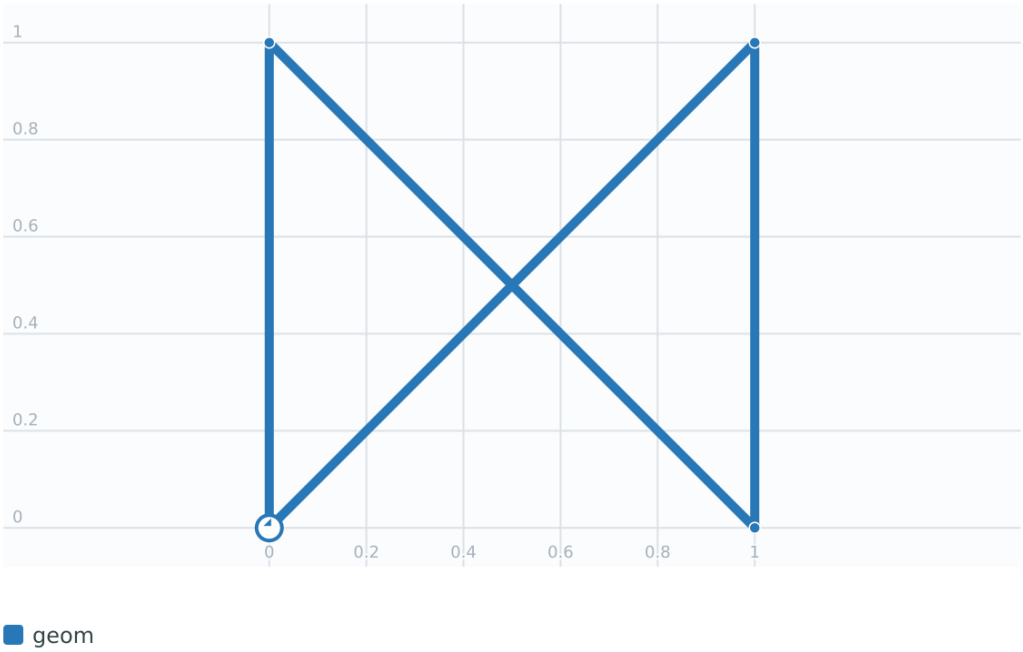
Code

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0,0 1,1 0,1 1,0 0)'::geometry AS geom) AS foo;
```

Output

```
st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 f         | t           | f
(1 row)
```

Figure



Voir aussi

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.25 ST_IsSimple

ST_IsSimple — Teste si une géométrie n'a pas de points d'auto-intersection ou d'auto-tangente.

Synopsis

boolean **ST_IsSimple**(geometry geomA);

Description

Renvoie TRUE si cette géométrie ne présente pas d'anomalie comme une auto intersection ou des segments tangentiels. Pour plus d'information sur les notions OGC de simplicité et de validité, se référer à "Ensuring OpenGIS compliance of geometries"

Note
La norme SQL-MM indique que le résultat de la fonction ST_IsSimple(NULL) doit être 0 ; PostGIS renvoie NULL.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.8

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

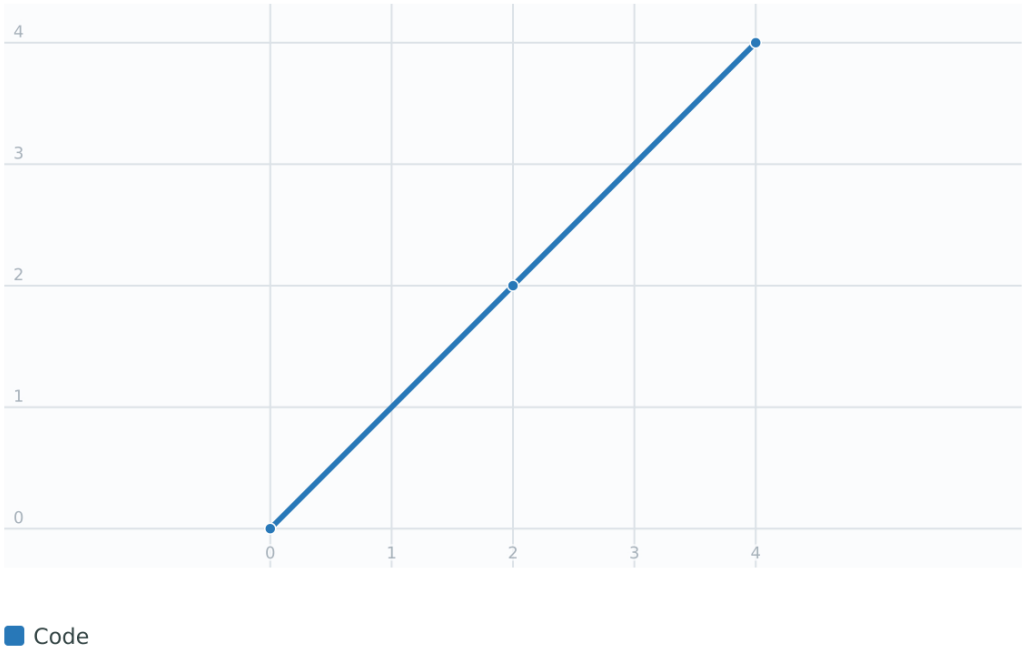
Code

```
SELECT ST_IsSimple('POLYGON((1 2,3 4,5 6,1 2))');
```

Output

```
f
```

Figure



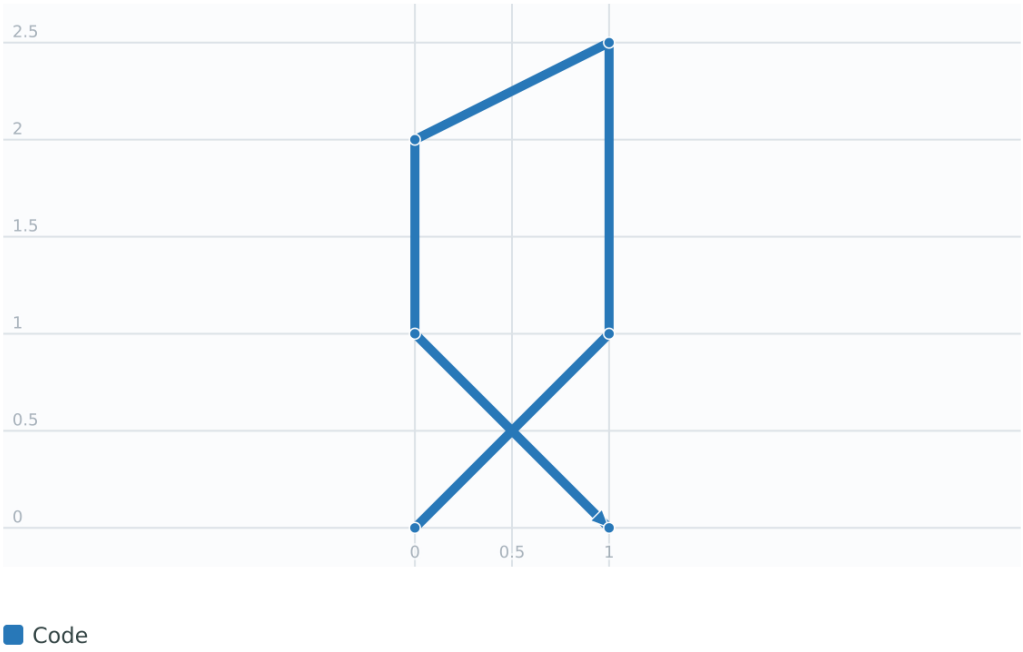
Code

```
SELECT ST_IsSimple('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)');
```

Output

```
f
```

Figure



Voir aussi

ST_IsValid

7.4.26 ST_M

ST_M — Renvoie la coordonnée M d'un point.

Synopsis

float **ST_M**(geometry a_point);

Description

Retourne les coordonnées M d'un point, ou NULL si non disponible. L'entrée doit être un point.

> **Note**
> Cette fonction ne fait pas (encore) partie de la spécification de l'OGC, mais elle est mentionnée ici pour compléter la liste des fonctions de l'extracteur de coordonnées de points.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_M('POINT(1 2 3 4)');
```

Output

```
4
```

Voir aussi

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.27 ST_MemSize

ST_MemSize — Renvoie la quantité d'espace mémoire que prend une géométrie.

Synopsis

integer **ST_MemSize**(geometry geomA);

Description

Renvoie la quantité d'espace mémoire (en octets) que prend la géométrie.

Ceci complète les fonctions intégrées de PostgreSQL fonctions d'objet de base de données pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relation_size.



Note
pg_relation_size qui donne la taille en octets d'une table peut renvoyer une taille en octets inférieure à ST_MemSize. Cela est dû au fait que pg_relation_size n'ajoute pas la contribution des tables toasted et que les grandes géométries sont stockées dans les tables TOAST.
pg_total_relation_size - comprend la table, les tables toasted et les index.
pg_column_size indique l'espace que prendrait une géométrie dans une colonne en tenant compte de la compression, et peut donc être inférieur à ST_MemSize

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Modifié : 2.2.0 nom modifié en ST_MemSize pour respecter la convention de nommage.

Exemples

Compare the memory used by all geometries in a Massachusetts town dataset with the subset for Boston.

Code

```
SELECT pg_size_pretty(sum(ST_MemSize(geom))) AS totgeomsum,
       pg_size_pretty(sum(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) AS ↩
    bossum,
       cast(
           sum(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END) * 100.0
           / sum(ST_MemSize(geom)) AS numeric(10, 2)
       ) AS perbos
FROM towns;
```

Output

```
totgeomsum | bossum | perbos
-----------+--------+--------
 1522 kB    | 30 kB  |   1.99
(1 row)
```

Return the memory size of a CircularString in bytes.

Code

```
SELECT ST_MemSize('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'::geometry);
```

Output

```
st_memsize
------------
         80
(1 row)
```

Calculate the percentage of a table's total size used by geometry data.

Code

```
SELECT pg_total_relation_size('public.neighborhoods') AS fulltable_size,
       sum(ST_MemSize(geom)) AS geomsize,
       sum(ST_MemSize(geom)) * 100.0
           / pg_total_relation_size('public.neighborhoods') AS pergeom
FROM neighborhoods;
```

Output

```
fulltable_size | geomsize |         pergeom
---------------+----------+-------------------------
         262144 |    96238 | 36.71188354492187500000
(1 row)
```

7.4.28 ST_NDims

ST_NDims — Renvoie la dimension des coordonnées d'une géométrie.

Synopsis

integer **ST_NDims**(geometry g1);

Description

Renvoie la dimension des coordonnées de la géométrie. PostGIS supporte 2 - (x,y) , 3 - (x,y,z) ou 2D avec mesure - x,y,m, et 4 - 3D avec espace de mesure x,y,z,m

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_NDims('POINT(1 1)') As d2point,
ST_NDims('POINT(1 1 2)') As d3point,
ST_NDims('POINTM(1 1 0.5)') As d2pointm;
```

Output

```
d2point | d3point | d2pointm
---------+---------+----------
       2 |       3 |         3
```

Voir aussi

ST_CoordDim, ST_Dimension, ST_GeomFromEWKT

7.4.29 ST_NPoints

ST_NPoints — Retourne le nombre de points (vertex) d'un objet géométrique.

Synopsis

integer **ST_NPoints**(geometry g1);

Description

Retourne le nombre de points d'un objet géométrique. Cela fonctionne pour tous les types de géométrie.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.



Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Count the points in a 2D LineString.

Code

```
SELECT ST_NPoints('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)');
```

Output

```
st_npoints
------------
          4
(1 row)
```

Count the points in a 3D LineString.

Code

```
SELECT ST_NPoints('LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31 -1,77.29 29.07 3)');
```

Output

```
st_npoints
------------
          4
(1 row)
```

Voir aussi

ST_NumPoints

7.4.30 ST_NRings

ST_NRings — Renvoie le nombre d'anneaux dans une géométrie polygonale.

Synopsis

integer **ST_NRings**(geometry geomA);

Description

Si la géométrie est un polygone ou un multi-polygone, renvoie le nombre d'anneaux. Contrairement à NumInteriorRings, il compte également les anneaux extérieurs.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

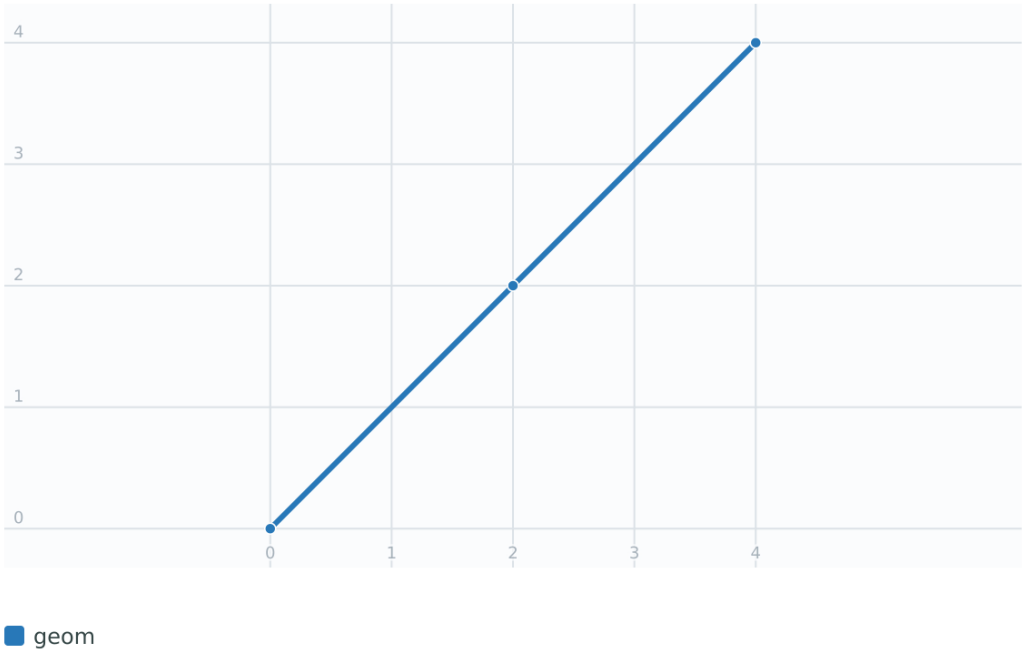
Code

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
FROM (SELECT 'POLYGON((1 2,3 4,5 6,1 2))'::geometry As geom) As foo;
```

Output

```
nrings | ninterrings
--------+-------------
      1 |           0
(1 row)
```

Figure



Voir aussi

ST_NumInteriorRings

7.4.31 ST_NumGeometries

ST_NumGeometries — Renvoie le nombre d'éléments dans une collection de géométrie.

Synopsis

integer **ST_NumGeometries**(geometry geom);

Description

Renvoie le nombre d'éléments d'une collection de géométries (GEOMETRYCOLLECTION ou MULTI*). Pour les géométries atomiques non vides, le résultat est 1. Pour les géométries vides, le résultat est 0.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Modifié : 2.0.0 Dans les versions précédentes, cette fonction renvoyait NULL si la géométrie n'était pas de type collection/-MULTI. 2.0.0+ renvoie maintenant 1 pour les géométries simples, par exemple POLYGONE, LINESTRING, POINT.

Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and return 1; use ST_NumPatches to count faces.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 9.1.4

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

This example returns 1; prior versions would have returned NULL.

Code

```
SELECT ST_NumGeometries('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)');
```

Output

```
st_numgeometries
------------------
                 1
(1 row)
```

A MultiPoint counts as one geometry within a GeometryCollection.

Code

```
SELECT ST_NumGeometries('GEOMETRYCOLLECTION(MULTIPOINT((-2 3),(-2 2)),
LINESTRING(5 5,10 10),
POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))');
```

Output

```
st_numgeometries
------------------
                 3
(1 row)
```

Voir aussi

ST_GeometryN, ST_Multi

7.4.32 ST_NumInteriorRings

ST_NumInteriorRings — Renvoie le nombre d'anneaux intérieurs (trous) d'un polygone.

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

Description

Renvoie le nombre d'anneaux intérieurs d'une géométrie polygonale. Retourne NULL si la géométrie n'est pas un polygone.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.2.5

Modifié : 2.0.0 - dans les versions antérieures, il permettait de passer un MULTIPOLYGONE, renvoyant le nombre d'anneaux intérieurs du premier POLYGONE.

Exemples

Count the interior rings of a Polygon.

Code

```
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;
```

For a MultiPolygon, dump its component Polygons and sum their interior ring counts.

Code

```
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(geom)) AS numholes
FROM (
    SELECT gid, field1, field2, (ST_Dump(geom)).geom AS geom
    FROM sometable
) AS foo
GROUP BY gid, field1, field2;
```

Voir aussi

ST_NumInteriorRing, ST_InteriorRingN

7.4.33 ST_NumInteriorRing

ST_NumInteriorRing — Returns the number of interior rings (holes) of a Polygon. Alias for ST_NumInteriorRings.

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

Description

This function is an alias for ST_NumInteriorRings.

Disponibilité : 1.2.2

Voir aussi

ST_NumInteriorRings, ST_InteriorRingN

7.4.34 ST_NumPatches

ST_NumPatches — Returns the number of faces in a PolyhedralSurface or TIN.

Synopsis

integer **ST_NumPatches**(geometry g1);

Description

Returns the number of faces in a PolyhedralSurface or TIN. Returns NULL for other geometries. Unlike ST_NumGeometries, this function counts individual faces.

Disponibilité : 2.0.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM ISO/IEC 13249-3 : 8.5

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_NumPatches('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )');
```

Output

```
st_numpatches
---------------
             6
(1 row)
```

Voir aussi

ST_GeomFromEWKT, ST_NumGeometries, ST_PatchN

7.4.35 ST_NumPoints

ST_NumPoints — Renvoie le nombre de points dans une LineString ou une CircularString.

Synopsis

integer **ST_NumPoints**(geometry g1);

Description

Retourne le nombre de points dans une ST_LineString ou ST_CircularString. Avant la version 1.4, cette fonction ne fonctionnait qu'avec les lignes, comme l'indiquent les spécifications. A partir de la version 1.4, il s'agit d'un alias de ST_NPoints qui renvoie le nombre de sommets, et pas seulement pour les lignes. Envisagez d'utiliser ST_NPoints à la place, qui est polyvalent et fonctionne avec de nombreux types de géométrie.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.2.4

Exemples

Code

```
SELECT ST_NumPoints('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)');
```

Output

```
st_numpoints
--------------
            4
(1 row)
```

Voir aussi

ST_NPoints

7.4.36 ST_PatchN

ST_PatchN — Returns the Nth face of a PolyhedralSurface or TIN.

Synopsis

geometry **ST_PatchN**(geometry geomA, integer n);

Description

Returns the 1-based Nth face of a PolyhedralSurface or TIN. Returns NULL for other geometries or an out-of-range index. Use this function to access faces, because ST_GeometryN treats PolyhedralSurface and TIN as unitary geometries.



Note
L'indice est basé sur 1.



Note
Si vous voulez extraire tous les éléments d'une géométrie, ST_Dump est plus efficace.

Disponibilité : 2.0.0

Cette méthode implémente la spécification SQL/MM. SQL-MM ISO/IEC 13249-3 : 8.5

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Extract the 2nd face of the polyhedral surface.

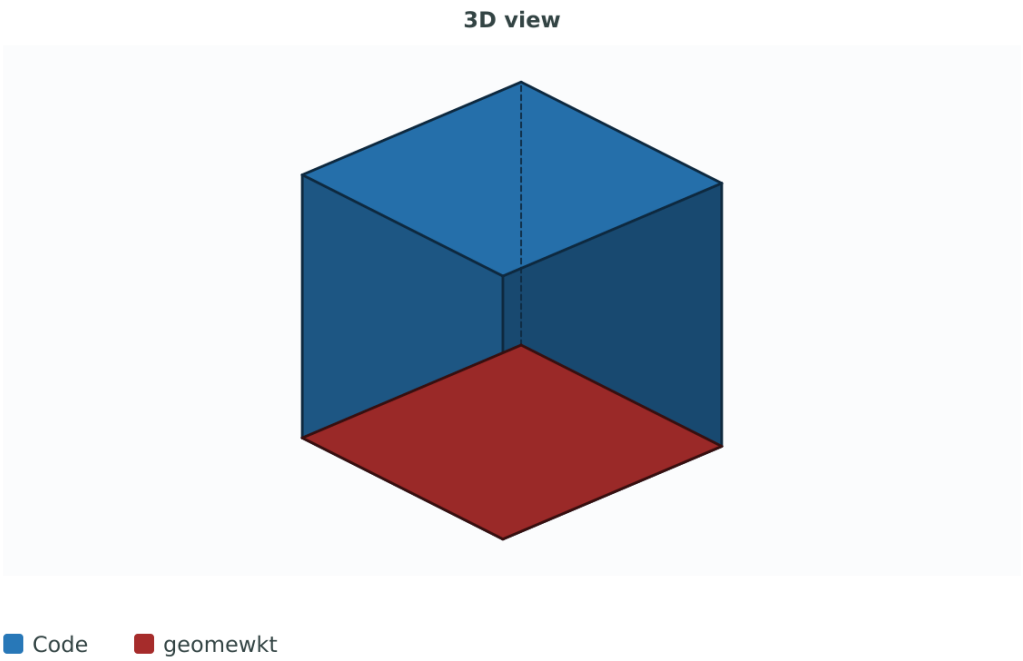
Code

```
SELECT ST_PatchN(geom, 2) As geomewkt
FROM (
VALUES ('POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry) ) As ↩
    foo(geom);
```

Output

```
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

Figure



Voir aussi

ST_AsEWKT, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries, ST_NumPatches

7.4.37 ST_PointN

ST_PointN — Renvoie le Nième point de la première LineString ou LineString circulaire d'une géométrie.

Synopsis

geometry **ST_PointN**(geometry a_linestring, integer n);

Description

Renvoie le Nième point d'une ligne ou d'une ligne circulaire dans la géométrie. Les valeurs négatives sont comptées à rebours à partir de la fin de la LineString, de sorte que -1 est le dernier point. Renvoie NULL s'il n'y a pas de ligne dans la géométrie.



Note
Index is 1-based as for OGC specs since version 0.8.0. Backward indexing (negative index) is not in OGC. Previous versions implemented this as 0-based instead.



Note
Si vous souhaitez obtenir le Nième point de chaque ligne dans une multiligne, utilisez ST_Dump en conjonction avec ST_Dump

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.2.5, 7.3.5

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Modifié : la version 2.0.0 ne fonctionne plus avec les multi-lignes à géométrie unique. Dans les anciennes versions de PostGIS, une multi-ligne d'une seule ligne fonctionnait parfaitement avec cette fonction et renvoyait le point de départ. Dans la version 2.0.0, elle renvoie simplement NULL comme n'importe quelle autre multi-ligne.

Modifié : 2.3.0 : indexation négative disponible (-1 est le dernier point)

Exemples

This example extracts all POINT geometries from a LineString.

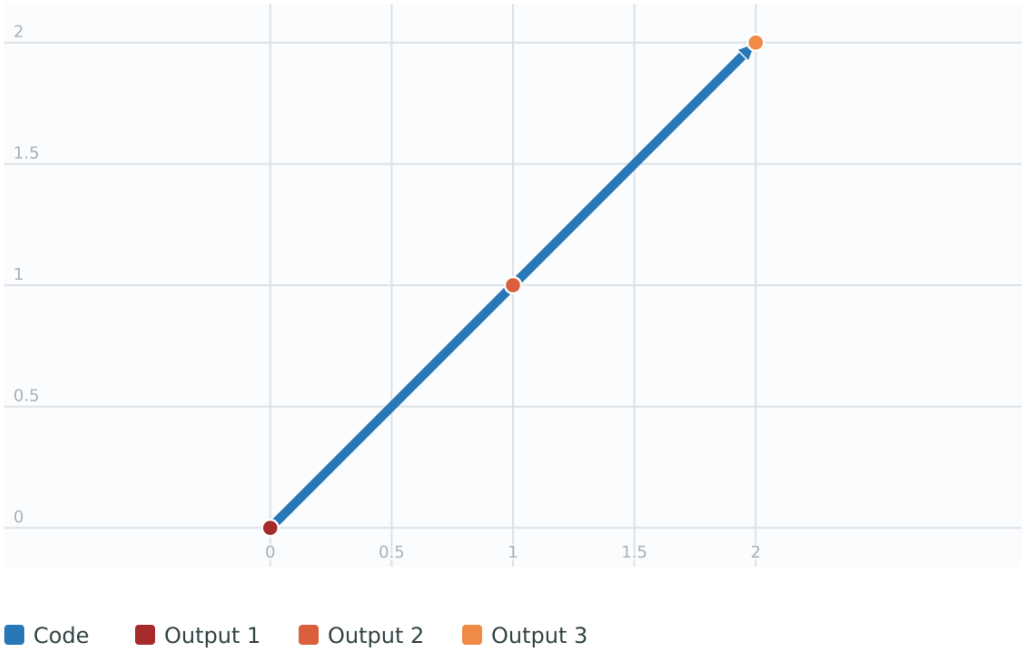
Code

```
SELECT ST_PointN(
      column1,
      generate_series(1, ST_NPoints(column1))
   )
FROM ( VALUES ('LINESTRING(0 0,1 1,2 2)'::geometry) ) AS foo;
```

Output

```
POINT(0 0)
 POINT(1 1)
 POINT(2 2)
(3 rows)
```

Figure



This example uses a CircularString.

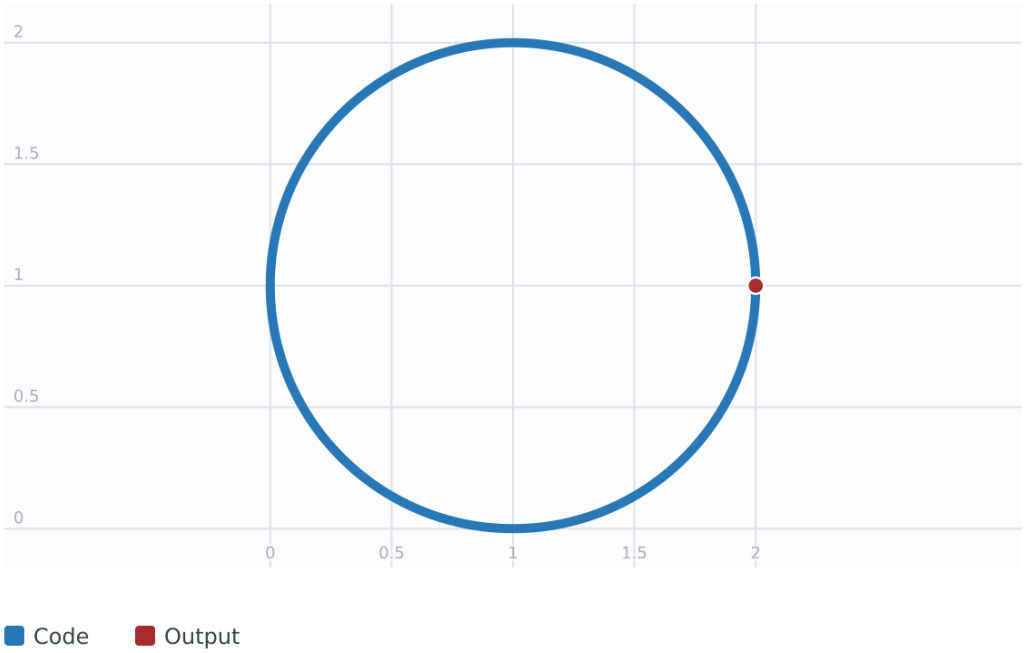
Code

```
SELECT ST_PointN('CIRCULARSTRING(1 2,3 2,1 2)', 2);
```

Output

```
POINT(3 2)
```

Figure



The index is 1-based. A negative index counts backward from the end of the LineString.

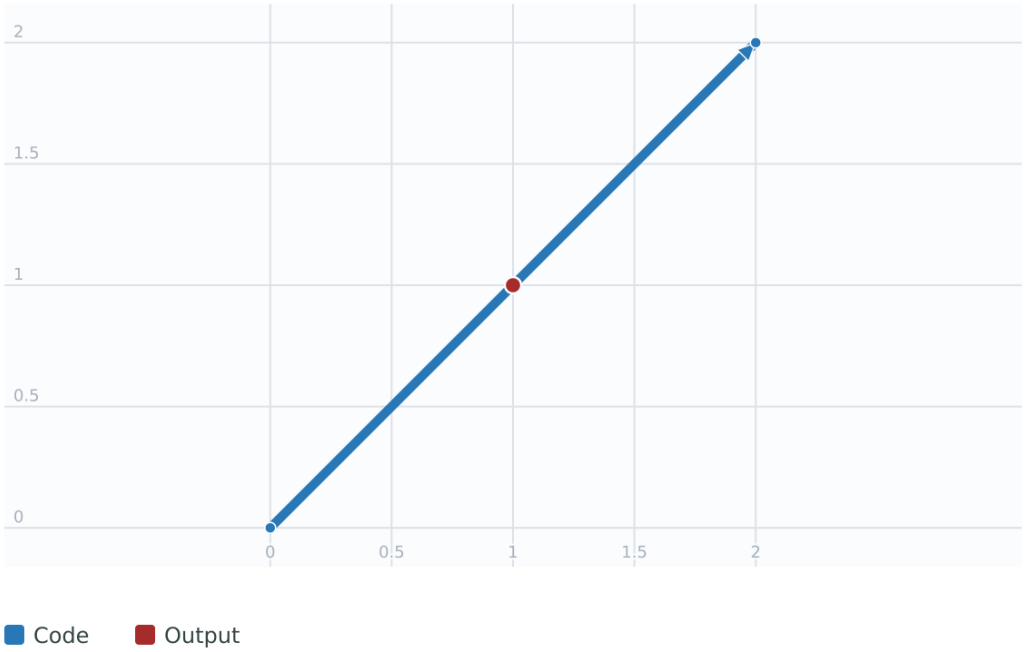
Code

```
SELECT f
FROM ST_GeomFromText('LINESTRING(0 0 0,1 1 1,2 2 2)') AS g
  CROSS JOIN ST_PointN(g, -2) AS f;
```

Output

```
POINT Z (1 1 1)
```

Figure



Voir aussi

ST_NPoints

7.4.38 ST_Points

ST_Points — Renvoie un MultiPoint contenant les coordonnées d'une géométrie.

Synopsis

geometry **ST_Points**(geometry geom);

Description

Renvoie un MultiPoint contenant toutes les coordonnées d'une géométrie. Les points en double sont conservés, y compris les points de départ et d'arrivée des géométries en anneau. (Si vous le souhaitez, les points en double peuvent être supprimés en appelant ST_RemoveRepeatedPoints sur le résultat).

Pour obtenir des informations sur la position de chaque coordonnée dans la géométrie parente, utilisez ST_DumpPoints.

Les coordonnées M et Z sont conservées si elles sont présentes.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Disponibilité : 2.3.0

Exemples

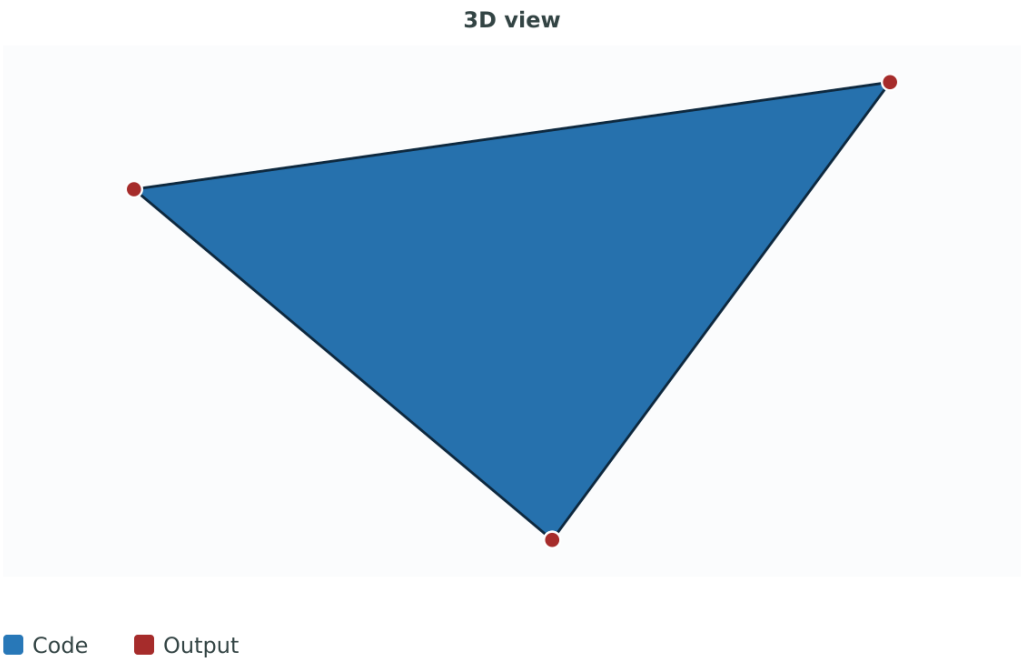
Code

```
SELECT ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6,30 10 4))');
```

Output

```
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

Figure



Voir aussi

ST_RemoveRepeatedPoints, ST_DumpPoints

7.4.39 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

Description

Returns the first point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry as a POINT. For other geometries, returns the first point in coordinate order.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.1.3

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Amélioré : 3.2.0 renvoie un point pour toutes les géométries. Le comportement précédent renvoyait NULL si l'entrée n'était pas une LineString.

Modifié : 2.0.0 ne fonctionne plus avec les MultiLineStrings à géométrie unique. Dans les anciennes versions de PostGIS, une MultiLineString à géométrie unique fonctionnait sans problème avec cette fonction et renvoyait le point de départ. Dans la version 2.0.0, elle renvoie simplement NULL comme toute autre MultiLineString. L'ancien comportement était une fonctionnalité non documentée, mais les personnes qui supposaient que leurs données étaient stockées en tant que LINESTRING peuvent voir ces données retourner NULL dans la version 2.0.0.

Exemples

Point de départ d'une LineString

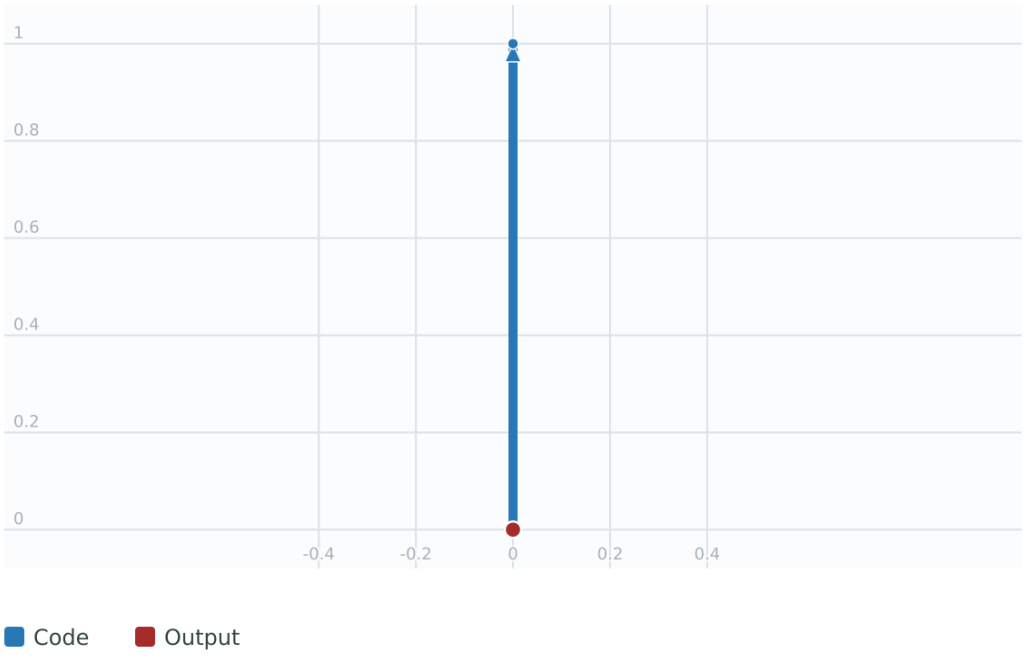
Code

```
SELECT ST_StartPoint('LINESTRING(0 1,0 2)'::geometry);
```

Output

```
POINT(0 1)
```

Figure



Start point of a Point is the Point itself

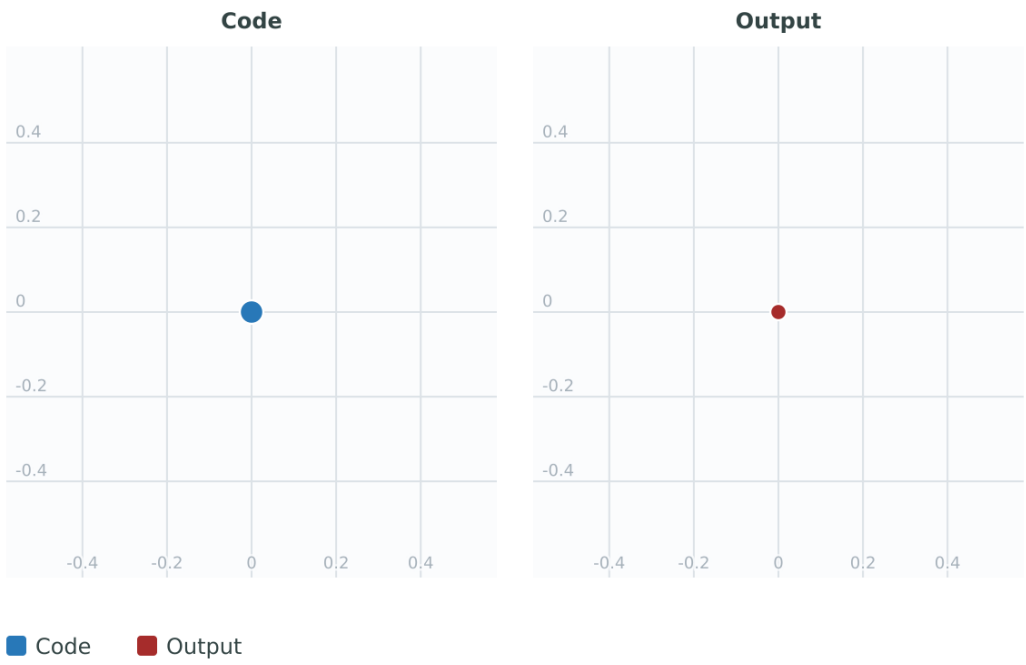
Code

```
SELECT ST_StartPoint('POINT(0 1)'::geometry);
```

Output

```
POINT(0 1)
```

Figure



Point de départ d'une LineString 3D

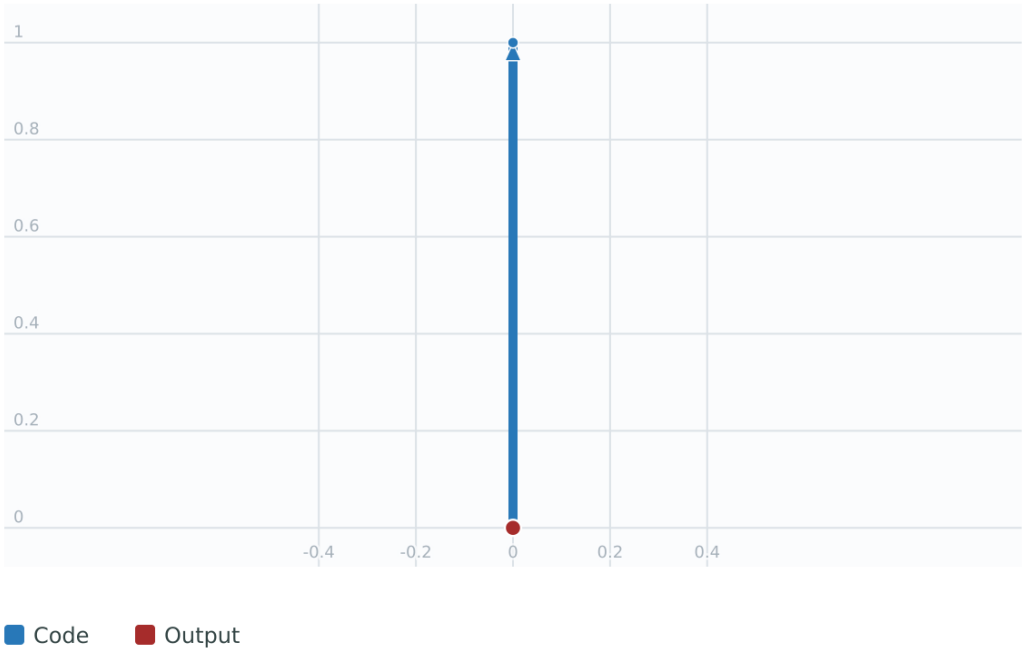
Code

```
SELECT ST_StartPoint('LINESTRING(0 1 1,0 2 2)'::geometry);
```

Output

```
POINT(0 1 1)
```

Figure



Point de départ d'une CircularString

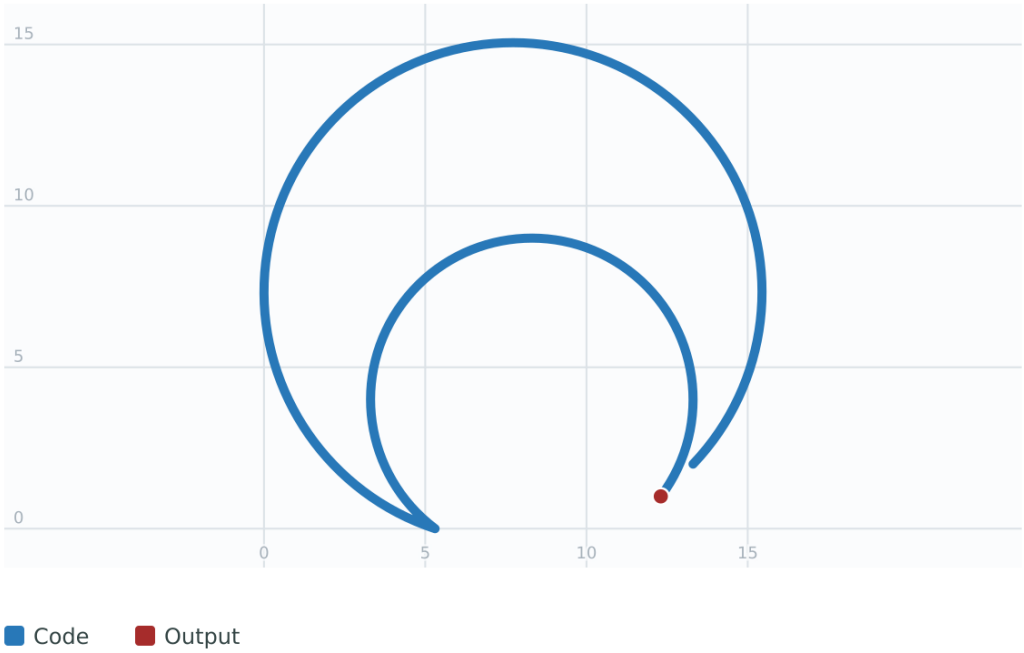
Code

```
SELECT ST_StartPoint('CIRCULARSTRING(5 2,-3 2,-2 1,-4 2,6 3)'::geometry);
```

Output

```
POINT(5 2)
```

Figure



Start point of a NURBSCurve

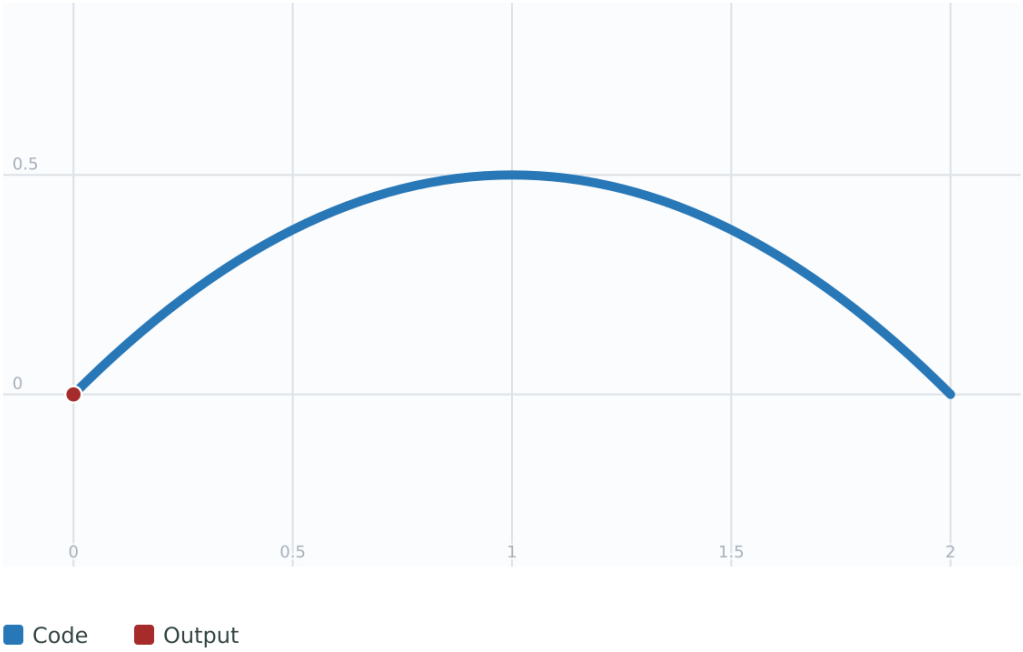
Code

```
SELECT ST_StartPoint('NURBSCURVE(2, (0 0, 1 1, 2 0))'::geometry);
```

Output

```
POINT(0 0)
```

Figure



Build edge node references from the start and end points of a LineString table. The example uses exact endpoint equality, so linework with nearly coincident endpoints should be snapped or otherwise cleaned first.

Code

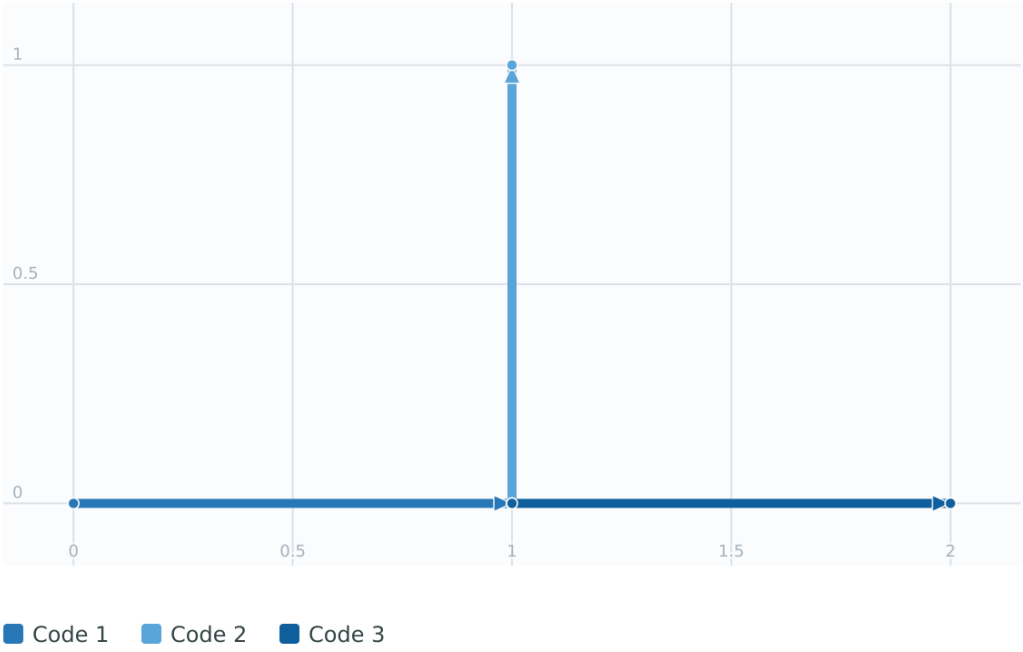
```
WITH roads(road_id, geom) AS (
  VALUES
    (1, 'LINESTRING(0 0, 1 0)'::geometry),
    (2, 'LINESTRING(1 0, 1 1)'::geometry),
    (3, 'LINESTRING(1 0, 2 0)'::geometry)
),
endpoints AS (
  SELECT road_id, 'from' AS end_name, ST_StartPoint(geom) AS geom,
         ST_AsEWKB(ST_StartPoint(geom)) AS endpoint_key
  FROM roads
  UNION ALL
  SELECT road_id, 'to' AS end_name, ST_EndPoint(geom) AS geom,
         ST_AsEWKB(ST_EndPoint(geom)) AS endpoint_key
  FROM roads
),
nodes AS (
  SELECT dense_rank() OVER (ORDER BY ST_X(geom), ST_Y(geom), endpoint_key) AS node_id,
         endpoint_key
  FROM (
    SELECT DISTINCT ON (endpoint_key) endpoint_key, geom
    FROM endpoints
    ORDER BY endpoint_key
  ) AS distinct_endpoints
```

```
)
SELECT e.road_id,
       max(n.node_id) FILTER (WHERE e.end_name = 'from') AS from_node,
       max(n.node_id) FILTER (WHERE e.end_name = 'to') AS to_node
FROM endpoints AS e
JOIN nodes AS n USING (endpoint_key)
GROUP BY e.road_id
ORDER BY e.road_id;
```

Output

```
road_id | from_node | to_node
---------+-----------+---------
     1 |         1 |       2
     2 |         2 |       3
     3 |         2 |       4
```

Figure



Voir aussi

ST_EndPoint, ST_PointN

7.4.40 ST_StartM

ST_StartM — Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

float8 **ST_StartM**(geometry geom);

Description

Returns the M coordinate of the first point of a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE geometry. Returns NULL if the input geometry does not have an M dimension or is not a LINESTRING, CIRCULARLINESTRING, or NURBSCURVE.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Start M value of a LineString

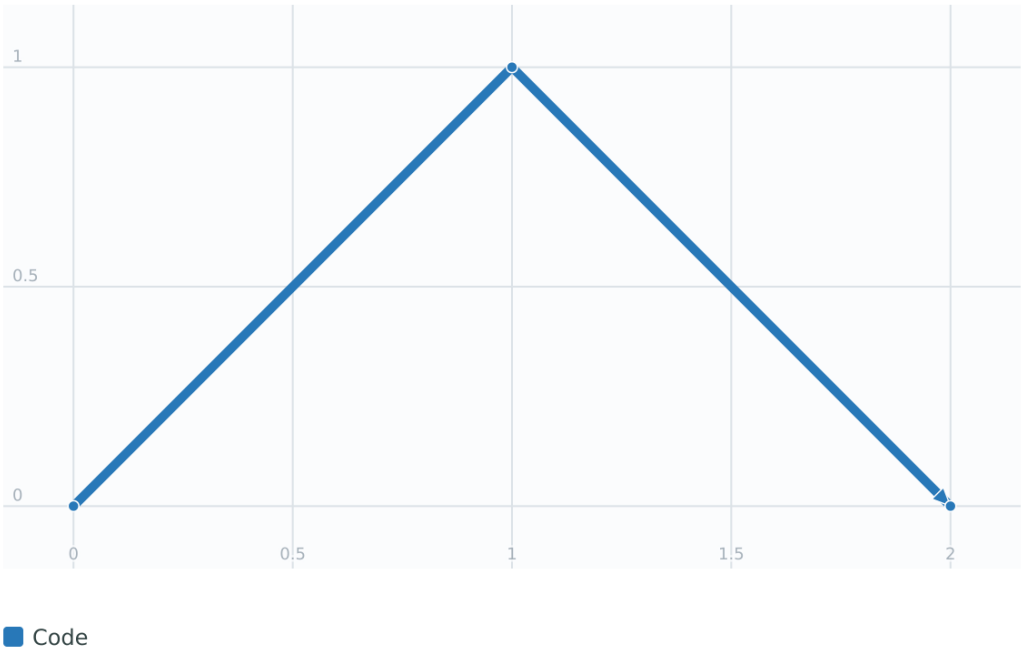
Code

```
SELECT ST_StartM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry);
```

Output

```
10
```

Figure



Start M value of a CircularString

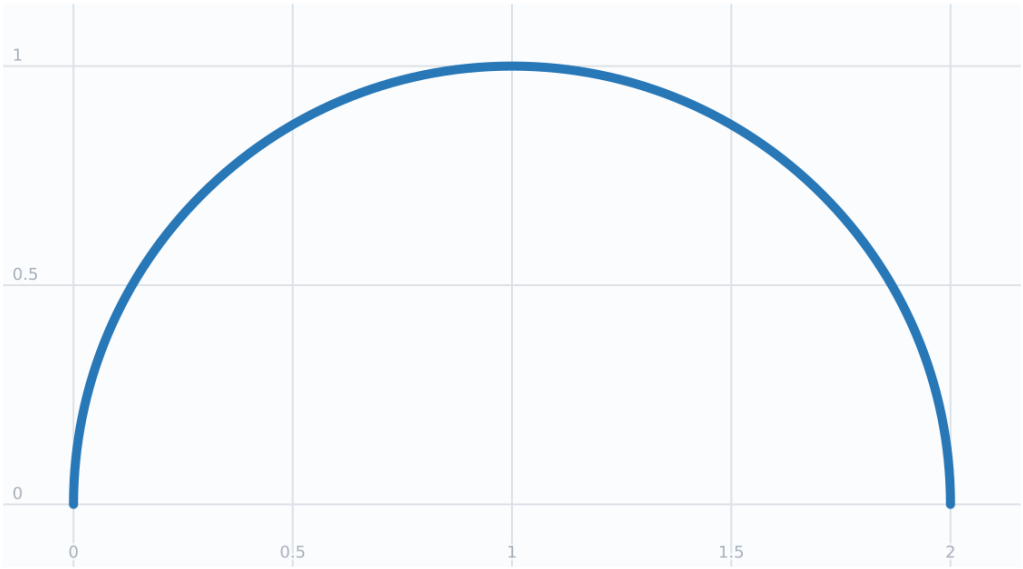
Code

```
SELECT ST_StartM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry);
```

Output

```
5
```

Figure



Start M value of a NURBSCurve

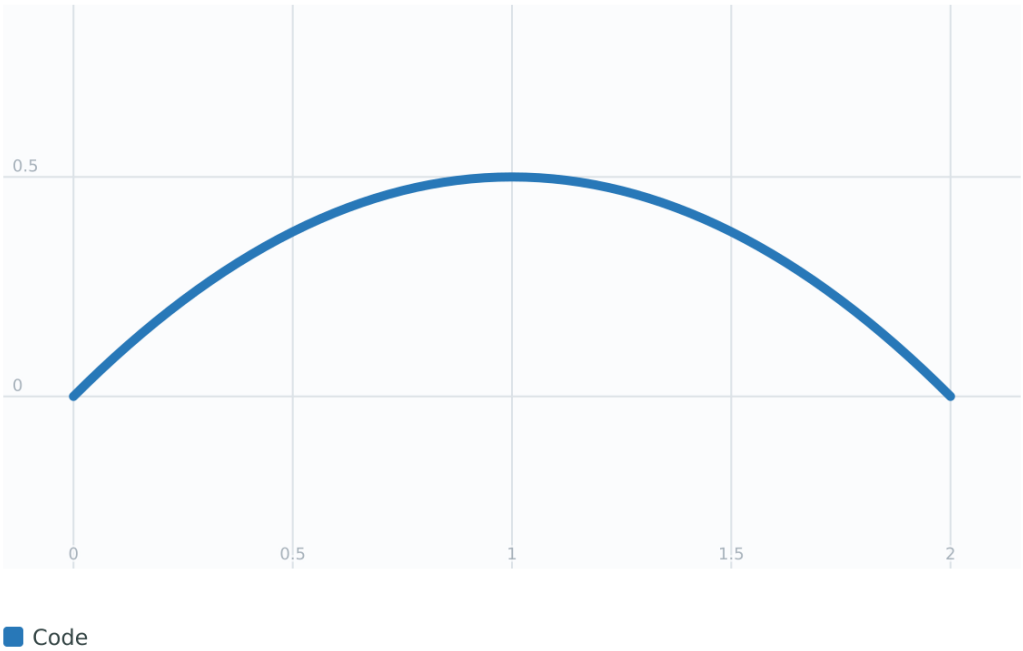
Code

```
SELECT ST_StartM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry);
```

Output

```
100
```

Figure



Geometry without M dimension returns NULL

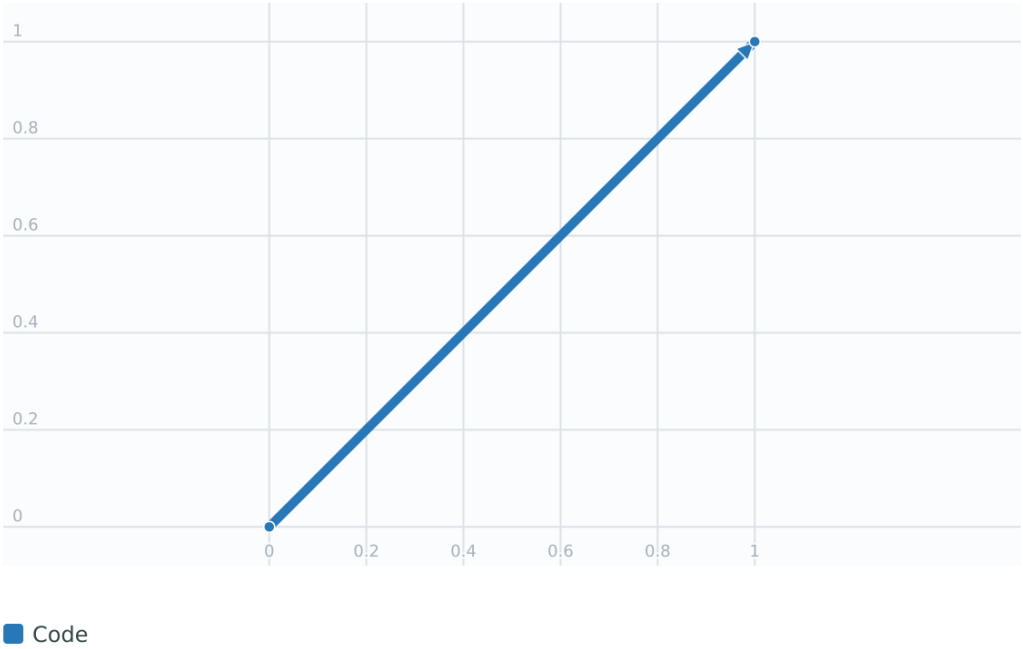
Code

```
SELECT ST_StartM('LINESTRING(0 0,1 1)'::geometry) IS NULL AS is_null;
```

Output

```
t
```

Figure



Voir aussi

ST_EndM, ST_SetStartM, ST_StartPoint

7.4.41 ST_EndM

ST_EndM — Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

float8 **ST_EndM**(geometry geom);

Description

Returns the M coordinate of the last point of a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry. Returns `NULL` if the input geometry does not have an M dimension or is not a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE`.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

End M value of a LineString

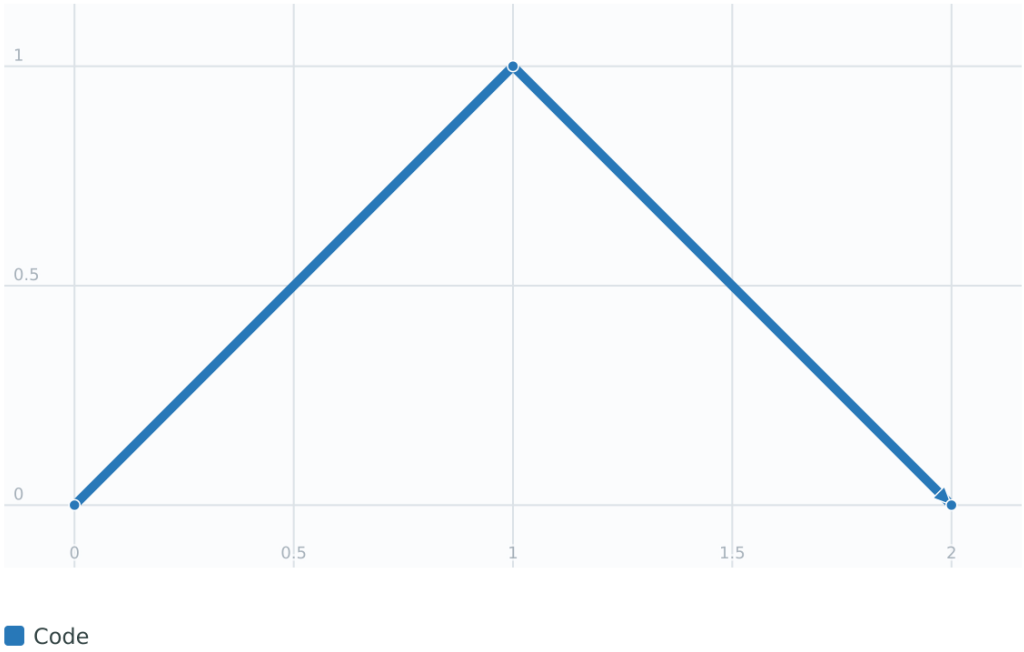
Code

```
SELECT ST_EndM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry);
```

Output

```
30
```

Figure



End M value of a CircularString

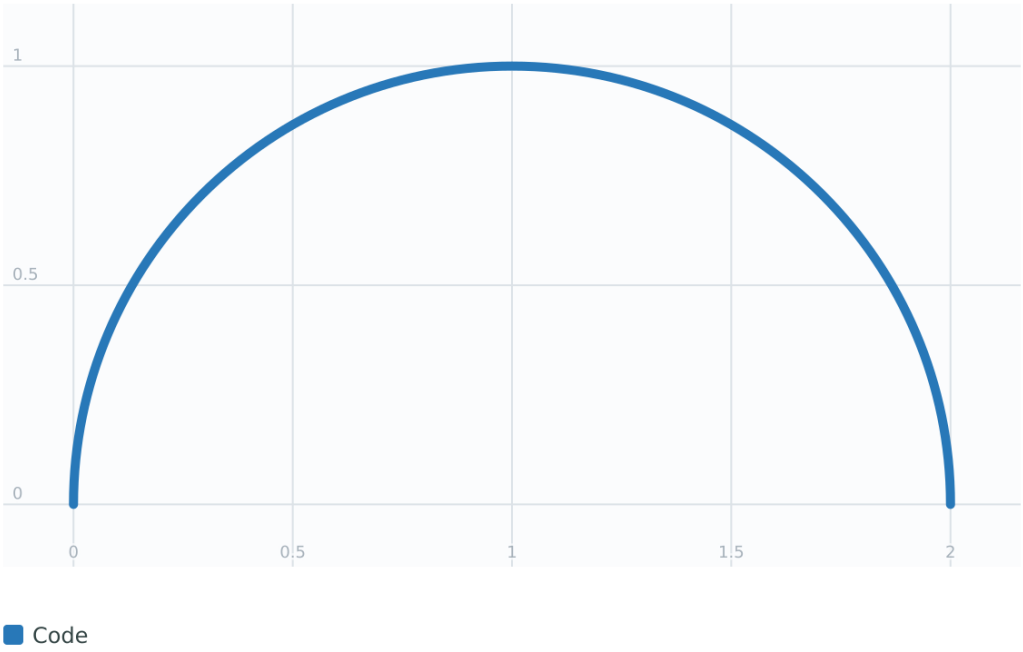
Code

```
SELECT ST_EndM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry);
```

Output

```
15
```

Figure



End M value of a NURBSCurve

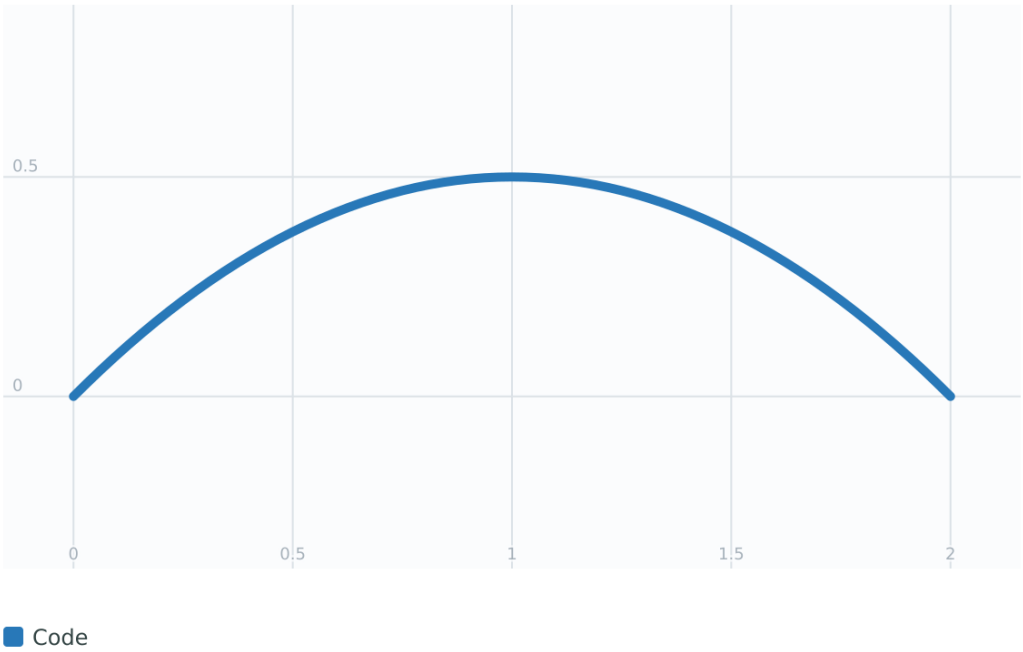
Code

```
SELECT ST_EndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry);
```

Output

```
300
```

Figure



Geometry without M dimension returns NULL

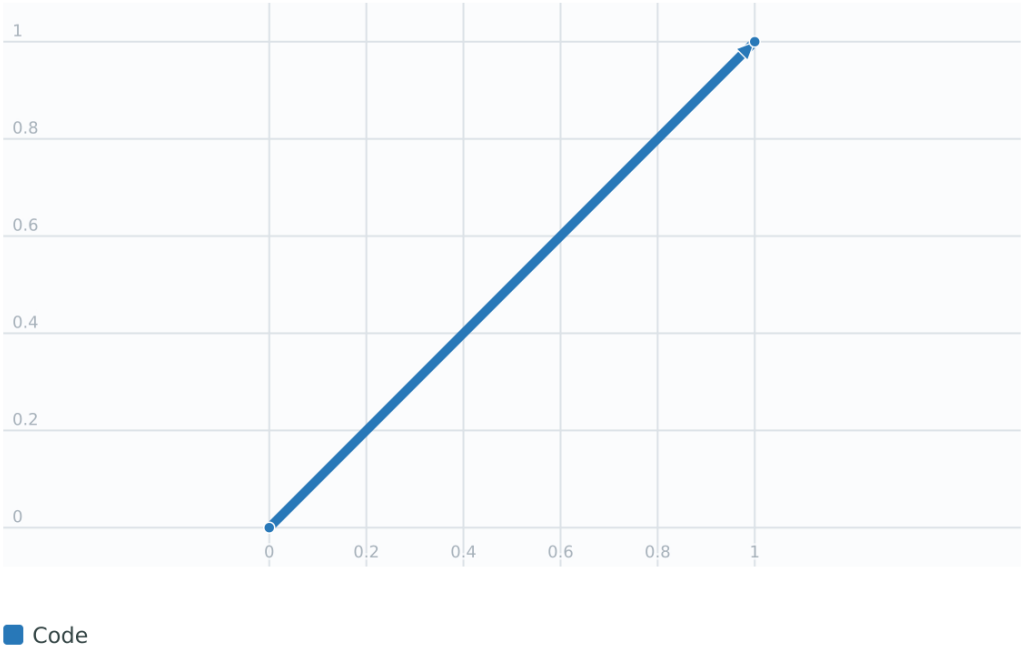
Code

```
SELECT ST_EndM('LINESTRING(0 0,1 1)'::geometry) IS NULL AS is_null;
```

Output

```
t
```

Figure



Voir aussi

ST_StartM, ST_SetEndM, ST_EndPoint

7.4.42 ST_SetStartM

ST_SetStartM — Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_SetStartM**(geometry geom, float8 m);

Description

Returns a copy of the input `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry with the M coordinate of the first point set to the specified value. If the input geometry does not have an M dimension, it will be added. Returns `NULL` if the input is not a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE`.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Set start M value of a LineString

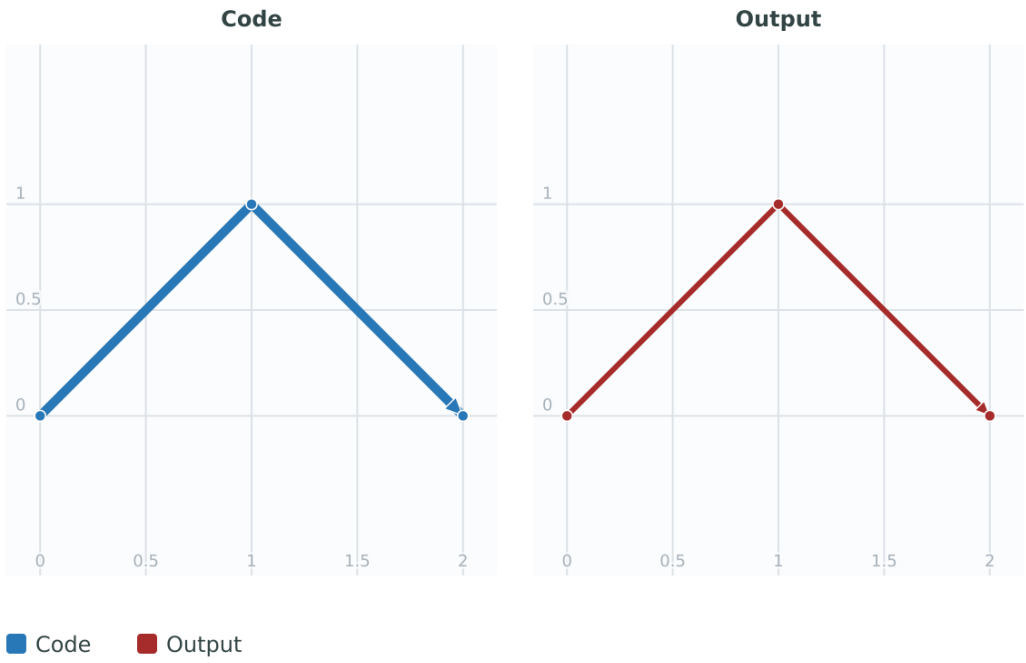
Code

```
SELECT ST_SetStartM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry, 100);
```

Output

```
LINESTRING M (0 0 100,1 1 20,2 0 30)
```

Figure



Add M dimension and set start M value

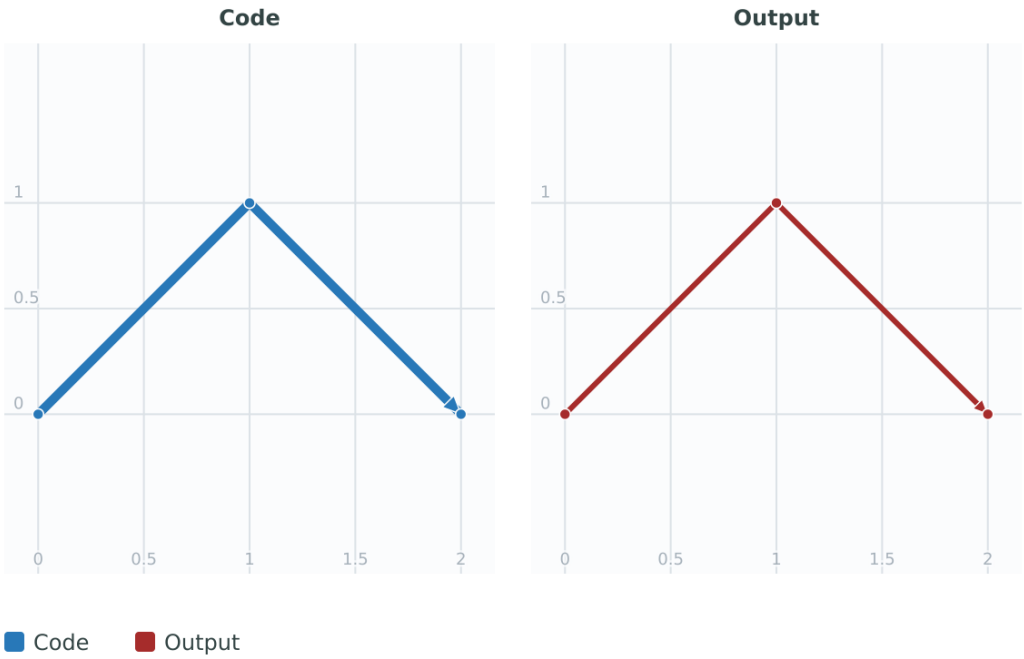
Code

```
SELECT ST_SetStartM('LINESTRING(0 0,1 1,2 0)'::geometry, 50);
```

Output

```
LINESTRING M (0 0 50,1 1 0,2 0 0)
```

Figure



Set start M value of a CircularString

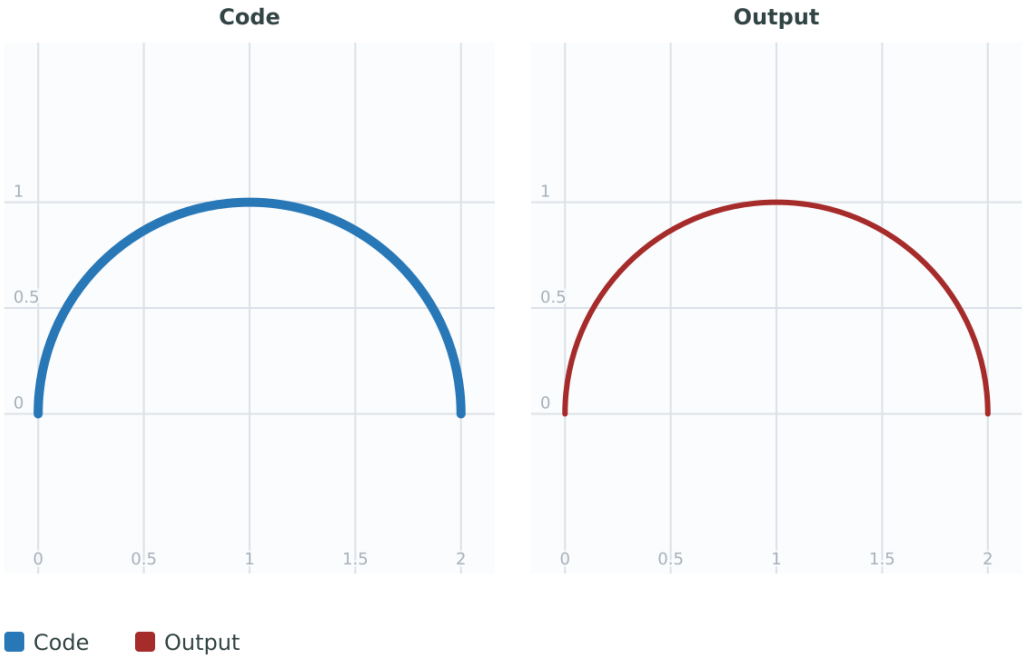
Code

```
SELECT ST_SetStartM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry, 99);
```

Output

```
CIRCULARSTRING M (0 0 99,1 1 10,2 0 15)
```

Figure



Set start M value of a NURBSCurve

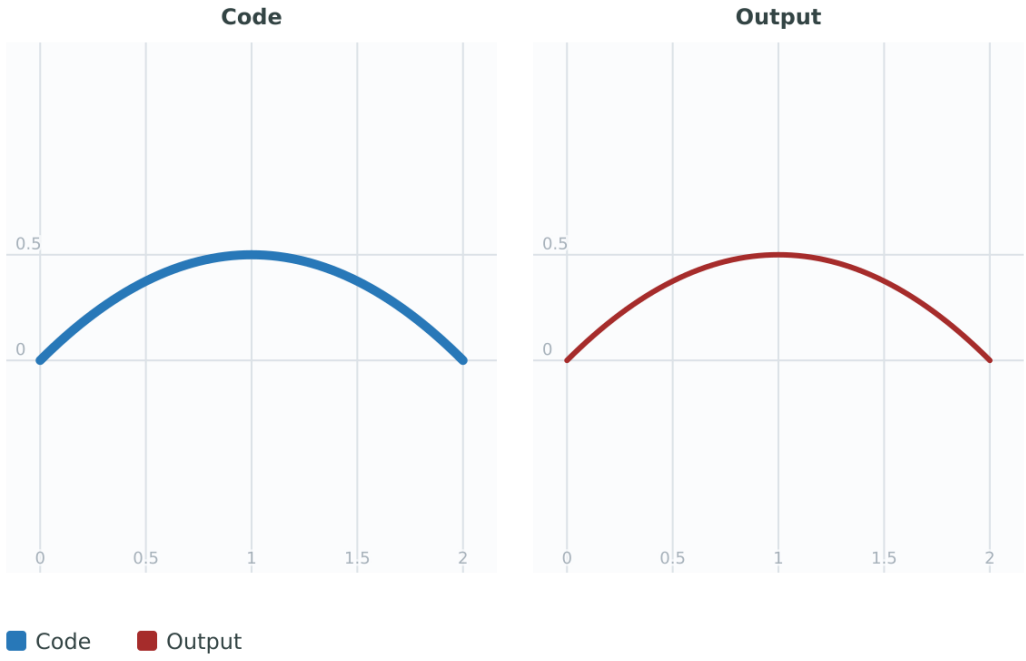
Code

```
SELECT ST_SetStartM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry, 500);
```

Output

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 500),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (1 1 ↩
     200),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (2 0 300),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



Voir aussi

ST_SetEndM, ST_StartM, ST_StartPoint

7.4.43 ST_SetEndM

ST_SetEndM — Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.

Synopsis

geometry **ST_SetEndM**(geometry geom, float8 m);

Description

Returns a copy of the input `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE` geometry with the M coordinate of the last point set to the specified value. If the input geometry does not have an M dimension, it will be added. Returns `NULL` if the input is not a `LINESTRING`, `CIRCULARLINESTRING`, or `NURBSCURVE`.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Set end M value of a LineString

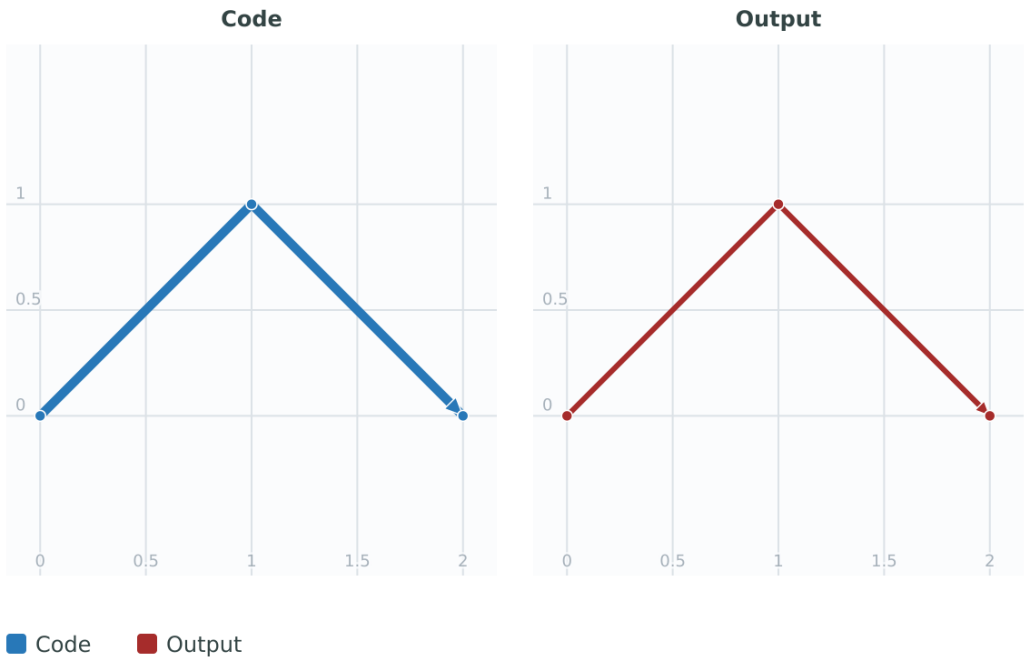
Code

```
SELECT ST_SetEndM('LINESTRING M(0 0 10,1 1 20,2 0 30)'::geometry, 999);
```

Output

```
LINESTRING M (0 0 10,1 1 20,2 0 999)
```

Figure



Add M dimension and set end M value

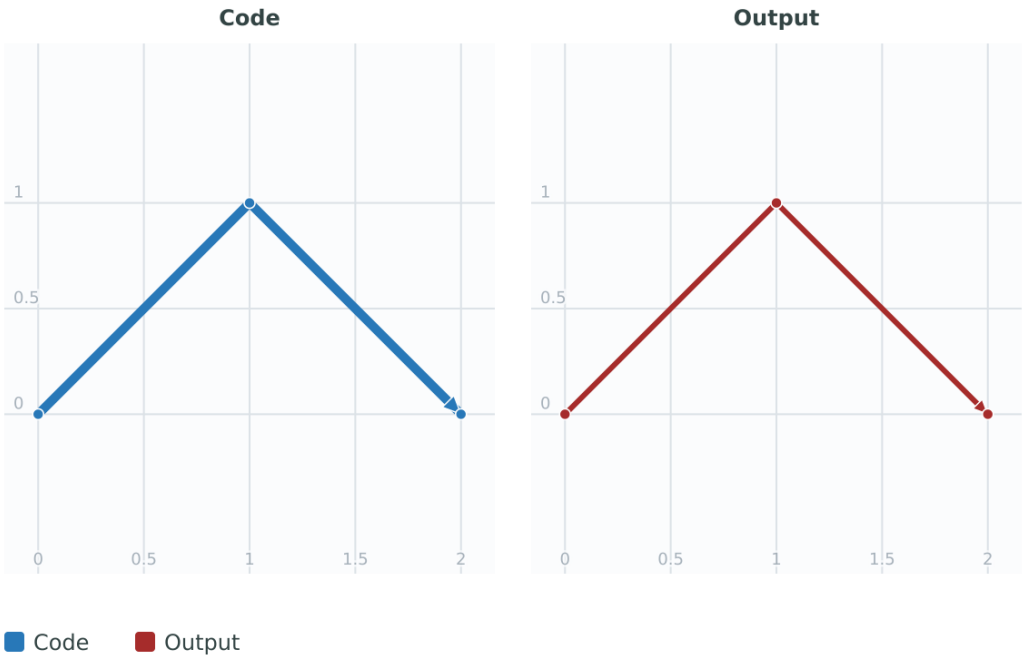
Code

```
SELECT ST_SetEndM('LINESTRING(0 0,1 1,2 0)'::geometry, 75);
```

Output

```
LINESTRING M (0 0 0,1 1 0,2 0 75)
```

Figure



Set end M value of a CircularString

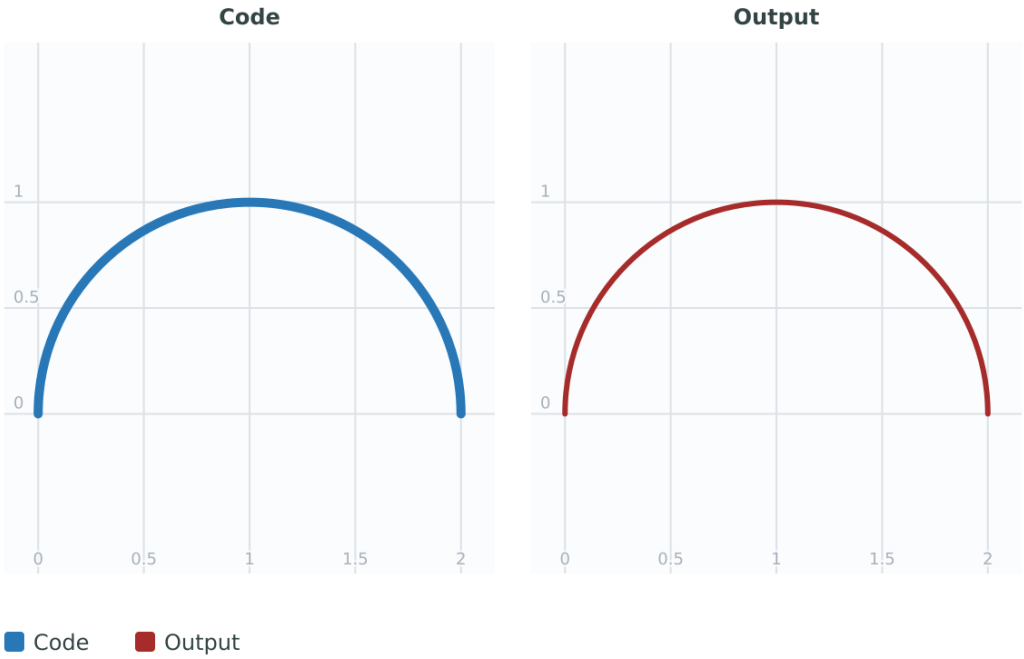
Code

```
SELECT ST_SetEndM('CIRCULARSTRING M(0 0 5,1 1 10,2 0 15)'::geometry, 88);
```

Output

```
CIRCULARSTRING M (0 0 5,1 1 10,2 0 88)
```

Figure



Set end M value of a NURBSCurve

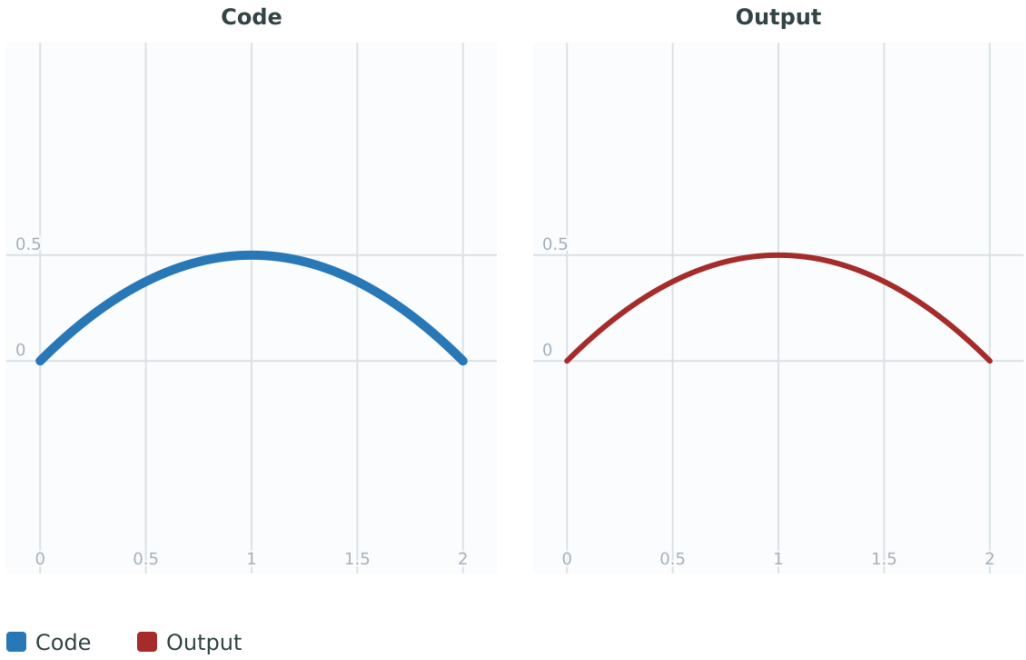
Code

```
SELECT ST_SetEndM('NURBSCURVE M(2, (0 0 100, 1 1 200, 2 0 300))'::geometry, 999);
```

Output

```
NURBSCURVE M (DEGREE 2,CONTROLPOINTS M (NURBSPOINT(WEIGHTEDPOINT M (0 0 100),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (1 1 ↩
      200),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT M (2 0 999),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3)))
```

Figure



Voir aussi

ST_SetStartM, ST_EndM, ST_EndPoint

7.4.44 ST_Summary

ST_Summary — Renvoie un résumé textuel du contenu d'une géométrie.

Synopsis

text **ST_Summary**(geometry g);
text **ST_Summary**(geography g);

Description

Renvoie un résumé textuel du contenu de la géométrie.

Les indicateurs indiqués entre crochets après le type de géométrie ont la signification suivante :

— M : possède une coordonnée M
— Z : possède une coordonnée Z
— B : possède une bounding box en cache

— G : est géodésique (geography)
— S : possède un système de référence spatiale

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Disponibilité : 1.2.2

Amélioré : la version 2.0.0 a ajouté la prise en charge du type geography

Amélioré : 2.1.0 Indicateur S pour indiquer si le système de référence spatiale est connu

Amélioré : 2.2.0 Ajout de la prise en charge des TIN et des courbes

Exemples

Code

```
SELECT ST_Summary('LINESTRING(0 0,1 1)'::geometry) as geom,
ST_Summary('POLYGON((0 0,1 1,1 2,1 1,0 0))'::geography) geog;
```

Output

```
geom              |          geog
------------------------------+--------------------------
 LineString[B] with 2 points | Polygon[BGS] with 1 rings
                              | ring 0 has 5 points
                              :
(1 row)
```

Code

```
SELECT ST_Summary('LINESTRING(0 0 1,1 1 1)'::geography) As geog_line,
ST_Summary('SRID=4326;POLYGON((0 0 1,1 1 2,1 2 3,1 1 1,0 0 1))'::geometry) As geom_poly;
```

Output

```
geog_line              |        geom_poly
------------------------------- +-------------------------
 LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
                                :    ring 0 has 5 points
                                :
(1 row)
```

Voir aussi

PostGIS_DropBBox, PostGIS_AddBBox, ST_Force3DM, ST_Force3DZ, ST_Force2D, geography

ST_IsValid, ST_IsValid, ST_IsValidReason, ST_IsValidDetail

7.4.45 ST_X

ST_X — Renvoie la coordonnée X d'un point.

Synopsis

float **ST_X**(geometry a_point);

Description

Renvoie la coordonnée X du point, ou NULL si elle n'est pas disponible. L'entrée doit être un point.



Note
Pour obtenir les valeurs X minimale et maximale des coordonnées géométriques, utilisez les fonctions ST_XMin et ST_XMax.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 6.1.3

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_X('POINT(1 2 3 4)');
```

Output

```
1
```

Code

```
SELECT ST_Y(ST_Centroid('LINESTRING(1 2 3 4,1 1 1 1)'::geometry));
```

Output

```
1.5
```

Voir aussi

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_XMax, ST_XMin, ST_Y, ST_Z

7.4.46 ST_Y

ST_Y — Renvoie la coordonnée Y d'un point.

Synopsis

float **ST_Y**(geometry a_point);

Description

Renvoie la coordonnée Y du point, ou NULL si elle n'est pas disponible. L'entrée doit être un point.



Note
Pour obtenir les valeurs minimale et maximale en Y des coordonnées géométriques, utilisez les fonctions ST_YMin et ST_YMax.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 6.1.4

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_Y('POINT(1 2 3 4)');
```

Output

```
2
```

Code

```
SELECT ST_Y(ST_Centroid('LINESTRING(1 2 3 4,1 1 1 1)'::geometry));
```

Output

```
1.5
```

Voir aussi

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_X, ST_YMax, ST_YMin, ST_Z

7.4.47 ST_Z

ST_Z — Renvoie la coordonnée Z d'un point.

Synopsis

float **ST_Z**(geometry a_point);

Description

Renvoie la coordonnée Z du point, ou NULL si elle n'est pas disponible. L'entrée doit être un point.



Note
Pour obtenir les valeurs minimale et maximale de Z des coordonnées géométriques, utilisez les fonctions ST_ZMin et ST_ZMax.

Cette méthode implémente la spécification SQL/MM.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_Z('POINT(1 2 3 4)');
```

Output

```
3
```

Voir aussi

ST_GeomFromEWKT, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.48 ST_ZMFlag

ST_ZMFlag — Retourne un code indiquant la dimension des coordonnées ZM d'une géométrie.

Synopsis

smallint **ST_ZMFlag**(geometry geomA);

Description

Retourne un code indiquant la dimension des coordonnées ZM d'une géométrie.

Les valeurs sont les suivantes : 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_ZMFlag('LINESTRING(1 2,3 4)');
```

Output

```
0
```

Code

```
SELECT ST_ZMFlag('LINESTRINGM(1 2 3, 3 4 3)');
```

Output

```
1
```

Code

```
SELECT ST_ZMFlag('CIRCULARSTRING(1 2 3,3 4 3,5 6 3)');
```

Output

```
2
```

Code

```
SELECT ST_ZMFlag('POINT(1 2 3 4)');
```

Output

```
3
```

Voir aussi

ST_CoordDim, ST_NDims, ST_Dimension

7.4.49 ST_HasZ

ST_HasZ — Vérifie si une géométrie possède une dimension Z.

Synopsis

boolean **ST_HasZ**(geometry geom);

Description

Vérifie si la géométrie d'entrée a une dimension Z et renvoie une valeur booléenne. Si la géométrie a une dimension Z, le résultat est vrai ; sinon, il est faux.

Les objets géométriques dotés d'une dimension Z représentent généralement des géométries tridimensionnelles (3D), tandis que ceux qui en sont dépourvus sont des géométries bidimensionnelles (2D).

Cette fonction est utile pour déterminer si une géométrie contient des informations sur l'altitude ou la hauteur.

Disponibilité : 3.5.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Exemples

Code

```
SELECT ST_HasZ('POINT(1 2 3)');
```

Output

```
t
```

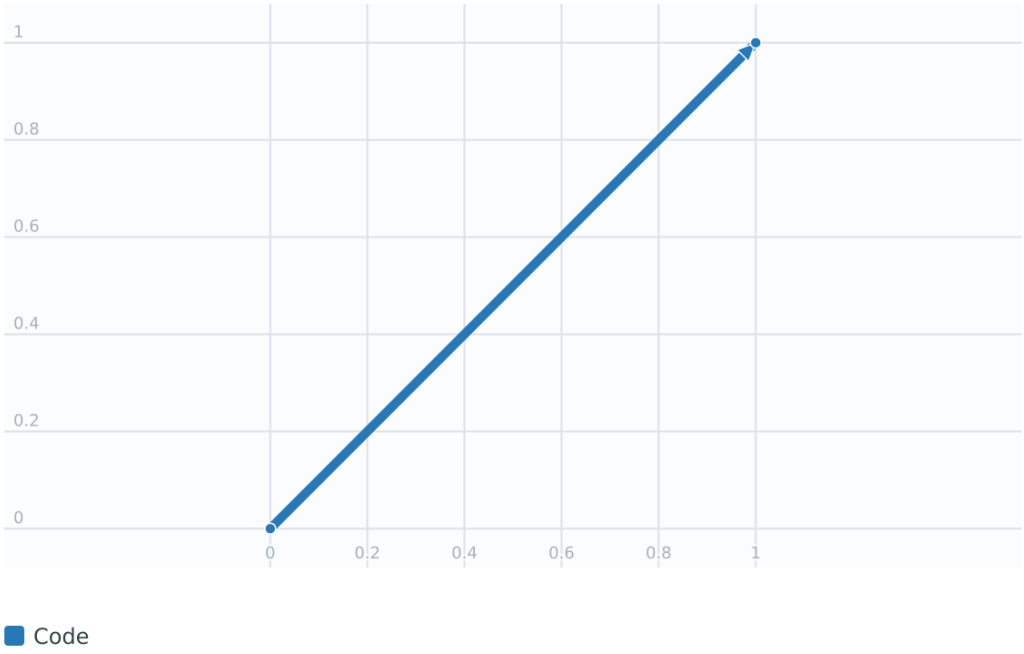
Code

```
SELECT ST_HasZ('LINESTRING(0 0,1 1)');
```

Output

```
f
```

Figure



Voir aussi

ST_ZMFlag

ST_HasM

7.4.50 ST_HasM

ST_HasM — Vérifie si une géométrie a une dimension M (mesure).

Synopsis

boolean **ST_HasM**(geometry geom);

Description

Vérifie si la géométrie d'entrée a une dimension M (mesure) et renvoie une valeur booléenne. Si la géométrie a une dimension M, il renvoie vrai ; sinon, il renvoie faux.

Les objets géométriques de dimension M représentent généralement des mesures ou des données supplémentaires associées à des entités spatiales.

Cette fonction est utile pour déterminer si une géométrie contient des informations de mesure.

Disponibilité : 3.5.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Exemples

Code

```
SELECT ST_HasM('POINTM(1 2 3)');
```

Output

```
t
```

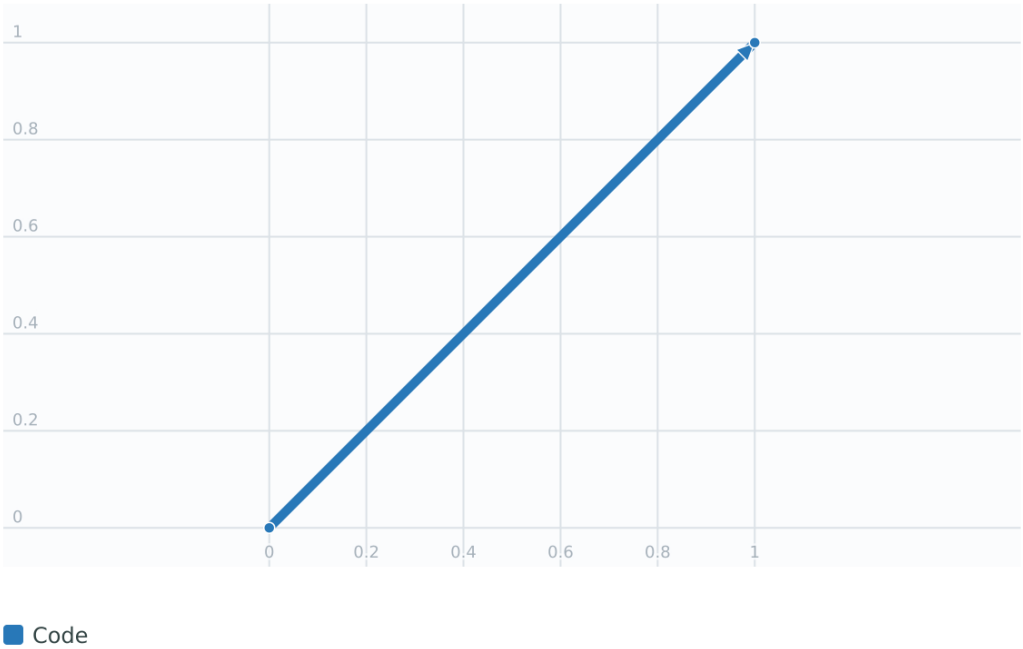
Code

```
SELECT ST_HasM('LINESTRING(0 0,1 1)');
```

Output

```
f
```

Figure



Voir aussi

ST_ZMFlag
ST_HasZ

7.4.51 ST_ControlPoints

ST_ControlPoints — Returns the control points of a NURBS curve as a MULTIPOINT geometry.

Synopsis

geometry **ST_ControlPoints**(geometry nurbscurve);

Description

Extracts the control points from a NURBS curve and returns them as a MULTIPOINT geometry. Control points define the control polygon that influences the shape of the NURBS curve.

The function raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Exemples

This example extracts control points from a NURBS curve.

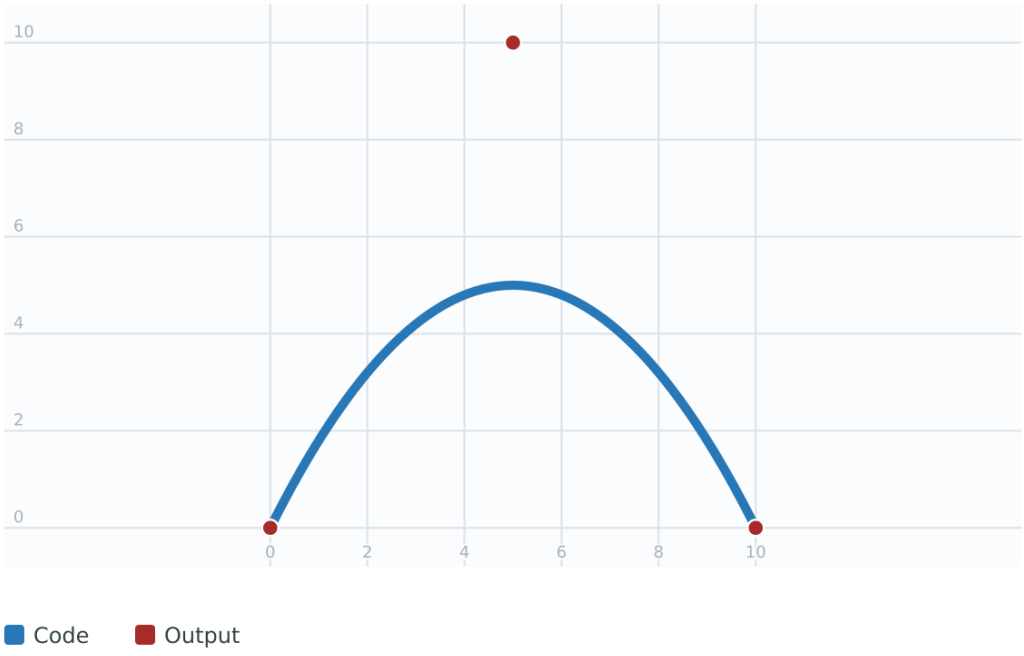
Code

```
SELECT ST_ControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
MULTIPOINT((0 0),(5 10),(10 0))
```

Figure



This example counts the control points.

Code

```
SELECT ST_NPoints(ST_ControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry));
```

Output

```
3
```

Voir aussi

ST_MakeNurbsCurve, ST_NumControlPoints, ST_Weights

7.4.52 ST_Degree

ST_Degree — Returns the polynomial degree of a NURBS curve.

Synopsis

integer **ST_Degree**(geometry nurbscurve);

Description

Returns the polynomial degree of a NURBS curve as an integer. The degree determines the curve's smoothness and flexibility. Higher degrees produce smoother curves but require more control points.

The function raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Exemples

This example returns the degree of a NURBS curve.

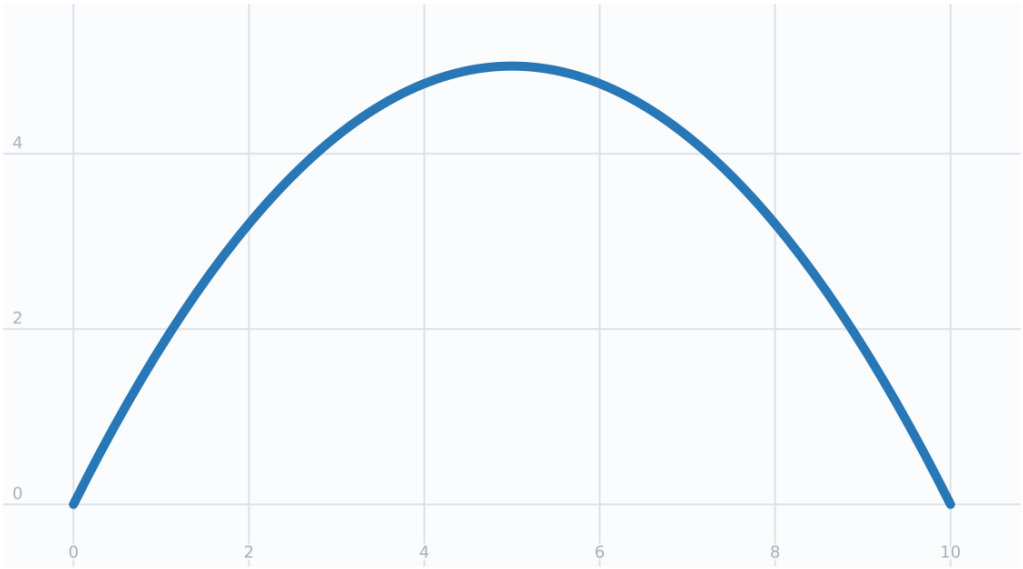
Code

```
SELECT ST_Degree('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
2
```

Figure



This example checks whether a NURBS curve is cubic.

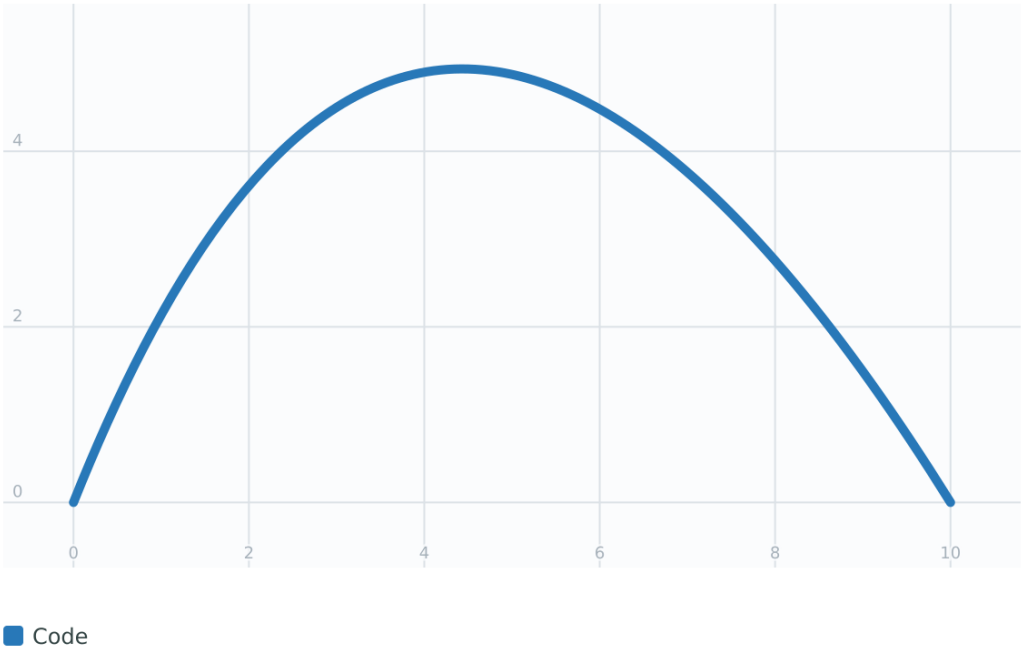
Code

```
SELECT ST_Degree('NURBSCURVE(3, (0 0, 2 5, 5 8, 10 0))'::geometry) = 3;
```

Output

```
t
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_NumControlPoints

7.4.53 ST_Weights

ST_Weights — Returns the weight array of a rational NURBS curve.

Synopsis

float8[] **ST_Weights**(geometry nurbscurve);

Description

Returns the weight values of a rational NURBS curve as an array of double precision values. Weights control the influence of each control point on the curve shape. Higher weights pull the curve closer to the corresponding control point.

Returns NULL if the curve has no weights (non-rational B-spline) or if the input is NULL. Raises an error if the input is not a NURBS curve.

Availability: 3.7.0

Exemples

This example returns weights from a rational NURBS curve.

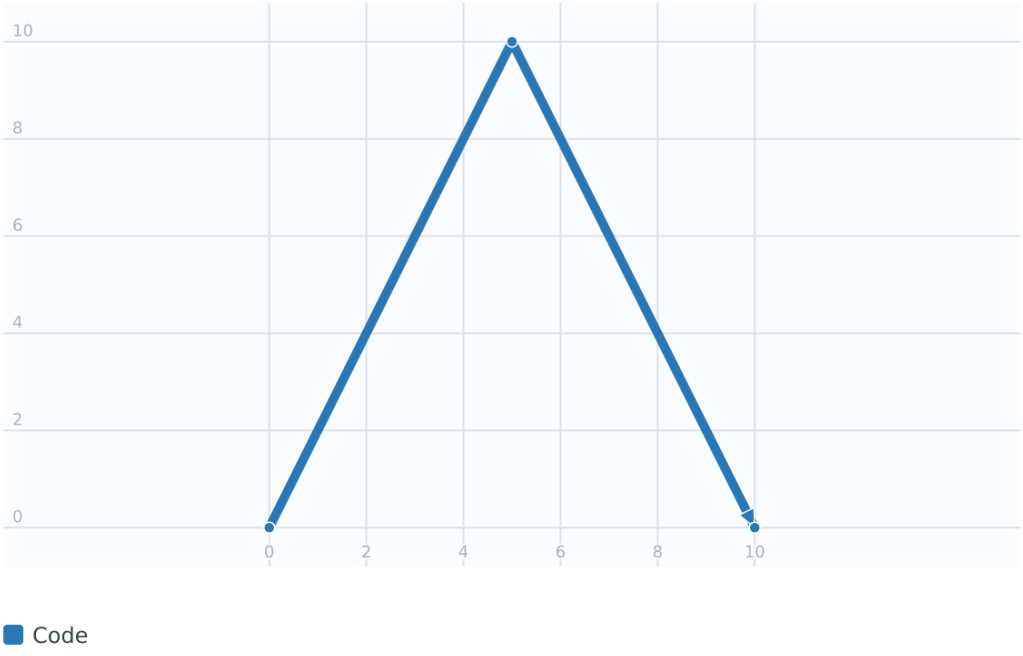
Code

```
SELECT ST_Weights(ST_MakeNurbsCurve(2, 'LINESTRING(0 0,5 10,10 0)'::geometry, ARRAY[1.0,  ↩
    2.0, 1.0]));
```

Output

```
{1,2,1}
```

Figure



This example checks the weights of a non-rational curve.

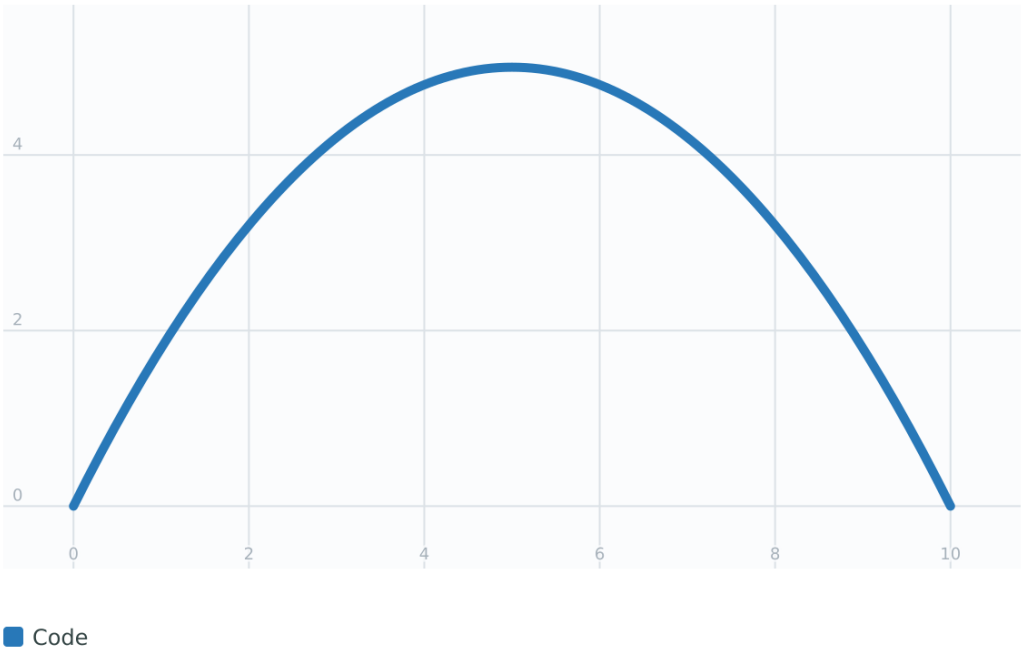
Code

```
SELECT ST_Weights('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
null
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_NurbsCurveIsRational, ST_ControlPoints

7.4.54 ST_Knots

ST_Knots — Returns the knot vector of a NURBS curve.

Synopsis

float8[] **ST_Knots**(geometry nurbscurve);

Description

Returns the knot vector of a NURBS curve as an array of double precision values. The knot vector defines the parameter space and controls where and how the control points affect the curve. It must be a non-decreasing sequence with length equal to (number of control points + degree + 1).

Returns NULL if the curve uses the default uniform clamped knot vector or if the input is NULL. Raises an error if the input is not a NURBS curve.

Availability: 3.7.0

Exemples

This example returns knots from a NURBS curve with a custom knot vector.

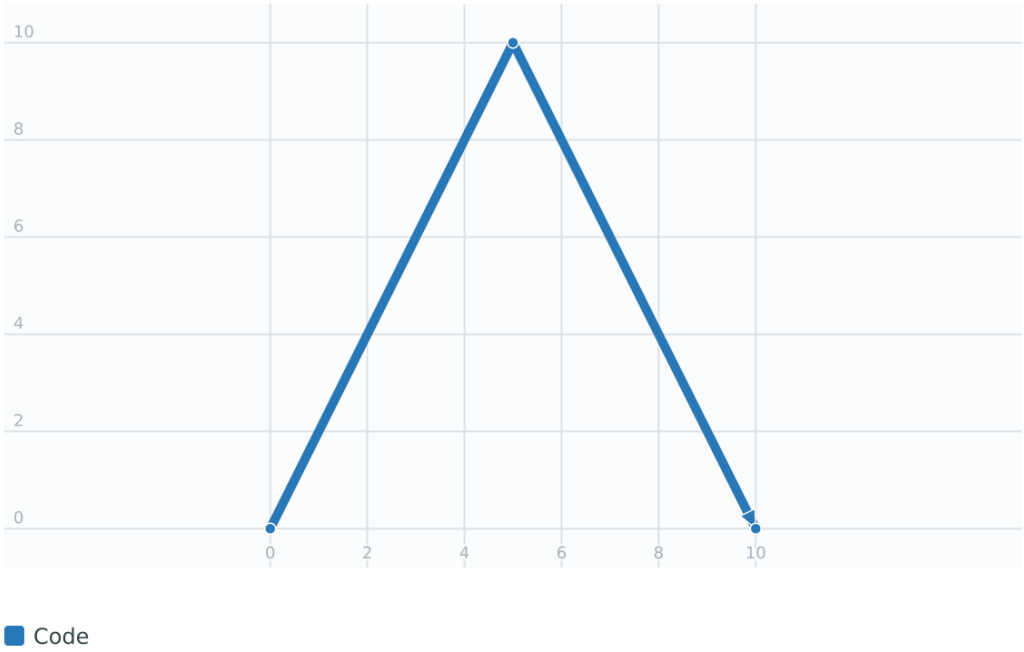
Code

```
SELECT ST_Knots(ST_MakeNurbsCurve(
        2,
        'LINESTRING(0 0,5 10,10 0)'::geometry,
        ARRAY[1.0, 2.0, 1.0],
        ARRAY[0, 0, 0, 1, 1, 1]));
```

Output

```
{0,0,0,1,1,1}
```

Figure



This example checks the knots of a curve using the default knot vector.

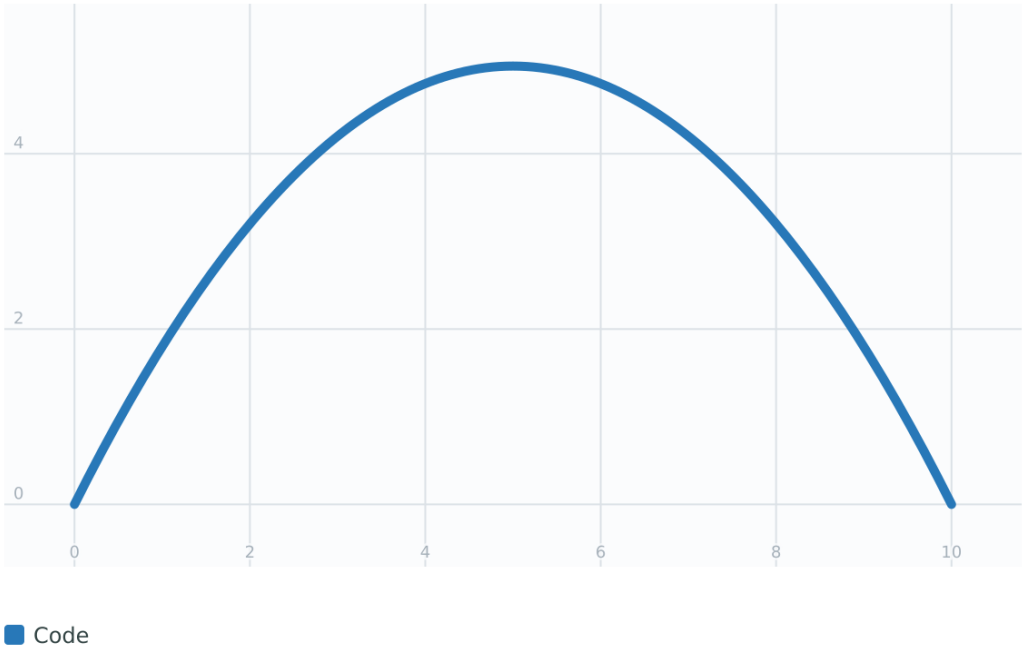
Code

```
SELECT ST_Knots('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
null
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_Degree, ST_NumControlPoints

7.4.55 ST_NumControlPoints

ST_NumControlPoints — Returns the number of control points in a NURBS curve.

Synopsis

integer **ST_NumControlPoints**(geometry nurbscurve);

Description

Returns the number of control points in a NURBS curve. The number of control points must be at least (degree + 1) for a valid NURBS curve.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Exemples

This example returns the number of control points.

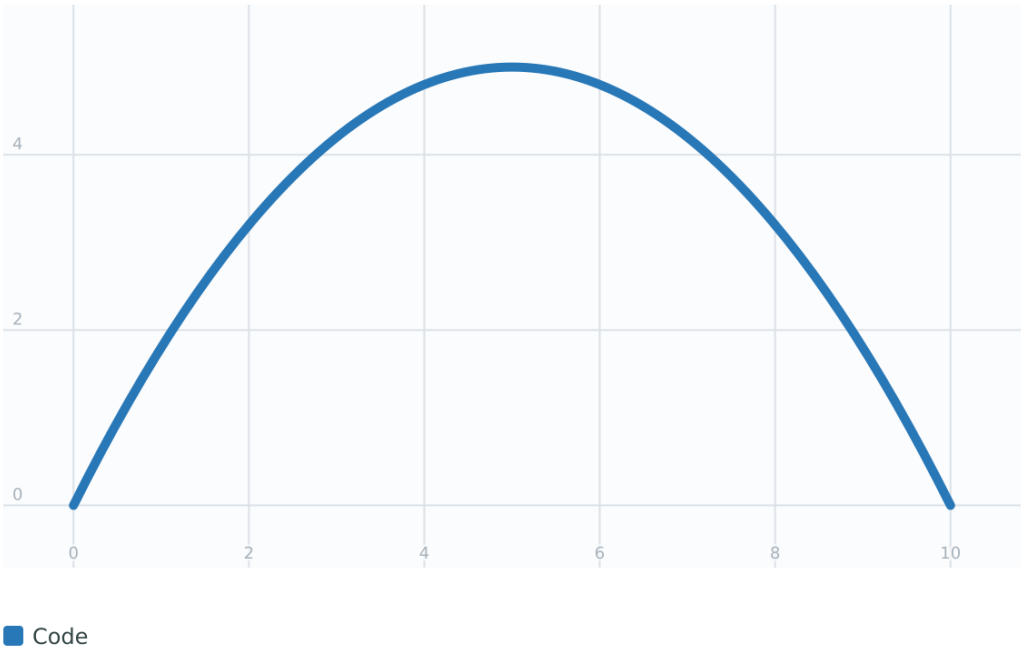
Code

```
SELECT ST_NumControlPoints('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
3
```

Figure



This example verifies that a curve has enough control points for its degree.

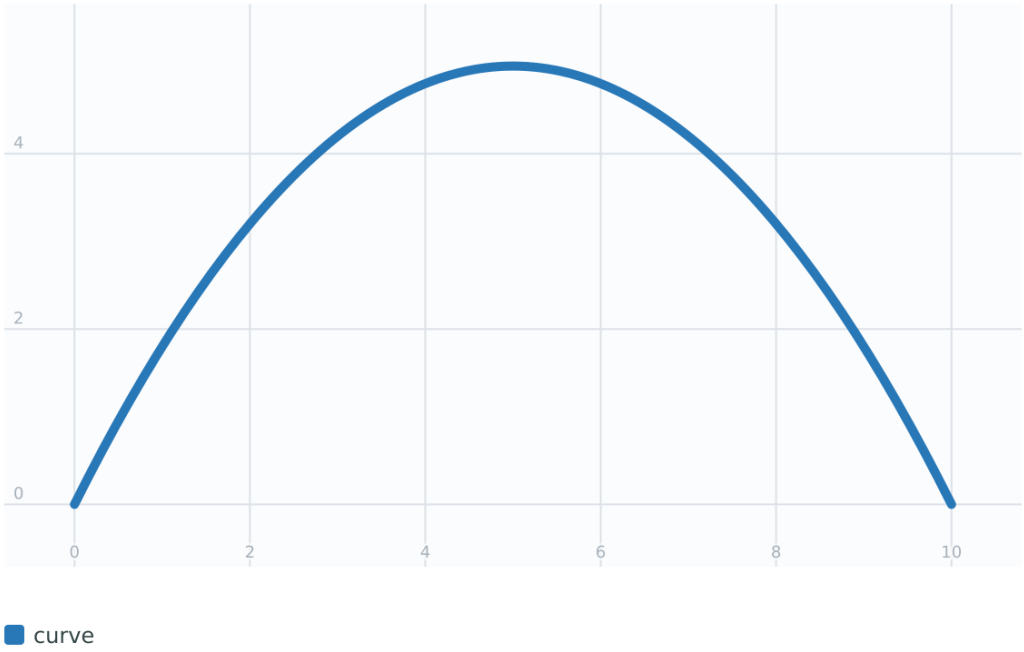
Code

```
SELECT ST_NumControlPoints(curve)
>= ST_Degree(curve) + 1 AS valid
FROM (SELECT 'NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry AS curve) AS t;
```

Output

```
t
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_ControlPoints, ST_Degree

7.4.56 ST_NurbsCurveIsRational

ST_NurbsCurveIsRational — Checks if a NURBS curve is rational (has weights).

Synopsis

boolean **ST_NurbsCurveIsRational**(geometry nurbscurve);

Description

Returns TRUE if the NURBS curve has weights assigned to its control points (rational NURBS), FALSE if it uses uniform weights (non-rational B-spline).

Rational NURBS curves provide more flexibility in curve shape control through variable weights. Non-rational B-splines use uniform weights and are a special case of NURBS curves.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Exemples

This example checks whether a curve is rational.

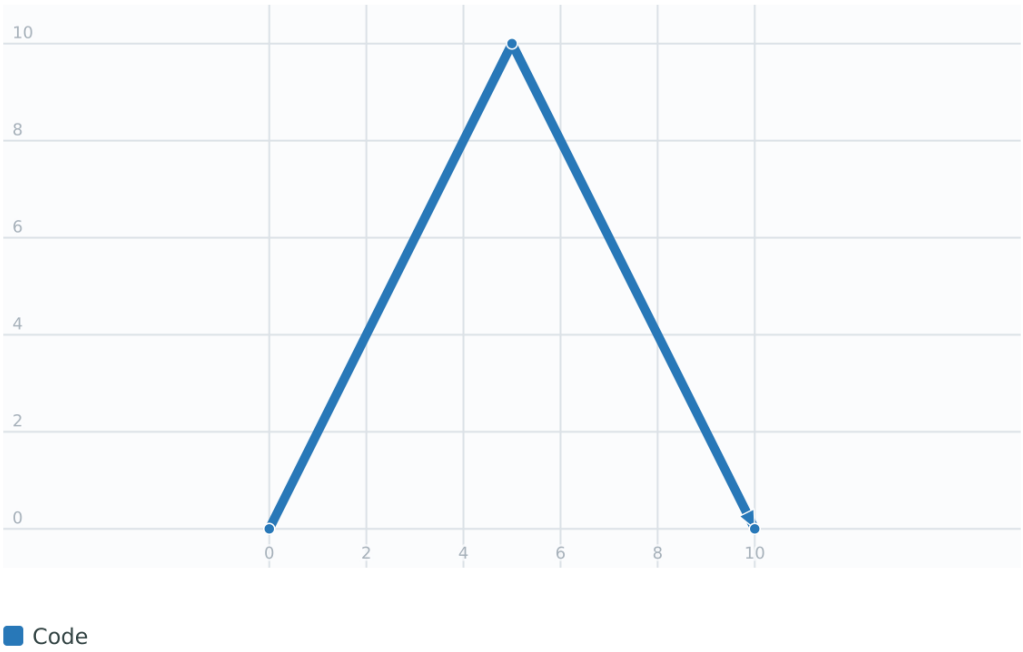
Code

```
SELECT ST_NurbsCurveIsRational(ST_MakeNurbsCurve(
        2,
        'LINESTRING(0 0,5 10,10 0)'::geometry,
        ARRAY[1.0, 2.0, 1.0]));
```

Output

```
t
```

Figure



This example checks a non-rational curve.

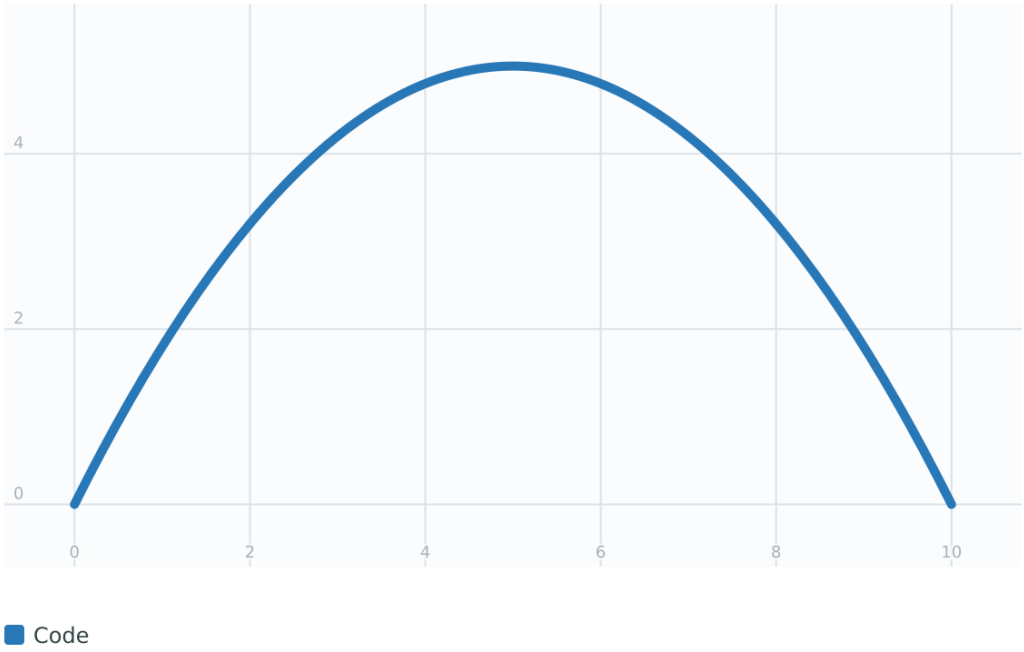
Code

```
SELECT ST_NurbsCurveIsRational('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
f
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_Weights, ST_IsNurbsCurve

7.4.57 ST_IsNurbsCurve

ST_IsNurbsCurve — Checks if a geometry is a NURBS curve.

Synopsis

boolean **ST_IsNurbsCurve**(geometry geom);

Description

Returns TRUE if the geometry is a NURBS curve, FALSE otherwise. This function can be used to test geometry types before applying NURBS-specific functions.

Returns NULL if the input is NULL.

Availability: 3.7.0

Exemples

This example checks whether a geometry is a NURBS curve.

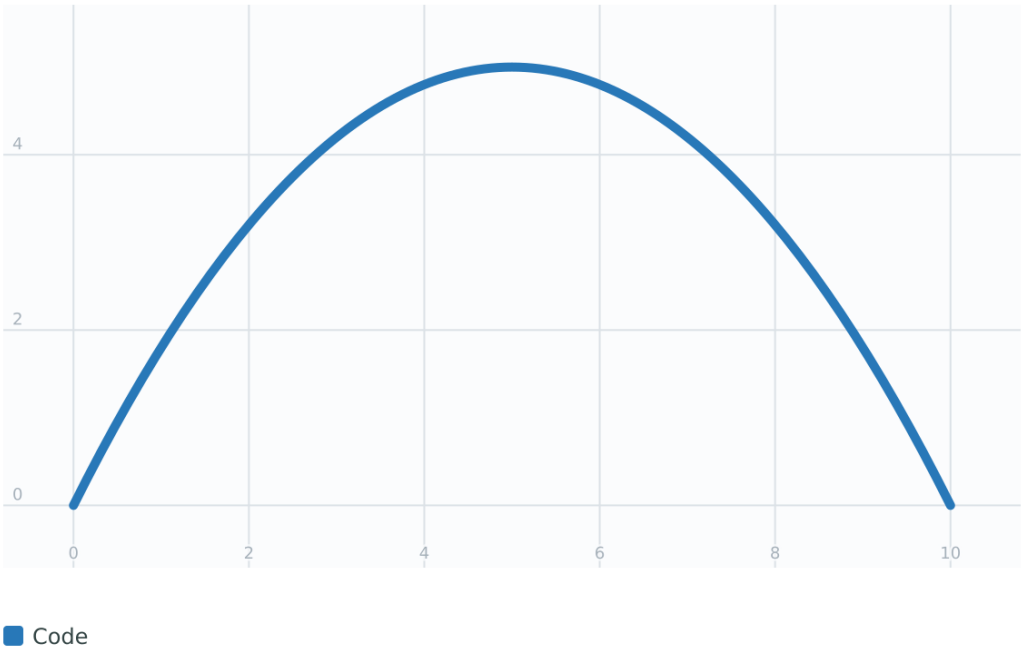
Code

```
SELECT ST_IsNurbsCurve('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
t
```

Figure



This example tests a regular LineString.

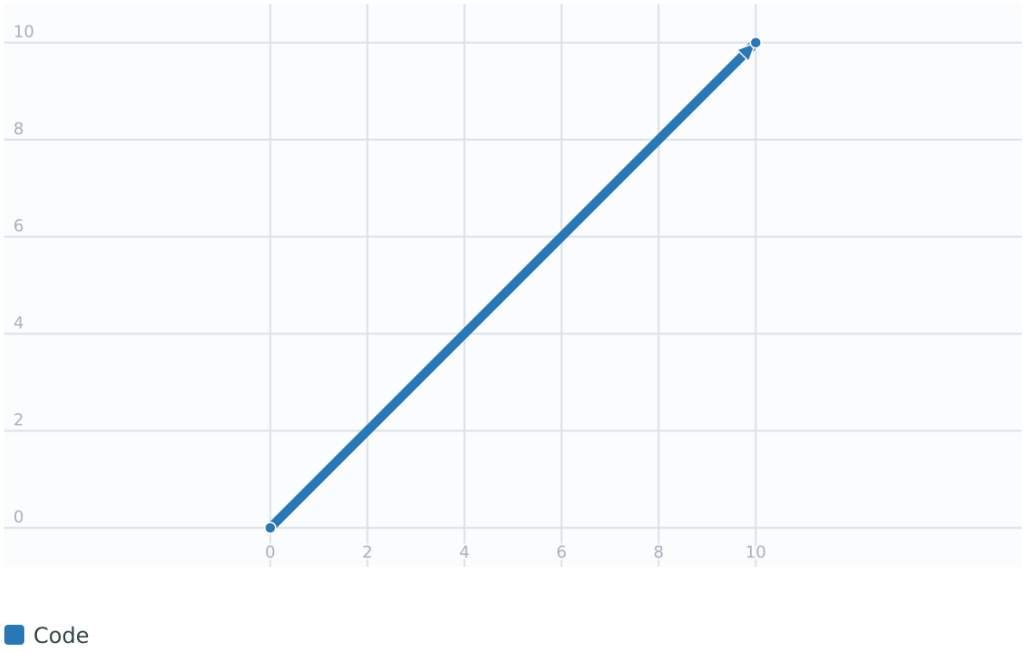
Code

```
SELECT ST_IsNurbsCurve('LINESTRING(0 0,10 10)'::geometry);
```

Output

```
f
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_GeometryType, ST_NurbsCurveIsRational

7.4.58 ST_Evaluate

ST_Evaluate — Evaluates a NURBS curve at a specific parameter value and returns the resulting point.

Synopsis

geometry **ST_Evaluate**(geometry nurbscurve, float8 parameter);

Description

Evaluates a NURBS curve at the specified parameter value and returns a POINT geometry representing the position on the curve. The parameter typically ranges from 0.0 (start of curve) to 1.0 (end of curve), though the actual range depends on the knot vector.

Raises an error if the input is not a NURBS curve or if evaluation fails. Returns NULL if either input is NULL.

Availability: 3.7.0

Exemples

This example returns the midpoint of a NURBS curve.

Code

```
SELECT ST_Evaluate('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 0.5);
```

This example samples multiple points along a curve.

Code

```
SELECT t, ST_Evaluate(curve, t) AS point
FROM (SELECT 'NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry AS curve) AS c,
         generate_series(0, 1, 0.25) AS t;
```

Voir aussi

ST_MakeNurbsCurve, ST_NurbsToLineString

7.4.59 ST_NurbsToLineString

ST_NurbsToLineString — Converts a NURBS curve to a LINESTRING by uniform sampling.

Synopsis

geometry **ST_NurbsToLineString**(geometry nurbscurve, integer num_segments=32);

Description

Converts a NURBS curve to a LINESTRING geometry by creating a piecewise linear approximation. The curve is sampled at uniformly distributed parameter values and the resulting points are connected with straight line segments.

The optional `num_segments` parameter specifies the number of line segments in the output (default: 32, range: 2-10000). Higher values produce smoother approximations but increase geometry complexity.

Raises an error if the input is not a NURBS curve. Returns NULL if the input is NULL.

Availability: 3.7.0

Exemples

This example converts a NURBS curve to a LineString using the default quality.

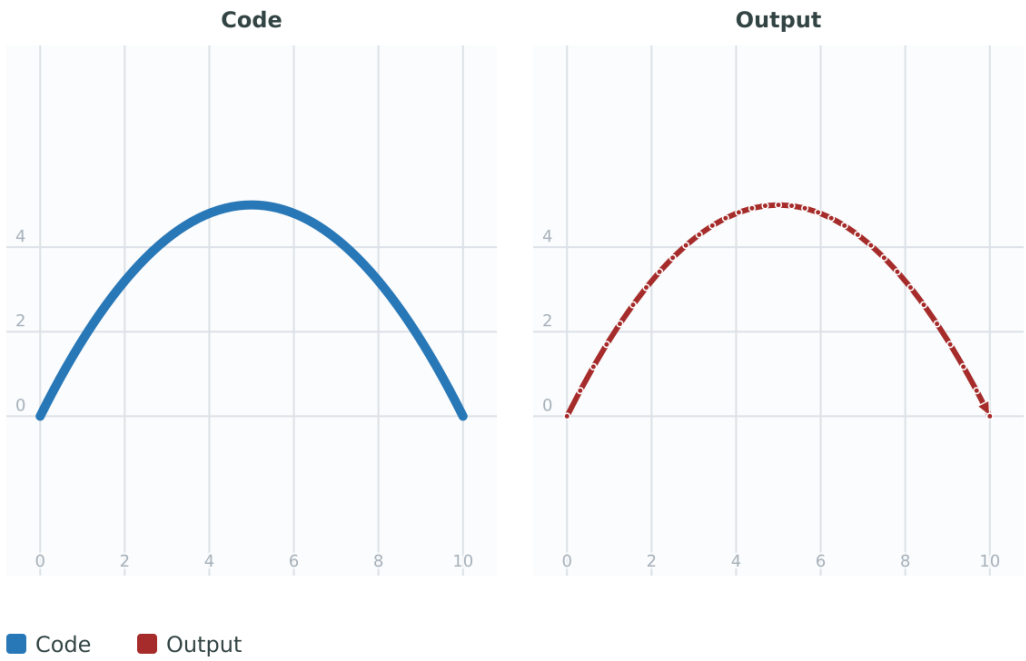
Code

```
SELECT ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry);
```

Output

```
LINESTRING(0 0,0.3125 0.60546875,0.625 1.171875,0.9375 1.69921875,1.25 2.1875,1.5625 2.63671875,1.875 3.046875,2.1875 ↩
    3.41796875,2.5 3.75,2.8125 4.04296875,3.125 4.296875,3.4375 4.51171875,3.75 4.6875,4.0625 4.82421875,4.375 ↩
    4.921875,4.6875 4.98046875,5 5,5.3125 4.98046875,5.625 4.921875,5.9375 4.82421875,6.25 4.6875,6.5625 ↩
    4.51171875,6.875 4.296875,7.1875 4.04296875,7.5 3.75,7.8125 3.41796875,8.125 3.046875,8.4375 2.63671875,8.75 ↩
    2.1875,9.0625 1.69921875,9.375 1.171875,9.6875 0.60546875,10 0)
```

Figure

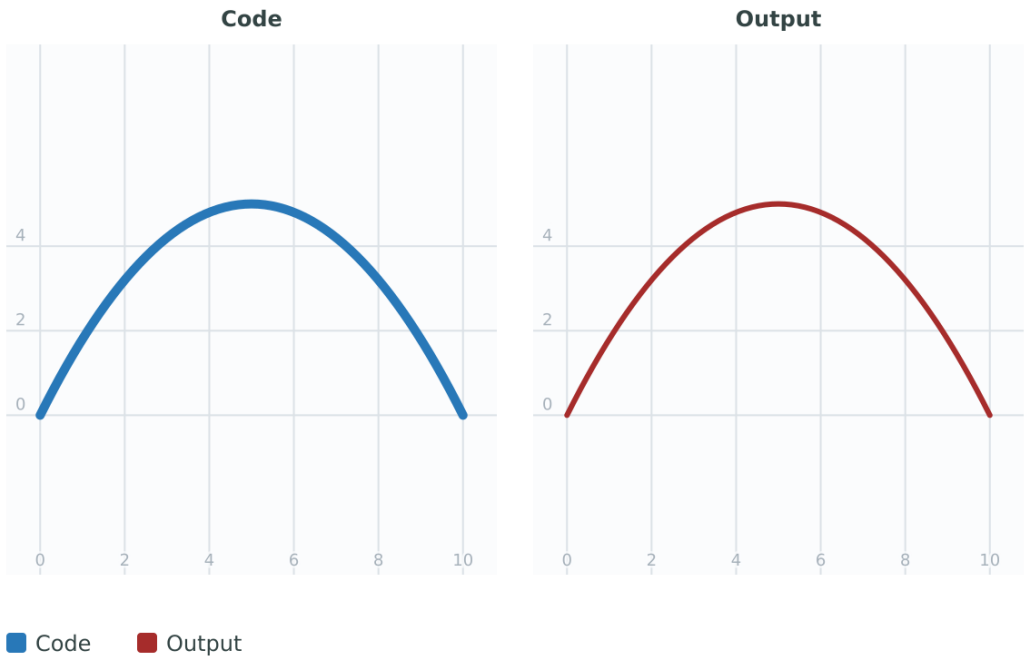


This example creates a high-quality approximation with 64 segments.

Code

```
SELECT ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 64);
```

Figure



This example creates a lower-quality, faster approximation with 16 segments.

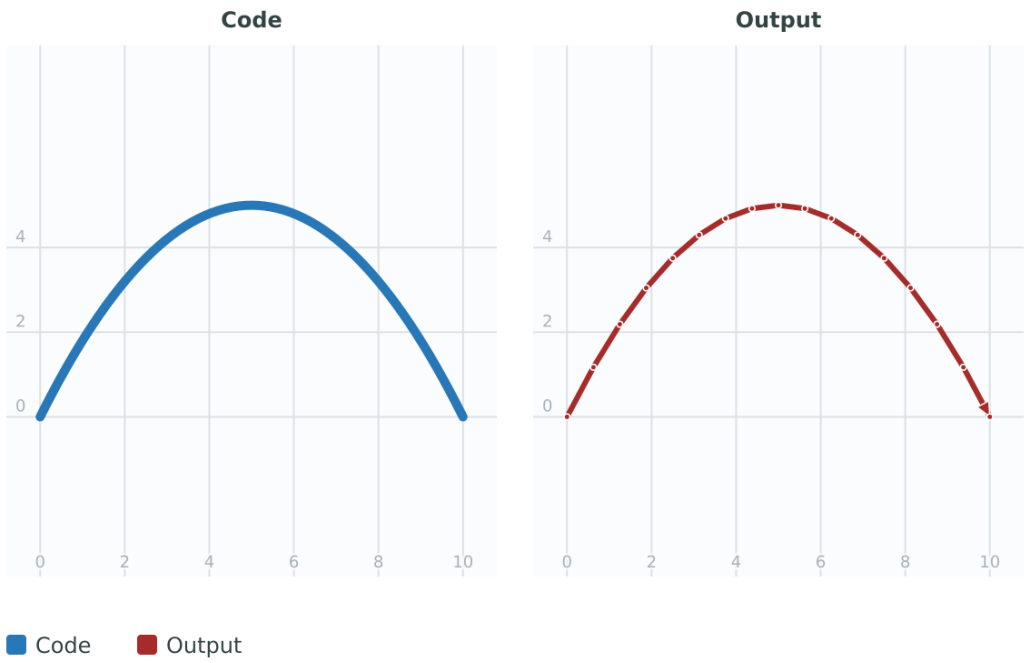
Code

```
SELECT ST_NurbsToLineString('NURBSCURVE(2, (0 0, 5 10, 10 0))'::geometry, 16);
```

Output

```
LINESTRING(0 0,0.625 1.171875,1.25 2.1875,1.875 3.046875,2.5 3.75,3.125 4.296875,3.75 4.6875,4.375 4.921875,5 5,5.625 ↩
    4.921875,6.25 4.6875,6.875 4.296875,7.5 3.75,8.125 3.046875,8.75 2.1875,9.375 1.171875,10 0)
```

Figure



Voir aussi

ST_MakeNurbsCurve, ST_Evaluate, ST_Segmentize

7.5 Éditeurs de géométrie

7.5.1 ST_AddPoint

ST_AddPoint — Ajoute un point à une LineString.

Synopsis

geometry **ST_AddPoint**(geometry linestring, geometry point);

geometry **ST_AddPoint**(geometry linestring, geometry point, integer position = -1);

Description

Ajoute un point à une LineString avant l'index *position* (en utilisant un index basé sur 0). Si le paramètre *position* est omis ou vaut -1, le point est ajouté à la fin de la LineString.

Disponibilité : 1.1.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Ajouter un point à la fin d'une ligne 3D

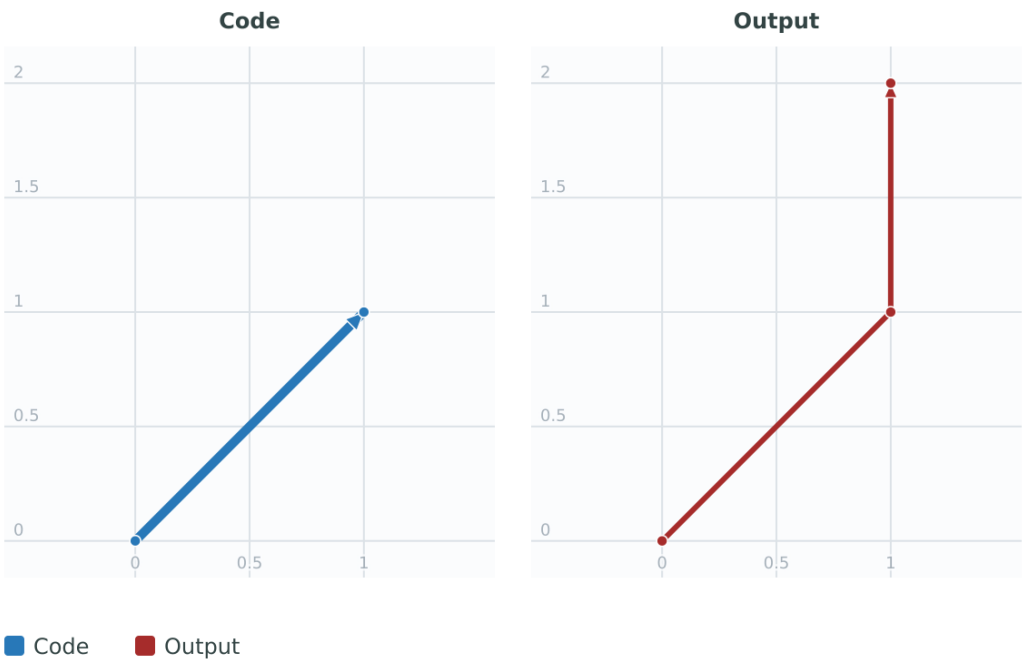
Code

```
SELECT ST_AddPoint('LINESTRING(0 0 1,1 1 1)', ST_MakePoint(1, 2, 3));
```

Output

```
LINESTRING(0 0 1,1 1 1,1 2 3)
```

Figure



Garantir que toutes les lignes d'une table sont fermées en ajoutant le point de départ de chaque ligne à la fin de la ligne uniquement pour celles qui ne sont pas fermées.

Code

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

Voir aussi

ST_RemovePoint, ST_SetPoint

7.5.2 ST_CollectionExtract

ST_CollectionExtract — Pour une collection de géométries spécifiée, renvoie une multi-géométrie contenant uniquement des éléments d'un type spécifié.

Synopsis

geometry **ST_CollectionExtract**(geometry collection);

geometry **ST_CollectionExtract**(geometry collection, integer type);

Description

Pour une collection de géométries spécifiée, renvoie une multi-géométrie homogène.

Si le `type` n'est pas spécifié, il renvoie une multi-géométrie contenant uniquement des géométries de la dimension la plus élevée. Les polygones sont donc préférés aux lignes, qui sont préférées aux points.

Si le `type` est spécifié, renvoie une multi-géométrie contenant uniquement ce type. S'il n'y a pas de sous-géométrie du bon type, une géométrie VIDE est renvoyée. Seuls les points, les lignes et les polygones sont pris en charge. Les numéros de type sont les suivants :

— 1 == POINT
— 2 == LINESTRING
— 3 == POLYGON

Pour les entrées de géométrie atomique, la géométrie est restituée inchangée si le type d'entrée correspond au type demandé. Sinon, le résultat est une géométrie VIDE du type spécifié. Si nécessaire, ces géométries peuvent être converties en géométries multiples en utilisant ST_Multi.



AVERTISSEMENT
La validité des résultats MultiPolygon n'est pas vérifiée. Si les composants du polygone sont adjacents ou se chevauchent, le résultat ne sera pas valide. (Par exemple, cela peut se produire lorsque cette fonction est appliquée à un résultat ST_Split). Cette situation peut être vérifiée avec ST_IsValid et réparée avec ST_MakeValid.

Disponibilité : 1.5.0



Note
Avant la version 1.5.3, cette fonction renvoyait les entrées atomiques inchangées, quel que soit leur type. Dans la version 1.5.3, les géométries simples non correspondantes renvoyaient un résultat NULL. Dans la version 2.0.0, les géométries simples non correspondantes renvoient un résultat VIDE du type demandé.

Exemples

Extraire le type de dimension le plus élevé :

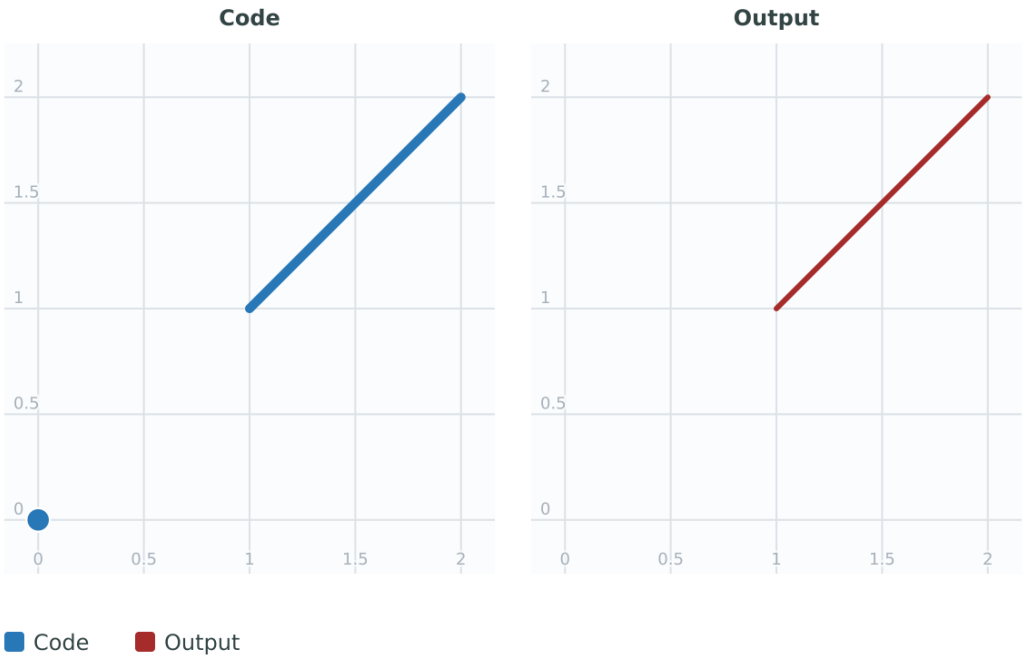
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1,2 2) )');
```

Output

```
MULTILINESTRING((1 1, 2 2))
```

Figure



Extraire des points (type 1 == POINT) :

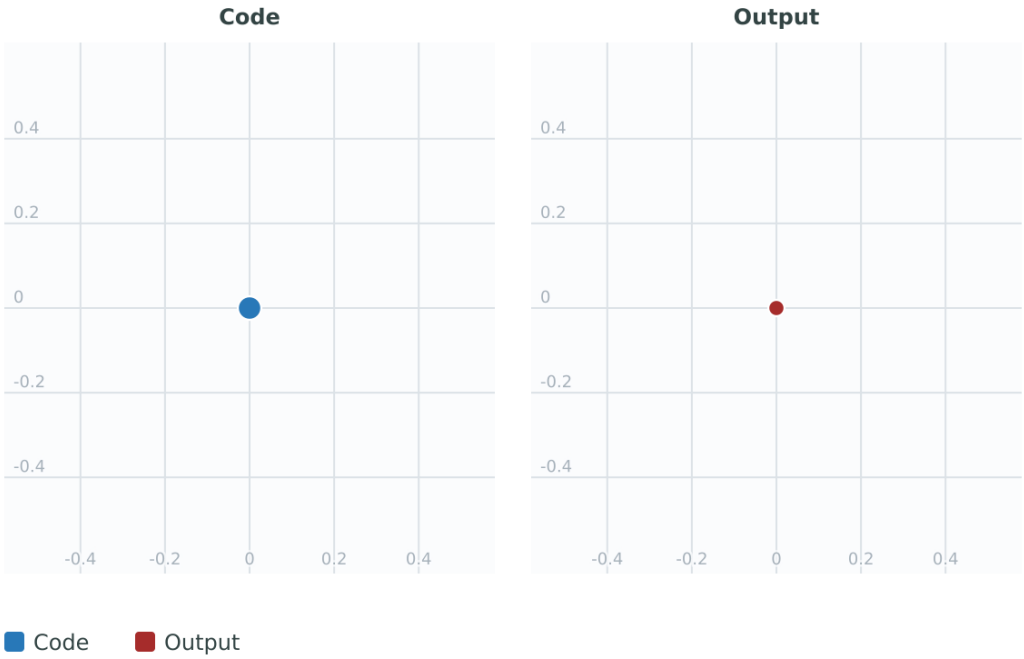
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0)))',
        1);
```

Output

```
MULTIPOINT((0 0))
```

Figure



Extraire des lignes (type 2 == LINESTRING) :

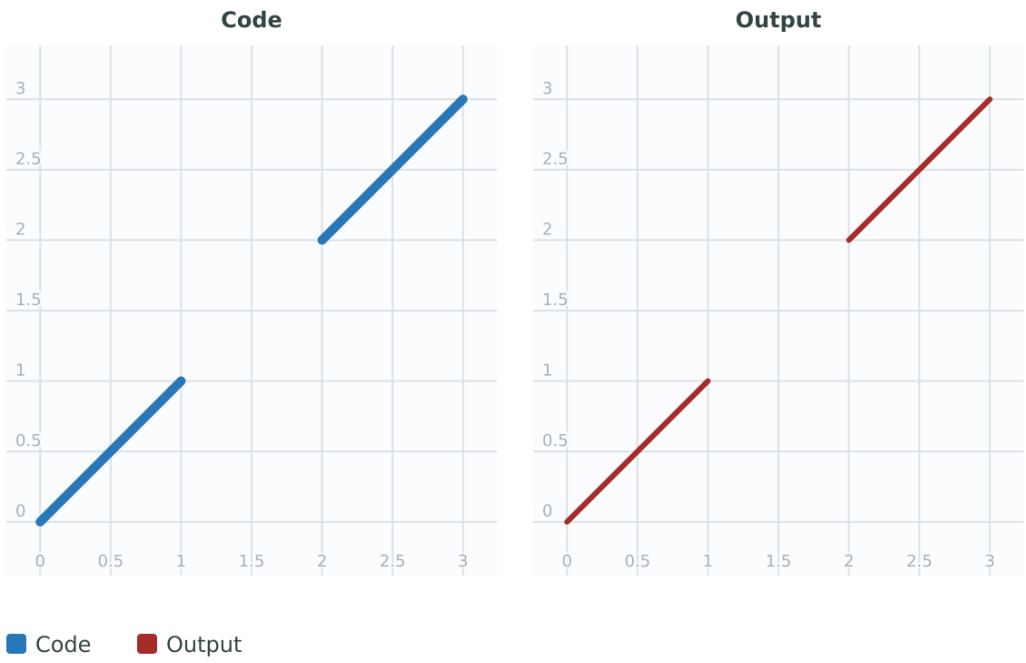
Code

```
SELECT ST_CollectionExtract('GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0,1 1)),  ↩
    LINESTRING(2 2,3 3))',
        2);
```

Output

```
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

Figure



Voir aussi

ST_CollectionHomogenize, ST_Multi, ST_IsValid, ST_MakeValid

7.5.3 ST_CollectionHomogenize

ST_CollectionHomogenize — Renvoie la représentation la plus simple d'une collection de géométries.

Synopsis

geometry **ST_CollectionHomogenize**(geometry collection);

Description

Pour une collection de géométries spécifiée, renvoie la représentation la plus simple du contenu.

— Les collections homogènes (uniformes) sont renvoyées sous la forme de la multi-géométrie appropriée.
— Les collections hétérogènes (mixtes) sont écrasées en une seule GeometryCollection.
— Les collections contenant un seul élément atomique sont renvoyées sous la forme de cet élément.

— Les géométries atomiques sont renvoyées telles quelles. Si nécessaire, elles peuvent être converties en une géométrie multiple en utilisant ST_Multi.



AVERTISSEMENT
Cette fonction ne garantit pas la validité du résultat. En particulier, une collection contenant des polygones adjacents ou se chevauchant créera un MultiPolygon invalide. Cette situation peut être vérifiée avec ST_IsValid et réparée avec ST_MakeValid.

Disponibilité : 2.0.0

Exemples

Collection d'un seul élément convertie en géométrie atomique

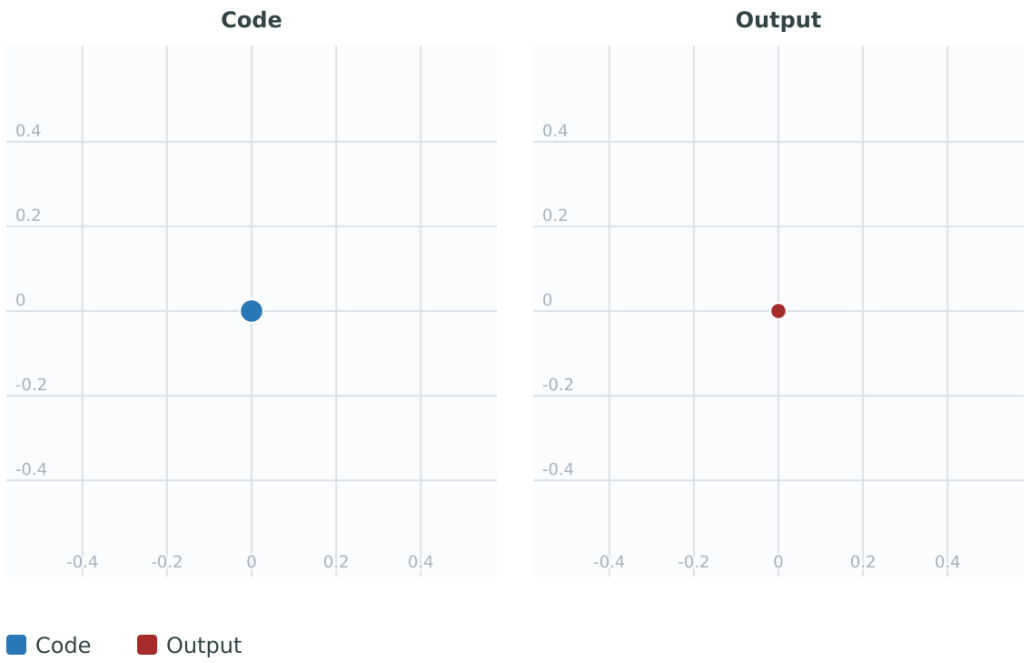
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))');
```

Output

```
POINT(0 0)
```

Figure



Collection imbriquée d'éléments simples convertie en géométrie atomique :

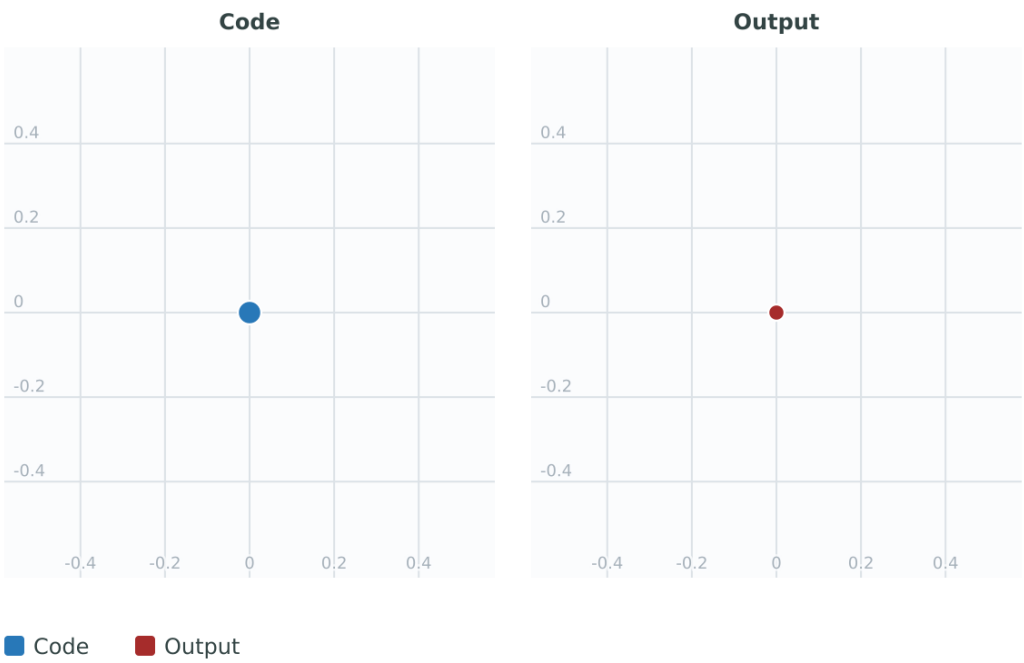
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))');
```

Output

```
POINT(0 0)
```

Figure



Collection convertie en une géométrie multiple :

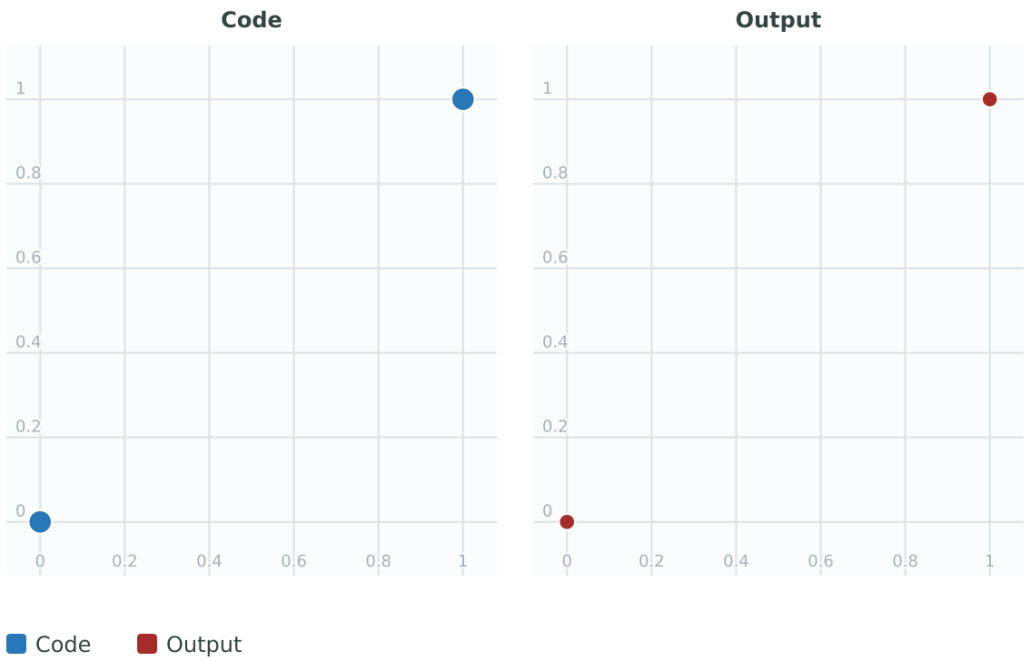
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), POINT(1 1))');
```

Output

```
MULTIPOINT((0 0),(1 1))
```

Figure



Collection hétérogène imbriquée écrasée en une GeometryCollection :

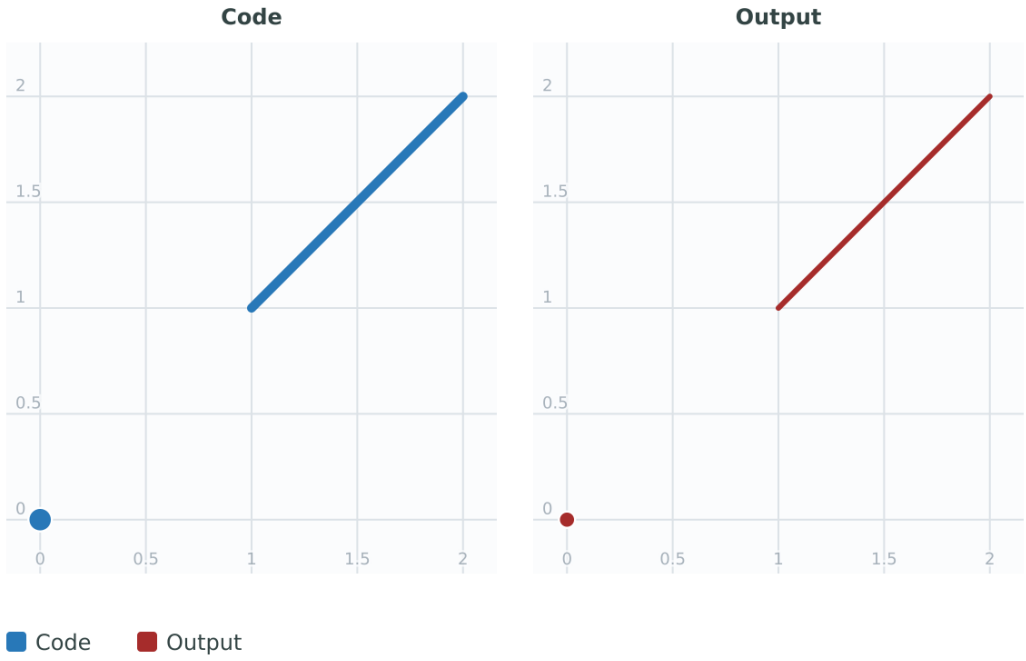
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION(
    LINESTRING(1 1,2 2)))');
```

Output

```
GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(1 1,2 2))
```

Figure



Collection de polygones convertis en un MultiPolygon (non valide) :

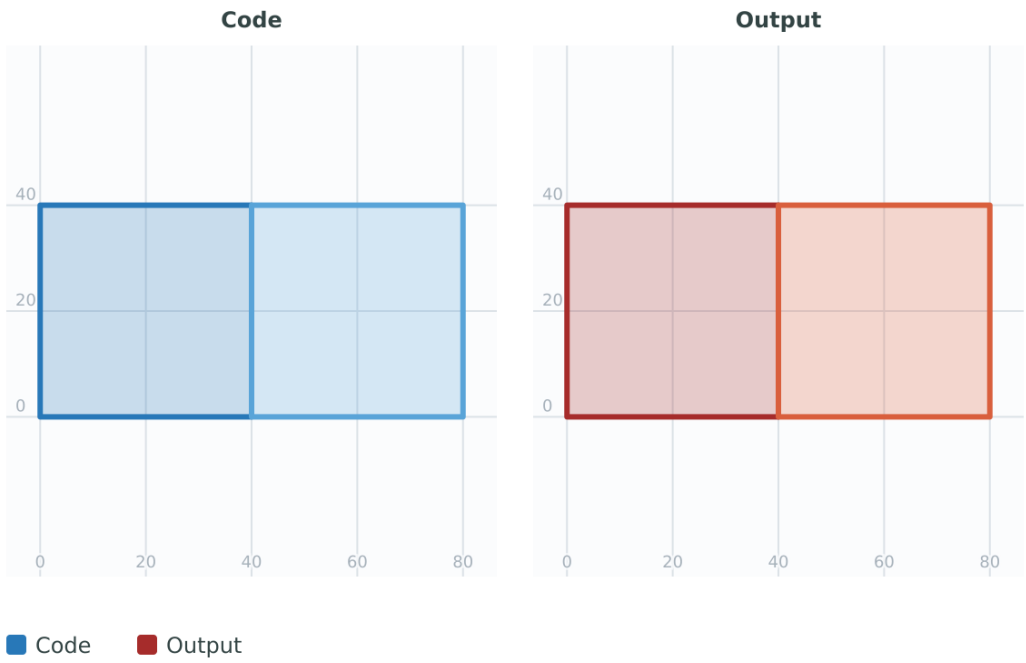
Code

```
SELECT ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50,50 50,50 10,10 10,10 50) ↩
    ),POLYGON ((90 50,90 10,50 10,50 50,90 50)))');
```

Output

```
MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

Figure



Voir aussi

ST_CollectionExtract, ST_Multi, ST_IsValid, ST_MakeValid

7.5.4 ST_CurveToLine

ST_CurveToLine — Convertit une géométrie contenant des courbes en une géométrie linéaire.

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);

Description

Convertit une CIRCULAR STRING en LINESTRING régulière ou CURVEPOLYGON en POLYGON ou MULTISURFACE en MULTIPOLYGON. Utile pour les sorties vers des appareils qui ne supportent pas les types de géométrie CIRCULARSTRING

Convertit une géométrie donnée en une géométrie linéaire. Chaque géométrie ou segment courbe est converti en une approximation linéaire en utilisant la `tolérance` et les options données (32 segments par quadrant et aucune option par défaut).

L'argument 'tolerance_type' détermine l'interprétation de l'argument `tolerance`. Il peut prendre les valeurs suivantes :

— 0 (par défaut) : La tolérance est le nombre maximum de segments par quadrant.
— 1 : La tolérance est l'écart maximal de la ligne par rapport à la courbe, en unités de source.
— 2 : La tolérance est l'angle maximal, en radians, entre les rayons générateurs.

L'argument 'flags' est un champ de bits. 0 par défaut. Les bits supportés sont :

— 1 : Sortie symétrique (indépendante de l'orientation).
— 2 : Conserver l'angle, évite de réduire les angles (longueur des segments) lors de la production d'une sortie symétrique. N'a aucun effet lorsque l'indicateur de symétrie est désactivé.

Disponibilité : 1.3.0

Amélioration : 2.4.0 a ajouté la prise en charge de la tolérance de l'écart maximal et de l'angle maximal, ainsi que de la sortie symétrique.

Amélioration : la version 3.0.0 a mis en place un nombre minimum de segments par arc linéarisé afin d'éviter une rupture topologique.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 7.1.7

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

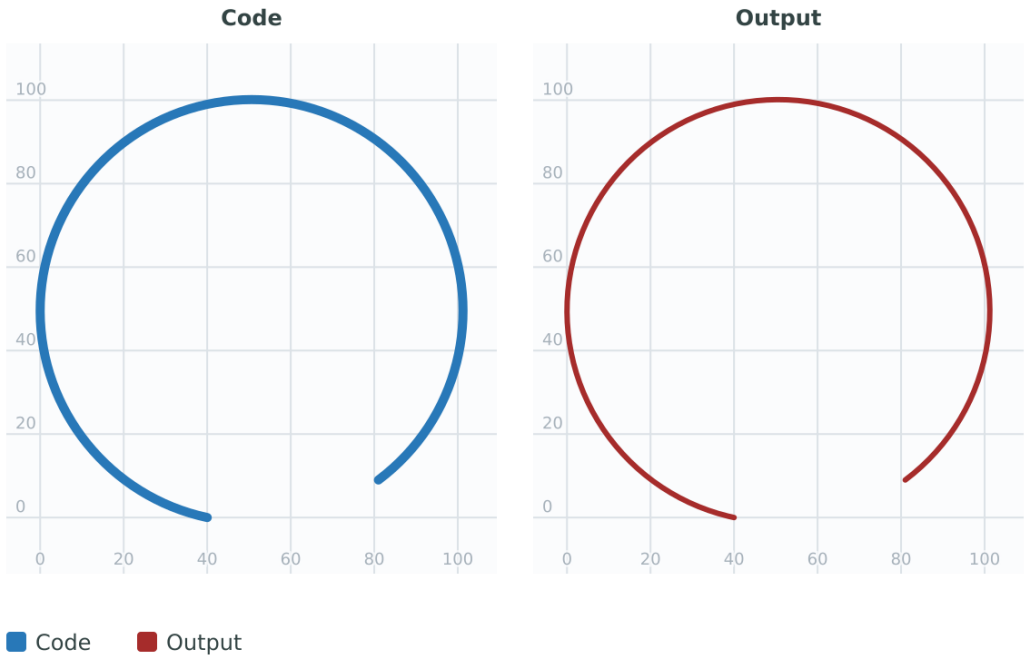
```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)');
```

Output

```
LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575 150418.17258804,220273.613787707 ↩
    150419.895736857,
220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847 150425.562198489,
220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347 150431.876723113,
220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099 150438.702620341,220286.147650624 ↩
    150441.066277505,
220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149 150448.342699654,
220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775 150455.77405574,
220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492 150463.199479347,
220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259 150470.458232479,220285.196316903 ↩
    150472.81345077,
220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384 150479.606668057,
220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122 150485.87804878,
220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362 150491.491836488,
220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731 150496.326509628,
220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106 150500.277412127,
220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878 150503.259018879,
220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107 150505.206787101,
220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968 150506.078553494,
220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804 150505.855446946,
220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724 150504.542297043,
220222.663718741 150503.86659104,220220.308500449 150503.074365683,
220217.994991777 150502.167529512,220215.72876617 150501.148267175,
220213.515283163 150500.019034164,220211.35987523 150498.7825509,
220209.267734939 150497.441796181,220207.243902439 150496,
220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732 150491.104263143,
220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592 150485.437801511,
220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092 150479.123276887,220191.739336189 ↩
    150476.89769814,
220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815 150469.933722495,
220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229 150462.657300346,
220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664 150455.22594426,
220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947 150447.800520653,
220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018 150440.541767521,
220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044 150433.60681495,
220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039 150427.14578372,220197.12195122 ↩
    150425.12195122,
```

```
220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077 150419.508163512,220203.826610682 ↩
    150417.804498867,
220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298 150413.253689397,220211.830006129 ↩
    150411.935663483,
220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756 150408.622717305,220220.824571561 ↩
    150407.740981121,
220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)
```

Figure



This example uses 3D input.

Code

```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

Output

```
LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,
220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857 1.05435185700189,....AD INFINITUM ↩
    ....
   220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)
```

Use only 2 segments to approximate quarter circle.

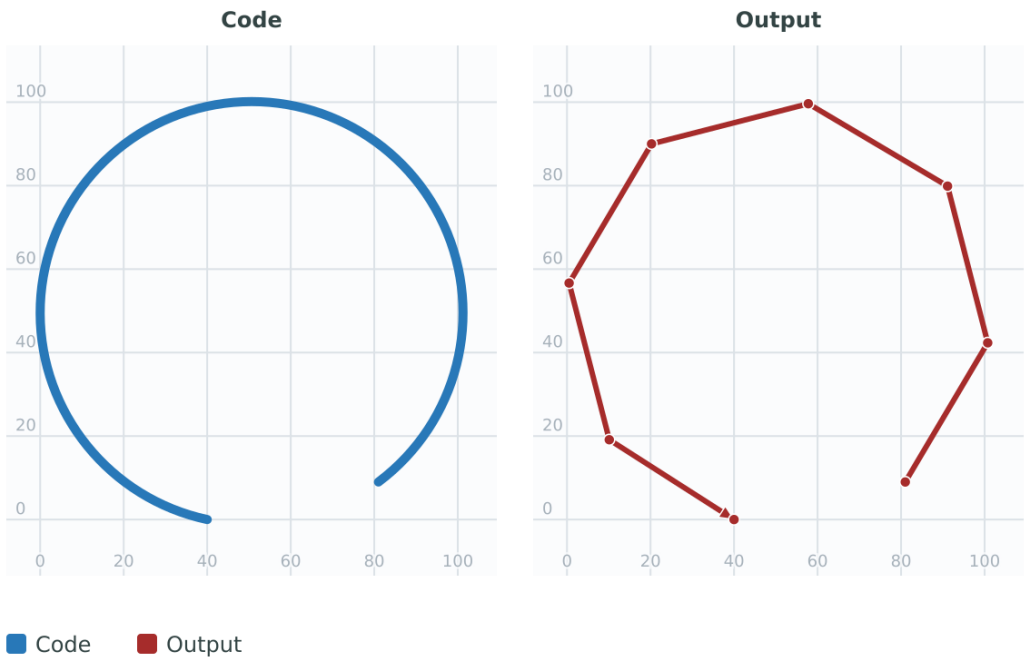
Code

```
SELECT ST_CurveToLine('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)', 2);
```

Output

```
LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122 150485.87804878,
220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,
220197.12195122 150425.12195122,220227 150406)
```

Figure



Ensure the approximated line is no more than 20 units away from the original curve, and make the result direction-neutral.

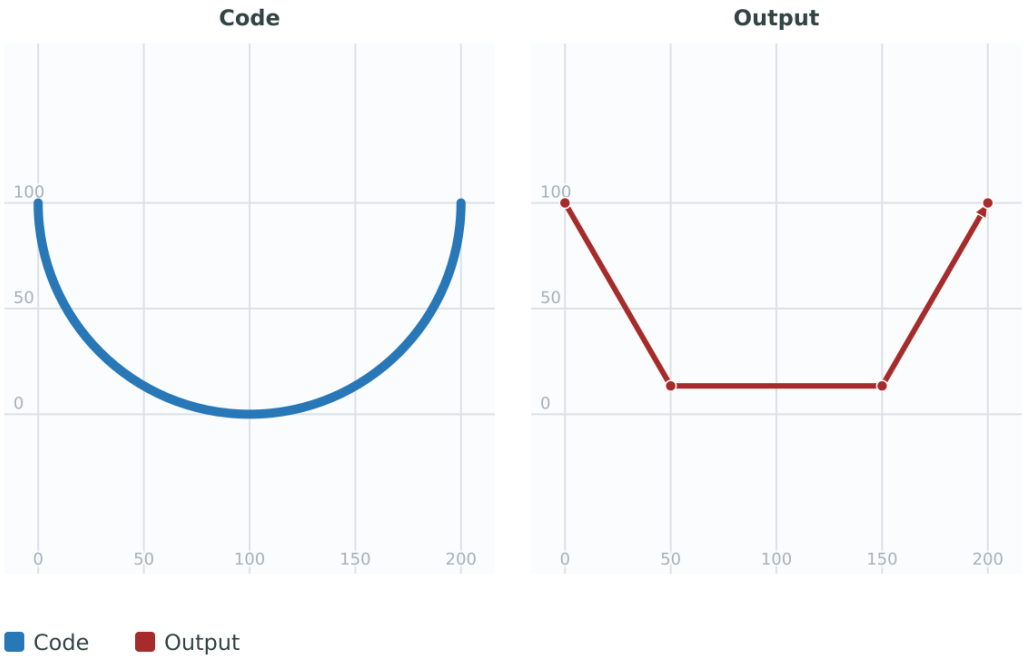
Code

```
SELECT ST_CurveToLine(
    'CIRCULARSTRING(0 0,100 -100,200 0)'::geometry,
    20,
    1,
    1
  );
```

Output

```
LINESTRING(0 0,49.99999999999996 -86.60254037844383,149.99999999999994 -86.6025403784439,200 0)
```

Figure



Voir aussi

ST_LineToCurve

7.5.5 ST_Scroll

ST_Scroll — Modifier le point de départ d'une LineString fermée.

Synopsis

geometry **ST_Scroll**(geometry linestring, geometry point);

Description

Modifie le point de départ/fin d'une LineString fermée en fonction du sommet `point` donné.

Disponibilité : 3.2.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Exemples

Make a closed line start at its 3rd vertex.

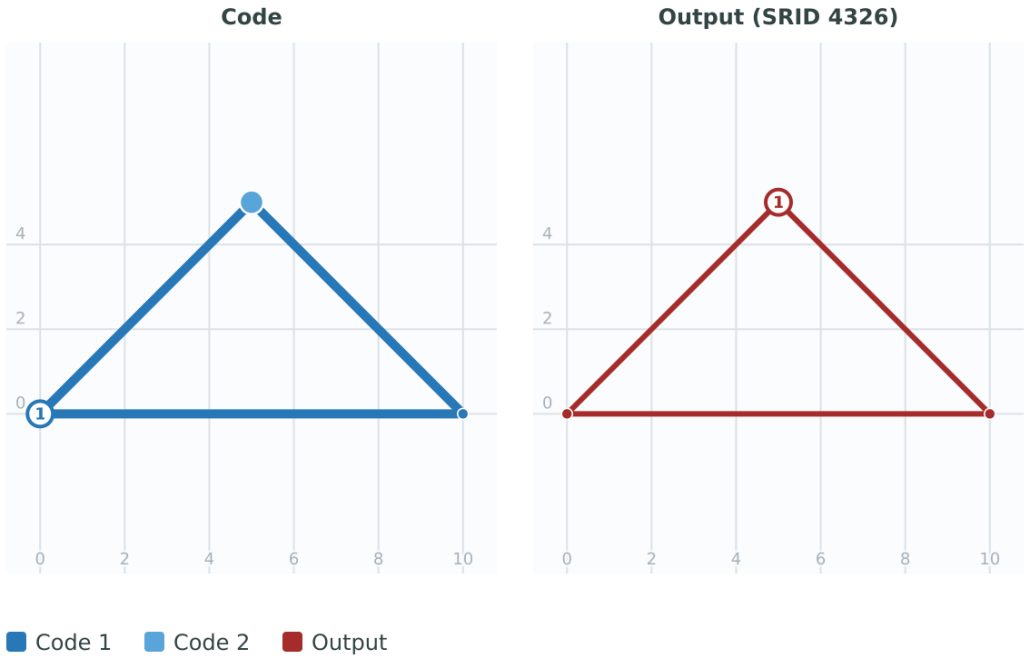
Code

```
SELECT ST_Scroll('SRID=4326;LINESTRING(0 0 0 1,10 0 2 0,5 5 4 2,0 0 0 1)', 'POINT(5 5 4 2)') ↩
    ;
```

Output

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

Figure



Voir aussi

ST_Normalize

7.5.6 ST_FlipCoordinates

ST_FlipCoordinates — Renvoie une version d'une géométrie dont les axes X et Y sont inversés.

Synopsis

geometry **ST_FlipCoordinates**(geometry geom);

Description

Renvoie une version de la géométrie donnée avec les axes X et Y inversés. Utile pour fixer les géométries qui contiennent des coordonnées exprimées en latitude/longitude (Y,X).

Disponibilité : 2.0.0

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemple

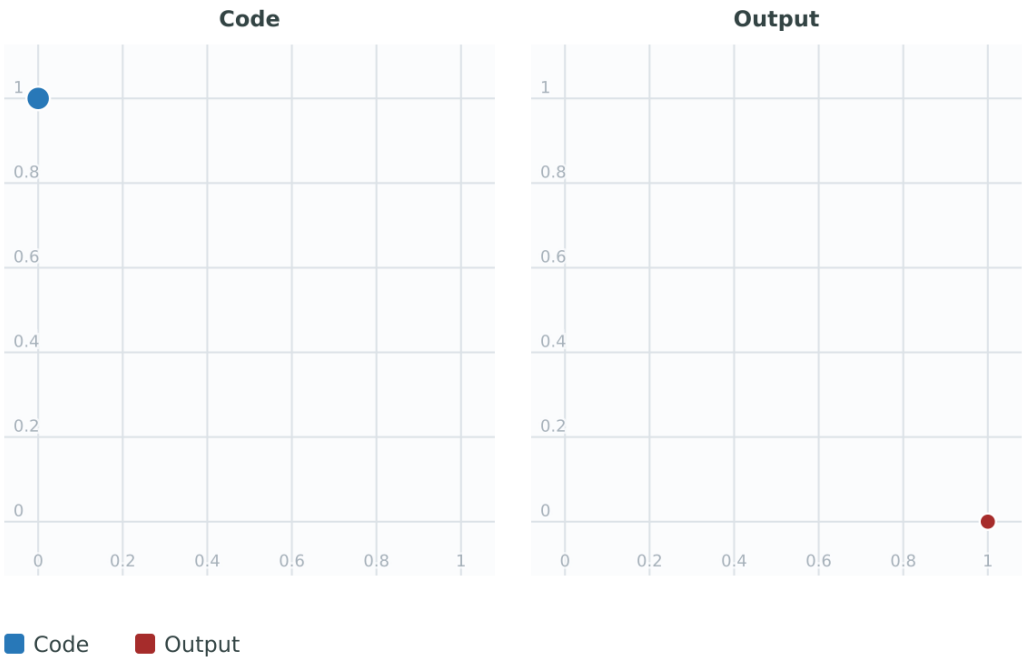
Code

```
SELECT ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)'));
```

Output

```
POINT(2 1)
```

Figure



Voir aussi

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — Forcer les géométries à passer en "mode bidimensionnel".

Synopsis

geometry **ST_Force2D**(geometry geomA);

Description

Force les géométries à passer en "mode bidimensionnel", de sorte que toutes les représentations de sortie n'auront que les coordonnées X et Y. Ceci est utile pour forcer une sortie conforme à l'OGC (puisque l'OGC ne spécifie que des géométries à 2 dimensions).

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_2D.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

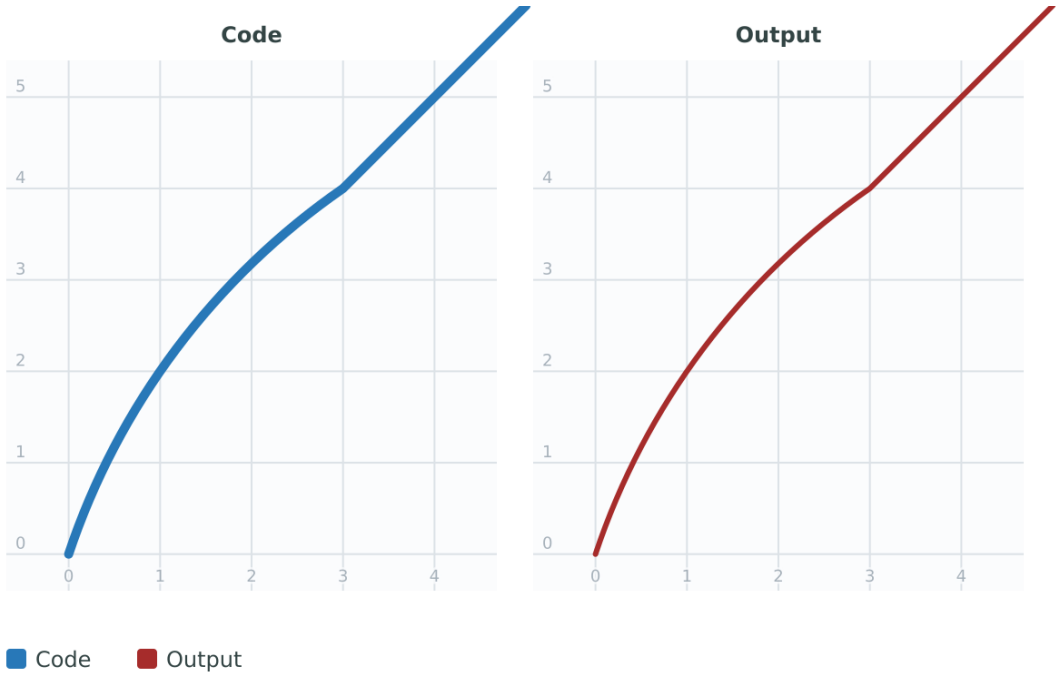
Code

```
SELECT ST_Force2D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)
```

Figure



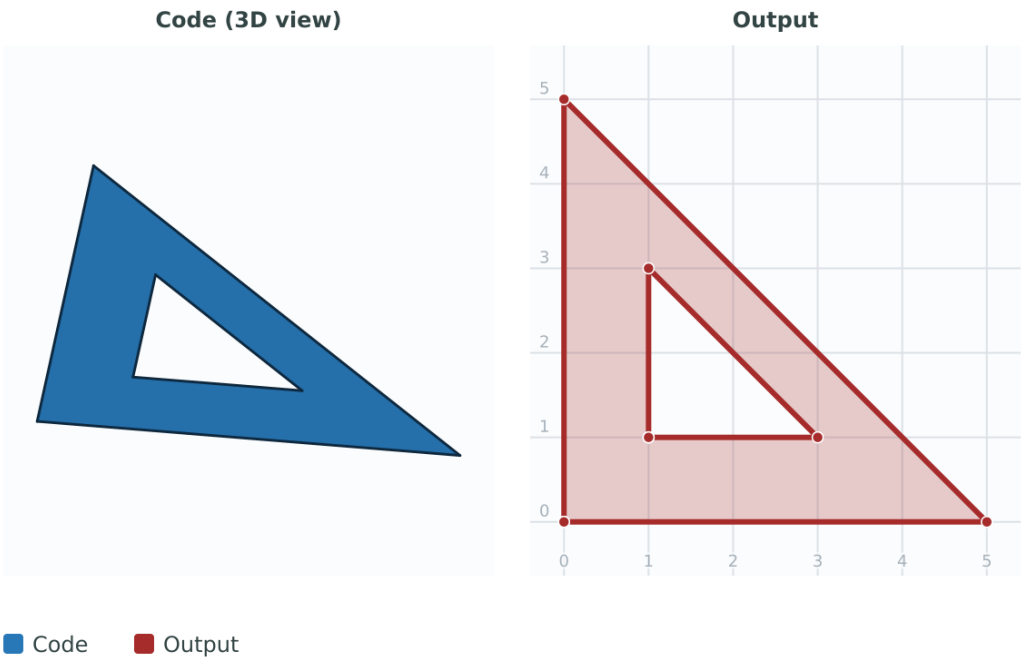
Code

```
SELECT ST_Force2D('POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))');
```

Output

```
POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))
```

Figure



Voir aussi

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.

Synopsis

geometry **ST_Force3D**(geometry geomA, float Zvalue = 0.0);

Description

Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ. Si une géométrie n'a pas de composante Z, une `Zvalue` coordonnée Z est ajoutée.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_3D.

Modifié : 3.1.0. Ajout de la prise en charge pour pouvoir passer une valeur Z non nulle.

Cette fonction prend en charge les surfaces Polyhedral.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Nothing happens to an already 3D geometry.

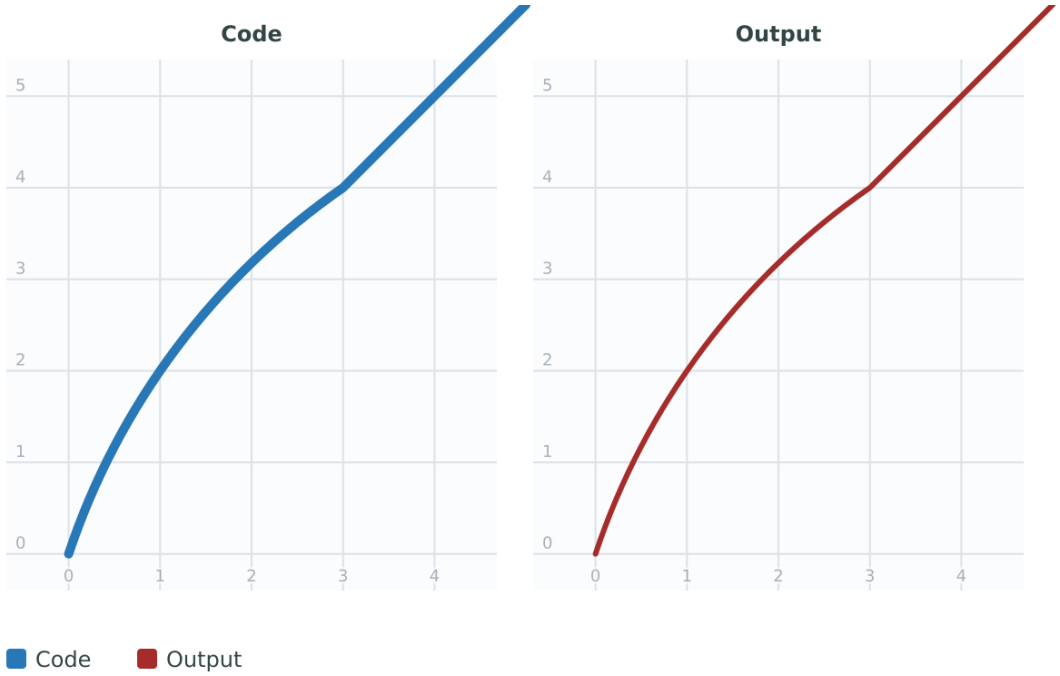
Code

```
SELECT ST_Force3D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

Figure



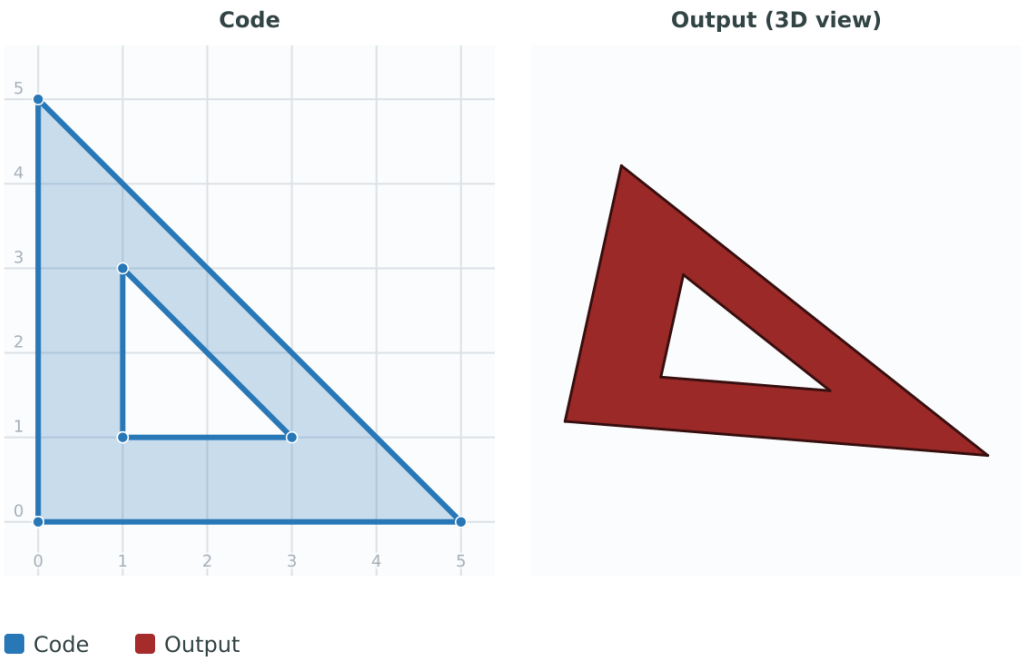
Code

```
SELECT ST_Force3D('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))');
```

Output

```
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Figure



Voir aussi

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3DZ

7.5.9 ST_Force3DZ

ST_Force3DZ — Forcer les géométries en mode XYZ.

Synopsis

geometry **ST_Force3DZ**(geometry geomA, float Zvalue = 0.0);

Description

Force les géométries à passer en mode XYZ. Si une géométrie n'a pas de composante Z, une `Zvalue` coordonnée Z est ajoutée.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_3DZ.

Modifié : 3.1.0. Ajout de la prise en charge pour pouvoir passer une valeur Z non nulle.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Nothing happens to an already 3D geometry.

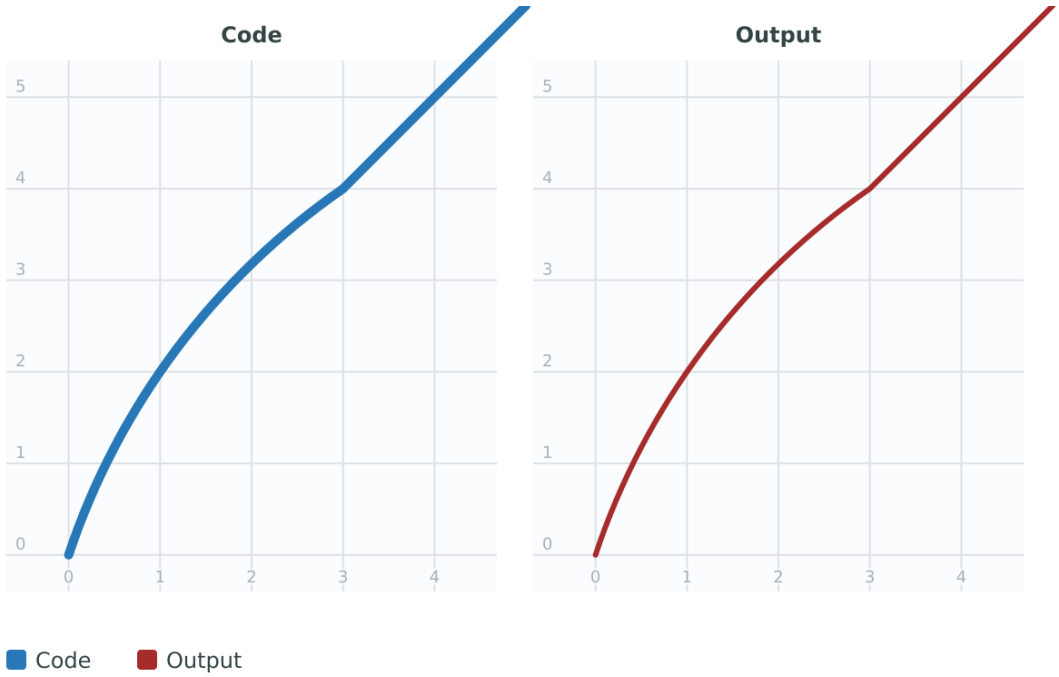
Code

```
SELECT ST_Force3DZ('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)
```

Figure



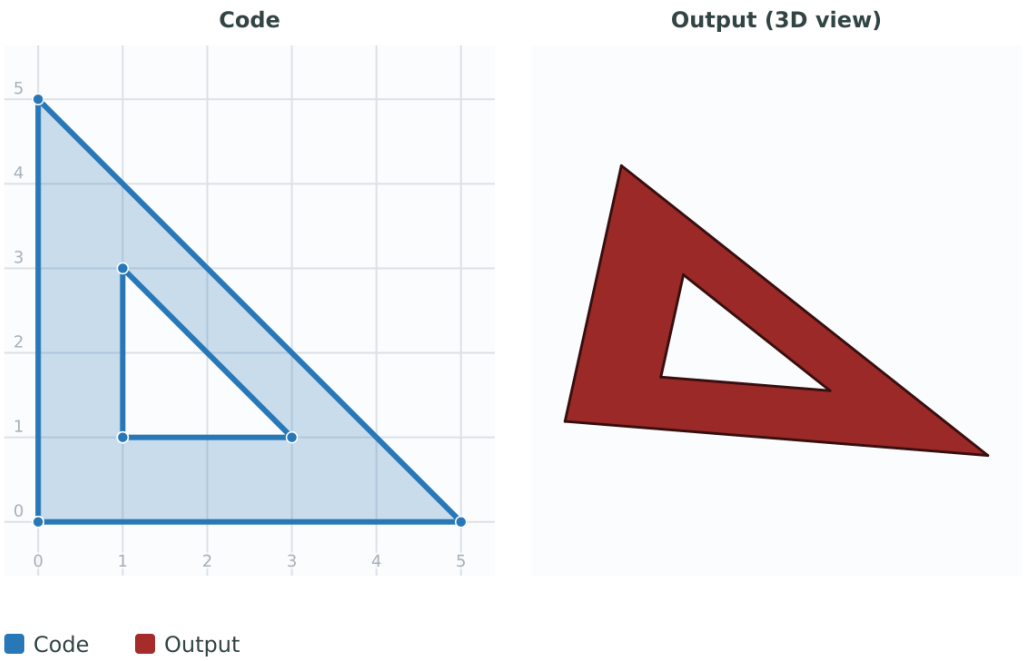
Code

```
SELECT ST_Force3DZ('POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))');
```

Output

```
POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Figure



Voir aussi

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.10 ST_Force3DM

ST_Force3DM — Forcer les géométries en mode XYM.

Synopsis

geometry **ST_Force3DM**(geometry geomA, float Mvalue = 0.0);

Description

Force les géométries à passer en mode XYM. Si une géométrie n'a pas de composante M, une `Mvalue` coordonnée M est ajoutée. Si la géométrie a une composante Z, la coordonnée Z est supprimée

Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_3DM.

Modifié : 3.1.0. Ajout de la prise en charge de pouvoir passer une valeur M non nulle.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Nothing happens to an already 3D geometry.

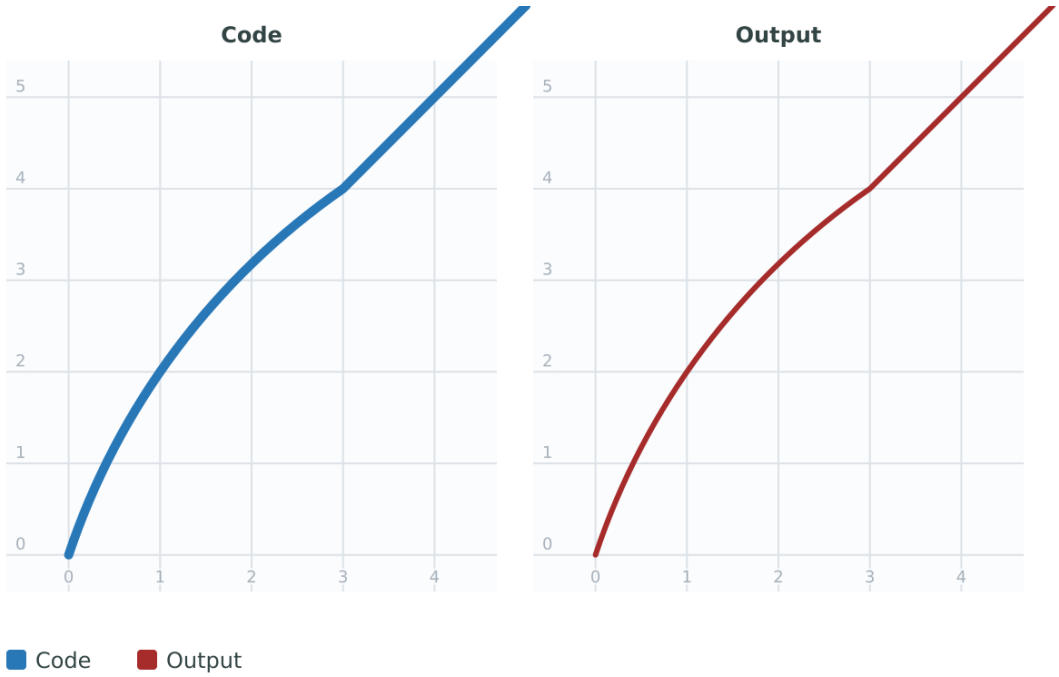
Code

```
SELECT ST_Force3DM('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)
```

Figure



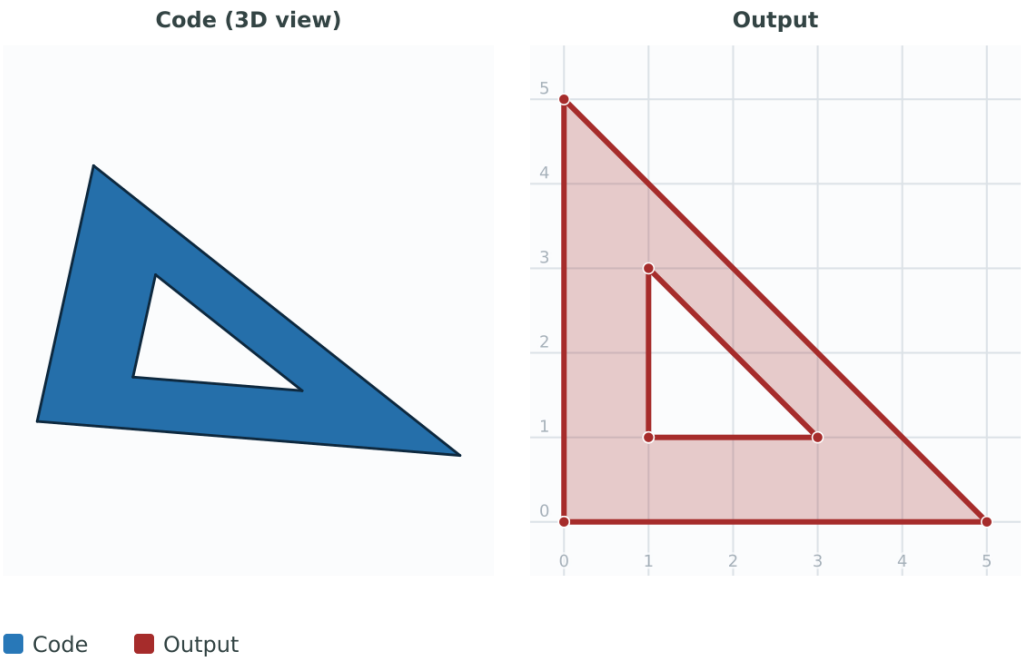
Code

```
SELECT ST_Force3DM('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))');
```

Output

```
POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))
```

Figure



Voir aussi

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.11 ST_Force4D

ST_Force4D — Forcer les géométries en mode XYZM.

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

Description

Force les géométries en mode XYZM. `Zvalue` et `Mvalue` sont ajoutés pour les dimensions Z et M manquantes, respectivement.

Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_4D.

Modifié : 3.1.0. Ajout de la prise en charge de pouvoir passer des valeurs Z et M non nulles.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Nothing happens to an already 3D geometry.

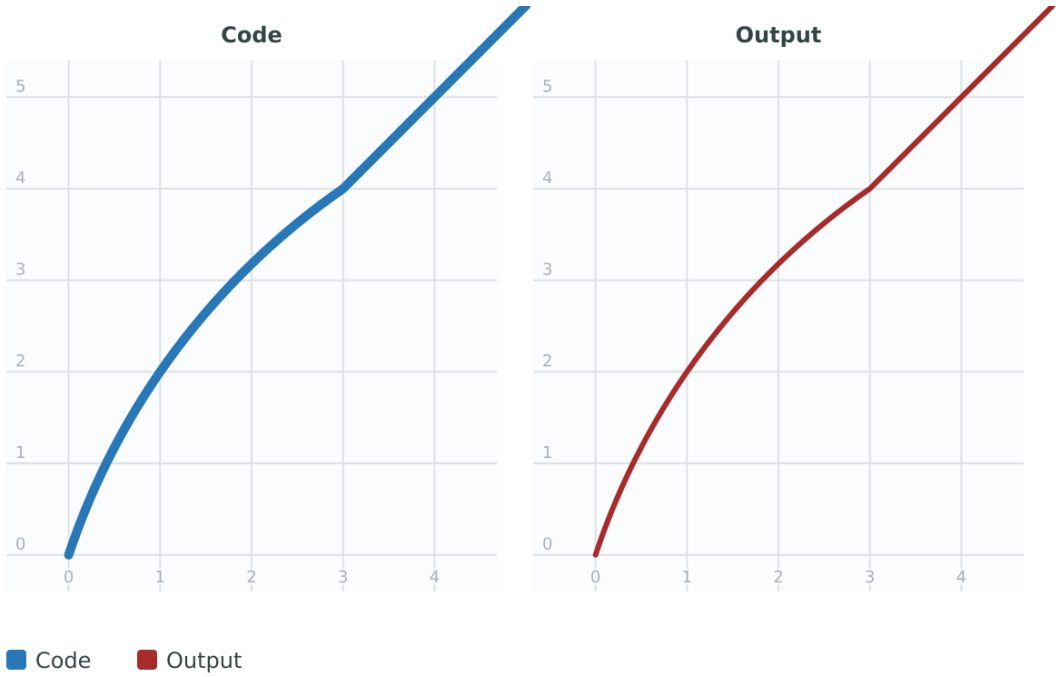
Code

```
SELECT ST_Force4D('CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)');
```

Output

```
CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)
```

Figure



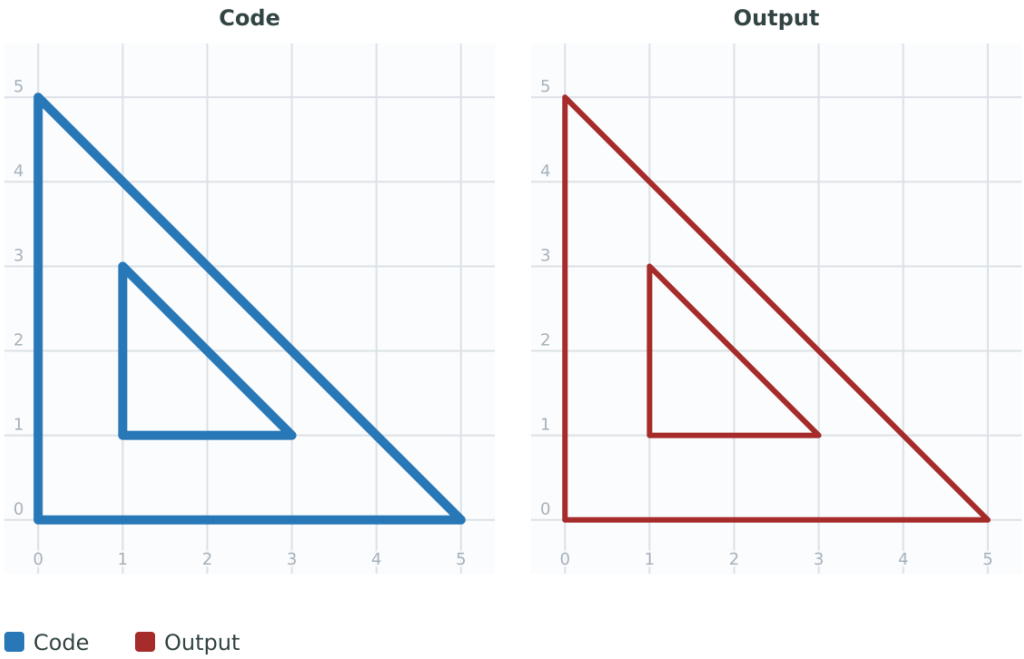
Code

```
SELECT ST_Force4D('MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3 1,1 1 1))');
```

Output

```
MULTILINESTRING((0 0 0 1,0 5 0 2,5 0 0 3,0 0 0 4),(1 1 0 1,3 1 0 1,1 3 0 1,1 1 0 1))
```

Figure



Voir aussi

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.12 ST_ForceCollection

ST_ForceCollection — Convertir la géométrie en une GEOMETRYCOLLECTION.

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

Description

Convertit la géométrie en une GEOMETRYCOLLECTION. Ceci est utile pour simplifier la représentation WKB.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Disponibilité : 1.2.2, avant la version 1.3.4 cette fonction plante avec les courbes. Ceci est corrigé dans la version 1.3.4+

Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Force_Collection.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

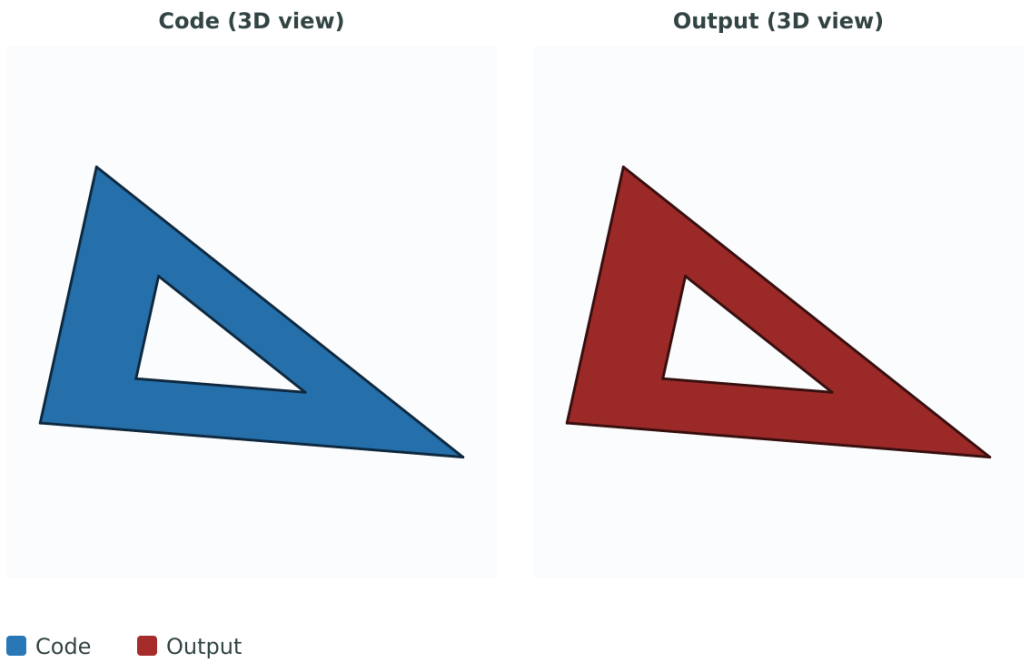
Code

```
SELECT ST_ForceCollection('POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1))');
```

Output

```
GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))
```

Figure



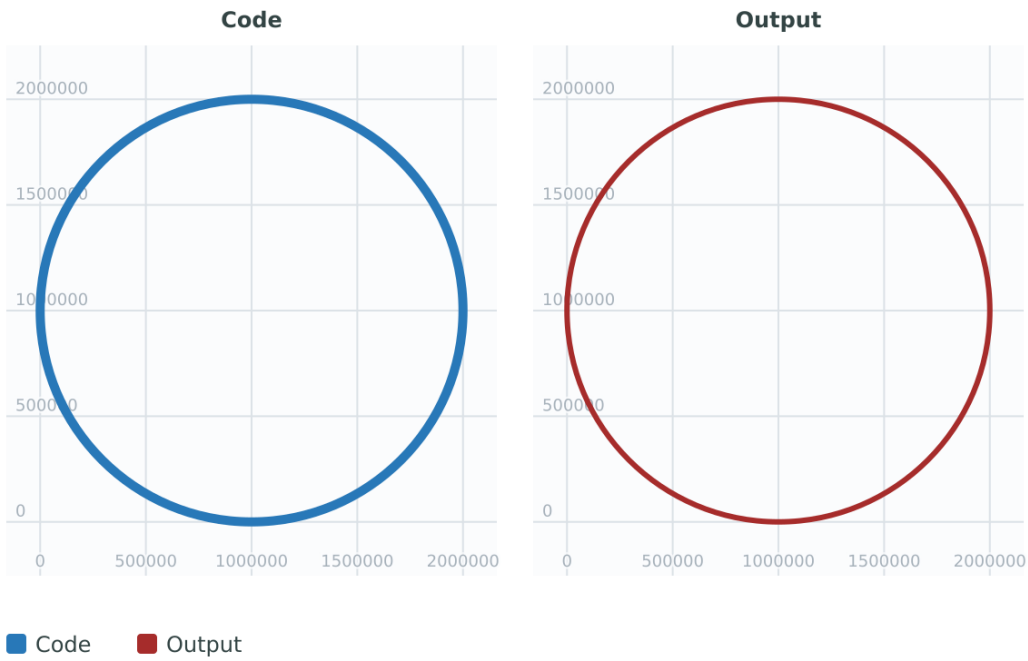
Code

```
SELECT ST_ForceCollection('CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)');
```

Output

```
GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
```

Figure



This example uses a PolyhedralSurface.

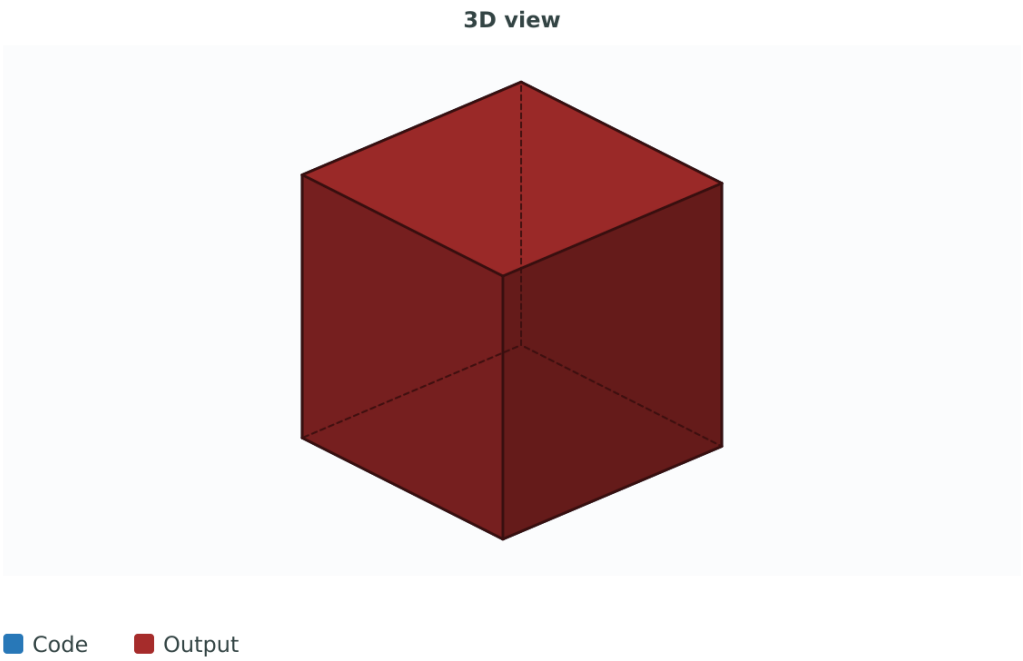
Code

```
SELECT ST_ForceCollection('POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))');
```

Output

```
GEOMETRYCOLLECTION(
  POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
  POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
  POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
  POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
  POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
  POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)
```

Figure



Voir aussi

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.13 ST_ForceCurve

ST_ForceCurve — Retransformation d'une géométrie dans son type de courbure, le cas échéant.

Synopsis

geometry **ST_ForceCurve**(geometry g);

Description

Transforme une géométrie en sa représentation courbe, le cas échéant : les lignes deviennent des courbes composées, les multilignes deviennent des multicourbes, les polygones deviennent des polygones courbes, les multipolygones deviennent des multisurfaces. Si la géométrie en entrée est déjà une représentation courbe, le résultat est identique à celui de l'entrée.

Disponibilité : 2.2.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

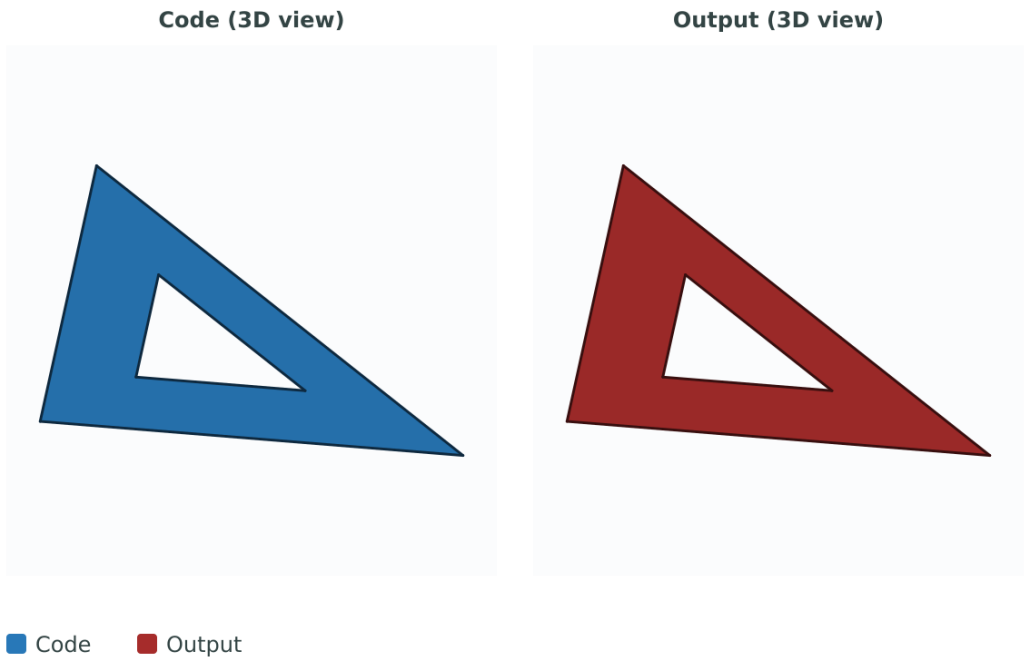
Code

```
SELECT ST_ForceCurve('POLYGON((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))':: ↩
    geometry
  );
```

Output

```
CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))
```

Figure



Voir aussi

ST_LineToCurve

7.5.14 ST_ForcePolygonCCW

ST_ForcePolygonCCW — Oriente tous les anneaux extérieurs dans le sens inverse des aiguilles d'une montre et tous les anneaux intérieurs dans le sens des aiguilles d'une montre.

Synopsis

geometry **ST_ForcePolygonCCW**(geometry geom);

Description

Force les (Multi)Polygones à utiliser une orientation dans le sens inverse des aiguilles d'une montre pour leur anneau extérieur, et une orientation dans le sens des aiguilles d'une montre pour leurs anneaux intérieurs. Les géométries non polygonales sont renvoyées inchangées.

Disponibilité : 2.4.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Exemples

Force a polygon and its hole to use counter-clockwise exterior rings and clockwise interior rings.

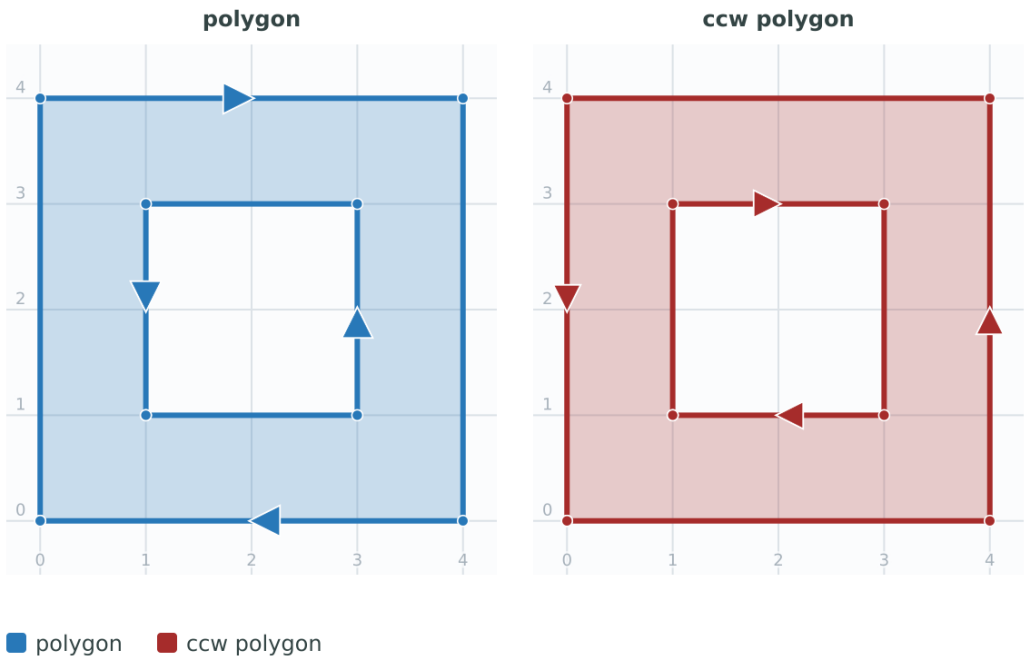
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,0 4,4 4,4 0,0 0),
                  (1 1,3 1,3 3,1 3,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_ForcePolygonCCW(geom) AS ccw_polygon
FROM data;
```

Output

```
POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1)) | POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1))
```

Figure



Voir aussi

ST_ForcePolygonCW, ST_IsPolygonCCW, ST_IsPolygonCW

7.5.15 ST_ForcePolygonCW

ST_ForcePolygonCW — Oriente tous les anneaux extérieurs dans le sens des aiguilles d'une montre et tous les anneaux intérieurs dans le sens inverse des aiguilles d'une montre.

Synopsis

geometry **ST_ForcePolygonCW**(geometry geom);

Description

Force les (Multi)Polygones à utiliser une orientation dans le sens des aiguilles d'une montre pour leur anneau extérieur, et une orientation dans le sens inverse des aiguilles d'une montre pour leurs anneaux intérieurs. Les géométries non polygonales sont renvoyées inchangées.

Disponibilité : 2.4.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Exemples

Force a polygon and its hole to use clockwise exterior rings and counter-clockwise interior rings.

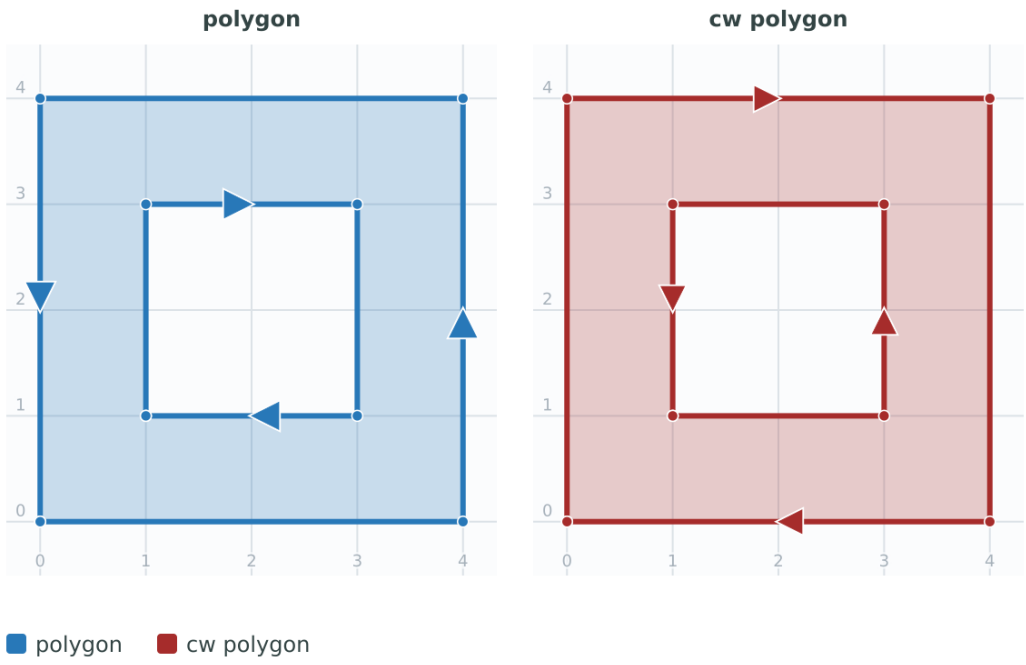
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,4 0,4 4,0 4,0 0),
                  (1 1,1 3,3 3,3 1,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       ST_ForcePolygonCW(geom) AS cw_polygon
FROM data;
```

Output

```
POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1)) | POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1))
```

Figure



Voir aussi

ST_ForcePolygonCCW, ST_IsPolygonCCW, ST_IsPolygonCW

7.5.16 ST_ForceSFS

ST_ForceSFS — Force geometries to use SFS 1.1 or 1.2 geometry types.

Synopsis

geometry **ST_ForceSFS**(geometry geomA);
geometry **ST_ForceSFS**(geometry geomA, text version);

Description

Forces a geometry to use the geometry types defined by the requested version of the OGC Simple Features Specification. The default is SFS 1.1. Curved geometries are converted to linear representations, `TRIANGLE` is converted to `POLYGON`, and `TIN` and `POLYHEDRALSURFACE` are converted to `GEOMETRYCOLLECTION` values containing `POLYGON` elements.

If *`version`* begins with `1.2`, curved geometries are still converted to linear representations, but `TRIANGLE`, `TIN`, and `POLYHEDRALSU` are preserved. Other version values use the default SFS 1.1 behavior. Geometry collections are processed recursively, and the SRID and Z/M coordinates are preserved.

Disponibilité : 2.1.0

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Compare the geometry types produced for SFS 1.1 (the default) and SFS 1.2. Read each row from the input type in the direction of the result columns.

Code

```
WITH inputs(geom) AS (VALUES
    ('CIRCULARSTRING(0 0, 1 1, 2 0)'::geometry),
    ('TRIANGLE((0 0, 0 2, 2 0, 0 0))'::geometry),
    ('TIN(((0 0, 0 2, 2 0, 0 0)))'::geometry),
    ('POLYHEDRALSURFACE(((0 0, 0 2, 2 0, 0 0)))'::geometry)
)
SELECT ST_GeometryType(geom) || ' →' AS input,
       ST_GeometryType(ST_ForceSFS(geom)) AS "SFS 1.1 (default)",
       ST_GeometryType(ST_ForceSFS(geom, '1.2')) AS "SFS 1.2"
FROM inputs;
```

Output

```
input          |   SFS 1.1 (default)   |       SFS 1.2
-----------------------+-----------------------+----------------------
 ST_CircularString →    | ST_LineString         | ST_LineString
 ST_Triangle →          | ST_Polygon            | ST_Triangle
 ST_Tin →               | ST_GeometryCollection | ST_Tin
 ST_PolyhedralSurface → | ST_GeometryCollection | ST_PolyhedralSurface
(4 rows)
```

Voir aussi

ST_CurveToLine, ST_ForceCurve

7.5.17 ST_ForceRHR

ST_ForceRHR — Force l'orientation des sommets d'un polygone à suivre la règle de la main droite.

Synopsis

geometry **ST_ForceRHR**(geometry g);

Description

Force l'orientation des sommets d'un polygone à suivre la règle de la main droite, dans laquelle la zone délimitée par le polygone se trouve à droite de la limite. En particulier, l'anneau extérieur est orienté dans le sens des aiguilles d'une montre et les anneaux intérieurs dans le sens inverse. Cette fonction est synonyme de ST_ForcePolygonCW



Note
La définition ci-dessus de la règle de la main droite est en conflit avec des définitions utilisées dans d'autres contextes. Pour éviter toute confusion, il est recommandé d'utiliser ST_ForcePolygonCW.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

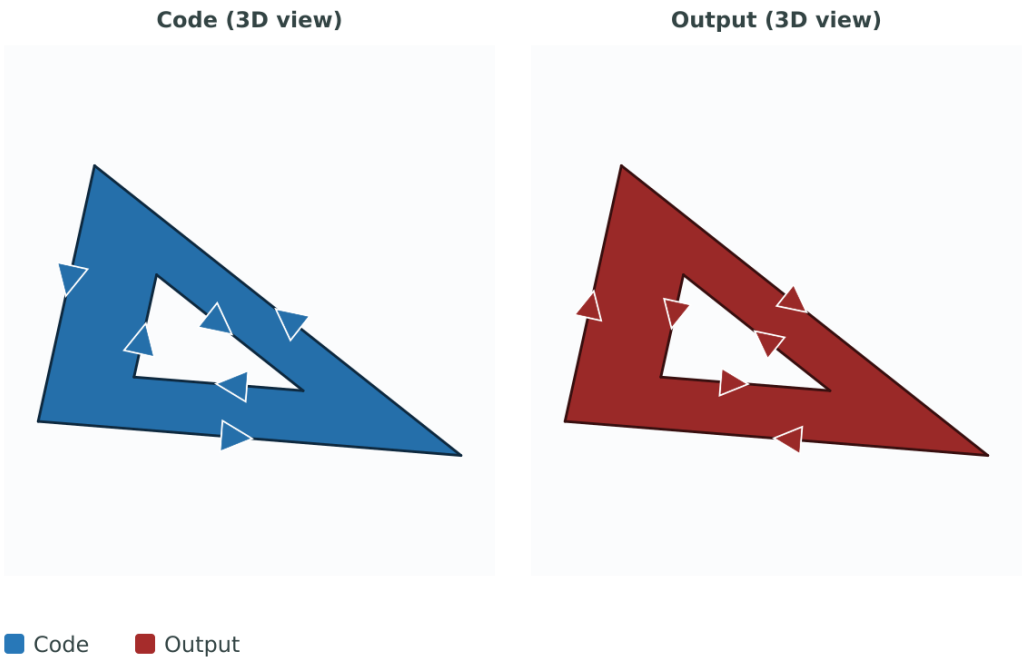
Code

```
SELECT ST_ForceRHR('POLYGON((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))'
  );
```

Output

```
POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))
```

Figure



Voir aussi

ST_ForcePolygonCCW, ST_ForcePolygonCW, ST_IsPolygonCCW, ST_IsPolygonCW, ST_BuildArea, ST_Polygonize, ST_Reverse

7.5.18 ST_LineExtend

ST_LineExtend — Renvoie une ligne prolongée vers l'avant et vers l'arrière selon les distances spécifiées.

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

Description

Renvoie une ligne prolongée vers l'avant et vers l'arrière en ajoutant de nouveaux points de départ (et d'arrivée) à la (aux) distance(s) donnée(s). Une distance de zéro n'ajoute pas de point. Seules les distances non négatives sont autorisées. La direction du ou des points ajoutés est déterminée par les deux premiers (et derniers) points distincts de la ligne. Les points en double sont ignorés.

Disponibilité : 3.4.0

Exemple : Prolonge une ligne de 5 unités vers l'avant et de 6 unités vers l'arrière

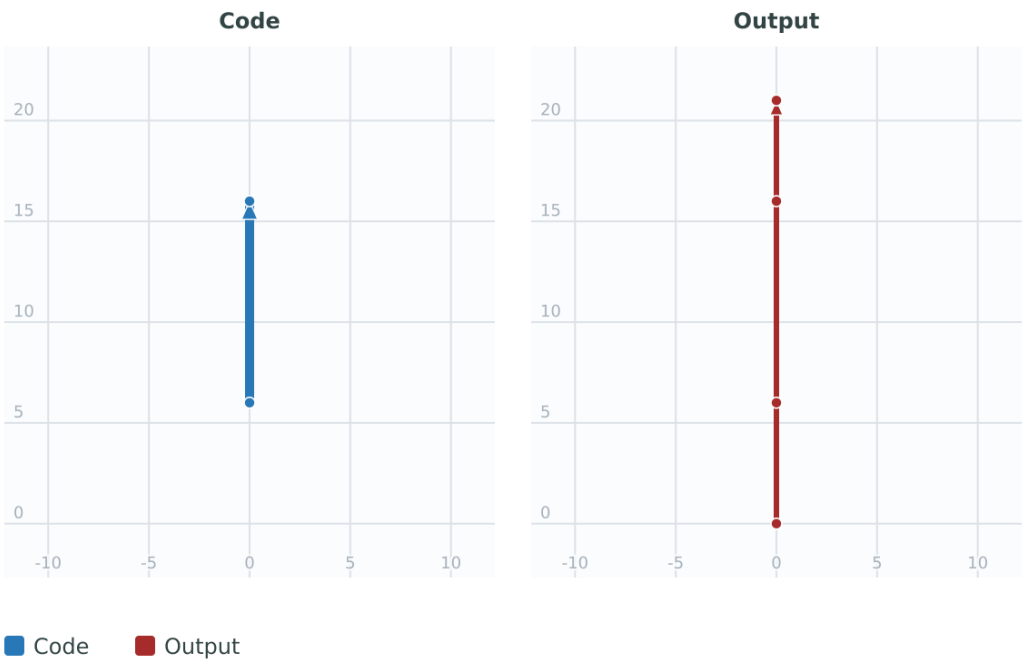
Code

```
SELECT ST_LineExtend('LINESTRING(0 0,0 10)'::geometry, 5, 6);
```

Output

```
LINESTRING(0 -6,0 0,0 10,0 15)
```

Figure



Voir aussi

ST_LineSubstring, ST_LocateAlong, ST_Project

7.5.19 ST_LineToCurve

ST_LineToCurve — Convertit une géométrie linéaire en une géométrie courbe.

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Description

Convertit les lignes et polygones simples en lignes circulaires et polygones courbes. Notez qu'il faut beaucoup moins de points pour décrire l'équivalent courbe.



Note
Si l'entrée LINESTRING/POLYGON n'est pas suffisamment courbée pour représenter clairement une courbe, la fonction renverra la même géométrie d'entrée.

Disponibilité : 1.3.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette méthode prend en charge les types Circular String et Curve.

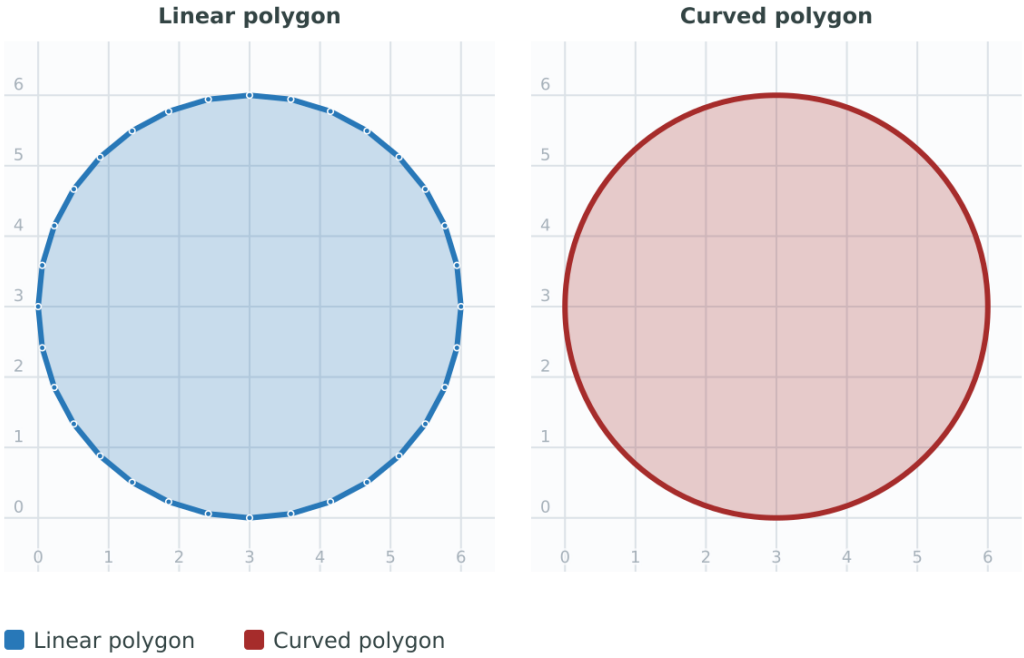
Exemples

This example converts a linear Polygon approximating a circle to a CurvePolygon.

Code

```
WITH data AS (
    SELECT ST_Buffer('POINT(1 3)'::geometry, 3) AS geom
)
SELECT geom AS "input_Linear polygon",
       ST_LineToCurve(geom) AS "Curved polygon"
FROM data;
```

Figure

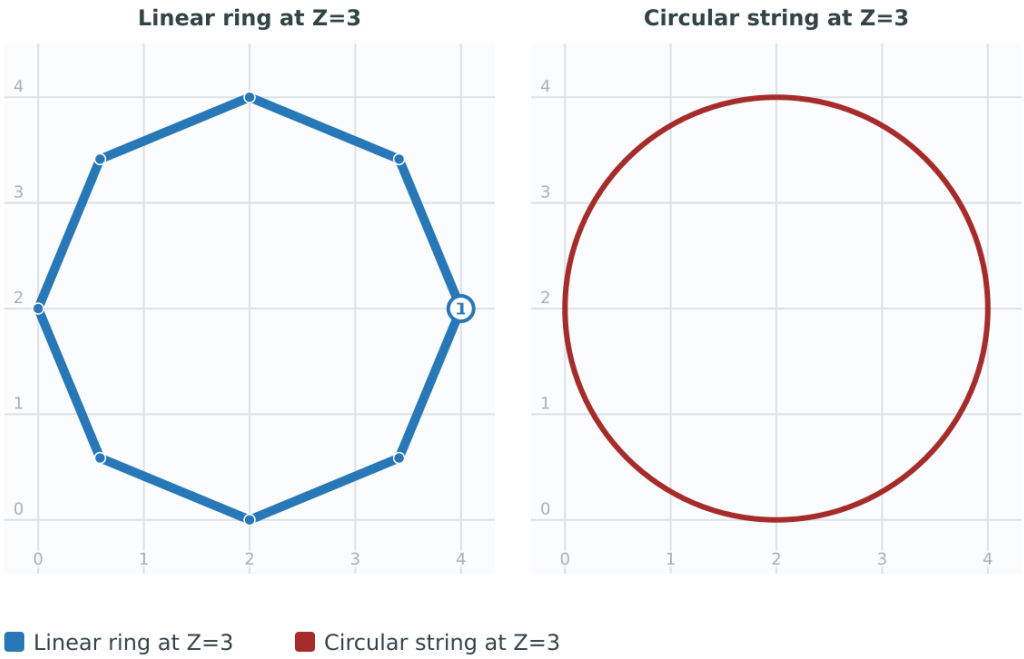


This example converts a 3D linear ring to a CircularString while preserving Z.

Code

```
WITH data AS (
    SELECT ST_Translate(
        ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1, 3), 2, 2))),
        0, 0, 3
    ) AS geom
)
SELECT geom AS "input_Linear ring at Z=3",
       ST_LineToCurve(geom) AS "Circular string at Z=3"
FROM data;
```

Figure



Voir aussi

ST_CurveToLine

7.5.20 ST_Multi

ST_Multi — Renvoie la géométrie sous la forme d'une géométrie MULTI*.

Synopsis

geometry **ST_Multi**(geometry geom);

Description

Renvoie la géométrie sous la forme d'une collection de géométries MULTI*. Si la géométrie est déjà une collection, elle est renvoyée inchangée.

Exemples

A Polygon is converted to a single-element MultiPolygon. The coordinates are unchanged.

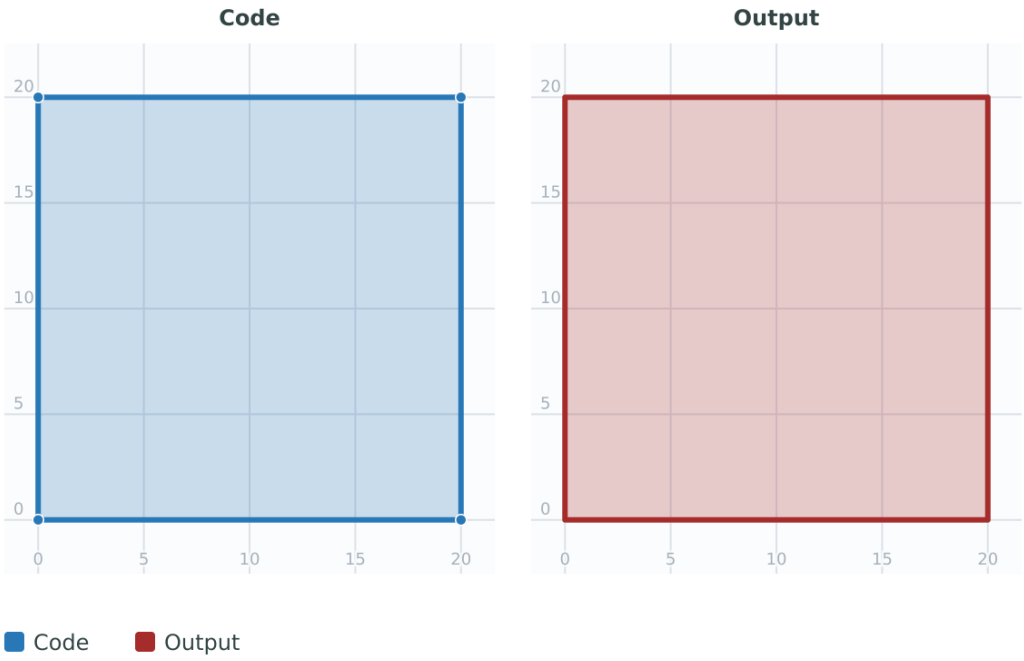
Code

```
SELECT ST_Multi('POLYGON ((10 30,30 30,30 10,10 10,10 30))');
```

Output

```
MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

Figure



Voir aussi

ST_AsText

7.5.21 ST_Normalize

ST_Normalize — Renvoie la géométrie sous sa forme canonique.

Synopsis

geometry **ST_Normalize**(geometry geom);

Description

Renvoie la géométrie sous sa forme normalisée/canonique. Peut réorganiser les sommets dans les anneaux de polygones, les anneaux dans un polygone, les éléments dans un complexe multi-géométrique.

La plupart du temps, il n'est utile qu'à des fins de test (comparaison des résultats attendus et obtenus).

Disponibilité : 2.3.0

Exemples

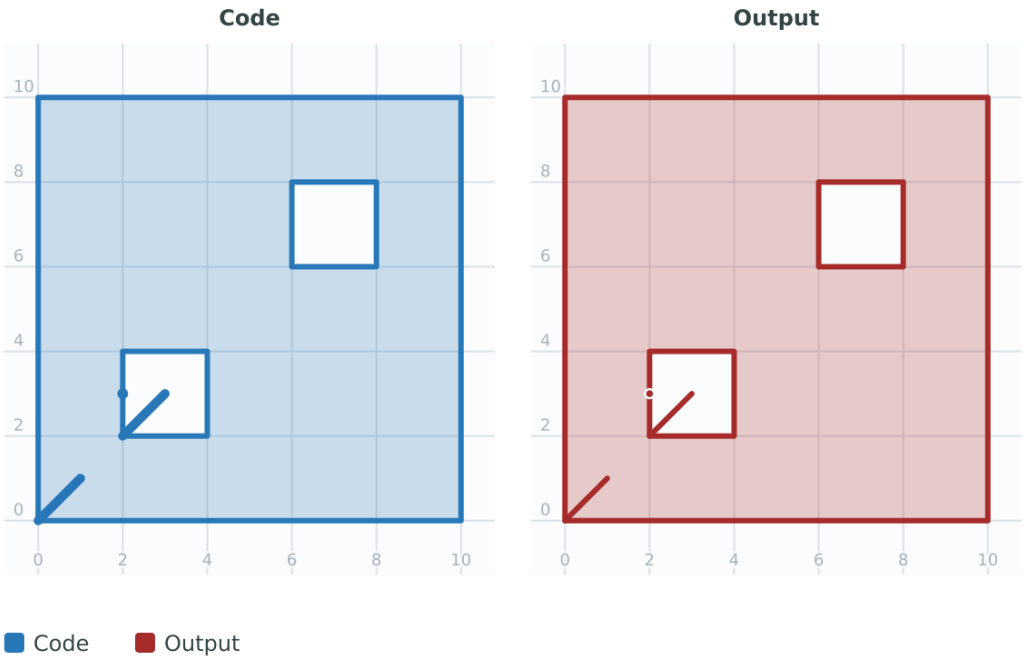
Code

```
SELECT ST_Normalize('GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0,1 1),(2 2,3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )');
```

Output

```
GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2 4,2 2)), MULTILINESTRING((2 ↩
    2,3 3),(0 0,1 1)), POINT(2 3))
```

Figure



Voir aussi

ST_Equals,

7.5.22 ST_Project

ST_Project — Renvoie un point projeté à partir d'un point de départ en fonction d'une distance et d'un azimut.

Synopsis

geometry **ST_Project**(geometry g1, float distance, float azimuth);
geometry **ST_Project**(geometry g1, geometry g2, float distance);
geography **ST_Project**(geography g1, float distance, float azimuth);
geography **ST_Project**(geography g1, geography g2, float distance);

Description

Renvoie un point projeté à partir d'un point le long d'une géodésique en utilisant une distance et un azimut donnés. C'est ce qu'on appelle le problème de la géodésique directe.

La version à deux points utilise la trajectoire entre le premier et le deuxième point pour définir implicitement l'azimut et utilise la distance comme précédemment.

La distance est indiquée en mètres. Les valeurs négatives sont prises en charge.

L'azimut (également appelé cap ou relèvement) est indiqué en radians. Il est mesuré dans le sens des aiguilles d'une montre à partir du vrai nord.

— Le nord est l'azimut zéro (0 degré)
— L'est est l'azimut $\pi/2$ (90 degrés)
— Le sud est l'azimut π (180 degrés)
— Ouest est l'azimut $3\pi/2$ (270 degrés)

Les valeurs d'azimut négatives et les valeurs supérieures à 2π (360 degrés) sont prises en charge.

Disponibilité : 2.0.0

Amélioration : 2.4.0 Autorise les distances négatives et les azimuts non normalisés.

Amélioration : 3.4.0 Autorise les arguments géométriques et la forme en deux points omettant l'azimut.

Exemple : Point projeté à 100,000 mètres et relèvement de 45 degrés

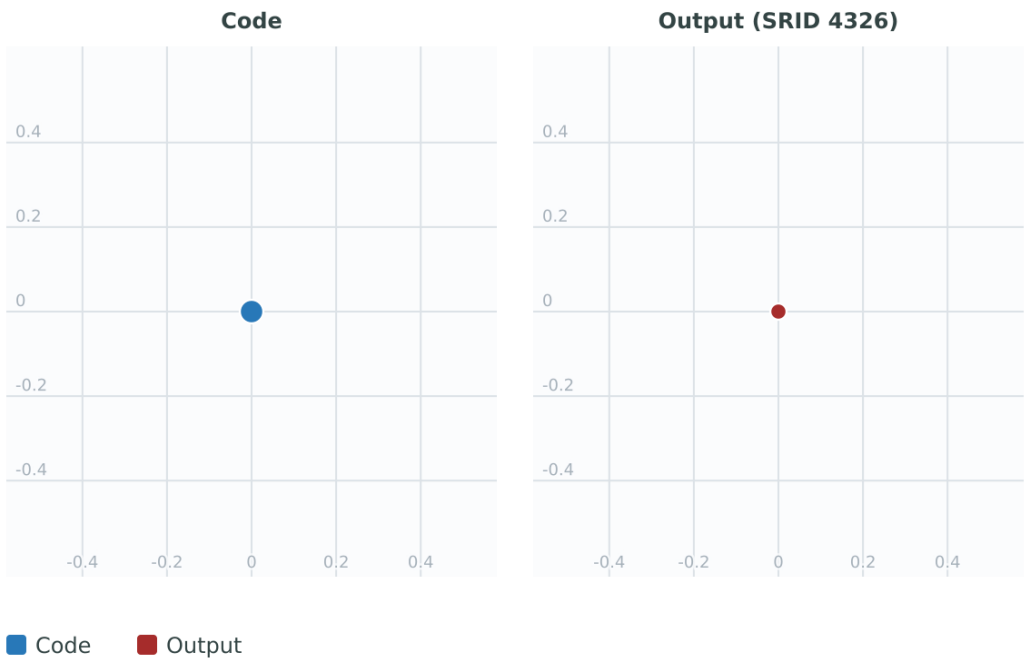
Code

```
SELECT ST_Project('POINT(0 0)'::geography, 100000, radians(45.0));
```

Output

```
POINT(0.635231029125537 0.639472334729198)
```

Figure



Voir aussi

ST_Azimuth, ST_Distance, Fonction PostgreSQL radians()

7.5.23 ST_QuantizeCoordinates

ST_QuantizeCoordinates — Met à zéro les bits de poids faible des coordonnées

Synopsis

geometry **ST_QuantizeCoordinates**(geometry g, int prec_x, int prec_y = NULL, int prec_z = NULL, int prec_m = NULL);

Description

`ST_QuantizeCoordinates` détermine le nombre de bits (`N`) requis pour représenter une valeur de coordonnées avec un nombre spécifié de chiffres après la virgule, puis met à zéro tous les bits sauf les `N` les plus significatifs. La valeur de coordonnées résultante sera toujours arrondie à la valeur d'origine, mais sa compressibilité sera améliorée. Il peut en résulter une réduction significative de l'utilisation du disque, à condition que la colonne géométrique utilise un type de stockage compressible. La fonction permet de spécifier un nombre différent de chiffres après la virgule dans chaque dimension ; les dimensions non spécifiées sont supposées avoir la précision de la dimension `x`. Les chiffres négatifs sont interprétés comme des chiffres à gauche de la virgule (c'est-à-dire que `prec_x=-2` conservera les valeurs des coordonnées à la centaine la plus proche).

Les coordonnées produites par `ST_QuantizeCoordinates` sont indépendantes de la géométrie qui contient ces coordonnées et de la position relative de ces coordonnées dans la géométrie. Par conséquent, les relations topologiques existantes entre les géométries ne sont pas affectées par l'utilisation de cette fonction. La fonction peut produire une géométrie non valide lorsqu'elle est appelée avec un nombre de chiffres inférieur à la précision intrinsèque de la géométrie.

Disponibilité : 2.5.0

Informations techniques

PostGIS stocke toutes les valeurs de coordonnées sous forme d'entiers à virgule flottante en double précision, qui peuvent représenter de manière fiable 15 chiffres significatifs. Cependant, PostGIS peut être utilisé pour gérer des données qui ont intrinsèquement moins de 15 chiffres significatifs. C'est le cas des données TIGER, qui sont fournies sous forme de coordonnées géographiques avec six chiffres de précision après la virgule (ce qui ne nécessite que neuf chiffres significatifs pour la longitude et huit chiffres significatifs pour la latitude)

Lorsque 15 chiffres significatifs sont disponibles, il existe de nombreuses représentations possibles d'un nombre à 9 chiffres significatifs. Un nombre à virgule flottante en double précision utilise 52 bits explicites pour représenter le significand (mantissa) de la coordonnée. Seuls 30 bits sont nécessaires pour représenter un mantissa de 9 chiffres significatifs, ce qui laisse 22 bits non significatifs ; nous pouvons leur donner la valeur que nous voulons et obtenir un nombre qui s'arrondit à notre valeur d'entrée. Par exemple, la valeur 100,123456 peut être représentée par les nombres à virgule flottante les plus proches de 100,123456000000, 100,123456000001 et 100,123456432199. Tous sont également valables, en ce sens que `ST_AsText(geom, 6)` renverra le même résultat avec n'importe laquelle de ces entrées. Comme nous pouvons fixer ces bits à n'importe quelle valeur, `ST_QuantizeCoo` fixe les 22 bits non significatifs à zéro. Pour une longue séquence de coordonnées, cela crée un motif de blocs de zéros consécutifs qui est compressé par PostgreSQL de manière plus efficace.



Note
Seule la taille sur disque de la géométrie est potentiellement affectée par `ST_QuantizeCoordinates`. ST_MemSize, qui indique l'utilisation en mémoire de la géométrie, renverra la même valeur quel que soit l'espace disque utilisé par une géométrie.

Exemples

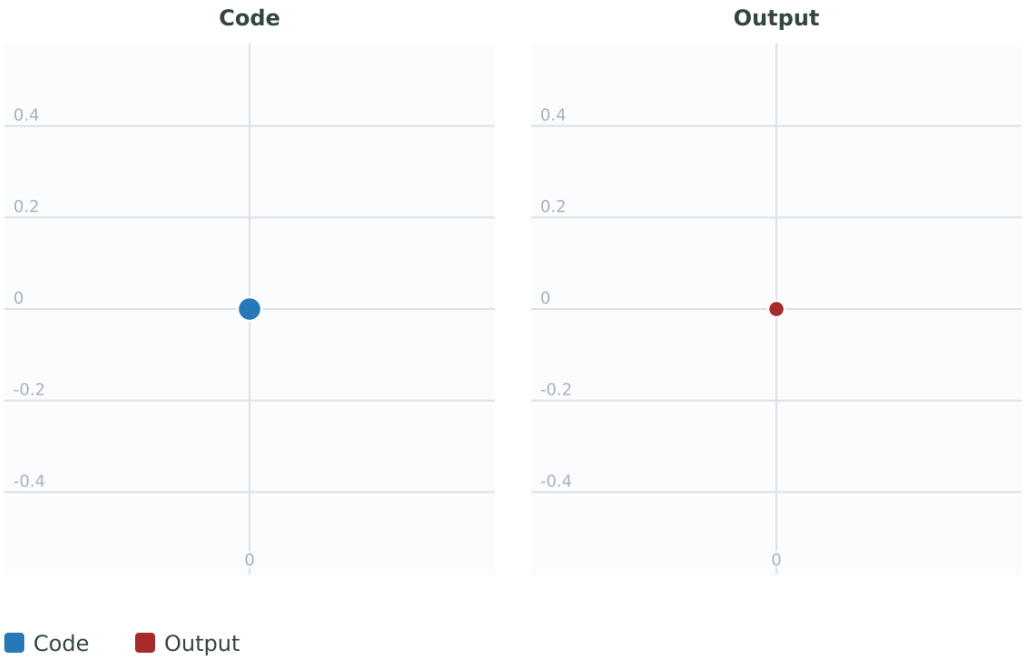
Code

```
SELECT ST_QuantizeCoordinates('POINT (100.123456 0)'::geometry, 4);
```

Output

```
POINT(100.12344360351562 0)
```

Figure



Code

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)'::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_QuantizeCoordinates(geom, digits)
FROM test, generate_series(15, -15, -1) AS digits;
```

Output

```
digits |                  encode                  |                 st_astext
--------+------------------------------------------+-------------------------------------------
     15 | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456)
     14 | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456 123.456789123456)
     13 | 01010000005e9a72083cdd5e405e9a72083cdd5e40 | POINT(123.45678912345599 123.45678912345599)
     12 | 0101000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.45678912345556 123.45678912345556)
     11 | 0101000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.45678912345466 123.45678912345466)
     10 | 0101000000009872083cdd5e40009872083cdd5e40 | POINT(123.45678912344738 123.45678912344738)
      9 | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336007 123.45678912336007)
      8 | 0101000000000070083cdd5e40000070083cdd5e40 | POINT(123.45678912103176 123.45678912103176)
      7 | 0101000000000060083cdd5e40000060083cdd5e40 | POINT(123.4567891061306 123.4567891061306)
      6 | 0101000000000000083cdd5e40000000083cdd5e40 | POINT(123.45678901672363 123.45678901672363)
      5 | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 123.456787109375)
      4 | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375 123.456787109375)
      3 | 0101000000000000038dd5e400000000038dd5e40 | POINT(123.45654296875 123.45654296875)
      2 | 0101000000000000000dd5e400000000000dd5e40 | POINT(123.453125 123.453125)
      1 | 0101000000000000000dc5e400000000000dc5e40 | POINT(123.4375 123.4375)
      0 | 0101000000000000000c05e400000000000c05e40 | POINT(123 123)
     -1 | 0101000000000000000005e400000000000005e40 | POINT(120 120)
     -2 | 0101000000000000000005840000000000005840 | POINT(96 96)
     -3 | 0101000000000000000005840000000000005840 | POINT(96 96)
     -4 | 0101000000000000000005840000000000005840 | POINT(96 96)
     -5 | 0101000000000000000005840000000000005840 | POINT(96 96)
```

```
     -6 | 0101000000000000000000584000000000000005840 | POINT(96 96)
     -7 | 0101000000000000000000584000000000000005840 | POINT(96 96)
     -8 | 0101000000000000000000584000000000000005840 | POINT(96 96)
     -9 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -10 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -11 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -12 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -13 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -14 | 0101000000000000000000584000000000000005840 | POINT(96 96)
    -15 | 0101000000000000000000584000000000000005840 | POINT(96 96)
(31 rows)
```

Voir aussi

ST_SnapToGrid

7.5.24 ST_RemovePoint

ST_RemovePoint — Supprime un point d'une ligne.

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

Description

Supprime un point d'une LineString, en fonction de son index (basé sur 0). Utile pour transformer une ligne fermée (anneau) en une ligne ouverte.

Amélioration : 3.2.0

Disponibilité : 1.1.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Garantit qu'aucune ligne n'est fermée en supprimant le point final des lignes fermées (anneaux). Suppose que geom est de type LINESTRING

Code

```
UPDATE sometable
    SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
    FROM sometable
    WHERE ST_IsClosed(geom);
```

Voir aussi

ST_AddPoint, ST_NPoints, ST_NumPoints

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — Renvoie une version d'une géométrie dont les points en double ont été supprimés.

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance = 0.0);

Description

Renvoie une version de la géométrie donnée dont les points consécutifs en double ont été supprimés. La fonction ne traite que les (Multi)LineStrings, les (Multi)Polygons et les MultiPoints, mais elle peut être appelée avec n'importe quel type de géométrie. Les éléments des collections de géométries sont traités individuellement. Les extrémités des lignes sont préservées.

If a non-zero *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates. The distance is computed in 2D (XY plane).

Amélioration : 3.2.0

Disponibilité : 2.2.0

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

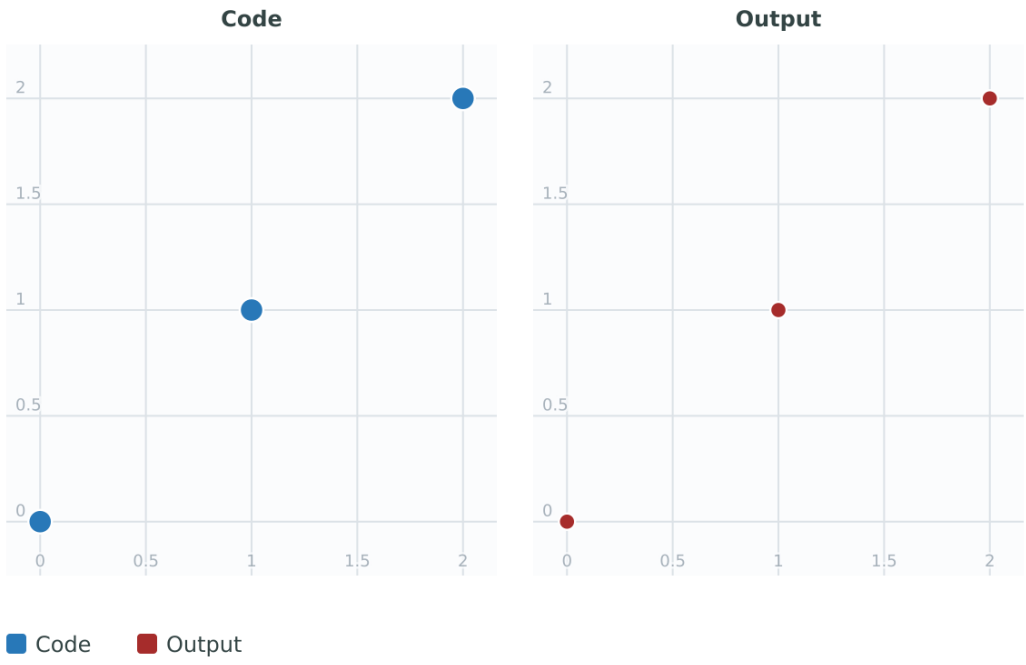
Code

```
SELECT ST_RemoveRepeatedPoints('MULTIPOINT ((1 1),(2 2),(3 3),(2 2))');
```

Output

```
MULTIPOINT((1 1),(2 2),(3 3))
```

Figure



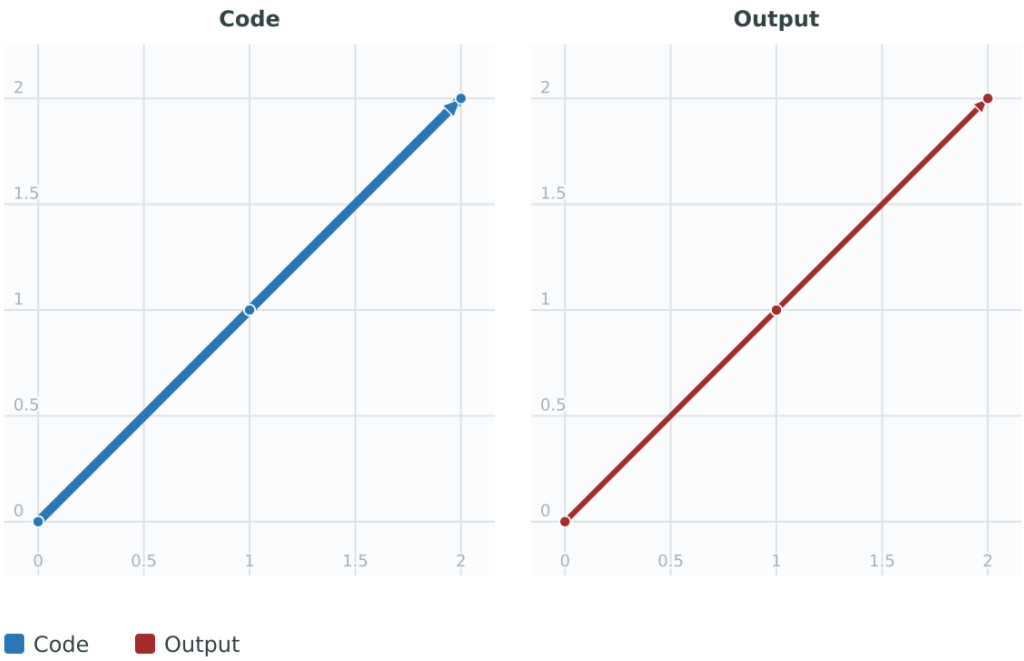
Code

```
SELECT ST_RemoveRepeatedPoints('LINESTRING (0 0,0 0,1 1,0 0,1 1,2 2)');
```

Output

```
LINESTRING(0 0,1 1,0 0,1 1,2 2)
```

Figure



Exemple: Les éléments de la collection sont traités individuellement.

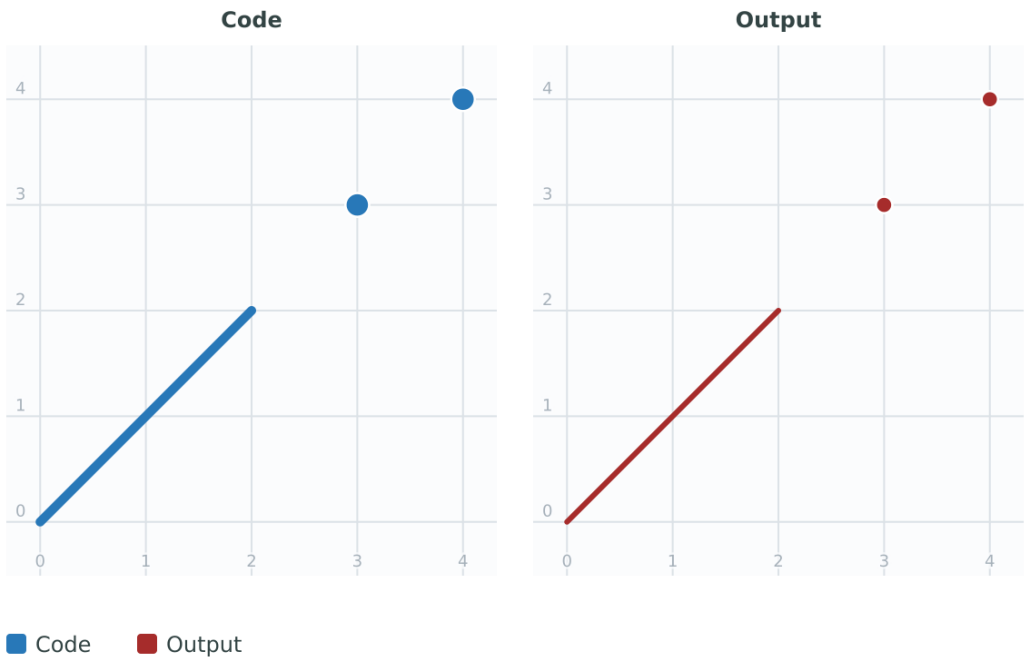
Code

```
SELECT ST_RemoveRepeatedPoints('GEOMETRYCOLLECTION (LINESTRING (1 1,2 2,2 2,3 3),POINT (4 4) ↩
    ,POINT (4 4),POINT (5 5))');
```

Output

```
GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3), POINT(4 4), POINT(4 4), POINT(5 5))
```

Figure



Exemple: Suppression répété de points avec une tolérance de distance.

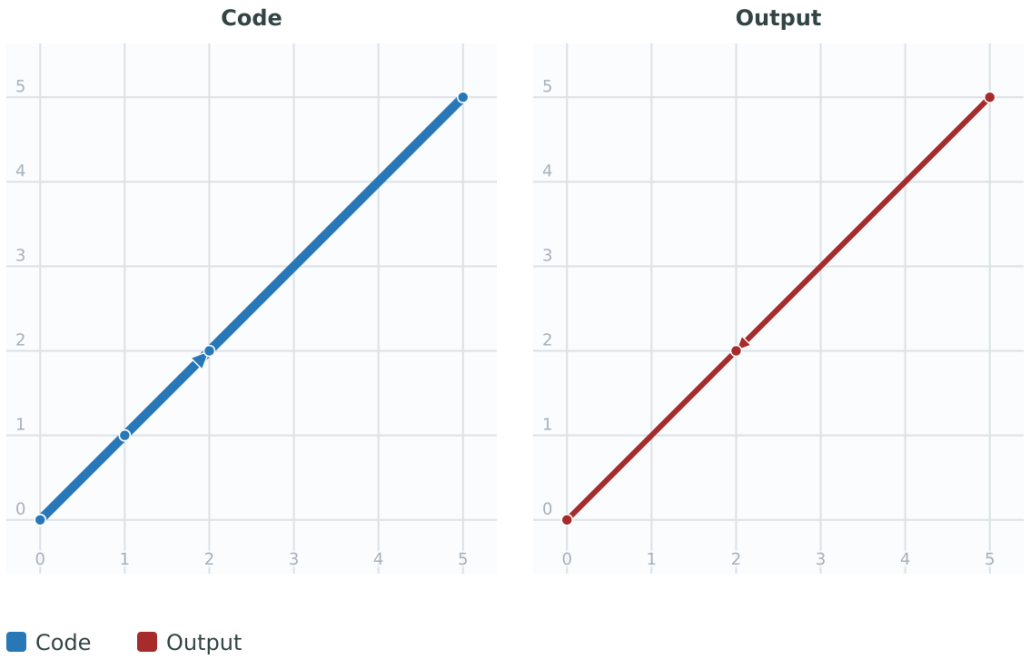
Code

```
SELECT ST_RemoveRepeatedPoints('LINESTRING (0 0,0 0,1 1,5 5,1 1,2 2)', 2);
```

Output

```
LINESTRING(0 0,5 5,2 2)
```

Figure



Voir aussi

ST_Simplify

7.5.26 ST_RemoveIrrelevantPointsForView

ST_RemoveIrrelevantPointsForView — Supprime les points qui ne sont pas pertinents pour le rendu d'une vue rectangulaire spécifique d'une géométrie.

Synopsis

geometry **ST_RemoveIrrelevantPointsForView**(geometry geom, box2d bounds, boolean cartesian_hint = false);

Description

Renvoie une geometry sans les points non pertinents pour le rendu de la géométrie dans une vue rectangulaire donnée.

Cette fonction peut être utilisée pour prétraiter rapidement les géométries qui ne doivent être rendues qu'à l'intérieur de certaines limites.

Seules les géométries de type (MULTI)POLYGONE et (MULTI)LINESTRING sont évaluées. Les autres géométries restent inchangées.

Contrairement à `ST_ClipByBox2D()`, cette fonction

— trie les points sans calculer de nouveaux points d'intersection, ce qui évite les erreurs d'arrondi et augmente généralement les performances,
— renvoie une géométrie dont le nombre de points est égal ou similaire,
— conduit au même résultat de rendu dans la vue spécifiée, et
— peut introduire des auto-intersections qui rendraient la géométrie résultante invalide (voir l'exemple ci-dessous).

Si `cartesian_hint` est défini à `true`, l'algorithme applique des optimisations supplémentaires impliquant des mathématiques cartésiennes pour réduire davantage le nombre de points résultant. Veuillez noter que l'utilisation de cette option peut introduire des artefacts de rendu si les coordonnées résultantes sont projetées dans un autre système de coordonnées (non cartésien) avant le rendu.



AVERTISSEMENT
Pour les polygones, cette fonction ne garantit pas actuellement que le résultat est valide. Cette situation peut être vérifiée avec ST_IsValid et réparée avec ST_MakeValid.

Disponibilité : 3.5.0

Exemples

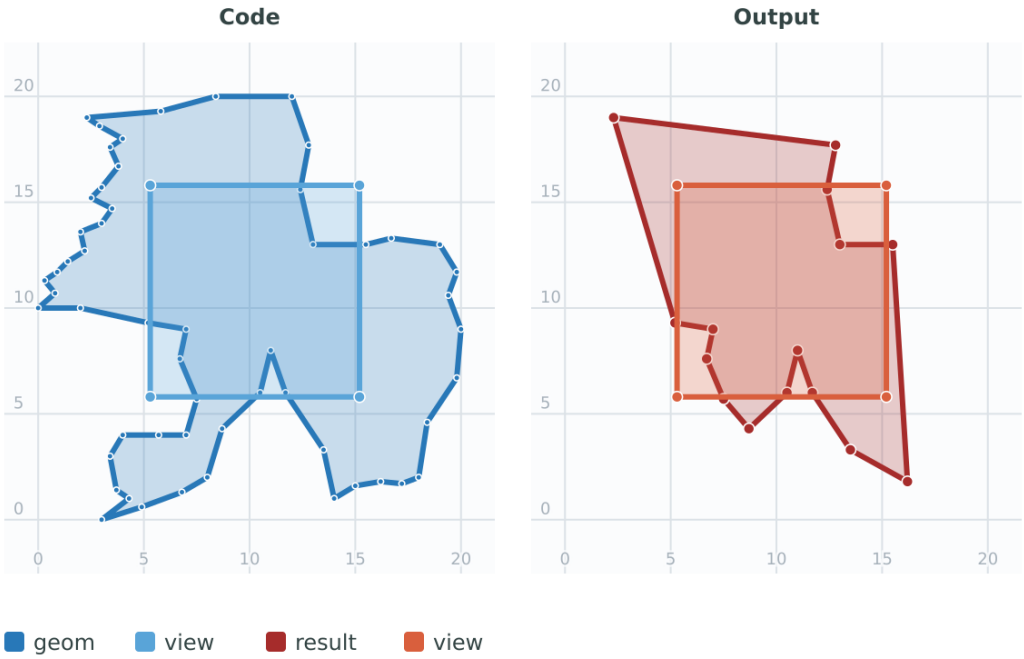
The input polygon has 55 points. The result retains 15 points, while rendering the same shape inside the view envelope.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((3.2 14.2,6.8 14.2,7.6 11.9,7.2 9.8,7.8 7.2,10.3 7.2,11.5 7.5,13.8 7.2,14.6 ↩
   5.9,14.2 4.8,14.8 3.2,14.6 0.9,13.2 -1.2,12.8 -3.8,12 -4.1,11 -4,9.8 -4.2,8.8 -4.8,8.3 ↩
   -2.5,6.5 0.2,5.8 2.2,5.3 0.2,3.5 -1.5,2.8 -3.8,1.6 -4.5,-0.3 -5.2,-2.2 -5.8,-0.9 ↩
   -4.8,-1.5 -4.4,-1.8 -2.8,-1.2 -1.8,0.5 -1.8,1.8 -1.8,2.3 -0.1,1.5 1.8,1.8 3.2,0 3.5,-3.2 ↩
    4.2,-5.2 4.2,-4.4 4.9,-4.9 5.5,-4.3 5.9,-3.8 6.4,-3 6.9,-3.2 7.8,-2.2 8.2,-1.7 8.9,-2.7 ↩
    9.4,-2.2 9.9,-1.4 10.9,-1.8 11.8,-1.2 12.2,-2.3 12.8,-2.9 13.2,0.6 13.5,3.2 14.2))':: ↩
   geometry,
```

```
  ST_MakeEnvelope(0.1, 0, 10, 10)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure

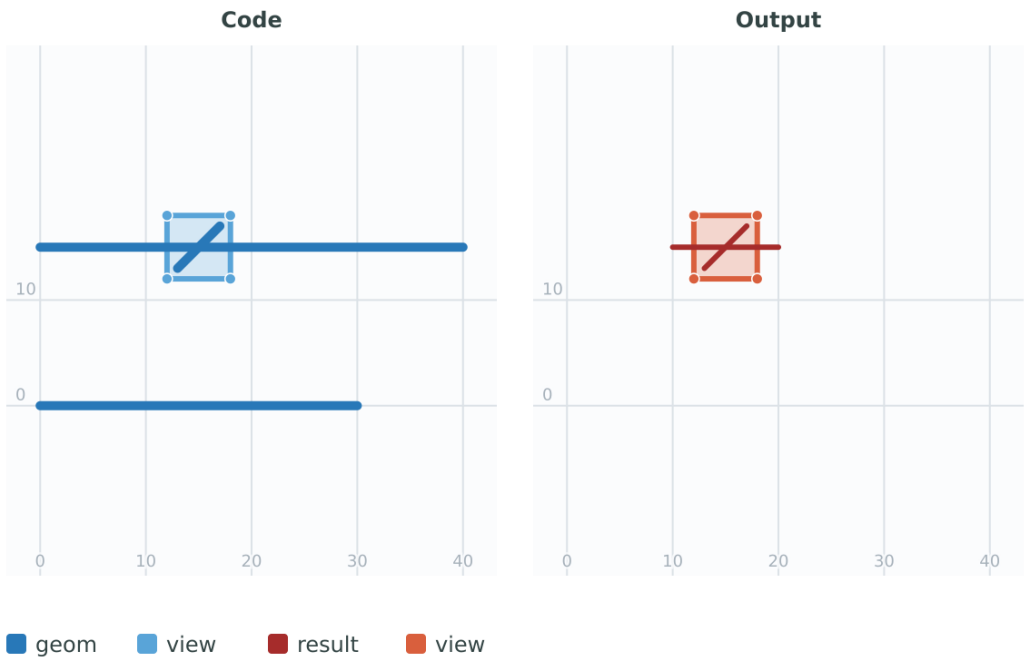


Only linework affecting the view is retained; existing vertices are reused.

Code

```
WITH data(geom, view) AS (VALUES (
  'MULTILINESTRING((0 0,10 0,20 0,30 0),(0 15,5 15,10 15,15 15,20 15,25 15,30 15,40 15),(13 ↩
    13,15 15,17 17))'::geometry,
  ST_MakeEnvelope(12, 12, 18, 18)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure



A line with no segment relevant to the view becomes empty.

Code

```
SELECT ST_RemoveIrrelevantPointsForView('LINESTRING(0 0,10 0,20 0,30 0)',
ST_MakeEnvelope(12, 12, 18, 18), true);
```

Output

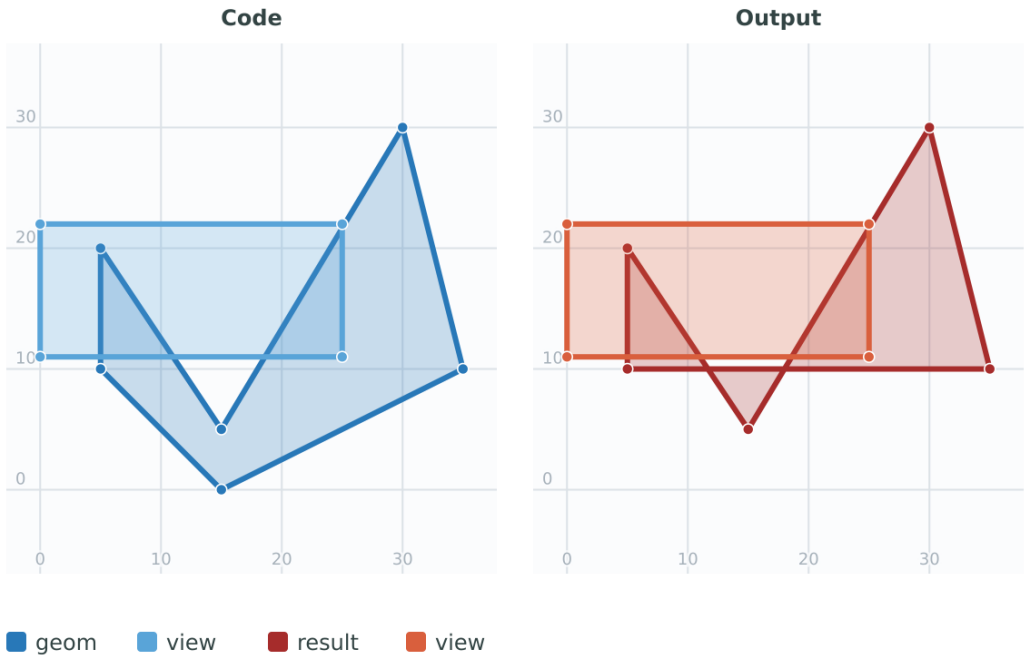
```
LINESTRING EMPTY
```

Removing vertices without computing intersections can introduce self-intersections in the result.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((0 10,0 0,10 -10,30 0,25 20,10 -5,0 10))'::geometry,
  ST_MakeEnvelope(-5, 1, 20, 12)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view, true) AS result,
       view AS view
FROM data;
```

Figure

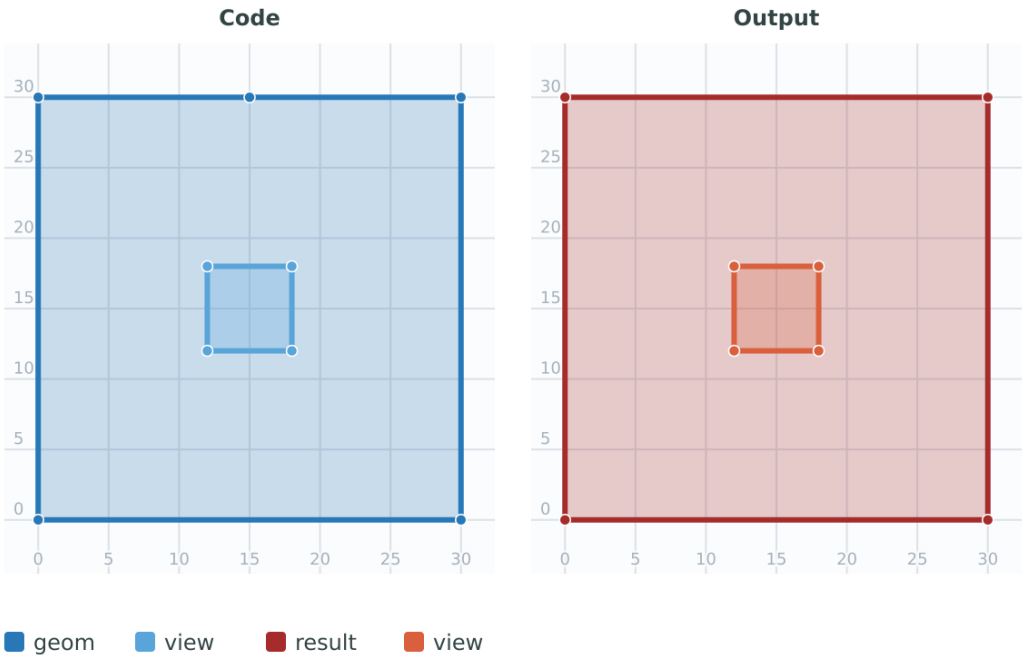


With the default *cartesian_hint*, the function removes only points known to be irrelevant to the view.

Code

```
WITH data(geom, view) AS (VALUES (
  'POLYGON((0 30,15 30,30 30,30 0,0 0,0 30))'::geometry,
  ST_MakeEnvelope(12, 12, 18, 18)
))
SELECT geom AS input_geom,
       view AS input_view,
       ST_RemoveIrrelevantPointsForView(geom, view) AS result,
       view AS view
FROM data;
```

Figure



Voir aussi

ST_ClipByBox2D, ST_Intersection

7.5.27 ST_RemoveSmallParts

ST_RemoveSmallParts — Supprime les petites parties (anneaux de polygone ou lignes) d'une géométrie.

Synopsis

geometry **ST_RemoveSmallParts**(geometry geom, double precision minSizeX, double precision minSizeY);

Description

Retourne un geometry sans les petites parties (anneaux de polygones extérieurs ou intérieurs, ou lignes).

Cette fonction peut être utilisée comme étape de prétraitement pour créer des cartes simplifiées, par exemple pour supprimer les petites îles ou les trous.

Il n'évalue que les géométries de type (MULTI)POLYGONE et (MULTI)LINESTRING. Les autres géométries restent inchangées.

Si `minSizeX` est supérieur à 0, les parties sont triées si leur largeur est inférieure à `minSizeX`.

Si `minSizeY` est supérieur à 0, les parties sont triées si leur hauteur est inférieure à `minSizeY`.

Les deux paramètres, `minSizeX` et `minSizeY` sont mesurés dans les unités du système de coordonnées de la géométrie.

Pour les objets de type polygone, l'évaluation est effectuée séparément pour chaque anneau, ce qui peut conduire à l'un des résultats suivants :

— la géométrie d'origine,
— un POLYGONE avec tous les anneaux ayant moins de sommets,

— un POLYGONE avec un nombre réduit d'anneaux intérieurs (ayant éventuellement moins de sommets),
— un POLYGONE VIDE, ou
— un MULTIPOLYGONE avec un nombre réduit de polygones (ayant éventuellement moins d'anneaux intérieurs ou de sommets), ou
— un MULTIPOLYGONE VIDE.

Pour les types linestring, l'évaluation est effectuée pour chaque ligne, ce qui peut conduire à l'un des résultats suivants :

— la géométrie d'origine,
— une LINESTRING avec un nombre réduit de sommets,
— une LINESTRING VIDE,
— une MULTILINESTRING avec un nombre réduit de lignes (ayant éventuellement moins de sommets), ou
— une MULTILINESTRING VIDE.

Disponibilité : 3.5.0

Exemples

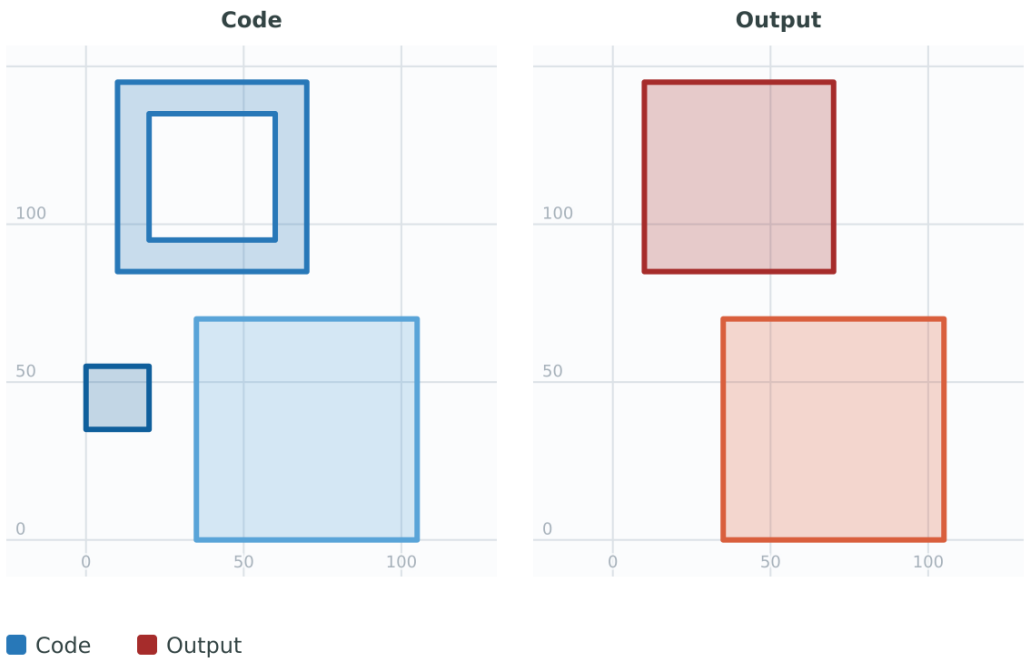
Code

```
SELECT ST_RemoveSmallParts('MULTIPOLYGON(
    ((60 160,120 160,120 220,60 220,60 160),(70 170,70 210,110 210,110 170,70 170)),
    ((85 75,155 75,155 145,85 145,85 75)),
    ((50 110,70 110,70 130,50 130,50 110)))',
    50, 50);
```

Output

```
MULTIPOLYGON(((60 160,120 160,120 220,60 220,60 160)),((85 75,155 75,155 145,85 145,85 75)))
```

Figure



Code

```
SELECT ST_RemoveSmallParts('LINESTRING(10 10,20 20)',
    50, 50);
```

Output

```
LINESTRING EMPTY
```

7.5.28 ST_Reverse

ST_Reverse — Retourne la géométrie avec l'ordre des sommets inversé.

Synopsis

geometry **ST_Reverse**(geometry g1);

Description

Peut être utilisé sur n'importe quelle géométrie et inverse l'ordre des sommets.

Amélioration : la prise en charge des courbes a été introduite dans la version 2.4.0.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

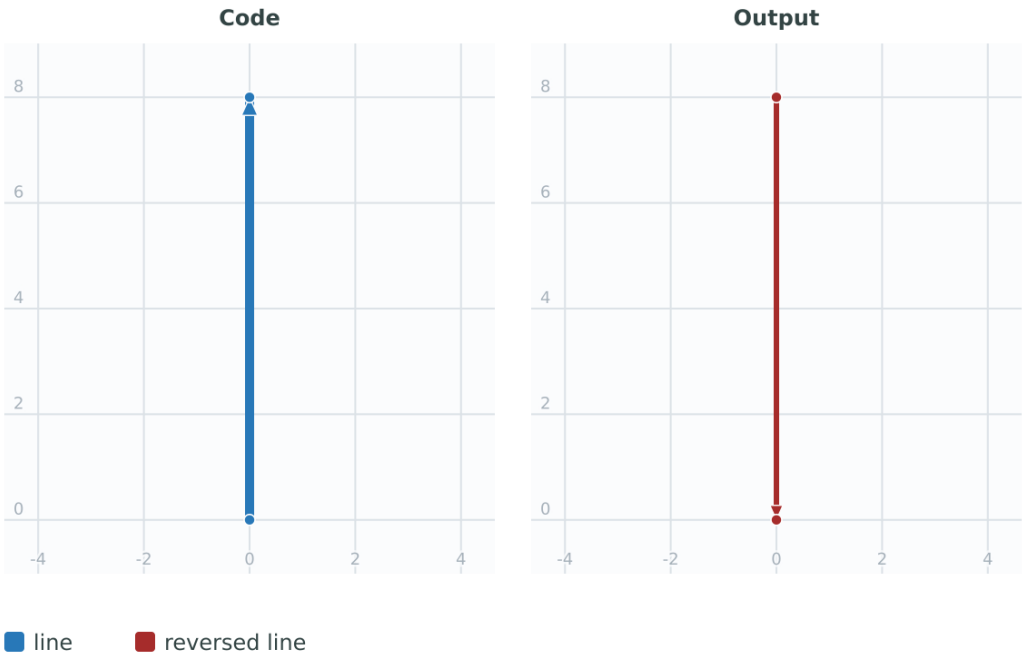
 Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
WITH data(geom) AS (VALUES (
  ST_MakeLine(ST_Point(1, 2), ST_Point(1, 10))
))
SELECT geom AS input_line,
       ST_Reverse(geom) AS reversed_line
FROM data;
```

Figure



7.5.29 ST_Segmentize

ST_Segmentize — Renvoie une geometry/geography modifiée dont aucun segment ne dépasse une distance donnée.

Synopsis

geometry **ST_Segmentize**(geometry geom, float max_segment_length);
geography **ST_Segmentize**(geography geog, float max_segment_length);

Description

Renvoie une geometry/geography modifiée dont aucun segment n'est plus long que `max_segment_length`. La longueur est calculée en 2D. Les segments sont toujours divisés en sous-segments de même longueur.

— Pour la géométrie, la longueur maximale est exprimée dans les unités du système de référence spatiale.
— En géographie, la longueur maximale est exprimée en mètres. Les distances sont calculées sur la sphère. Les sommets ajoutés sont créés le long des arcs de grands cercles sphériques définis par les extrémités des segments.



Note
Cette opération ne fait que raccourcir les segments longs. Elle n'allonge pas les segments plus courts que la longueur maximale.



AVERTISSEMENT
Pour les entrées contenant de longs segments, la spécification d'un `max_segment_length` relativement court peut entraîner l'ajout d'un très grand nombre de sommets. Cela peut se produire involontairement si l'argument est spécifié accidentellement comme un nombre de segments, plutôt que comme une longueur maximale.

Disponibilité : 1.2.2

Amélioration : 3.0.0 La segmentation de géométrie produit désormais des sous-segments de longueur égale

Amélioration : 2.3.0 La segmentation d'objets geography produit désormais des sous-segments de longueur égale

Amélioration : la prise en charge des objets de type geography a été introduite dans la version 2.1.0.

Modifié : 2.1.0 Suite à l'introduction de la prise en charge du type geography, l'utilisation `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` provoque une erreur de fonction ambiguë. L'entrée doit être correctement typée en tant que geometry ou geography. Utilisez ST_GeomFromText, ST_GeogFromText ou un cast vers le type requis (par exemple, `ST_Segmentize('LINEST 2, 3 4)'::geometry, 0.5) )`

Exemples

La segmentation d'une ligne. Les segments longs sont divisés de manière égale et les segments courts ne sont pas divisés.

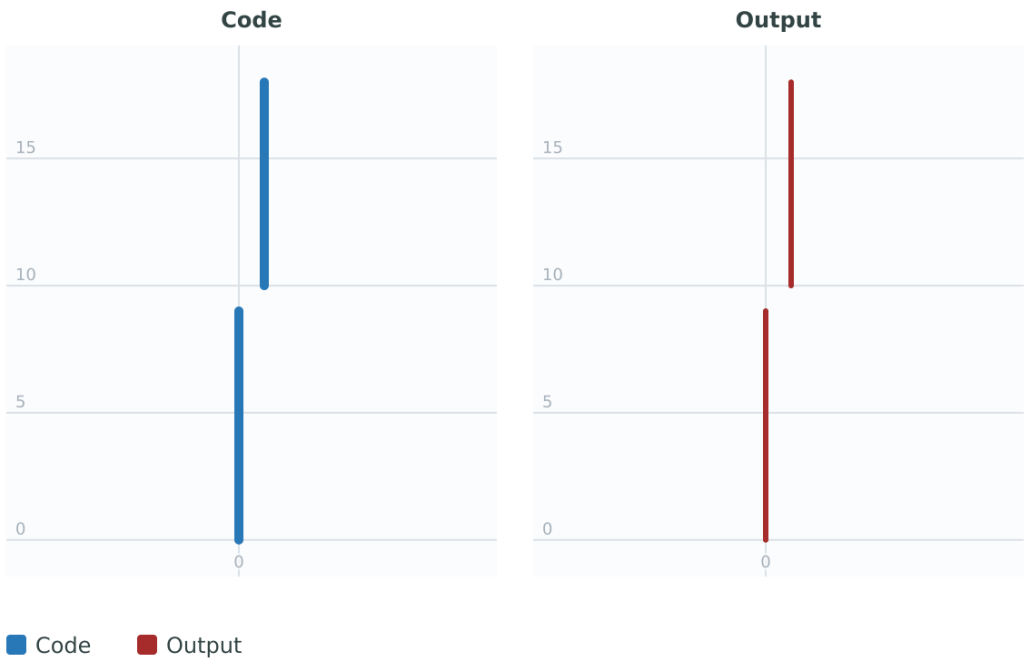
Code

```
SELECT ST_Segmentize('MULTILINESTRING((0 0,0 1,0 9),(1 10,1 18))'::geometry,
5);
```

Output

```
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Figure



Segmentation d'un polygone :

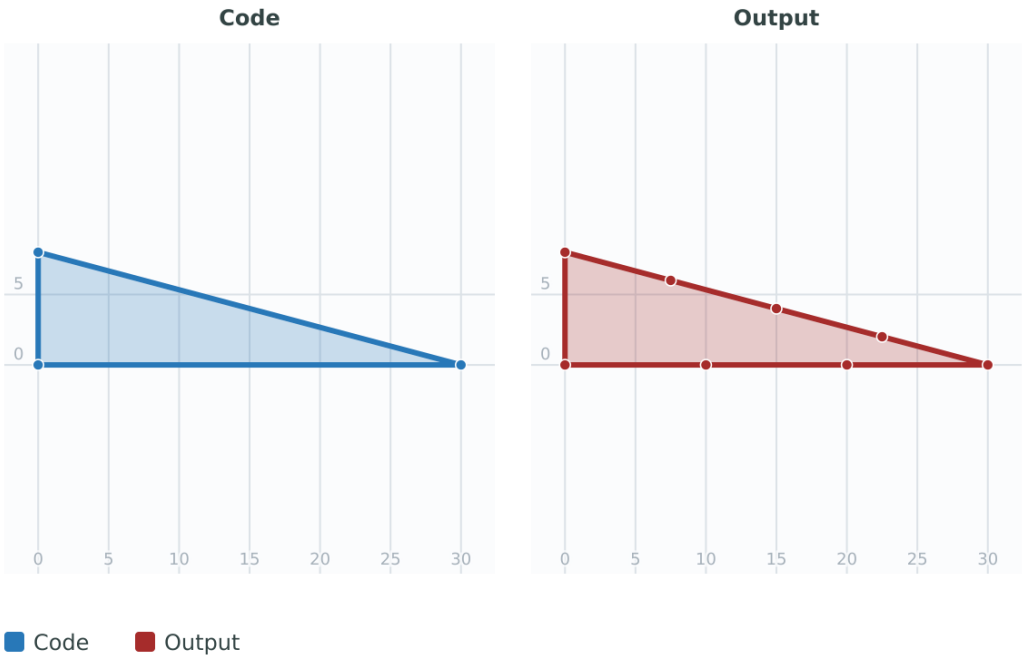
Code

```
SELECT ST_Segmentize('POLYGON((0 0,0 8,30 0,0 0))'::geometry,
        10);
```

Output

```
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Figure



Segmentation d'une ligne géographique, en utilisant une longueur de segment maximale de 2000 kilomètres. Les sommets sont ajoutés le long de l'arc de grand cercle reliant les points d'extrémité.

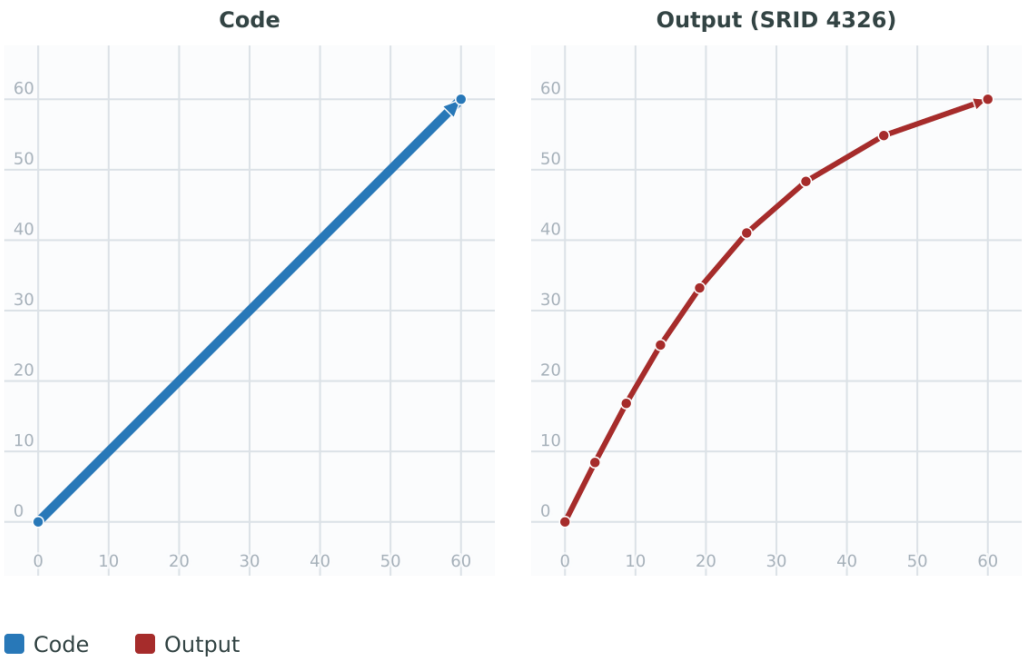
Code

```
SELECT ST_Segmentize('LINESTRING (0 0,60 60)'::geography), 2000000);
```

Output

```
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404 16.824093489701564,13.550465473227048 ↩
    25.107950473646188,19.1066053508691 33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954 ↩
    48.337222885886,45.238153936612264 54.84733442373889,60 60)
```

Figure



Voir aussi

ST_LineSubstring

7.5.30 ST_SetPoint

ST_SetPoint — Remplacer le point d'une ligne par un point donné.

Synopsis

geometry **ST_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

Description

Remplace le point N de la ligne par le point donné. L'index est basé sur 0. Les index négatifs sont comptés à rebours, de sorte que -1 est le dernier point. Cette fonction est particulièrement utile dans les triggers lorsqu'il s'agit de maintenir la relation entre les articulations lorsqu'un sommet se déplace.

Disponibilité : 1.1.0

Mise à jour 2.3.0 : indexation négative

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Change first point in line string from -1 3 to -1 1.

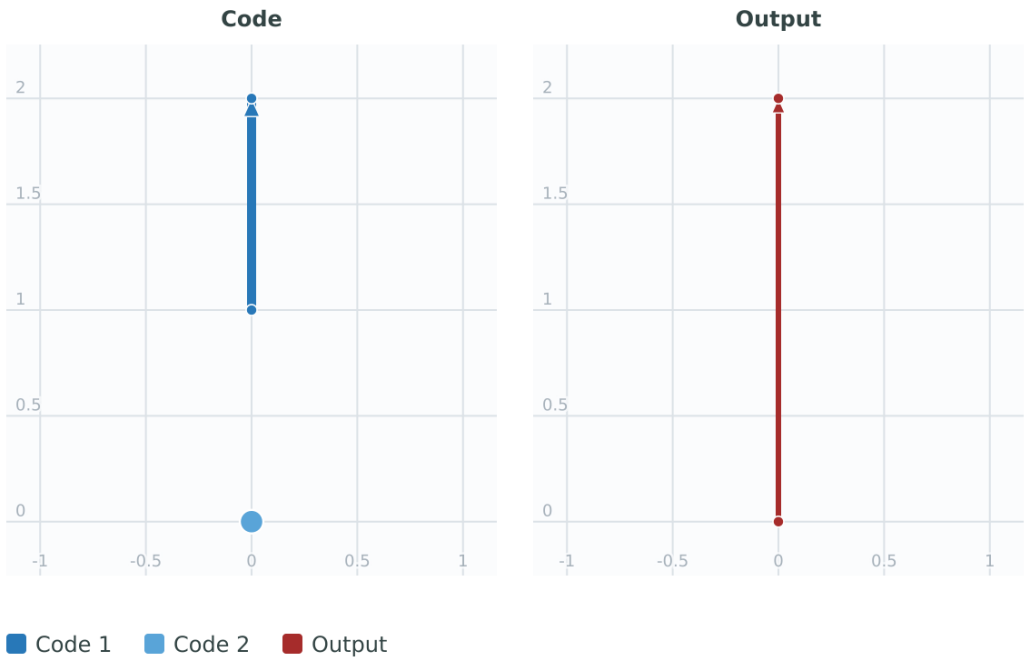
Code

```
SELECT ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)');
```

Output

```
LINESTRING(-1 1,-1 3)
```

Figure



Change last point in a line string (let's play with 3D linestring this time)

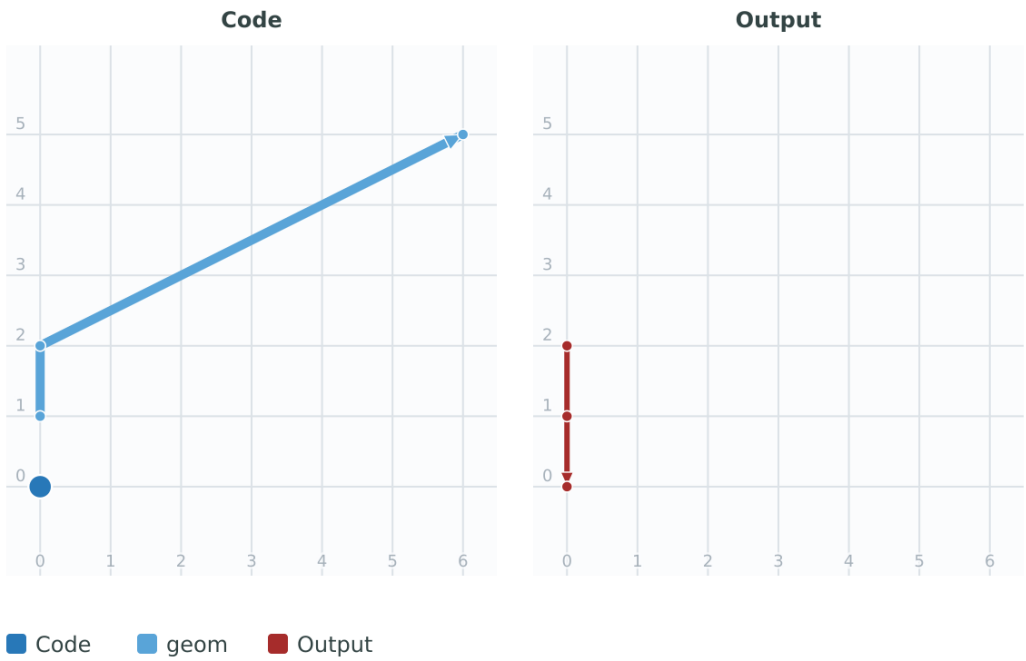
Code

```
SELECT ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, 'POINT(-1 1 3)')
FROM (SELECT 'LINESTRING(-1 2 3,-1 3 4,5 6 7)'::geometry As geom) As foo;
```

Output

```
LINESTRING(-1 2 3,-1 3 4,-1 1 3)
```

Figure



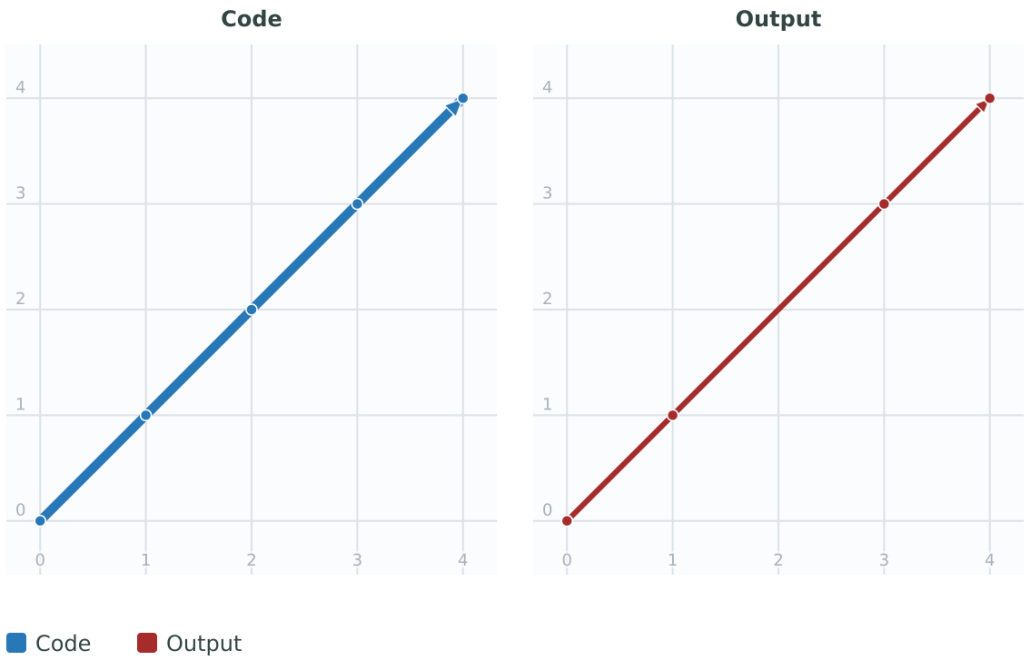
Code

```
SELECT ST_SetPoint(g, -3, p)
FROM ST_GeomFromText('LINESTRING(0 0,1 1,2 2,3 3,4 4)') AS g
    CROSS JOIN ST_PointN(g, 1) as p;
```

Output

```
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

Figure



Voir aussi

ST_AddPoint, ST_NPoints, ST_NumPoints, ST_PointN, ST_RemovePoint

7.5.31 ST_ShiftLongitude

ST_ShiftLongitude — Décale les coordonnées de longitude d'une géométrie entre -180..180 et 0..360.

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

Description

Lit chaque point/vertex d'une géométrie et déplace sa coordonnée de longitude de -180..0 à 180..360 et vice versa si elle se trouve entre ces plages. Cette fonction est symétrique, de sorte que le résultat est une représentation 0..360 d'une donnée -180..180 et une représentation -180..180 d'une donnée 0..360.



Note
Ceci n'est utile que pour les données dont les coordonnées sont exprimées en longitude/latitude ; par exemple, SRID 4326 (WGS 84 géographique)



AVERTISSEMENT
Un bug antérieur à la version 1.3.4 empêchait le fonctionnement pour MULTIPOINT. La version 1.3.4+ fonctionne également avec MULTIPOINT.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques.

NOTE : cette fonction a été renommée "ST_Shift_Longitude" dans la version 2.2.0


Cette fonction prend en charge les surfaces Polyhedral.


Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

This example performs a forward longitude shift on a single point.

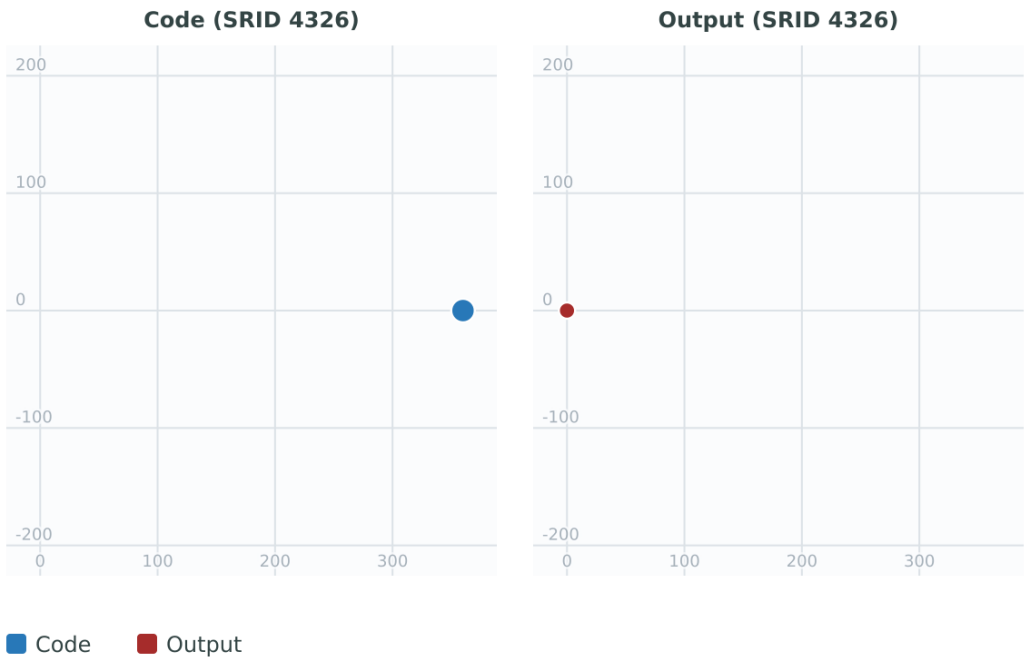
Code

```
SELECT ST_ShiftLongitude('SRID=4326;POINT(270 0)'::geometry)
```

Output

```
POINT(-90 0)
```

Figure



This example performs a reverse longitude shift on a single point.

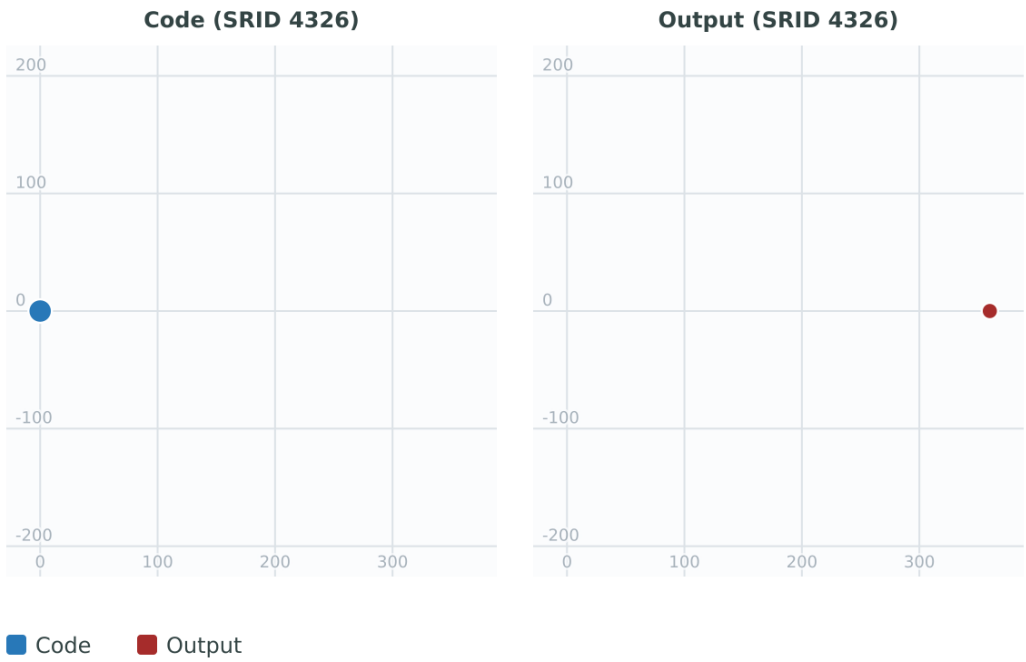
Code

```
SELECT ST_ShiftLongitude('SRID=4326;POINT(-90 0)'::geometry)
```

Output

```
POINT(270 0)
```

Figure



This example shows that LineStrings are shifted only where coordinates require it.

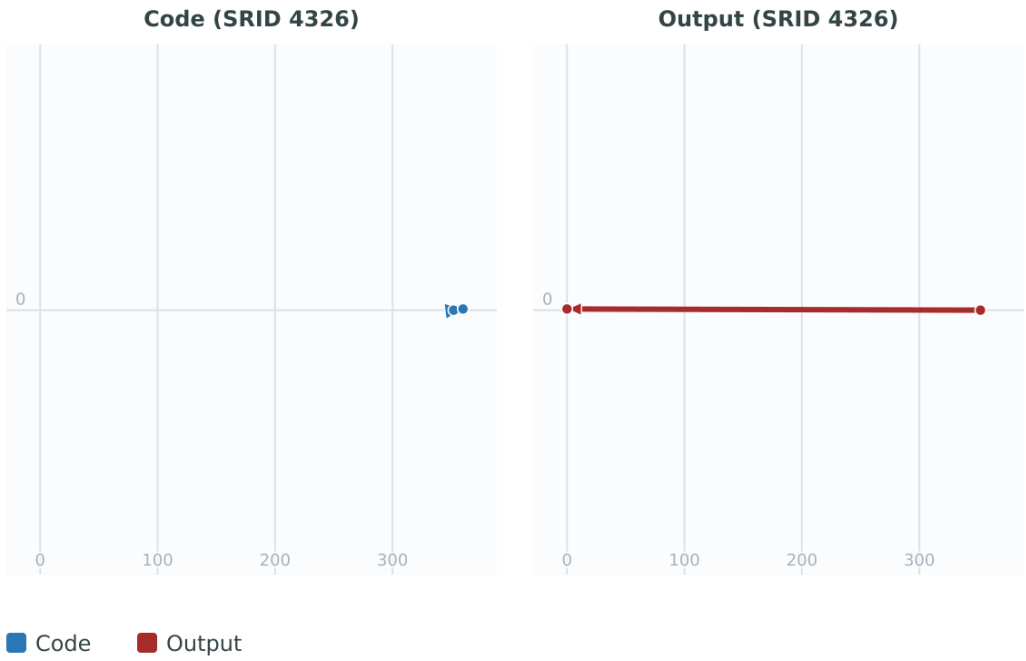
Code

```
SELECT ST_ShiftLongitude('SRID=4326;LINESTRING(174 12,182 13)'::geometry)
```

Output

```
LINESTRING(174 12,-178 13)
```

Figure



Voir aussi

ST_WrapX

7.5.32 ST_WrapX

ST_WrapX — Enveloppe une géométrie autour d'une valeur X.

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

Description

Cette fonction divise les géométries d'entrée et déplace ensuite chaque composant résultant tombant à droite (pour un "move" négatif) ou à gauche (pour un "move" positif) de la ligne "wrap" donnée dans la direction spécifiée par le paramètre "move", pour finalement réassembler les morceaux.



Note
Cette fonction est utile pour "recentrer" les données à long terme afin que les caractéristiques intéressantes ne soient pas produites d'un côté à l'autre.

Disponibilité : 2.3.0 nécessite GEOS

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Move all components whose bounding boxes fall completely to the left of `x = 0` by +360.

Code

```
SELECT ST_WrapX(geom, 0, 360);
```

Move all components whose bounding boxes fall completely to the left of `x = -30` by +360.

Code

```
SELECT ST_WrapX(geom, -30, 360);
```

Voir aussi

ST_ShiftLongitude

7.5.33 ST_SnapToGrid

ST_SnapToGrid — Accrocher tous les points de la géométrie d'entrée à une grille régulière.

Synopsis

geometry **ST_SnapToGrid**(geometry geomA, float originX, float originY, float sizeX, float sizeY);
geometry **ST_SnapToGrid**(geometry geomA, float sizeX, float sizeY);
geometry **ST_SnapToGrid**(geometry geomA, float size);
geometry **ST_SnapToGrid**(geometry geomA, geometry pointOrigin, float sizeX, float sizeY, float sizeZ, float sizeM);

Description

Variante 1,2,3 : Accrocher tous les points de la géométrie d'entrée à la grille définie par l'origine et la taille des cellules. Supprime les points consécutifs tombant sur la même cellule, et renvoie éventuellement NULL si les points de sortie ne sont pas suffisants pour définir une géométrie du type donné. Les géométries écrasées d'une collection sont supprimées de celle-ci. Utile pour réduire la précision.

Variante 4 : Introduite dans la version 1.1.0 - Accroche tous les points de la géométrie d'entrée à la grille définie par son origine (le deuxième argument doit être un point) et la taille des cellules. Spécifiez 0 comme taille pour toute dimension que vous ne voulez pas accrocher à une grille.



Note
La géométrie renvoyée peut perdre sa simplicité (voir ST_IsSimple).



Note
Avant la version 1.1.0, cette fonction renvoyait toujours une géométrie 2D. A partir de la version 1.1.0, la géométrie renvoyée aura la même dimensionnalité que la géométrie d'entrée, les valeurs de dimensions supérieures n'étant pas modifiées. Utilisez la version prenant un second argument géométrique pour définir toutes les dimensions de la grille.

Disponibilité : 1.0.0RC1

Disponibilité : 1.1.0 - Prise en charge des Z et M

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Snap a geometry column to a precision grid of 10^-3.

Code

```
UPDATE mytable
   SET geom = ST_SnapToGrid(geom, 0.001);
```

Display how the same grid size rounds a LineString.

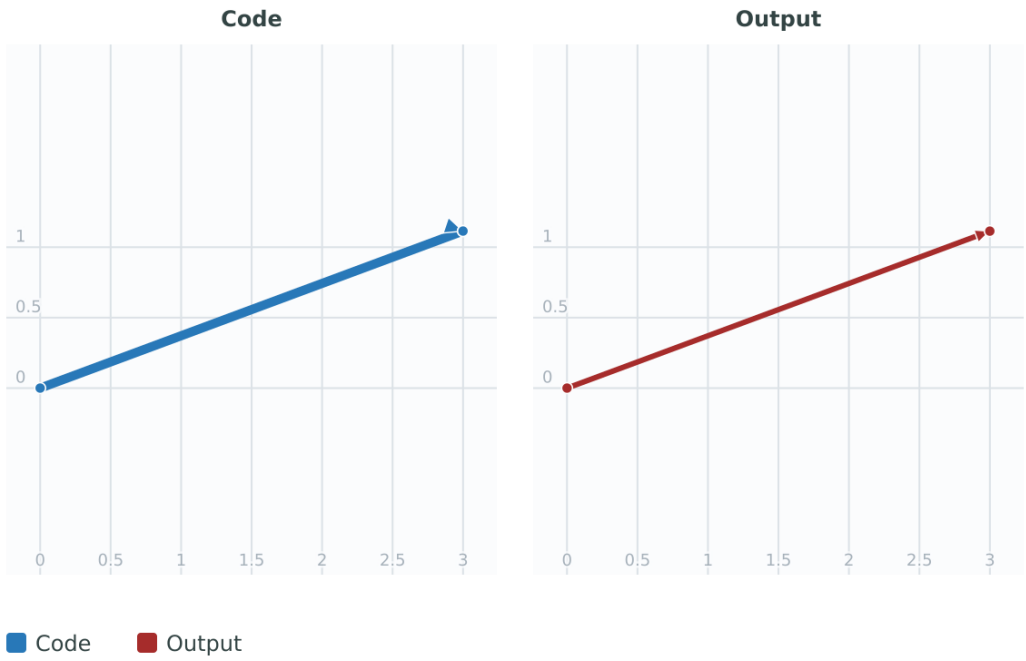
Code

```
SELECT ST_SnapToGrid('LINESTRING(1.1115678 2.123,4.111111 3.2374897,4.11112 3.23748667)':: ↩
    geometry,
        0.001);
```

Output

```
LINESTRING(1.112 2.123,4.111 3.237)
```

Figure



This example snaps a 4D geometry.

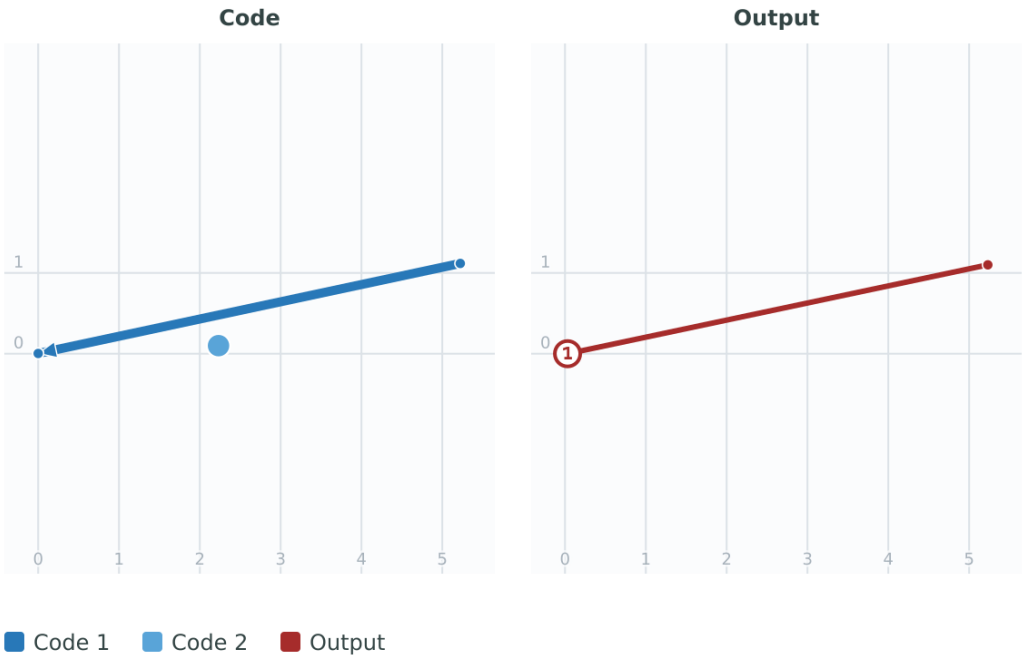
Code

```
SELECT ST_SnapToGrid('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
        4.111111 3.2374897 3.1234 1.1111,-1.11111112 2.123 2.3456 1.1111112)'::geometry,
        'POINT(1.12 2.22 3.2 4.4444)',
        0.1, 0.1, 0.1, 0.01);
```

Output

```
LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)
```

Figure



With a 4D geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m and z the same.

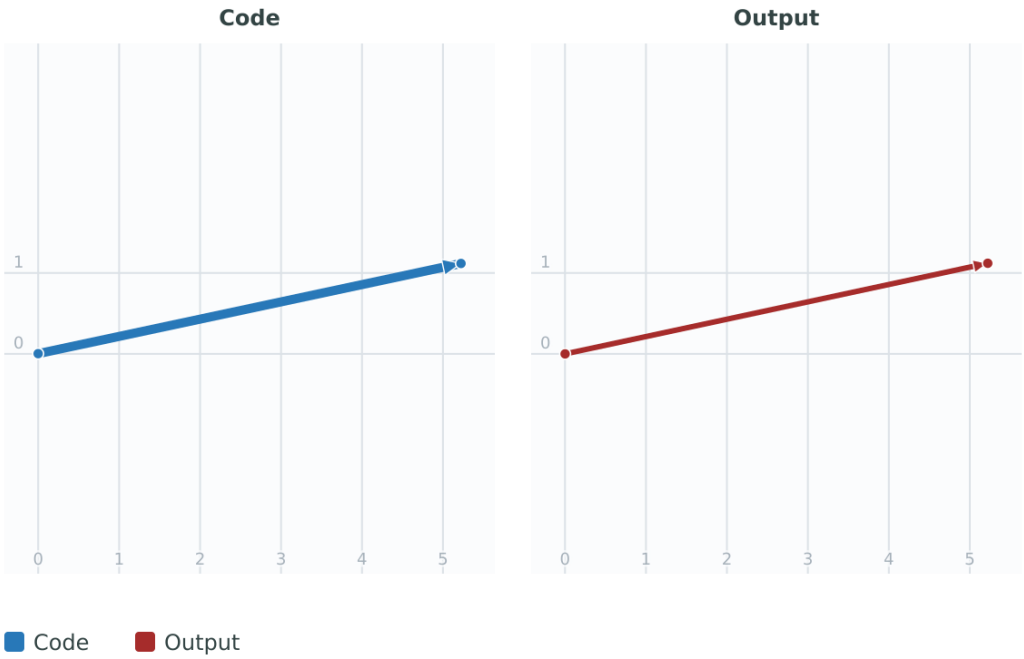
Code

```
SELECT ST_SnapToGrid('LINESTRING(-1.1115678 2.123 3 2.3456,
        4.111111 3.2374897 3.1234 1.1111)'::geometry,
       0.01);
```

Output

```
LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

Figure



Voir aussi

ST_Snap, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_GeomFromEWKT, ST_Simplify

7.5.34 ST_Snap

ST_Snap — Accrocher les segments et les sommets de la géométrie d'entrée aux sommets d'une géométrie de référence.

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

Description

Accroche les sommets et les segments d'une géométrie aux sommets d'une autre géométrie. Une tolérance de distance d'accrochage est utilisée pour contrôler l'endroit où l'accrochage est effectué. La géométrie résultante est la géométrie d'entrée avec les sommets accrochés. Si aucun accrochage n'a lieu, la géométrie d'entrée est renvoyée inchangée.

L'accrochage d'une géométrie à une autre peut améliorer la robustesse des opérations de superposition en éliminant les arêtes presque coïncidentes (qui posent des problèmes lors du noding et du calcul de l'intersection).

Un accrochage trop important peut entraîner la création d'une topologie non valide, c'est pourquoi le nombre et l'emplacement des sommets accrochés sont décidés à l'aide d'une heuristique pour déterminer quand il est sûr d'accrocher. Cela peut toutefois entraîner l'omission de certains accrochages potentiels.



Note
La géométrie renvoyée peut perdre sa simplicité (voir ST_IsSimple) et sa validité (voir ST_IsValid).

Effectué par le module GEOS.

Disponibilité : 2.0.0

Exemples

A MultiPolygon snapped to a LineString at 1.01 times their distance.

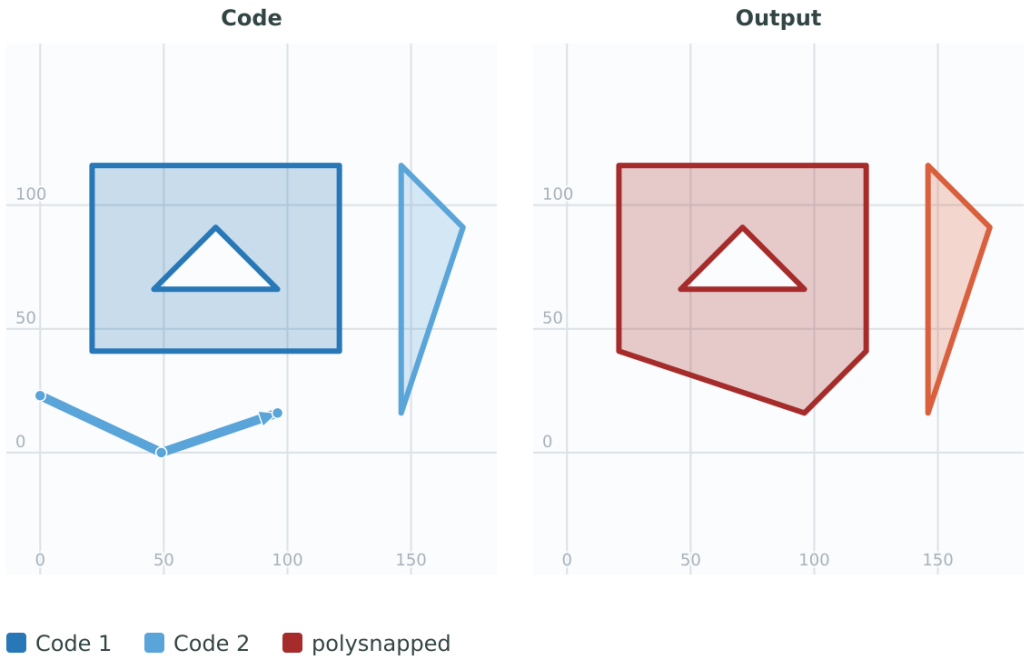
Code

```
SELECT ST_Snap(poly, line, ST_Distance(poly, line)*1.01) AS polysnapped
FROM (SELECT
   ST_GeomFromText('MULTIPOLYGON(
     ((26 125,26 200,126 200,126 125,26 125 ),
      ( 51 150,101 150,76 175,51 150 )),
      (( 151 100,151 200,176 175,151 100 )))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
MULTIPOLYGON(((26 125,26 200,126 200,126 125,101 100,26 125),
(51 150,101 150,76 175,51 150)),((151 100,151 200,176 175,151 100)))
```

Figure



A MultiPolygon snapped to a LineString at 1.25 times their distance.

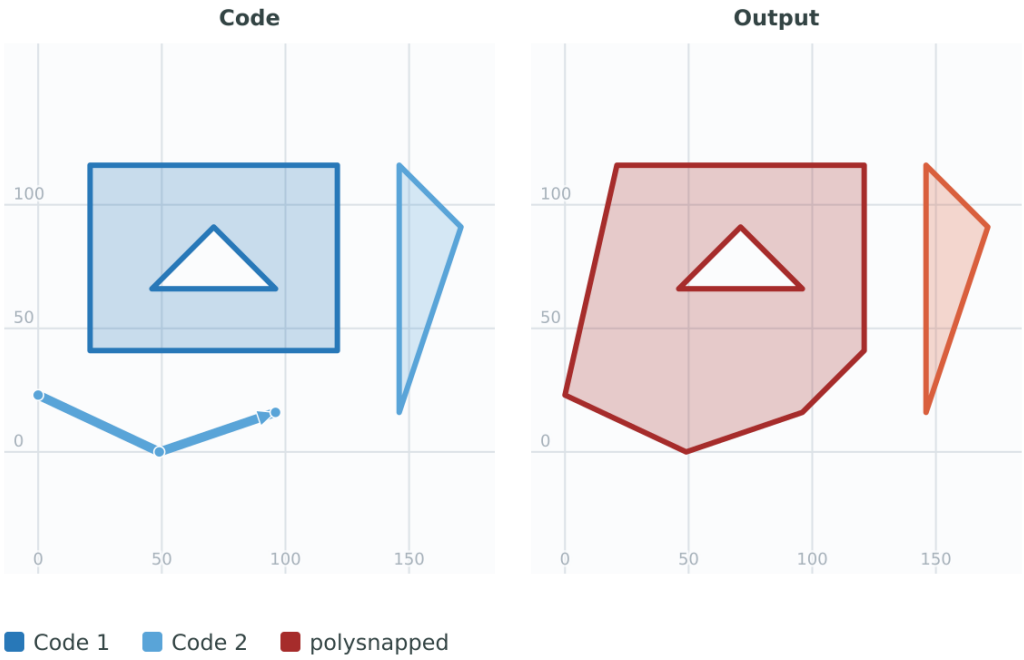
Code

```
SELECT ST_Snap(poly, line, ST_Distance(poly, line)*1.25) AS polysnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    (( 26 125,26 200,126 200,126 125,26 125 ),
      ( 51 150,101 150,76 175,51 150 )),
      (( 151 100,151 200,176 175,151 100 )))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
MULTIPOLYGON(((5 107,26 200,126 200,126 125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151 100,151 200,176 175,151 100)))
```

Figure



The LineString snapped to the original MultiPolygon at 1.01 times their distance.

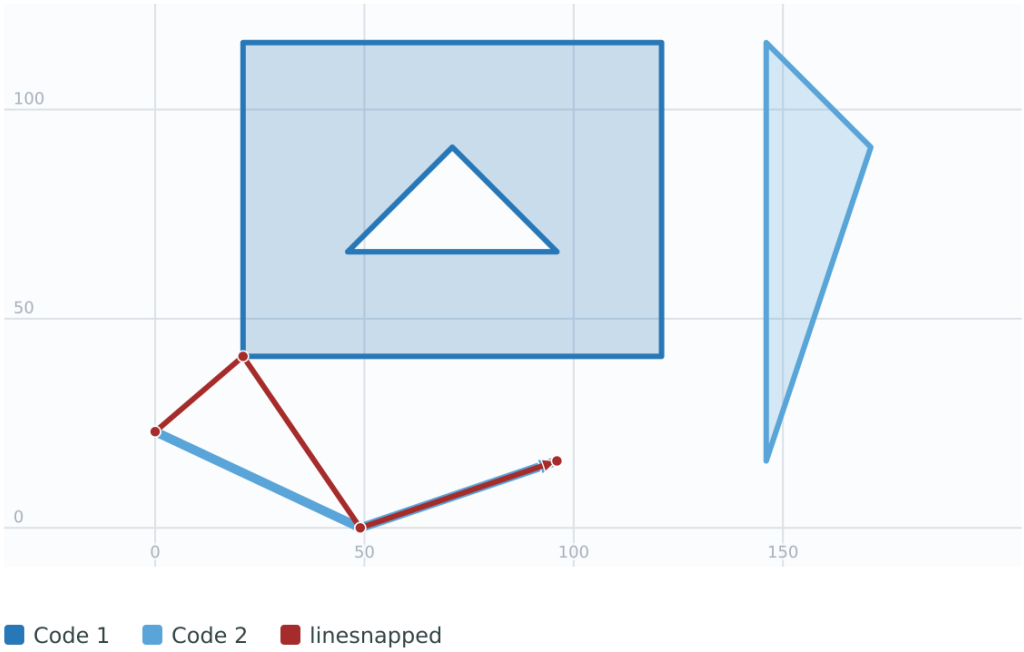
Code

```
SELECT ST_Snap(line, poly, ST_Distance(poly, line)*1.01) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
     ((26 125,26 200,126 200,126 125,26 125),
      (51 150,101 150,76 175,51 150 )),
      ((151 100,151 200,176 175,151 100)))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
LINESTRING(5 107,26 125,54 84,101 100)
```

Figure



The LineString snapped to the original MultiPolygon at 1.25 times their distance.

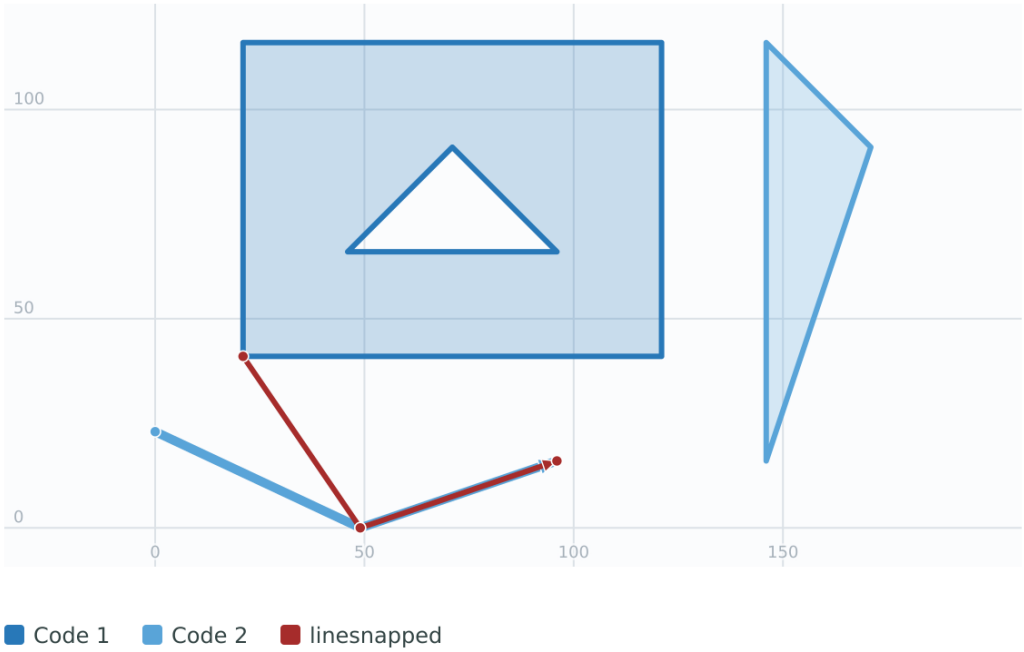
Code

```
SELECT ST_Snap(line, poly, ST_Distance(poly, line)*1.25) AS linesnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
     (( 26 125,26 200,126 200,126 125,26 125 ),
      (51 150,101 150,76 175,51 150 )),
      ((151 100,151 200,176 175,151 100 )))') As poly,
       ST_GeomFromText('LINESTRING (5 107,54 84,101 100)') As line
    ) As foo;
```

Output

```
LINESTRING(26 125,54 84,101 100)
```

Figure



Voir aussi

ST_SnapToGrid

7.5.35 ST_SwapOrdinates

ST_SwapOrdinates — Renvoie une version de la géométrie donnée avec les valeurs d'ordonnées permutées.

Synopsis

geometry **ST_SwapOrdinates**(geometry geom, cstring ords);

Description

Renvoie une version de la géométrie donnée dont les ordonnées ont été interverties.

Le paramètre `ords` est une chaîne de 2 caractères désignant les ordonnées à permuter. Les noms valides sont : x, y, z et m.

Disponibilité : 2.2.0

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemple

This example scales the M value by 2.

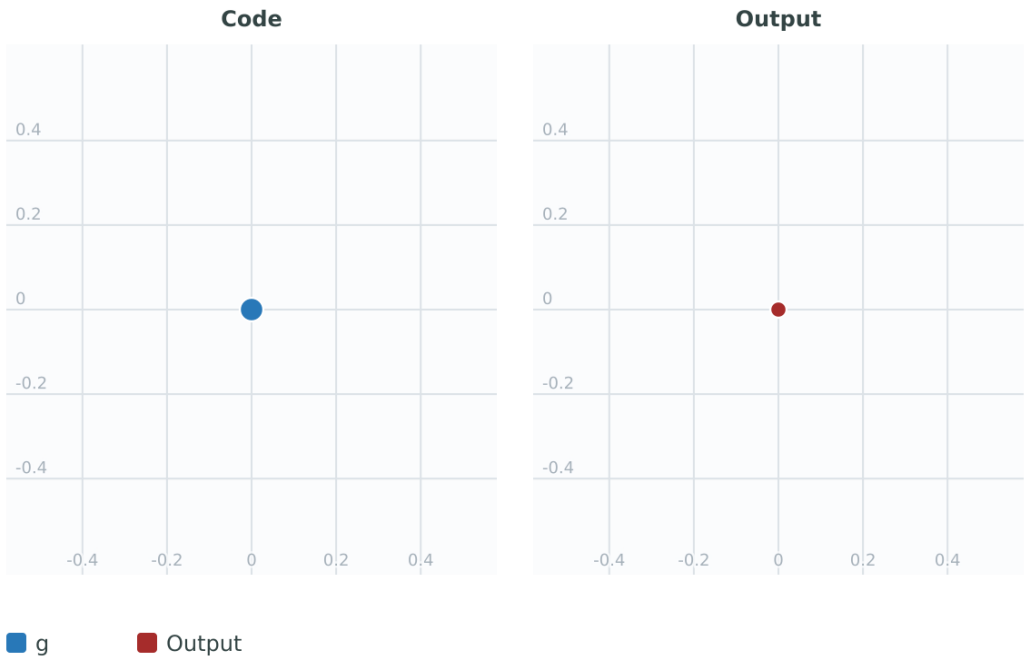
Code

```
SELECT ST_SwapOrdinates(ST_Scale(ST_SwapOrdinates(g, 'xm'),
      2, 1
    ),
  'xm') FROM ( SELECT 'POINT ZM (0 0 0 2)'::geometry g ) foo;
```

Output

```
POINT ZM (0 0 0 4)
```

Figure



Voir aussi

ST_FlipCoordinates

7.6 Validation de la géométrie

7.6.1 ST_IsValid

ST_IsValid — Teste si une géométrie est bien formée en 2D.

Synopsis

boolean **ST_IsValid**(geometry g);
boolean **ST_IsValid**(geometry g, integer flags);

Description

Teste si une valeur ST_Geometry est bien formée et valide en 2D selon les règles de l'OGC. Pour les géométries à 3 et 4 dimensions, la validité est toujours testée uniquement en 2 dimensions. Pour les géométries qui ne sont pas valides, une NOTICE PostgreSQL est émise fournissant les détails de la raison pour laquelle elle n'est pas valide.

Pour la version avec le paramètre `flags`, les valeurs prises en charge sont documentées dans ST_IsValidDetail Cette version n'imprime pas de NOTICE expliquant l'invalidité.

Pour plus d'informations sur la définition de la validité des géométries, reportez-vous à Section 4.4



Note
SQL-MM définit le résultat de ST_IsValid(NULL) comme étant 0, alors que PostGIS renvoie NULL.

Effectué par le module GEOS.

La version acceptant les flags est disponible à partir de la version 2.0.0.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.



Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.9



Note
Les spécifications de l'OGC-SFS et de SQL-MM ne comprennent pas d'argument de type flag pour ST_IsValid. L'indicateur est une extension de PostGIS.

Exemples

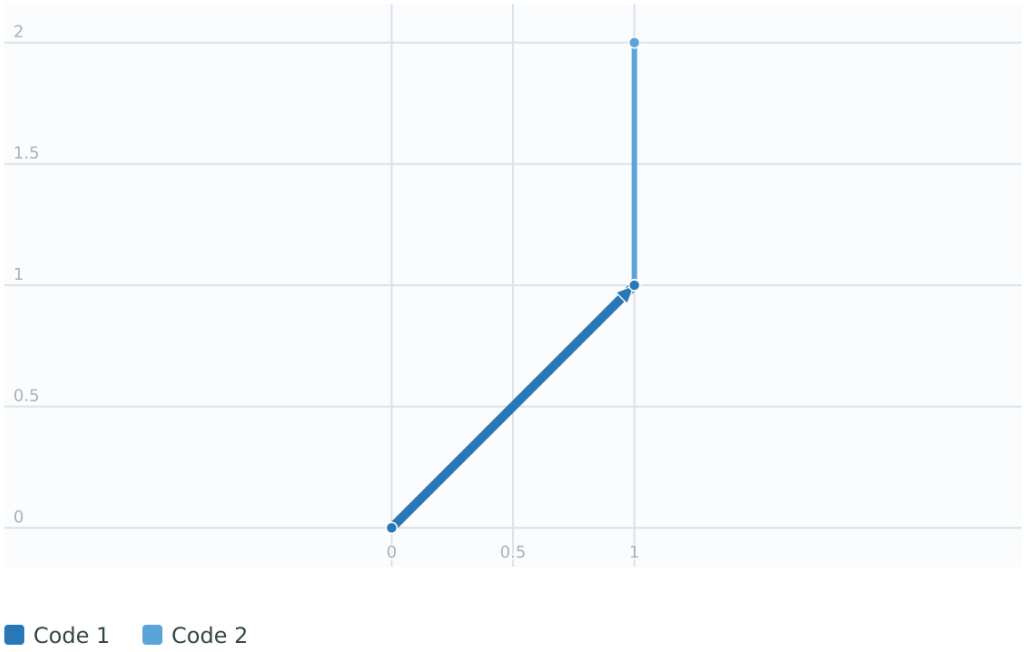
Code

```
SELECT ST_IsValid('LINESTRING(0 0,1 1)') As good_line,
ST_IsValid('POLYGON((0 0,1 1,1 2,1 1,0 0))') As bad_poly;
```

Output

```
NOTICE:  Self-intersection at or near point 0 0
 good_line | bad_poly
-----------+----------
 t         | f
```

Figure



Find tables with invalid geometries in registered geometry columns.

Code

```
CREATE TEMP TABLE invalid_geometry_columns (
    table_name text,
    column_name text,
    invalid_count bigint
);

DO $$
DECLARE
    rec record;
    invalid_count bigint;
BEGIN
    FOR rec IN
        SELECT f_table_schema, f_table_name, f_geometry_column
        FROM geometry_columns
        ORDER BY f_table_schema, f_table_name, f_geometry_column
    LOOP
        EXECUTE format(
            'SELECT count(*) FROM %I.%I WHERE NOT ST_IsValid(%I, 0)',
            rec.f_table_schema,
            rec.f_table_name,
            rec.f_geometry_column
        ) INTO invalid_count;

        IF invalid_count
> 0 THEN
            INSERT INTO invalid_geometry_columns
            VALUES (
                format('%I.%I', rec.f_table_schema, rec.f_table_name),
                rec.f_geometry_column,
                invalid_count
            );
        END IF;
    END LOOP;
```

```
END;
$$;

SELECT * FROM invalid_geometry_columns;
```

Voir aussi

ST_IsSimple, ST_IsValidReason, ST_IsValidDetail,

7.6.2 ST_IsValidDetail

ST_IsValidDetail — Renvoie une ligne `valid_detail` indiquant si une géométrie est valide ou sinon une raison et une localisation.

Synopsis

valid_detail **ST_IsValidDetail**(geometry geom, integer flags);

Description

Renvoie une ligne `valid_detail`, contenant un booléen (`valid`) indiquant si une géométrie est valide, un varchar (`reason`) indiquant une raison pour laquelle elle est invalide et une géométrie (`location`) indiquant où elle est invalide.

Utile pour améliorer la combinaison de ST_IsValid et ST_IsValidReason afin de générer un rapport détaillé des géométries invalides.

Le paramètre facultatif `flags` est un champ de type bit. Il peut avoir les valeurs suivantes :

— 0 : utiliser la sémantique de validité habituelle de l'OGC SFS.

— 1 : Considérer certains types d'anneaux auto-touchants (coquilles inversées et trous exverts) comme valides. Ceci est également connu sous le nom de "flag ESRI", car c'est le modèle de validité utilisé par ces outils. Notez que cela n'est pas valide selon le modèle OGC.

Effectué par le module GEOS.

Disponibilité : 2.0.0

Exemples

This example returns the first three rejects from a successful quintuplet experiment.

Code

```
SELECT gid, reason(ST_IsValidDetail(geom)), location(ST_IsValidDetail(geom)) as location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
    FROM generate_series(-4,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,8) z1
    WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
    INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),y1*1, ↩
     z1*2) As line
    FROM generate_series(-3,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,10) z1
    WHERE x1
```

```
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

Output

```
 gid  |      reason       |  location
------+-------------------+-------------
 5330 | Self-intersection | POINT(32 5)
 5340 | Self-intersection | POINT(42 5)
 5350 | Self-intersection | POINT(52 5)
```

This example reports a valid LineString.

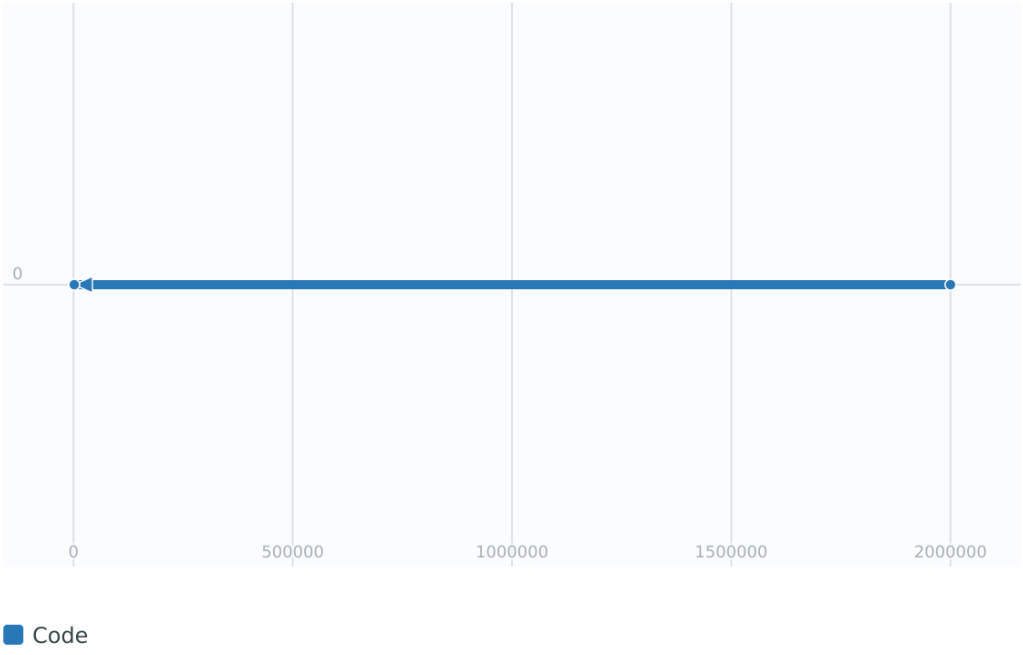
Code

```
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

Output

```
 valid | reason | location
-------+--------+----------
 t     |        |
```

Figure



Voir aussi

ST_IsValid, ST_IsValidReason

7.6.3 ST_IsValidReason

ST_IsValidReason — Renvoie un texte indiquant si une géométrie est valide, ou la raison de son invalidité.

Synopsis

text **ST_IsValidReason**(geometry geomA);
text **ST_IsValidReason**(geometry geomA, integer flags);

Description

Renvoie un texte indiquant si une géométrie est valide ou, si elle est invalide, une raison pour laquelle elle l'est.

Utile en combinaison avec ST_IsValid pour générer un rapport détaillé des géométries invalides et des raisons.

Les `flags` autorisés sont documentés dans ST_IsValidDetail.

Effectué par le module GEOS.

Disponibilité : 1.4

Disponibilité : la version 2.0 prend des flags.

Exemples

This example uses an invalid bow-tie polygon.

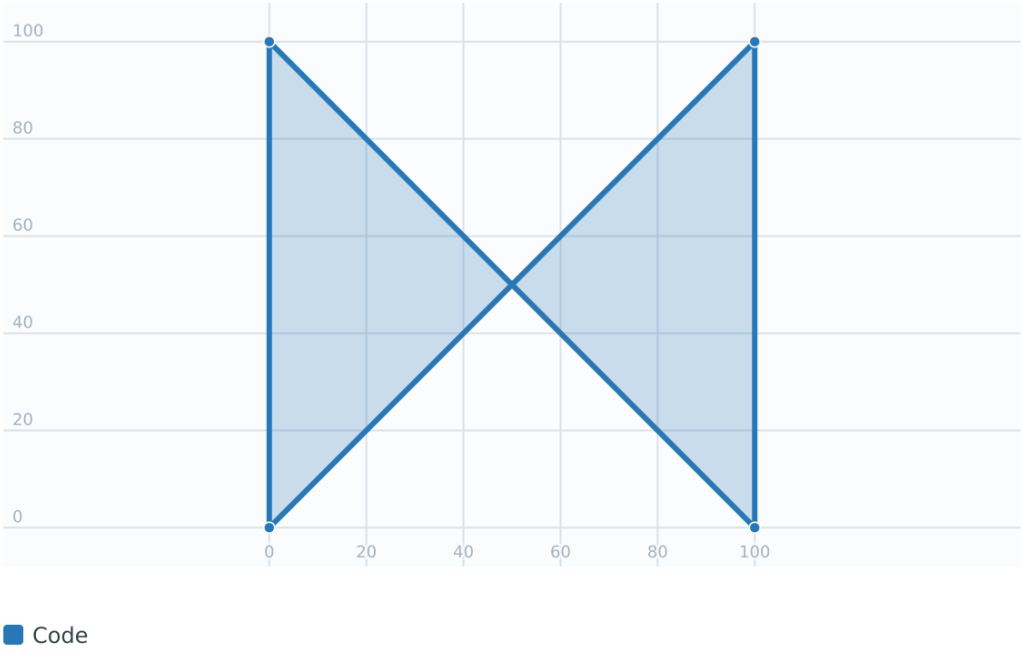
Code

```
SELECT ST_IsValidReason(
    'POLYGON ((100 200,100 100,200 200,
     200 100,100 200))'::geometry) as validity_info;
```

Output

```
Self-intersection[150 150]
```

Figure



This example returns the first three rejects from a successful quintuplet experiment.

Code

```
SELECT gid, ST_IsValidReason(geom) as validity_info
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
    FROM generate_series(-4,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,8) z1
    WHERE x1
> y1*0.5 AND z1 < x1*y1) As e
    INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_Point(x1*10,y1), z1)),y1*1, ↩
     z1*2) As line
    FROM generate_series(-3,6) x1
    CROSS JOIN generate_series(2,5) y1
    CROSS JOIN generate_series(1,10) z1
    WHERE x1
> y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff)
> 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(geom) = false
ORDER BY gid
LIMIT 3;
```

Output

```
 gid  |      validity_info
------+-------------------------
 5330 | Self-intersection [32 5]
 5340 | Self-intersection [42 5]
 5350 | Self-intersection [52 5]
```

Simple example.

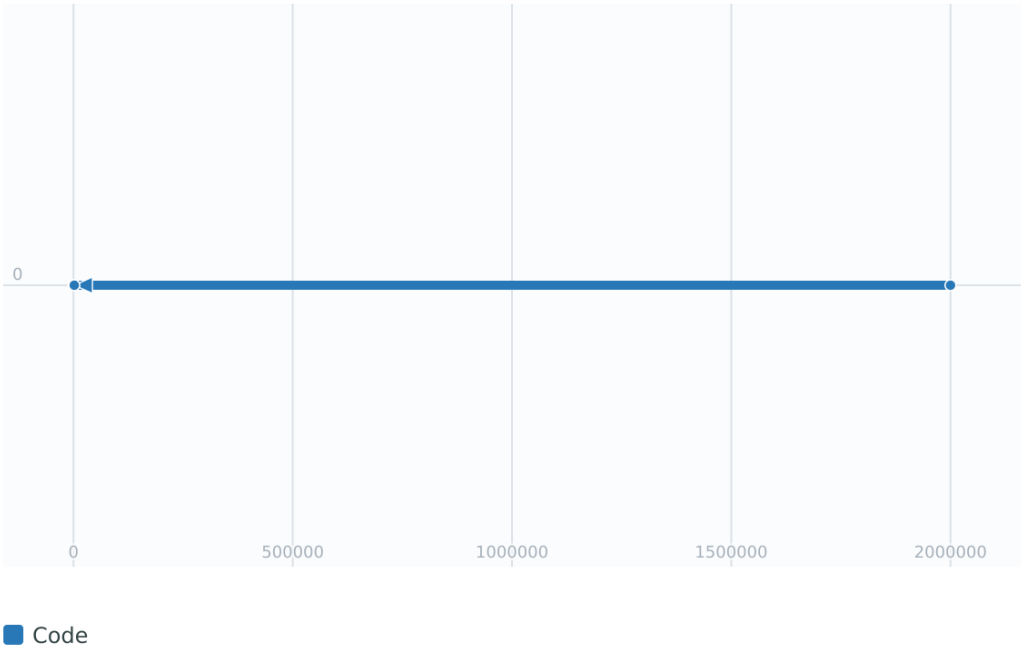
Code

```
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

Output

```
 st_isvalidreason
------------------
 Valid Geometry
```

Figure



Voir aussi

ST_IsValid, ST_Summary

7.6.4 ST_MakeValid

ST_MakeValid — Tente de rendre valide une géométrie invalide sans perdre de sommets.

Synopsis

geometry **ST_MakeValid**(geometry input);
geometry **ST_MakeValid**(geometry input, text params);

Description

La fonction tente de créer une représentation valide d'une géométrie invalide donnée sans perdre aucun des sommets d'entrée. Les géométries valides sont retournées inchangées.

Les types de données pris en charge sont : POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS et GEOMETRYCOLLECTIONS contenant n'importe quel mélange de ces éléments.

En cas de diminution de dimension (total ou partiel), la géométrie de sortie peut être une collection de géométries de dimension inférieure à égale, ou une géométrie de dimension inférieure.

Les polygones simples peuvent devenir des multigéométries en cas d'auto-intersections.

L'argument `params` peut être utilisé pour fournir une chaîne d'options permettant de sélectionner la méthode à utiliser pour construire une géométrie valide. La chaîne d'options est au format "method=linework|structure keepcollapsed=true|false". Si aucun argument "params" n'est fourni, l'algorithme "linework" sera utilisé par défaut.

La clé "method" a deux valeurs.

— "linework" is the default algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework. The requirement that no vertices are lost can generate complex collections as outputs.
— "structure" is an algorithm that distinguishes between interior and exterior rings, building new geometry by unioning exterior rings, and then differencing all interior rings. The results tend to be more intuitive, where collapsed lines are discarded in the output.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false".

— Keep collapsed of "false" is the default, and means that geometry components that collapse to a lower dimensionality, for example a one-point linestring will be dropped.
— Keep collapsed of "true" means geometry components that collapse to a lower dimensionality will be retained.

Effectué par le module GEOS.

Disponibilité : 2.0.0

Amélioré : 2.0.1, améliorations de la vitesse

Amélioration : 2.1.0, ajout du support pour GEOMETRYCOLLECTION et MULTIPOINT.

Amélioration : 3.1.0, suppression des coordonnées avec des valeurs NaN.

Amélioration : 3.2.0, ajout d'options d'algorithme, 'linework' et 'structure' qui nécessite GEOS >= 3.10.0.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

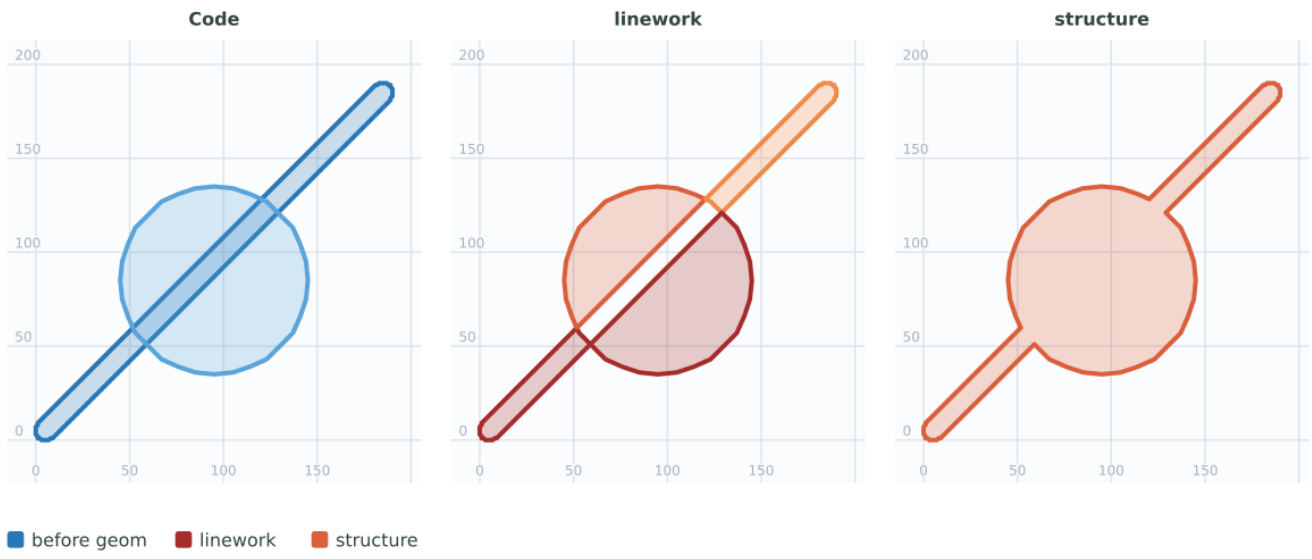
Exemples

Make valid a MultiPolygon of two overlapping polygons using the linework and structure methods.

Code

```
SELECT ST_MakeValid(before_geom) AS linework,
  ST_MakeValid(before_geom, 'method=structure') AS structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194,194 193,195 192,195 191,
195 190,195 189,195 188,194 187,194 186,14 6,13 6,12 5,11 5,
10 5,9 5,8 5,7 6,6 6,6 7,5 8,5 9,5 10,5 11,5 12,6 13,6 14,186 194)),
((150 90,149 80,146 71,142 62,135 55,128 48,119 44,110 41,100 40,
90 41,81 44,72 48,65 55,58 62,54 71,51 80,50 90,51 100,
54 109,58 118,65 125,72 132,81 136,90 139,100 140,110 139,
119 136,128 132,135 125,142 118,146 109,149 100,150 90)))'::geometry AS before_geom) AS f;
```

Figure

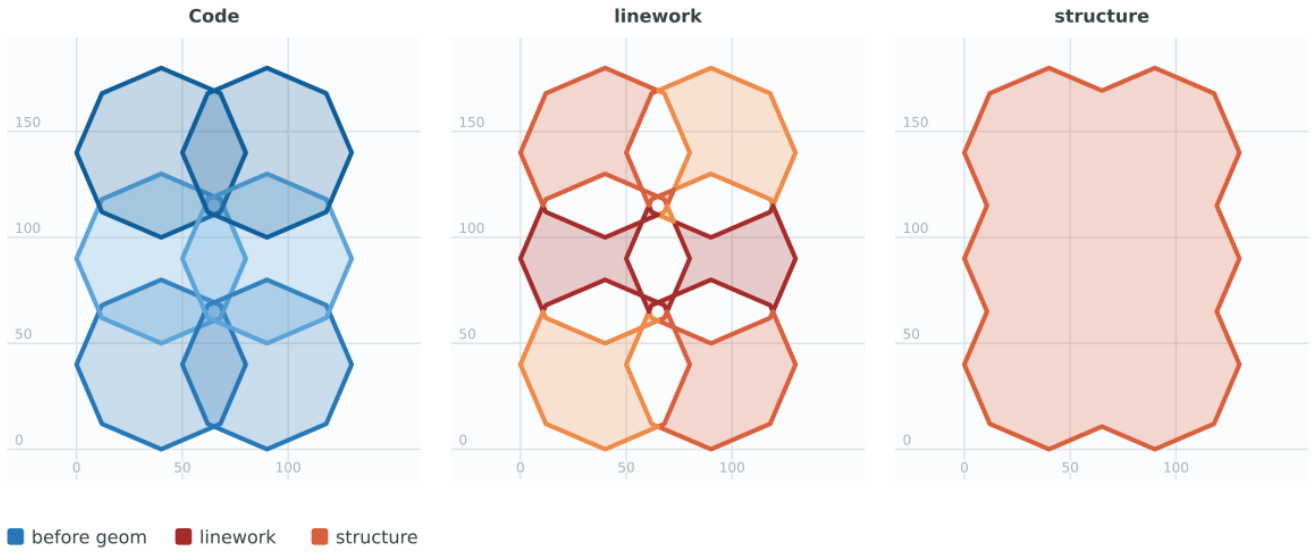


Make valid a MultiPolygon of six overlapping polygons using the linework and structure methods.

Code

```
SELECT ST_MakeValid(before_geom) AS linework,
  ST_MakeValid(before_geom, 'method=structure') AS structure
FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 50)),
      ((91 100,79 72,51 60,23 72,11 100,23 128,51 140,79 128,91 100)),
      ((91 150,79 122,51 110,23 122,11 150,23 178,51 190,79 178,91 150)),
      ((141 50,129 22,101 10,73 22,61 50,73 78,101 90,129 78,141 50)),
      ((141 100,129 72,101 60,73 72,61 100,73 128,101 140,129 128,141 100)),
      ((141 150,129 122,101 110,73 122,61 150,73 178,101 190,129 178,141 150)))'::geometry ↩
    AS before_geom) AS c;
```

Figure



Collapsed Geometry Examples

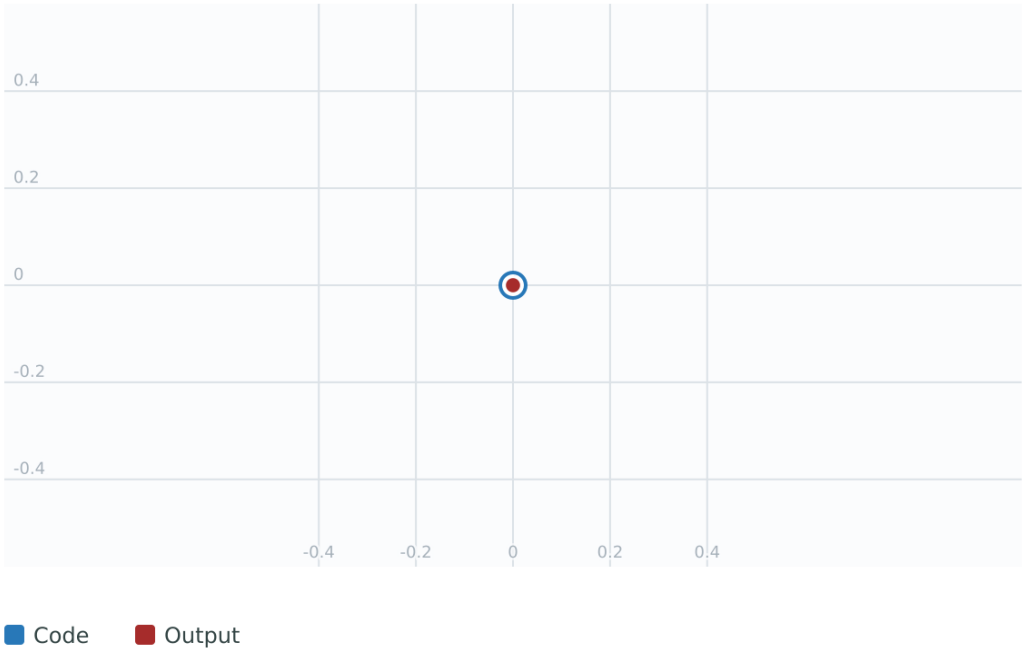
Code

```
SELECT ST_MakeValid(
'LINESTRING(0 0,0 0)',
'method=structure keepcollapsed=true'
);
```

Output

```
POINT(0 0)
```

Figure



Code

```
SELECT ST_MakeValid(
'LINESTRING(0 0,0 0)',
'method=structure keepcollapsed=false'
);
```

Output

```
LINESTRING EMPTY
```

Voir aussi

ST_IsValid, ST_Collect, ST_CollectionExtract

7.7 Fonctions des systèmes de référence spatiale

7.7.1 ST_InverseTransformPipeline

ST_InverseTransformPipeline — Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent en utilisant l'inverse d'un pipeline de transformation de coordonnées défini.

Synopsis

geometry **ST_InverseTransformPipeline**(geometry geom, text pipeline, integer to_srid);

Description

Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent à l'aide d'un pipeline de transformation de coordonnées défini pour aller dans la direction inverse.

Voir ST_TransformPipeline pour plus de détails sur l'écriture d'un pipeline de transformation.

Disponibilité : 3.4.0

Le SRID de la géométrie en entrée est ignoré et le SRID de la géométrie de sortie sera mis à zéro à moins qu'une valeur ne soit fournie via le paramètre facultatif `to_srid`. Lors de l'utilisation de ST_TransformPipeline, le pipeline est exécuté dans le sens direct. En utilisant `ST_InverseTransformPipeline()`, le pipeline est exécuté dans le sens inverse.

Les transformations utilisant des pipelines sont une version spéciale de ST_Transform. Dans la plupart des cas, `ST_Transform` choisira les opérations correctes pour convertir entre les systèmes de coordonnées, et devrait être préféré.

Exemples

Changer WGS 84 long lat en UTM 31N en utilisant la conversion EPSG:16031

This example transforms in the inverse direction.

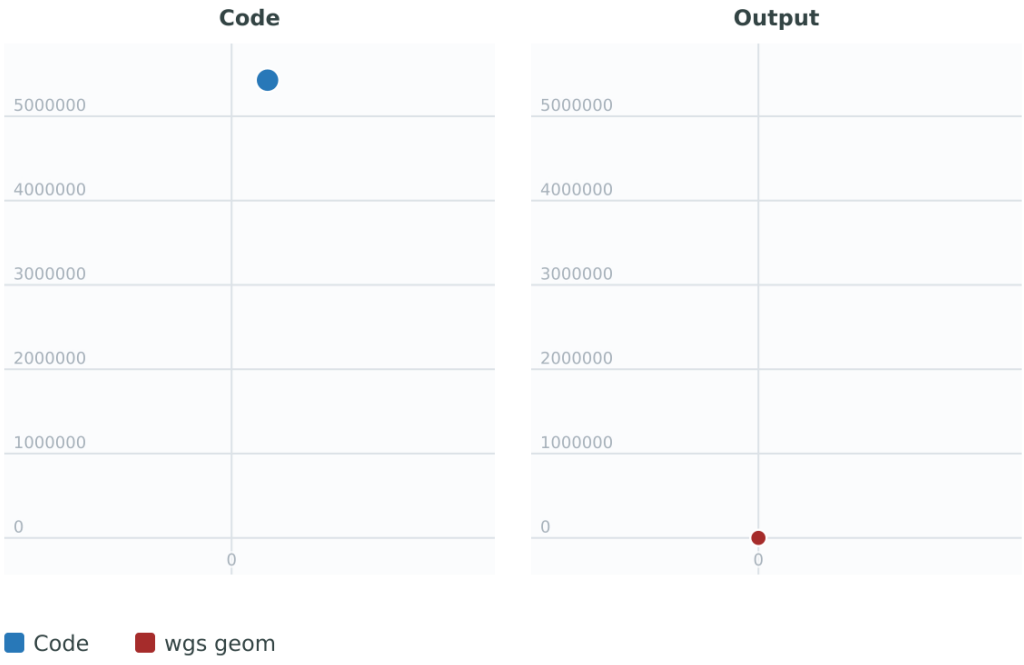
Code

```
SELECT ST_InverseTransformPipeline(
        'POINT(426857.9877165967 5427937.523342293)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::16031'
    ) AS wgs_geom;
```

Output

```
POINT(2 48.99999999999999)
```

Figure



Exemple GDA2020.

This example uses ST_Transform with automatic selection of a conversion pipeline.

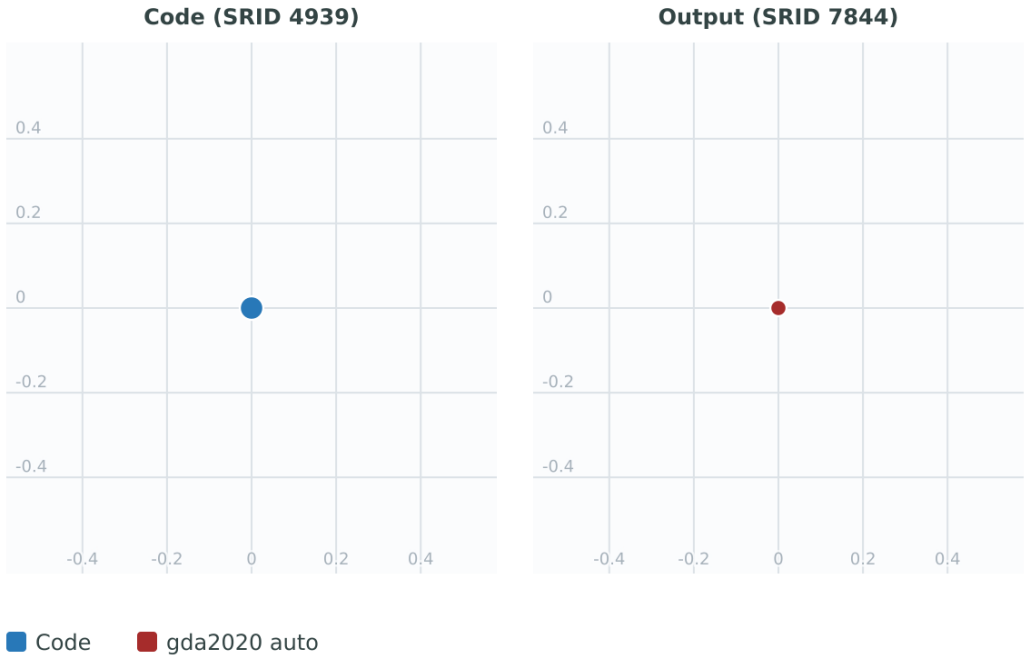
Code

```
SELECT ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844) AS gda2020_auto;
```

Output

```
POINT(143.00000635638918 -36.999986706128176)
```

Figure



Voir aussi

ST_Transform, ST_TransformPipeline

7.7.2 ST_SetSRID

ST_SetSRID — Définir le SRID d'une géométrie.

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

Description

Définit le SRID d'une géométrie à une valeur entière particulière. Utile pour construire des boîtes de délimitation pour les requêtes.



Note
Cette fonction ne transforme en aucune façon les coordonnées de la géométrie - elle définit simplement les métadonnées définissant le système de référence spatiale dans lequel la géométrie est supposée se trouver. Utilisez ST_Transform si vous souhaitez transformer la géométrie dans une nouvelle projection.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Mark a point as WGS 84 longitude/latitude.

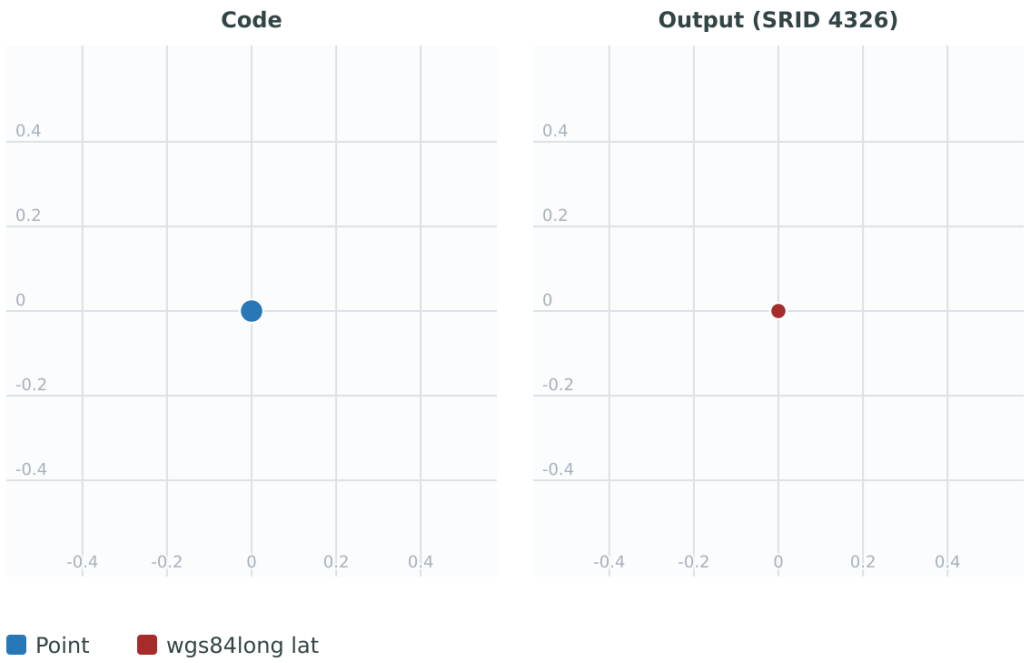
Code

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611), 4326) AS wgs84long_lat;
```

Output

```
SRID=4326;POINT(-123.365556 48.428611)
```

Figure



Mark a point as WGS 84 longitude/latitude and then transform it to Web Mercator.

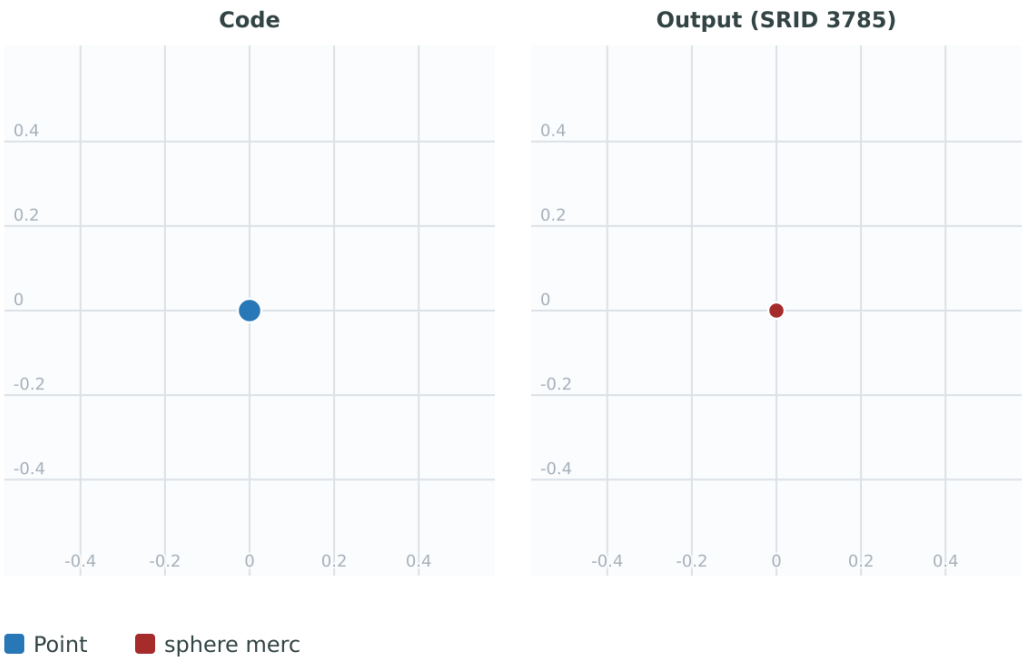
Code

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611), 4326), 3785) AS sphere_merc ↩
    ;
```

Output

```
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

Figure



Voir aussi

Section 4.5, ST_SRID, ST_Transform, UpdateGeometrySRID

7.7.3 ST_SRID

ST_SRID — Renvoie l'identifiant de référence spatiale d'une géométrie.

Synopsis

integer **ST_SRID**(geometry g1);

Description

Renvoie l'identifiant de référence spatiale pour la ST_Geometry tel que défini dans la table spatial_ref_sys. Section 4.5

> **Note**
> La table spatial_ref_sys est une table qui répertorie tous les systèmes de référence spatiale connus de PostGIS et qui est utilisée pour les transformations d'un système de référence spatiale à un autre. Il est donc important de vérifier que vous disposez du bon identifiant de système de référence spatiale si vous envisagez de transformer vos géométries.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.5

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)', 4326));
```

Output

```
st_srid
---------
    4326
(1 row)
```

Voir aussi

Section 4.5, ST_SetSRID, ST_Transform, ST_SRID, ST_SRID

7.7.4 ST_Transform

ST_Transform — Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent.

Synopsis

geometry **ST_Transform**(geometry g1, integer srid);
geometry **ST_Transform**(geometry geom, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, integer to_srid);

Description

Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatiale différent. La référence spatiale de destination `to_srid` peut être identifiée par un paramètre entier SRID valide (c'est-à-dire qu'elle doit exister dans la table `spatial_ref_sys`). Il est également possible d'utiliser une référence spatiale définie comme une chaîne PROJ.4 pour `to_proj` et/ou `from_proj`, mais ces méthodes ne sont pas optimisées. Si le système de référence spatiale de destination est exprimé avec une chaîne PROJ.4 au lieu d'un SRID, le SRID de la géométrie de sortie sera mis à zéro. A l'exception des fonctions avec `from_proj`, les géométries d'entrée doivent avoir un SRID défini.

ST_Transform est souvent confondu avec ST_SetSRID. ST_Transform modifie en fait les coordonnées d'une géométrie d'un système de référence spatiale à un autre, tandis que ST_SetSRID() modifie simplement l'identifiant SRID de la géométrie.

ST_Transform sélectionne automatiquement un pipeline de conversion approprié en fonction des systèmes de référence spatiale source et cible. Pour utiliser une méthode de conversion spécifique, utilisez ST_TransformPipeline.



Note
PostGIS doit être compilé avec le support PROJ. Utilisez PostGIS_Full_Version pour confirmer que vous avez compilé le support PROJ.



Note
Si l'on utilise plus d'une transformation, il est utile de disposer d'un index fonctionnel sur les transformations les plus couramment utilisées afin de tirer parti de l'utilisation de l'index.



Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Amélioration : la version 2.3.0 a introduit la prise en charge du texte PROJ.4 direct.

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.6

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Changer la géométrie des pieds US du plan de l'État du Massachusetts en WGS 84 long lat

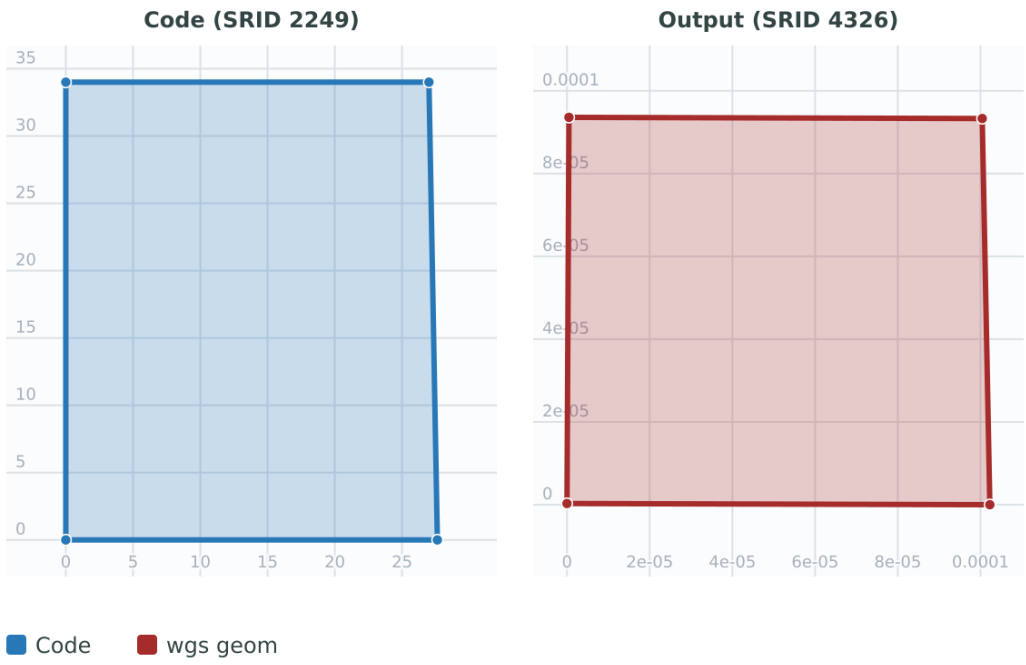
Code

```
SELECT ST_Transform(
  'SRID=2249;POLYGON((743238 2967416,743238 2967450,
    743265 2967450,743265.625 2967416,743238 2967416))'::geometry,
  4326) AS wgs_geom;
```

Output

```
SRID=4326;POLYGON((-71.1776848522251 42.3902896512903,-71.17768437663261 42.390382947801015,-71.17758443054647 ↩
    42.390382667791854,-71.17758259272304 42.390289364798825,-71.1776848522251 42.3902896512903))
```

Figure



3D Circular String example.

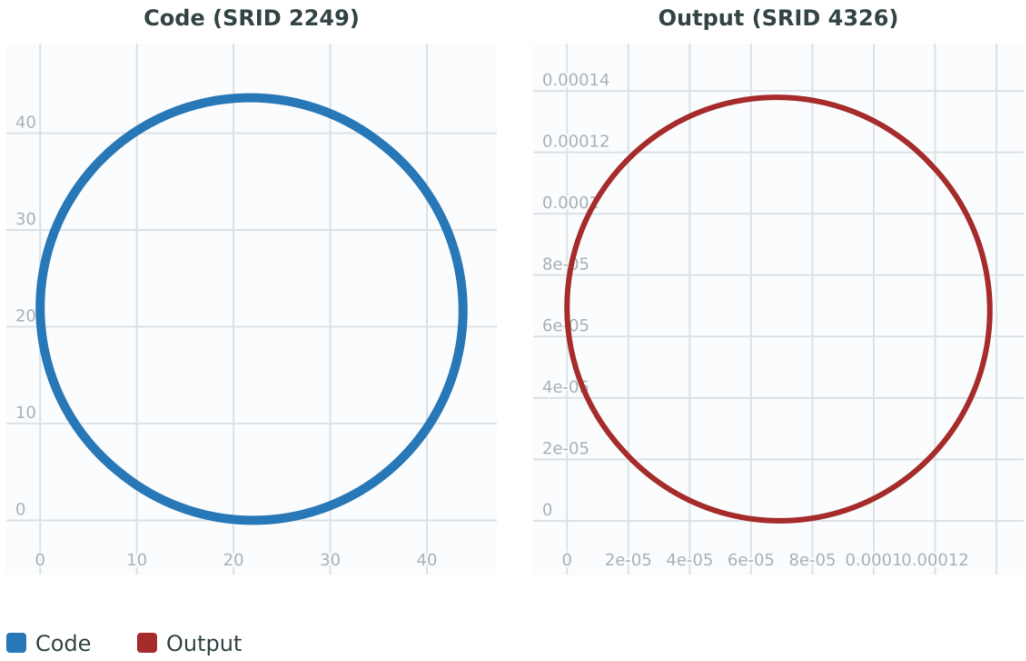
Code

```
SELECT ST_Transform('SRID=2249;CIRCULARSTRING(743238 2967416 1,743238 2967450 2,743265 ↩
    2967450 3,743265.625 2967416 3,743238 2967416 4)'::geometry,
    4326);
```

Output

```
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512903 1,-71.17768437663261 42.390382947801015 2,
-71.17758443054647 42.390382667791854 3,
-71.17758259272304 42.390289364798825 3,-71.1776848522251 42.3902896512903 4)
```

Figure



Exemple de création d'un index fonctionnel partiel. Pour les tables dont vous n'êtes pas sûr que toutes les géométries seront renseignées, il est préférable d'utiliser un index partiel qui laisse de côté les géométries nulles, ce qui permet à la fois de conserver de l'espace et de rendre votre index plus petit et plus efficace.

Code

```
CREATE INDEX idx_geom_26986_parcels
  ON parcels
  USING gist
  (ST_Transform(geom, 26986))
  WHERE geom IS NOT NULL;
```

Exemples d'utilisation du texte PROJ.4 pour transformer avec des références spatiales personnalisées.

Find intersection of two polygons near the North pole, using a custom Gnomic projection. See http://boundlessgeo.com/2012/02/flattening
the-peel/.

Code

```
WITH data AS (
  SELECT
    ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
    ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
```

```
    '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
)
SELECT ST_Transform(ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
  gnom, 4326)
FROM data;
```

Output

```
POLYGON((-140.9999840290591 73.42768865981984,-141 68,-170 68,-170.0000026915361 74.05492179637055,-140.9999840290591 ↩
    73.42768865981984))
```

Figure



Configurer le comportement de transformation

Parfois, la transformation de coordonnées impliquant un décalage de grille peut échouer, par exemple si PROJ.4 n'a pas été construit avec des fichiers de décalage de grille ou si la coordonnée ne se trouve pas dans la plage pour laquelle le décalage de grille est défini. Par défaut, PostGIS génère une erreur si un fichier de décalage de grille n'est pas présent, mais ce comportement peut être configuré pour chaque SRID, soit en testant différentes valeurs `to_proj` de texte PROJ.4, soit en modifiant la valeur `proj4text` dans la table `spatial_ref_sys`.

Par exemple, le paramètre proj4text +datum=NAD87 est une forme abrégée du paramètre +nadgrids suivant :

Code

```
+nadgrids=@conus, @alaska, @ntv2_0.gsb, @ntv1_can.dat
```

Le préfixe @ signifie qu'aucune erreur n'est signalée si les fichiers ne sont pas présents, mais si la fin de la liste est atteinte sans qu'aucun fichier n'ait été approprié (c'est-à-dire trouvé et se chevauchant), une erreur est émise.

Si, à l'inverse, vous voulez vous assurer qu'au moins les fichiers standard sont présents, mais que si tous les fichiers ont été analysés sans résultat, une transformation nulle est appliquée, vous pouvez utiliser :

Code

```
+nadgrids=@conus, @alaska, @ntv2_0.gsb, @ntv1_can.dat, null
```

Le fichier de décalage de grille nul est un fichier de décalage de grille valide couvrant le monde entier et n'appliquant aucun décalage. Ainsi, pour un exemple complet, si vous vouliez modifier PostGIS de manière à ce que les transformations du SRID 4267 qui ne se situent pas dans la plage correcte n'entraînent pas d'ERREUR, vous utiliseriez ce qui suit :

Code

```
UPDATE spatial_ref_sys SET proj4text = '+proj=longlat +ellps=clrk66 +nadgrids=@conus,@alaska ↩
    ,@ntv2_0.gsb,@ntv1_can.dat,null +no_defs' WHERE srid = 4267;
```

Voir aussi

Section 4.5, ST_SetSRID, ST_SRID, UpdateGeometrySRID, ST_TransformPipeline

7.7.5 ST_TransformPipeline

ST_TransformPipeline — Retourner une nouvelle géométrie avec des coordonnées transformées dans un système de référence spatial différent à l'aide d'un pipeline de transformation de coordonnées défini.

Synopsis

geometry **ST_TransformPipeline**(geometry g1, text pipeline, integer to_srid);

Description

Retourner une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent à l'aide d'un pipeline de transformation de coordonnées défini.

Les pipelines de transformation sont définis à l'aide de l'un des formats de texte suivants :

— `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Notez qu'une simple chaîne de caractère `EPSG:CODE` n'identifie pas de manière unique une opération de coordonnées : le même code EPSG peut être utilisé pour une définition CRS.

— Une chaîne de caractères de pipeline PROJ de la forme : `+proj=pipeline ...`. La normalisation automatique des axes ne sera pas appliquée et, si nécessaire, l'appelant devra ajouter une étape de pipeline supplémentaire ou supprimer les étapes `axisswap`.

— Opérations concaténées de la forme : `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,`

Disponibilité : 3.4.0

Le SRID de la géométrie d'entrée est ignoré, et le SRID de la géométrie de sortie sera mis à zéro à moins qu'une valeur ne soit fournie via le paramètre optionnel `to_srid`. Lors de l'utilisation de `ST_TransformPipeline()`, le pipeline est exécuté dans la direction avant. En utilisant ST_InverseTransformPipeline le pipeline est exécuté dans le sens inverse.

Les transformations utilisant des pipelines sont une version spéciale de ST_Transform. Dans la plupart des cas, `ST_Transform` choisira les opérations correctes pour convertir entre les systèmes de coordonnées, et devrait être préféré.

Exemples

Changer WGS 84 long lat en UTM 31N en utilisant la conversion EPSG:16031

This example transforms in the forward direction.

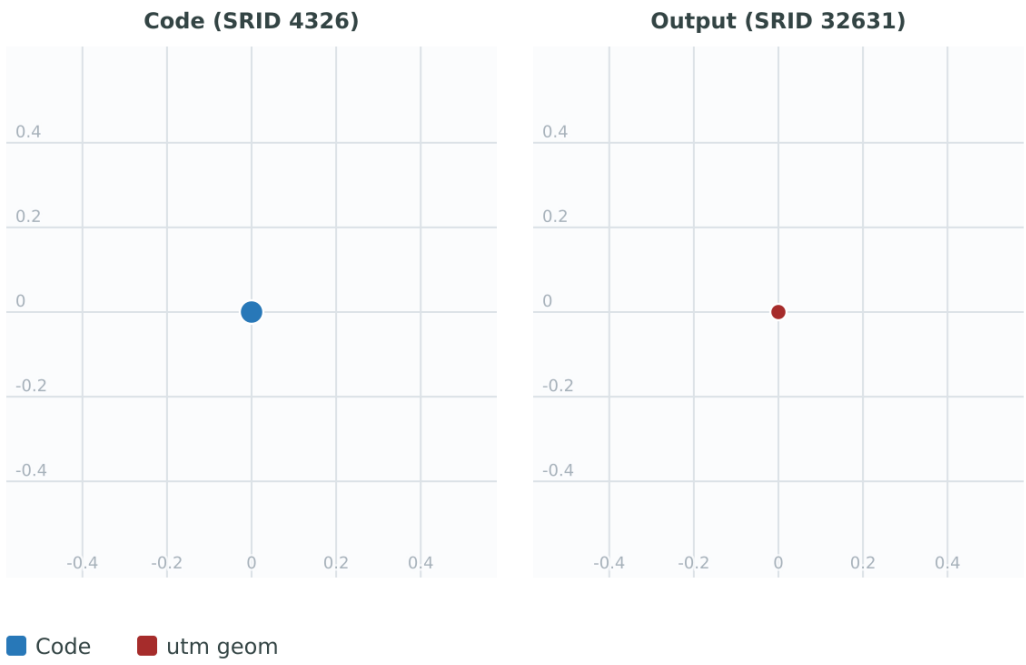
Code

```
SELECT
    ST_SetSRID(ST_TransformPipeline(
        'SRID=4326;POINT(2 49)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::16031'
    ), 32631) AS utm_geom;
```

Output

```
SRID=32631;POINT(426857.9877165967 5427937.523342293)
```

Figure



This example transforms in the inverse direction.

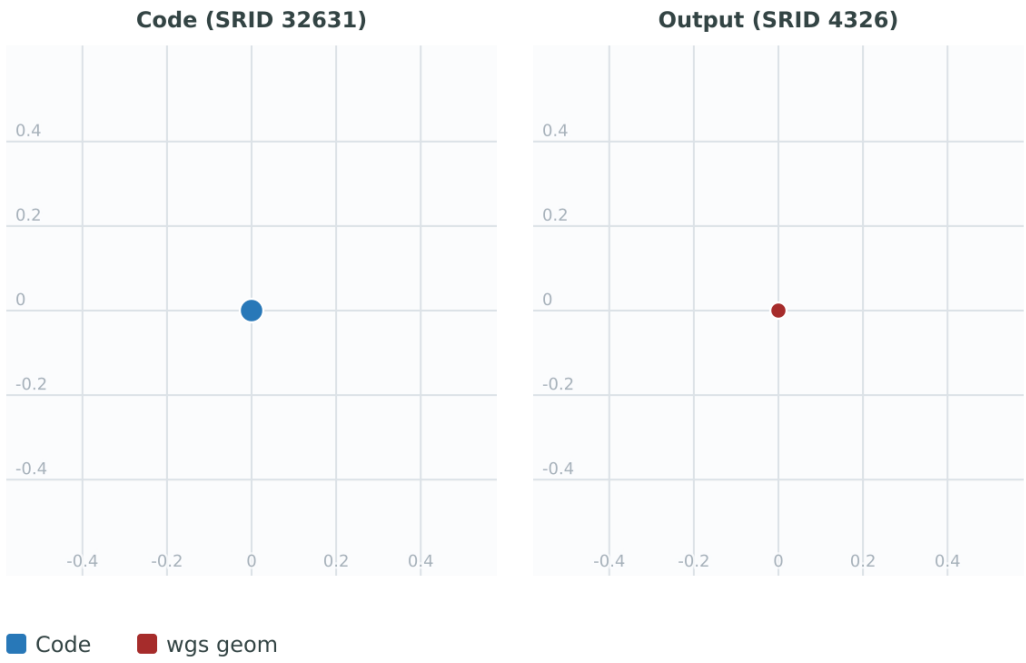
Code

```
SELECT
    ST_SetSRID(ST_InverseTransformPipeline(
        'SRID=32631;POINT(426857.9877165967 5427937.523342293)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::16031'
    ), 4326) AS wgs_geom;
```

Output

```
SRID=4326;POINT(2 48.99999999999999)
```

Figure



Exemple GDA2020.

This example uses ST_Transform with automatic selection of a conversion pipeline.

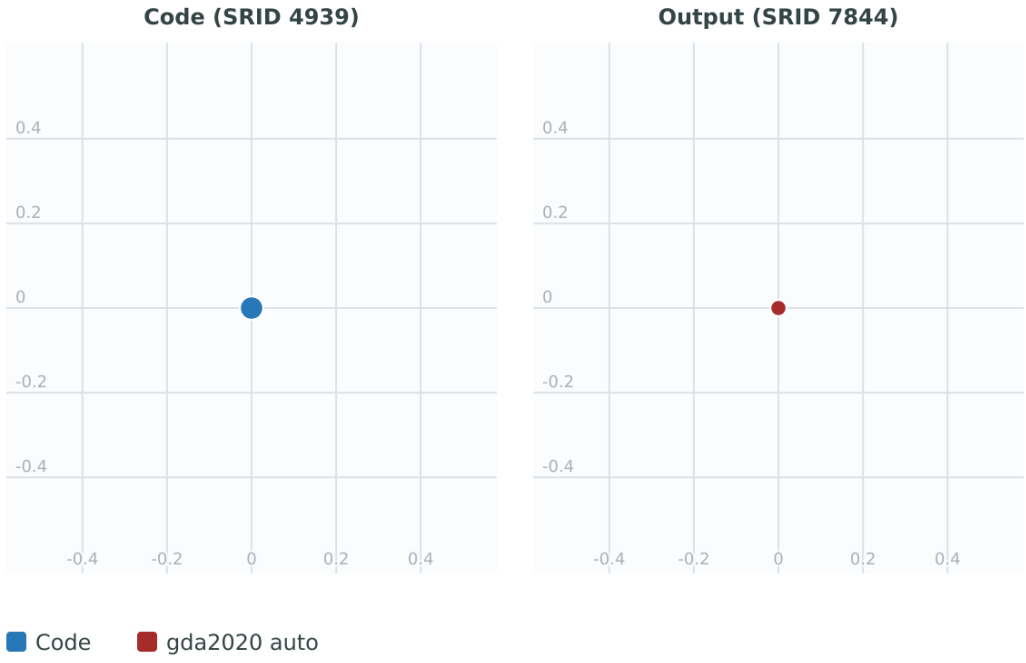
Code

```
SELECT ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844) AS gda2020_auto;
```

Output

```
POINT(143.00000635638918 -36.999986706128176)
```

Figure



Using a defined conversion (EPSG:8447)

Code

```
SELECT
    ST_TransformPipeline(
        'SRID=4939;POINT(143.0 -37.0)'::geometry,
        'urn:ogc:def:coordinateOperation:EPSG::8447'
    ) AS gda2020_code;
```

Output

```
POINT(143.0000063280214 -36.999986718287545)
```

This example uses a PROJ pipeline definition matching EPSG:8447, as returned from projinfo. Any axisswap steps must be removed.

Code

```
SELECT
    ST_TransformPipeline(
        'SRID=4939;POINT(143.0 -37.0)'::geometry,
        '+proj=pipeline
   +step +proj=unitconvert +xy_in=deg +xy_out=rad
   +step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
   +step +proj=unitconvert +xy_in=rad +xy_out=deg'
    ) AS gda2020_pipeline;
```

Output

```
POINT(143.0000063280214 -36.999986718287545)
```

Voir aussi

ST_Transform, ST_InverseTransformPipeline

7.7.6 postgis_srs_codes

postgis_srs_codes — Renvoie la liste des codes SRS associés à l'autorité donnée.

Synopsis

setof text **postgis_srs_codes**(text auth_name);

Description

Renvoie un ensemble de tous les `auth_srid` pour le `auth_name` donné.

Disponibilité : 3.4.0

Proj version 6+

Exemples

Indiquez les dix premiers codes associés à l'autorité EPSG.

Code

```
SELECT * FROM postgis_srs_codes('EPSG') ORDER BY 1 LIMIT 10;
```

Output

```
10156
10157
10158
10160
10162
10163
10164
10165
10166
10167
```

Voir aussi

postgis_srs, postgis_srs_all, postgis_srs_search

7.7.7 postgis_srs

postgis_srs — Renvoyer une fiche de métadonnées pour l'autorité et le srid demandés.

Synopsis

setof record **postgis_srs**(text auth_name, text auth_srid);

Description

Renvoie un enregistrement de métadonnées pour le `auth_srid` demandé pour le `auth_name` donné. L'enregistrement contiendra les éléments suivants : `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, et les angles de la zone d'utilisation, `point_sw` et `point_ne`.

Disponibilité : 3.4.0

Proj version 6+

Exemples

Obtenir les métadonnées pour EPSG:3005.

Code

```
SELECT auth_name, auth_srid, srname,point_sw AS point_sw,point_ne AS point_ne
FROM postgis_srs('EPSG', '3005');
```

Output

```
-[ RECORD 1 ]-------------------
auth_name | EPSG
auth_srid | 3005
srname    | NAD83 / BC Albers
point_sw  | POINT(-139.04 48.25)
point_ne  | POINT(-114.08 60.01)
```

Voir aussi

postgis_srs_codes, postgis_srs_all, postgis_srs_search

7.7.8 postgis_srs_all

postgis_srs_all — Renvoie des enregistrements de métadonnées pour chaque système de référence spatiale dans la base de données Proj sous-jacente.

Synopsis

setof record **postgis_srs_all**(void);

Description

Renvoie un ensemble d'enregistrements de métadonnées dans la base de données Proj sous-jacente. Les enregistrements auront le `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, et les angles de la zone d'utilisation, `point_sw` et `point_ne`.

Disponibilité : 3.4.0

Proj version 6+

Exemples

Obtenir les 10 premiers enregistrements de métadonnées de la base de données Proj.

Code

```
SELECT auth_name, auth_srid, srname FROM postgis_srs_all() ORDER BY auth_name, auth_srid ↩
    LIMIT 10;
```

Output

```
auth_name | auth_srid |                                srname
-----------+-----------+-------------------------------------------------------------------------
 EPSG      | 10156     | ETRS89 + MSL UK & Ireland VORF08 depth
 EPSG      | 10157     | ETRS89 + CD UK & Ireland VORF08 depth
 EPSG      | 10158     | S34J-IRF
 EPSG      | 10160     | S34J reconstruction east-orientated
 EPSG      | 10162     | JGD2011 / Japan Plane Rectangular CS I + JGD2011 (vertical) height
 EPSG      | 10163     | JGD2011 / Japan Plane Rectangular CS II + JGD2011 (vertical) height
 EPSG      | 10164     | JGD2011 / Japan Plane Rectangular CS III + JGD2011 (vertical) height
 EPSG      | 10165     | JGD2011 / Japan Plane Rectangular CS IV + JGD2011 (vertical) height
 EPSG      | 10166     | JGD2011 / Japan Plane Rectangular CS V + JGD2011 (vertical) height
 EPSG      | 10167     | JGD2011 / Japan Plane Rectangular CS VI + JGD2011 (vertical) height
```

Voir aussi

postgis_srs_codes, postgis_srs, postgis_srs_search

7.7.9 postgis_srs_search

postgis_srs_search — Renvoyer les enregistrements de métadonnées pour les systèmes de coordonnées projetées dont les zones d'utilisation contiennent entièrement le paramètre bounds.

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

Description

Renvoie un ensemble d'enregistrements de métadonnées pour les systèmes de coordonnées projetées dont les zones d'utilisation contiennent entièrement le paramètre bounds. Chaque enregistrement contiendra le `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, et les angles de la zone d'utilisation, `point_sw` et `point_ne`.

La recherche ne porte que sur les systèmes de coordonnées projetées et vise à permettre aux utilisateurs d'explorer les systèmes possibles en fonction de l'étendue de leurs données.

Disponibilité : 3.4.0

Proj version 6+

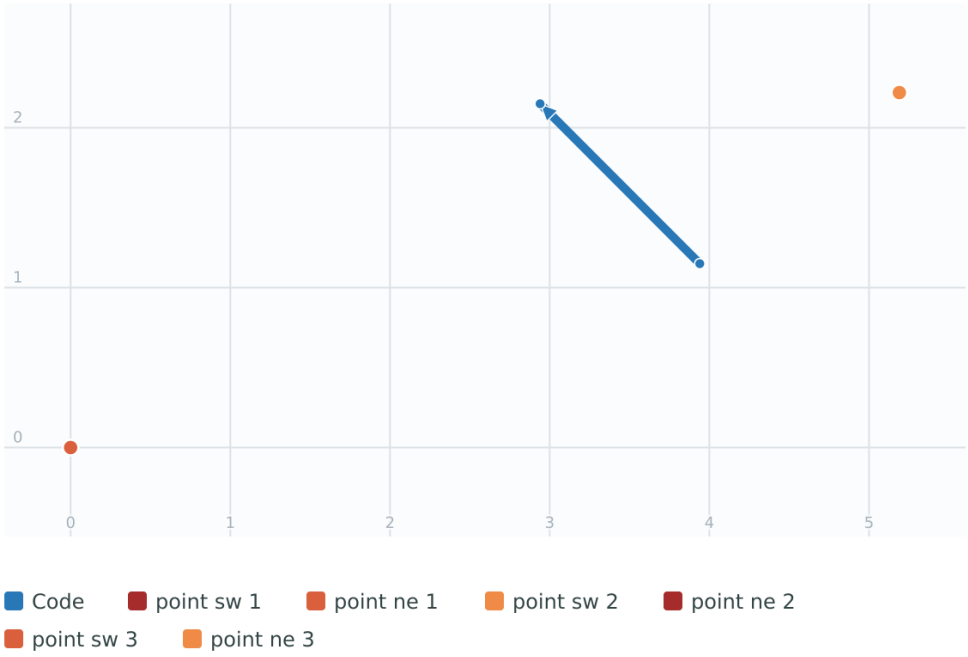
Exemples

Recherche de systèmes de coordonnées projetées en Louisiane.

Code

```
SELECT auth_name, auth_srid, srname,point_sw AS point_sw,point_ne AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30,-91 31)')
WHERE auth_srid IN ('2801', '3452', '3457')
ORDER BY auth_name, auth_srid::integer
```

Figure



Examinez une table pour connaître l'étendue maximale et trouvez les systèmes de coordonnées projetées qui pourraient convenir.

Code

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
SELECT auth_name, auth_srid, srname,
  point_sw AS point_sw,
  point_ne AS point_ne
FROM ext
```

```
CROSS JOIN postgis_srs_search(ST_SetSRID(ext.geom, ext.srid))
LIMIT 3;
```

Voir aussi

postgis_srs_codes, postgis_srs_all, postgis_srs

7.8 Import de géométrie

7.8.1 Well-Known Text (WKT)

7.8.1.1 ST_BdPolyFromText

ST_BdPolyFromText — Construit un Polygon à partir d'une collection de lignes fermées, exprimées sous forme de MultiLineString en représentation Well-Known text.

Synopsis

geometry **ST_BdPolyFromText**(text WKT, integer srid);

Description

Construit un Polygon à partir d'une collection de lignes fermées, exprimées sous forme de MultiLineString en représentation Well-Known text.



Note
Renvoie une erreur si le WKT n'est pas une MULTILINESTRING. Renvoie une erreur si le résultat est un MULTIPOLYGON. Utiliser ST_BdMPolyFromText dans ce cas, ou voir ST_BuildArea() pour une approche basée sur une fonction spécifique.


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Effectué par le module GEOS.

Disponibilité : 1.1.0

Voir aussi

ST_BuildArea, ST_BdMPolyFromText

7.8.1.2 ST_BdMPolyFromText

ST_BdMPolyFromText — Construit un MultiPolygon à partir d'une collection de lignes fermées, exprimées sous forme de MultiLineString en représentation Well-Known text.

Synopsis

geometry **ST_BdMPolyFromText**(text WKT, integer srid);

Description

Construit un Polygon à partir d'une collection de lignes fermées, de polygones ou de MultiLineStrings exprimés en représentation Well-Known text.



Note
Renvoie une erreur si le WKT n'est pas une MULTILINESTRING. Force le type de retour en MULTIPOLYGON même si le résultat est en fait composé d'un seul POLYGON. Utiliser ST_BdPolyFromText si l'on est sûr que le résultat produit des Polygon, ou voir la fonction spécifique PostGIS ST_BuildArea().

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Effectué par le module GEOS.

Disponibilité : 1.1.0

Voir aussi

ST_BuildArea, ST_BdPolyFromText

7.8.1.3 ST_GeogFromText

ST_GeogFromText — Retourne un objet de type geography à partir de sa représentation Well-Know Text (WKT ou EWKT).

Synopsis

geography **ST_GeogFromText**(text EWKT);

Description

Retourne un objet de type geography à partir de sa représentation Well-Know Text (WKT ou EWKT). Le SRID 4326 est pris par défaut. Ceci est un alias pour ST_GeographyFromText. Les coordonnées des points sont exprimées en longitude latitude.

Exemples

Add a WGS 84 geography point column to a table.

Code

```
ALTER TABLE sometable ADD COLUMN geog geography(POINT, 4326);
```

Populate the geography column from longitude and latitude columns.

Code

```
UPDATE sometable
SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')');
```

Specify a geography point using EPSG:4267, NAD27.

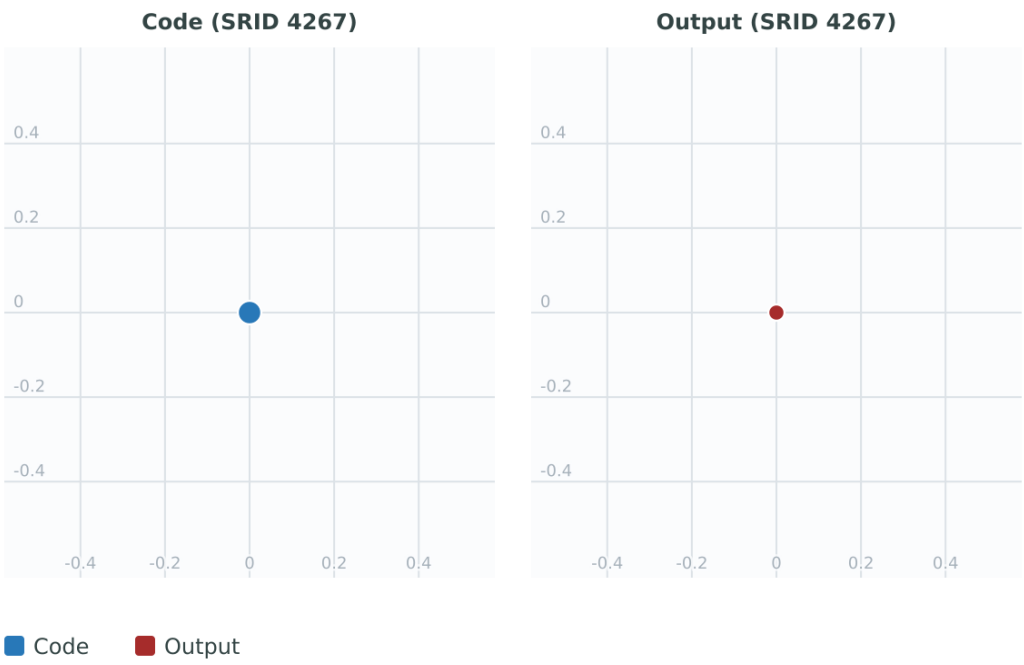
Code

```
SELECT ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)');
```

Output

```
SRID=4267;POINT(-77.0092 38.889588)
```

Figure



Voir aussi

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — Retourne un objet de type geography à partir de sa représentation Well-Know Text (WKT ou EWKT).

Synopsis

geography **ST_GeographyFromText**(text EWKT);

Description

Retourne un objet de type geography à partir de sa représentation Well-Know Text (WKT ou EWKT). SRID 4326 par défaut.

Voir aussi

ST_GeogFromText, ST_AsText

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — Crée une collection Geometry à partir de la collection WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.

Synopsis

geometry **ST_GeomCollFromText**(text WKT, integer srid);
geometry **ST_GeomCollFromText**(text WKT);

Description

Crée une géométrie de collection à partir de la représentation Well-Known-Text (WKT) avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.

OGC SPEC 3.2.6.2 - l'option SRID est issue des tests de conformité

Retourne null si le WKT n'est pas une GEOMETRYCOLLECTION



Note
Si vous êtes absolument sûrs que toutes les géométries WKT sont des collections, ne pas utiliser cette fonction. Elle est plus lente que ST_GeomFromText à cause d'une étape de validation supplémentaire.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2



Cette méthode implémente la spécification SQL/MM.

Exemples

Code

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2), LINESTRING(1 2,3 4))');
```

Voir aussi

ST_GeomFromText, ST_SRID

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — Retourne un objet ST_Geometry à partir de sa représentation textuelle étendue (Extended Well-Known Text representation, EWKT).

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

Description

Retourne un objet ST_Geometry à partir de sa représentation textuelle étendue OGC (Extended Well-Known Text representation, EWKT).



Note
Le format EWKT n'est pas une norme OGC, mais un format spécifique à PostGIS incluant l'identifiant du système de référence des coordonnées (SRID)

Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Create a LineString with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)') AS geom;
```

Output

```
geometry_type | srid
--------------+------
 ST_LineString | 4269
(1 row)
```

Create a MultiLineString with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932))') AS geom;
```

Output

```
geometry_type    | srid
-----------------+------
 ST_MultiLineString  | 4269
(1 row)
```

Create a Point with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)') AS geom;
```

Output

```
geometry_type | srid
--------------+------
 ST_Point      | 4269
(1 row)
```

Create a Polygon with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;POLYGON((-71.1776585052917 ↩
    42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↩
    42.3902909739571))') AS geom;
```

Output

```
geometry_type | srid
--------------+------
 ST_Polygon   | 4269
(1 row)
```

Create a MultiPolygon with SRID 4269.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('SRID=4269;MULTIPOLYGON(((-71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873 ↩
    42.315113108546)))') AS geom;
```

Output

```
geometry_type   | srid
----------------+------
 ST_MultiPolygon | 4269
(1 row)
```

Create a 3D CircularString without an SRID.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)') AS ↩
    geom;
```

Output

```
geometry_type    | srid
-----------------+------
 ST_CircularString  | 0
(1 row)
```

Create a PolyhedralSurface without an SRID.

Code

```
SELECT ST_GeometryType(geom) AS geometry_type, ST_SRID(geom) AS srid
FROM ST_GeomFromEWKT('POLYHEDRALSURFACE(
    ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
    ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
    ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
)') AS geom;
```

Output

```
geometry_type       | srid
--------------------+------
 ST_PolyhedralSurface     | 0
(1 row)
```

Voir aussi

ST_AsEWKT, ST_GeomFromText

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — Prend les données géographiques MARC21/XML en entrée et renvoie un objet géométrique PostGIS.

Synopsis

geometry **ST_GeomFromMARC21** (text marcxml);

Description

Cette fonction crée une géométrie PostGIS à partir d'un enregistrement MARC21/XML, qui peut contenir un `POINT` ou un `POLYGON`. En cas d'entrées de données géographiques multiples dans le même enregistrement MARC21/XML, un `MULTIPOINT` ou `MULTIPOLYGON` sera renvoyé. Si la notice contient des types de géométrie mixtes, une `GEOMETRYCOLLECTION` sera retournée. Il renvoie NULL si l'enregistrement MARC21/XML ne contient pas de données géographiques (datafield :034).

Prise en charge des versions LOC MARC21/XML :

— MARC21/XML 1.1

Disponibilité : 3.3.0, nécessite libxml2 2.6+



Note
Les données mathématiques cartographiques codées MARC21/XML ne fournissent actuellement aucun moyen de décrire le système de référence spatiale des coordonnées codées, de sorte que cette fonction retournera toujours une géométrie avec `SRID 0`.



Note

Les géométries `POLYGON` retournées seront toujours orientées dans le sens des aiguilles d'une montre.

Exemples

Conversion de données géographiques MARC21/XML contenant un seul `POINT` encodé en `hddd.dddddd`

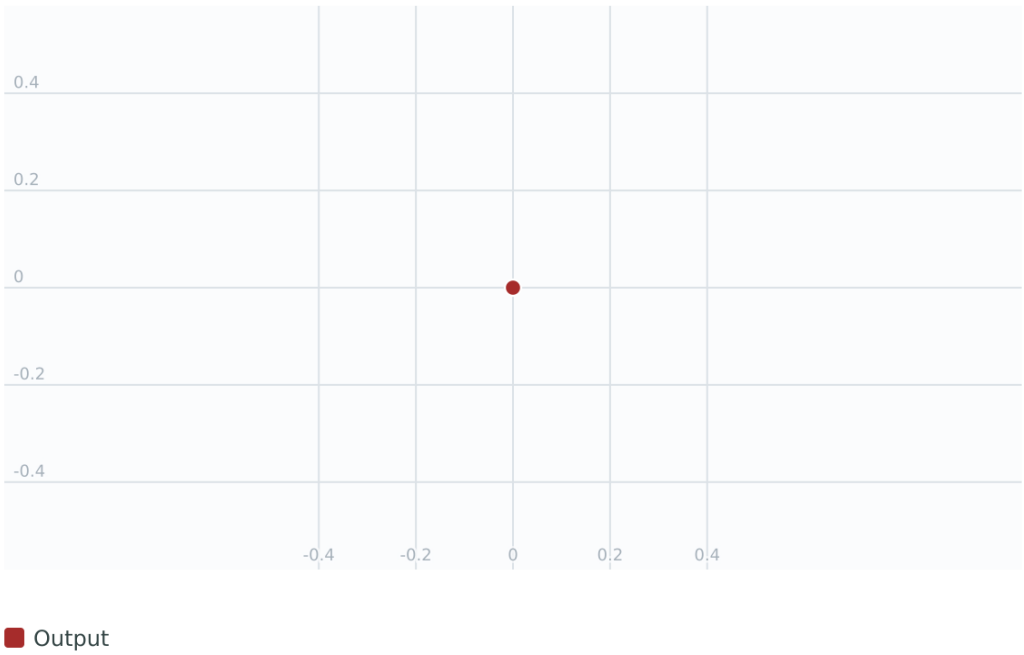
Code

```
SELECT ST_GeomFromMARC21(
'<record xmlns="http://www.loc.gov/MARC21/slim">
    <leader
>00000nz a2200000nc 4500</leader>
    <controlfield tag="001"
>040277569</controlfield>
    <datafield tag="034" ind1=" " ind2=" ">
        <subfield code="d"
>W004.500000</subfield>
        <subfield code="e"
>W004.500000</subfield>
        <subfield code="f"
>N054.250000</subfield>
        <subfield code="g"
>N054.250000</subfield>
    </datafield>
</record
>'
);
```

Output

```
POINT(-4.5 54.25)
```

Figure



Conversion de données géographiques MARC21/XML contenant un seul `POLYGON` encodé en `hdddmmss`

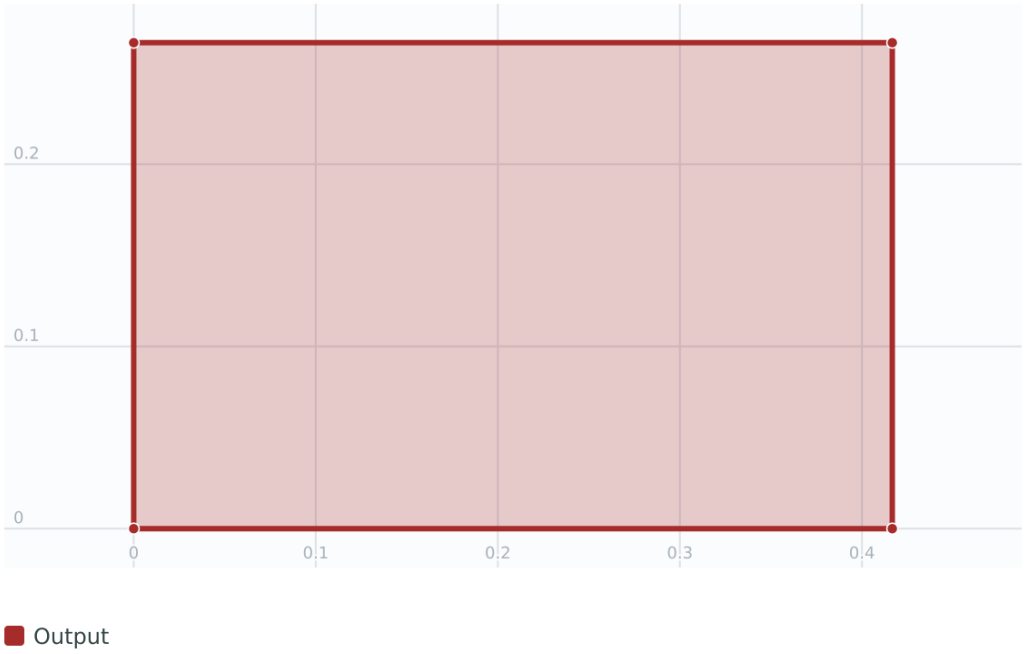
Code

```
SELECT ST_GeomFromMARC21(
'<record xmlns="http://www.loc.gov/MARC21/slim">
    <leader
>01062cem a2200241 a 4500</leader>
    <controlfield tag="001"
>   84696781 </controlfield>
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="b"
>50000</subfield>
        <subfield code="d"
>E0130600</subfield>
        <subfield code="e"
>E0133100</subfield>
        <subfield code="f"
>N0523900</subfield>
        <subfield code="g"
>N0522300</subfield>
    </datafield>
</record
>'
);
```

Output

```
POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65))
```

Figure



Conversion de données géographiques MARC21/XML contenant un `POLYGON` et un `POINT` :

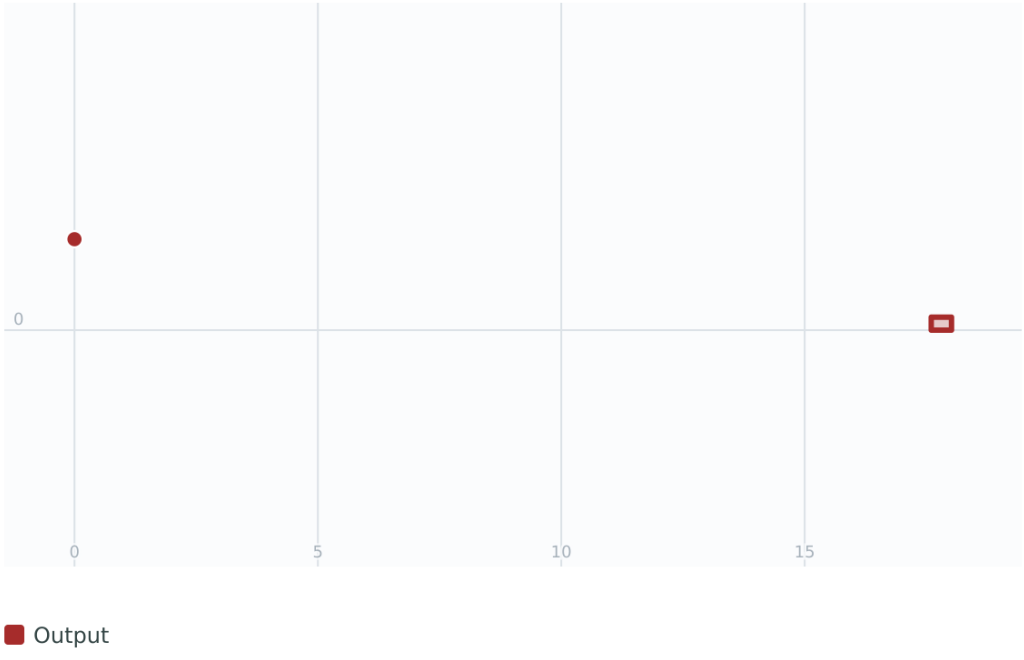
Code

```
SELECT ST_GeomFromMARC21(
'<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="b"
>50000</subfield>
        <subfield code="d"
>E0130600</subfield>
        <subfield code="e"
>E0133100</subfield>
        <subfield code="f"
>N0523900</subfield>
        <subfield code="g"
>N0522300</subfield>
    </datafield>
    <datafield tag="034" ind1=" " ind2=" ">
        <subfield code="d"
>W004.500000</subfield>
        <subfield code="e"
>W004.500000</subfield>
        <subfield code="f"
>N054.250000</subfield>
        <subfield code="g"
>N054.250000</subfield>
    </datafield>
</record
>'
);
```

Output

```
GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↩
    52.38333333333333,13.1 52.65)), POINT(-4.5 54.25))
```

Figure



Voir aussi

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Returns a geometry from Well-Known Text (WKT). Alias for ST_GeomFromText.

Synopsis

geometry **ST_GeometryFromText**(text WKT);
geometry **ST_GeometryFromText**(text WKT, integer srid);

Description

This function is an alias for ST_GeomFromText and provides the same signatures.

Disponibilité : 1.2.2

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.40

Voir aussi

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — Retourne un objet ST_Geometry à partir de sa représentation textuelle Well-Known Text (WKT).

Synopsis

geometry **ST_GeomFromText**(text WKT);
geometry **ST_GeomFromText**(text WKT, integer srid);

Description

Construit un objet Postgis de type geometry à partir d'une représentation OGC Well-Known Text WKT.



Note
Il existe deux variantes de la fonction ST_GeomFromText. La première ne prend pas de SRID et renvoie une géométrie sans système de référence spatiale défini (SRID=0). La seconde prend un SRID comme deuxième argument et renvoie une géométrie qui inclut ce SRID comme partie de ses métadonnées.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - l'option SRID est issue des tests de conformité.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.40

Cette méthode prend en charge les types Circular String et Curve.



Note
Bien que non conforme à l'OGC, ST_MakePoint est plus rapide que ST_GeomFromText et ST_PointFromText. Il est également plus facile à utiliser pour les valeurs de coordonnées numériques. ST_Point est une autre option similaire en vitesse à ST_MakePoint et est conforme à l'OGC, mais ne prend pas en charge autre chose que les points 2D.

Changement : 2.0.0 dans les version précédentes de PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') etait autorisé. C'est désormais interdit dans PostGIS 2.0.0 pour respecter la norme SQL/MM. La forme privilégiée désormais est : ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')

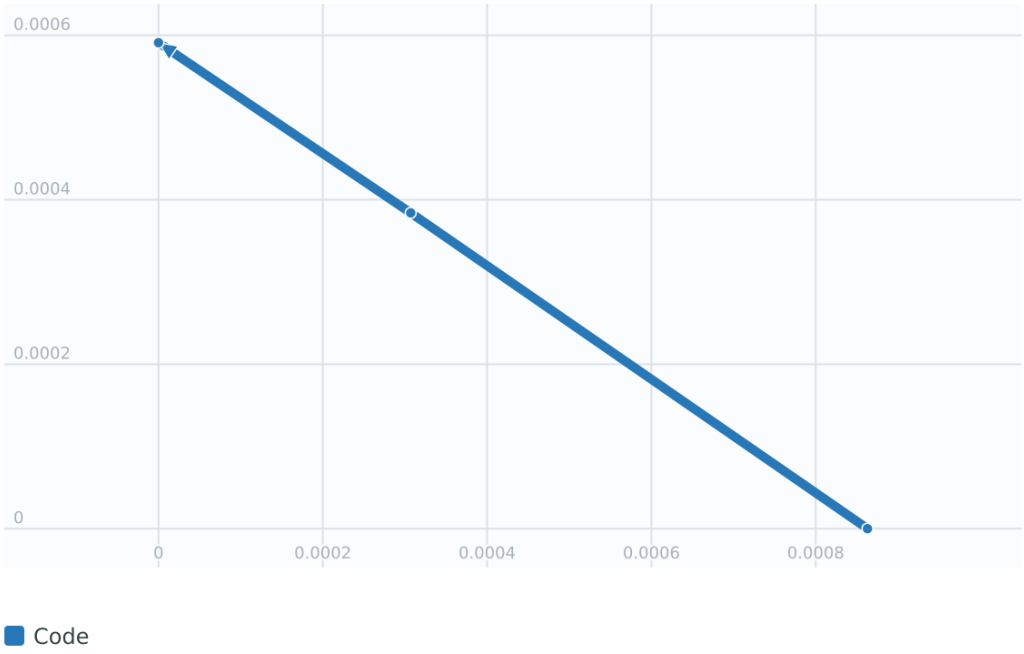
Exemples

Create a LineString with SRID 0.

Code

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)'));
```

Figure

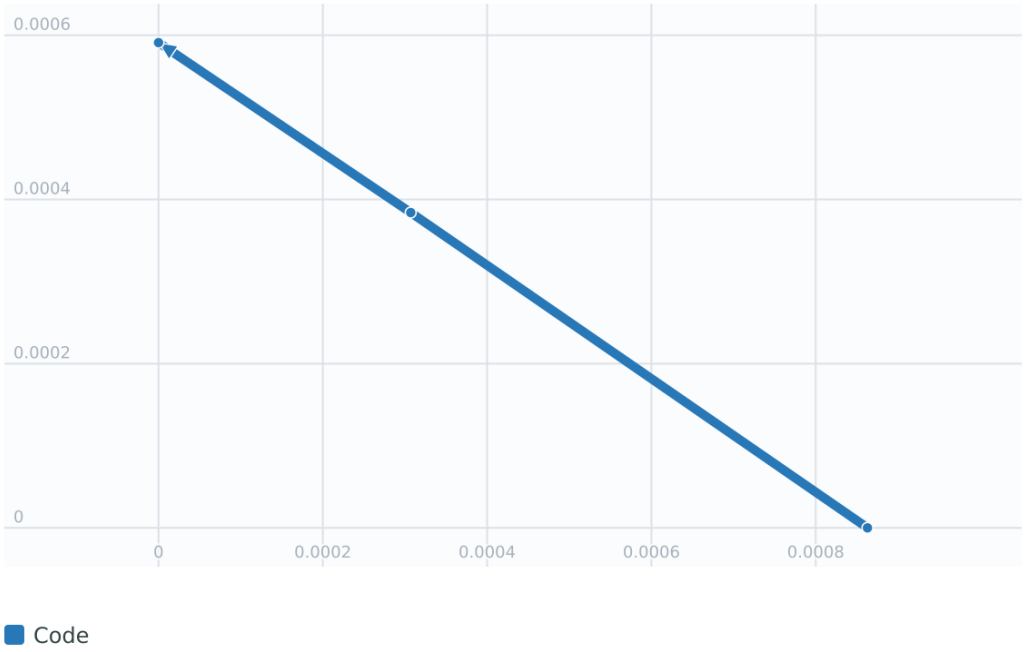


Create the same LineString with SRID 4269.

Code

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932)', 4269));
```

Figure

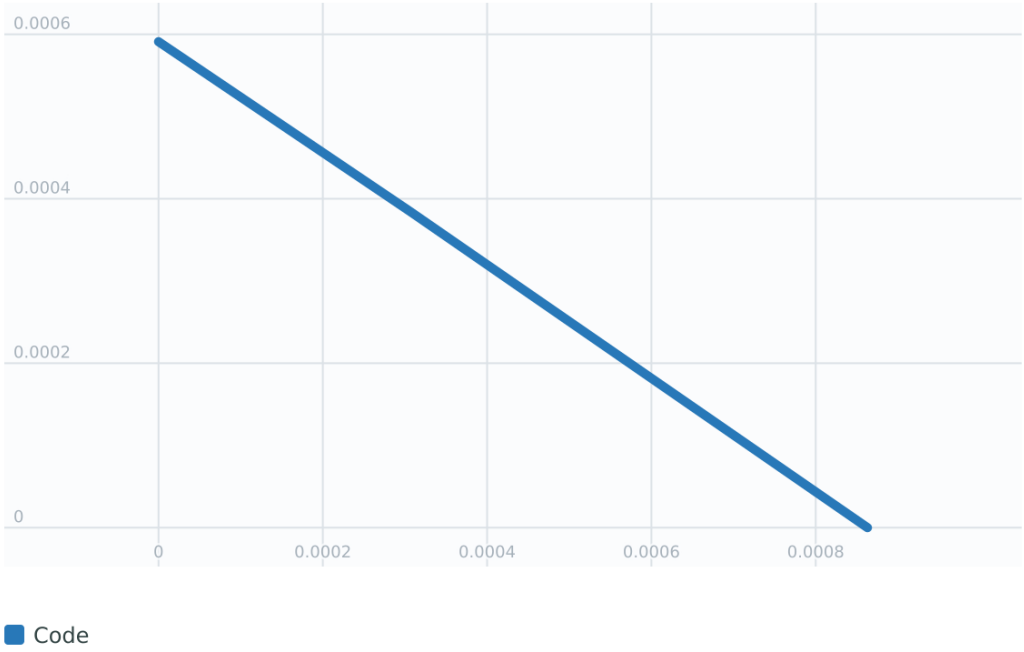


Create a MultiLineString.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('MULTILINESTRING((-71.160281 42.258729,-71.160837 ↩
    42.259113,-71.161144 42.25932))'));
```

Figure



Create a Point.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('POINT(-71.064544 42.28787)'));
```

Output

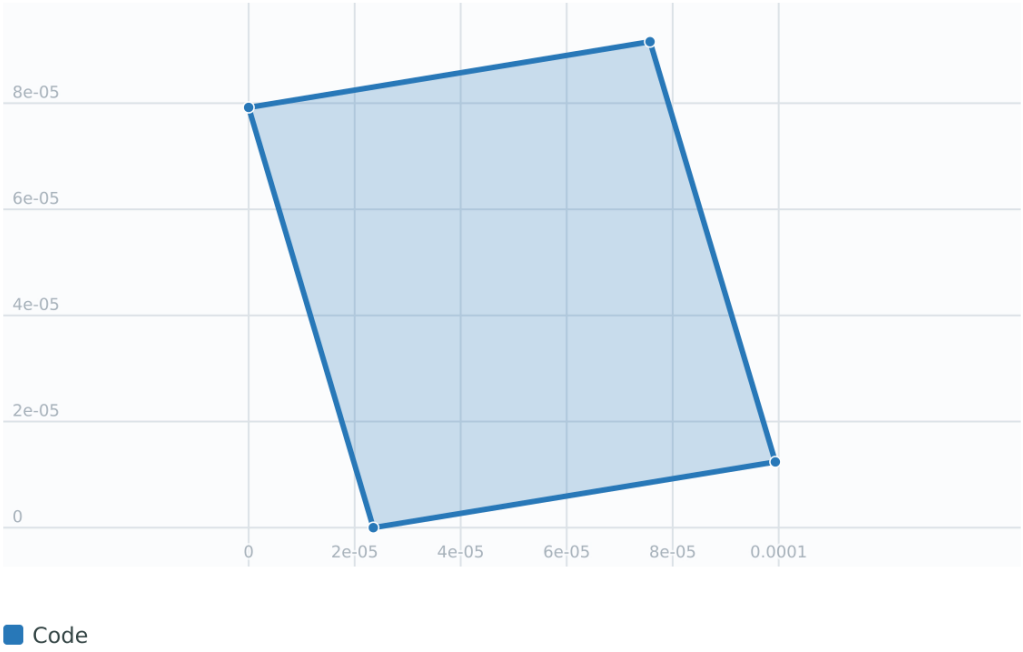
```
ST_Point
```

Create a Polygon.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('POLYGON((-71.1776585052917 ↩
    42.3902909739571,-71.1776820268866 42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↩
    42.3902909739571))'));
```

Figure



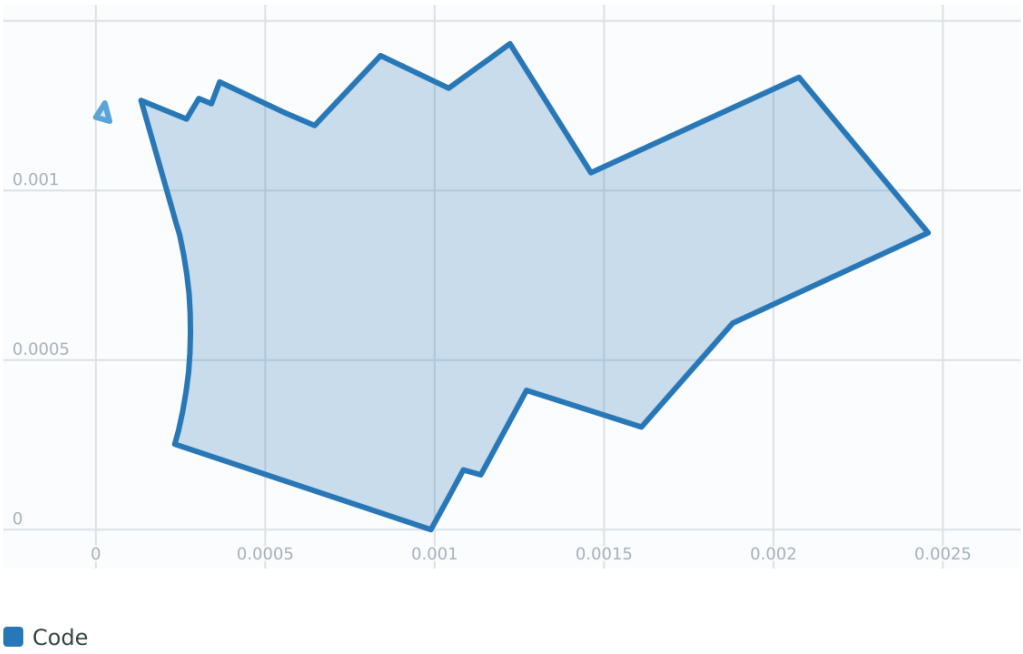
Create a MultiPolygon with SRID 4326.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('MULTIPOLYGON(((-71.1031880899493 42.3152774590236,
-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,
-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,
-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,
-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,
-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,
-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,
-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,
-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,
-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,
-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,
-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,
-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,
-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,
-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,
```

```
-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,
-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,
-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,
-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,
-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,
-71.1031880899493 42.3152774590236)),
((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,
-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873
    42.315113108546)))', 4326));
```

Figure

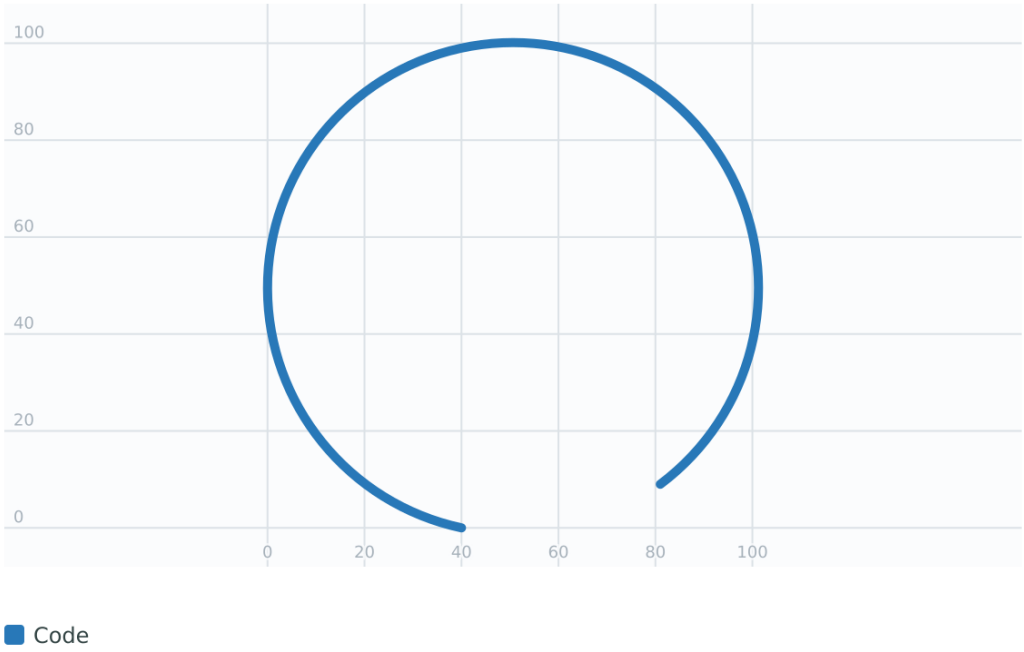


Create a CircularString.

Code

```
SELECT ST_GeometryType(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227
    150406)'));
```

Figure



Voir aussi

ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.8.1.10 ST_LineFromText

ST_LineFromText — Construit une géométrie à partir d'une représentation WKT avec le SRID donné. Si aucun SRID n'est donné, la valeur par défaut est 0.

Synopsis

geometry **ST_LineFromText**(text WKT);
geometry **ST_LineFromText**(text WKT, integer srid);

Description

Crée une géométrie à partir de WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0. Si le WKT passé n'est pas un LINESTRING, null est retourné.



Note
OGC SPEC 3.2.6.2 - SRID optionnel issu des tests de conformité.



Note
Si vous êtes sûrs que toutes les géométries WKT sont des LINESTRINGS, la fonction ST_GeomFromText est plus efficace car elle ne contrôle pas le type de la géométrie renvoyée.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.2.8

Exemples

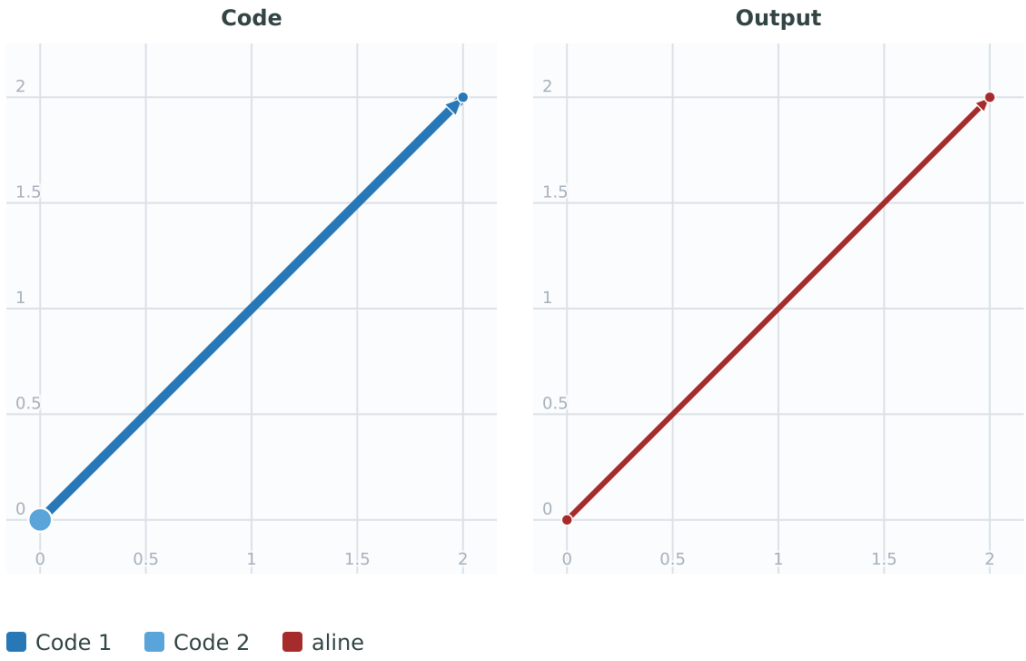
Code

```
SELECT ST_LineFromText('LINESTRING(1 2,3 4)') AS aline,
ST_LineFromText('POINT(1 2)') IS NULL AS null_return;
```

Output

```
aline                | null_return
-------------------------------------
LINESTRING(1 2,3 4)  | t
```

Figure



Voir aussi

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — Retourne un objet de type ST_MultiLineString à partir de sa représentation WKT.

Synopsis

geometry **ST_MLineFromText**(text WKT, integer srid);
geometry **ST_MLineFromText**(text WKT);

Description

Crée une géométrie à partir du texte connu (WKT) avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.

OGC SPEC 3.2.6.2 - l'option SRID est issue des tests de conformité

Retourne NULL si le WKT n'est pas une MULTILINESTRING



Note
Si vous êtes absolument sûrs que toutes les géométries WKT sont des points, ne pas utiliser cette fonction. Elle est plus lente que ST_GeomFromText à cause d'une étape de validation supplémentaire.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2



Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 9.4.4

Exemples

Code

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2,3 4),(4 5,6 7))');
```

Voir aussi

ST_GeomFromText

7.8.1.12 ST_MPointFromText

ST_MPointFromText — Créé une Geometry depuis un WKT avec le SRID donné. Si le SRID n'est pas fourni, il sera défini par défaut à 0.

Synopsis

geometry **ST_MPointFromText**(text WKT, integer srid);
geometry **ST_MPointFromText**(text WKT);

Description

Créé une Geometry depuis un WKT avec le SRID donné. Si le SRID n'est pas fourni, il sera défini par défaut à 0.

OGC SPEC 3.2.6.2 - l'option SRID est issue des tests de conformité

Retourne NULL si le WKT n'est pas une MULTIPOINT



Note
Si vous êtes absolument sûrs que toutes les géométries WKT sont des points, ne pas utiliser cette fonction. Elle est plus lente que ST_GeomFromText à cause d'une étape de validation supplémentaire.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2



Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 9.2.4

Exemples

Code

```
SELECT ST_MPointFromText('MULTIPOINT((1 2),(3 4))');
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180),(-70.9611 42.1223))', 4326);
```

Voir aussi

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — Crée une géométrie multi-polygone à partir de WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.

Synopsis

geometry **ST_MPolyFromText**(text WKT, integer srid);
geometry **ST_MPolyFromText**(text WKT);

Description

Crée un MultiPolygone à partir de WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.

OGC SPEC 3.2.6.2 - l'option SRID est issue des tests de conformité

Retourne une erreur si le WKT n'est pas un MULTIPOLYGON



Note
Si vous êtes absolument sûrs que toutes les géométries WKT sont des multipolygones, ne pas utiliser cette fonction. Elle est plus lente que ST_GeomFromText à cause d'une étape de validation supplémentaire.


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 9.6.4

Exemples

Create a 3D MultiPolygon with an interior ring and SRID 0.

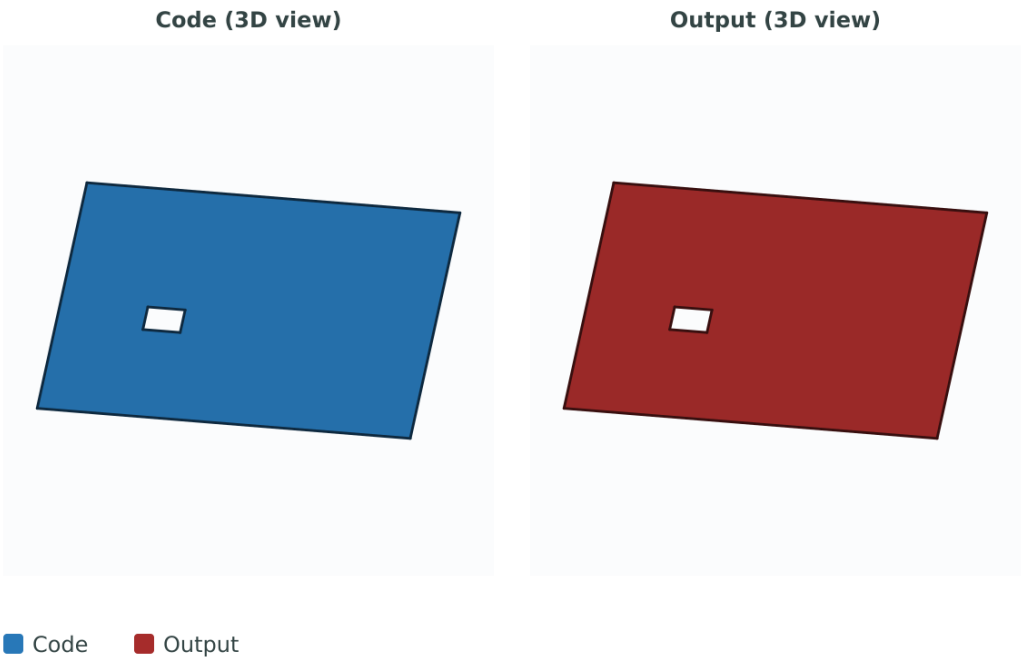
Code

```
SELECT ST_MPolyFromText(
    'MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),
                   (5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))'
);
```

Output

```
MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 3,7 5 3,5 5 3)))
```

Figure

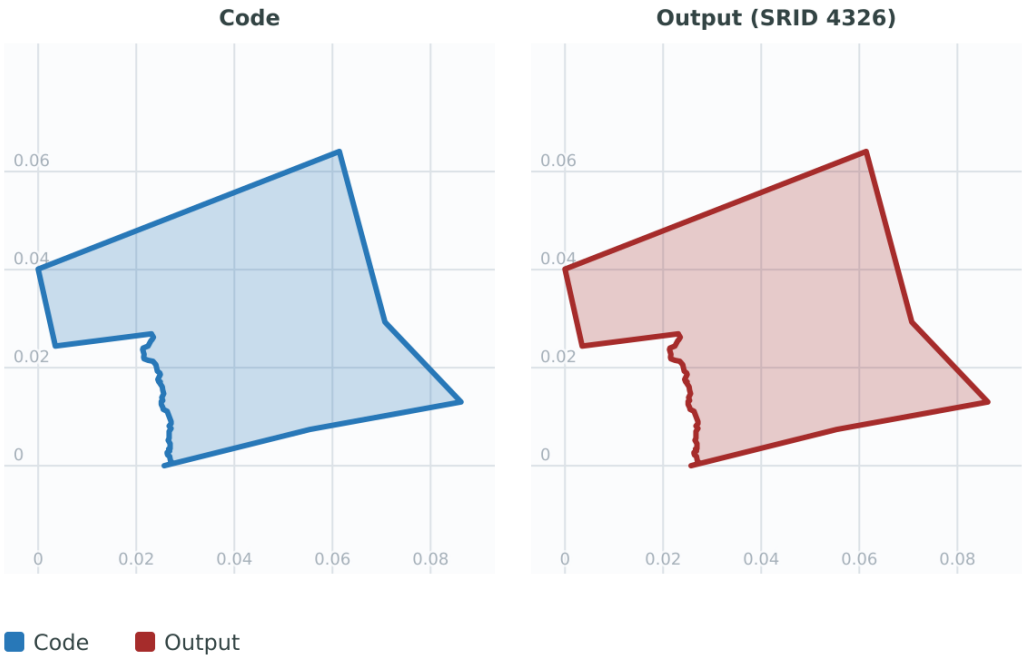


Create a MultiPolygon with SRID 4326.

Code

```
SELECT ST_MPolyFromText(
    'MULTIPOLYGON(((-70.916 42.1002,-70.9468 42.0946,-70.9765 42.0872,
      -70.9754 42.0875,-70.9749 42.0879,-70.9752 42.0881,-70.9754 42.0891,
      -70.9758 42.0894,-70.9759 42.0897,-70.9759 42.0899,-70.9754 42.0902,
      -70.9756 42.0906,-70.9753 42.0907,-70.9753 42.0917,-70.9757 42.0924,
      -70.9755 42.0928,-70.9755 42.0942,-70.9751 42.0948,-70.9755 42.0953,
      -70.9751 42.0958,-70.9751 42.0962,-70.9759 42.0983,-70.9767 42.0987,
      -70.9768 42.0991,-70.9771 42.0997,-70.9771 42.1003,-70.9768 42.1005,
      -70.977 42.1011,-70.9766 42.1019,-70.9768 42.1026,-70.9769 42.1033,
      -70.9775 42.1042,-70.9773 42.1043,-70.9776 42.1043,-70.9778 42.1048,
      -70.9773 42.1058,-70.9774 42.1061,-70.9779 42.1065,-70.9782 42.1078,
      -70.9788 42.1085,-70.9798 42.1087,-70.9806 42.109,-70.9807 42.1093,
      -70.9806 42.1099,-70.9809 42.1109,-70.9808 42.1112,-70.9798 42.1116,
      -70.9792 42.1127,-70.979 42.1129,-70.9787 42.1134,-70.979 42.1139,
      -70.9791 42.1141,-70.9987 42.1116,-71.0022 42.1273,-70.9408 42.1513,
      -70.9315 42.1165,-70.916 42.1002)))',
    4326
);
```

Figure



Voir aussi

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — Construit une géométrie point à partir d'une représentation WKT et le SRID donné. Si aucun SRID n'est donné, la valeur par défaut est 0.

Synopsis

geometry **ST_PointFromText**(text WKT);
geometry **ST_PointFromText**(text WKT, integer srid);

Description

Construit un objet point ST_Geometry de PostGIS à partir de la représentation textuelle Well-Known de l'OGC. Si le SRID n'est pas donné, il prend par défaut la valeur inconnue (actuellement 0). Si la géométrie n'est pas une représentation de point WKT, retourne null. Si la représentation WKT n'est pas du tout valide, une erreur est générée.



Note
Il existe 2 versions de la fonction ST_PointFromText : la première ne prend pas de SRID en paramètre et retourne une geometry sans système de coordonnées. La seconde prend un SRID en second paramètre et retourne une ST_Geometry incluant un SRID dans ses métadonnées. Ce SRID doit obligatoirement exister dans la table spatial_ref_sys.



Note
Si vous êtes absolument sûrs que toutes les géométries WKT sont des points, ne pas utiliser cette fonction. Elle est plus lente que ST_GeomFromText à cause d'une étape de validation supplémentaire. Si le point doit être construit à partir de coordonnées latitude longitude et que la performance est recherchée, utiliser la fonction ST_MakePoint ou son équivalent OGC ST_Point.

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - l'option SRID est issue des tests de conformité.

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 6.1.8

Exemples

Create a point with the default SRID 0.

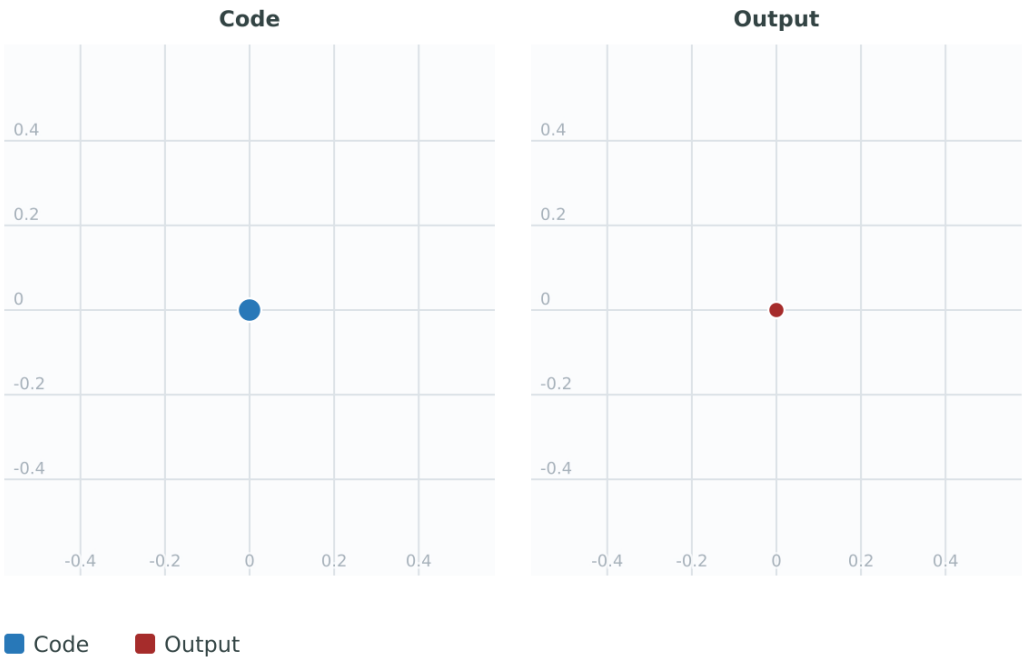
Code

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
```

Output

```
POINT(-71.064544 42.28787)
```

Figure



Create a point with SRID 4326.

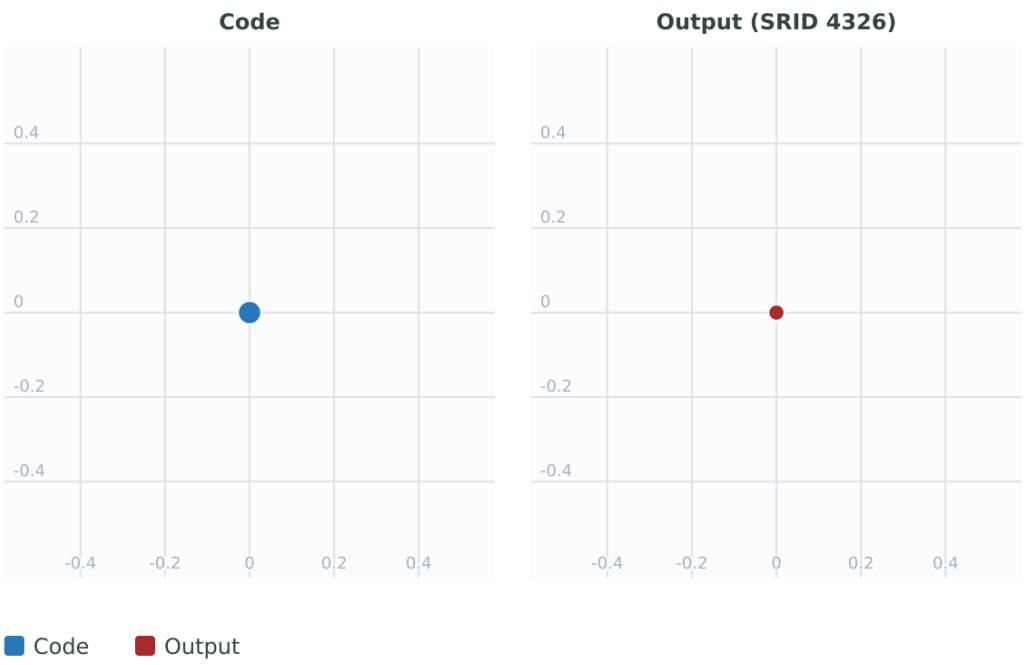
Code

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

Output

```
SRID=4326;POINT(-71.064544 42.28787)
```

Figure



Voir aussi

ST_GeomFromText, ST_MakePoint, ST_Point, ST_SRID

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — Créé une Geometry depuis un WKT avec le SRID donné. Si le SRID n'est pas fourni, il sera défini par défaut à 0.

Synopsis

geometry **ST_PolygonFromText**(text WKT);
geometry **ST_PolygonFromText**(text WKT, integer srid);

Description

Crée une géométrie à partir de WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0. Retourne null si WKT n'est pas un polygone.

OGC SPEC 3.2.6.2 - l'option SRID est issue des tests de conformité



Note
Si vous êtes absolument sûrs que toutes les géométries WKT sont des polygones, ne pas utiliser cette fonction. Elle est plus lente que ST_GeomFromText à cause d'une étape de validation supplémentaire.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 8.3.6

Exemples

The result is returned as a native geometry. The documentation test runner compares that value directly with the readable EWKT shown below and uses the same geometry for the generated figure.

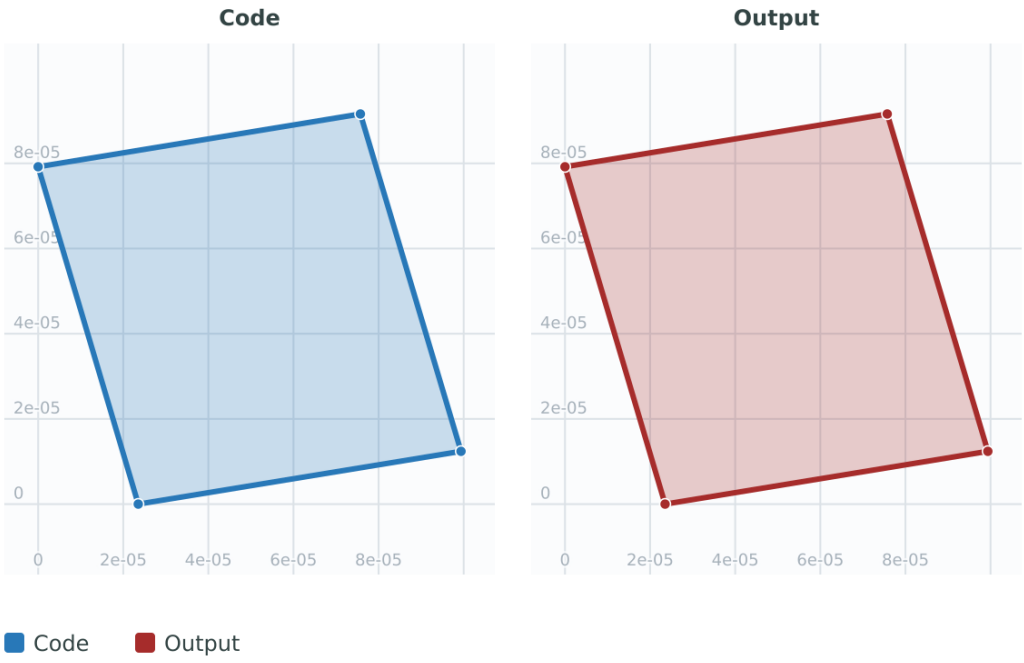
Code

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 ↩
    42.3903701743239,
-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 ↩
    42.3902909739571))');
```

Output

```
POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866 42.3903701743239,-71.1776063012595 ↩
    42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917 42.3902909739571))
```

Figure



Code

```
SELECT ST_PolygonFromText('POINT(1 2)') IS NULL AS point_is_notpoly;
```

Output

```
t
```

Voir aussi

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Returns a geometry from Well-Known Text (WKT). SQL/MM alias for ST_GeomFromText.

Synopsis

geometry **ST_WKTToSQL**(text WKT);

Description

This function is the SQL/MM alias for ST_GeomFromText with no SRID argument.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.34

Voir aussi

ST_GeomFromText

7.8.2 Well-Known Binary (WKB)

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — Retourne un objet de type geography à partir de sa représentation binaire Well-Know Binary (WKB ou EWKB).

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

Description

`ST_GeogFromWKB` prend en paramètre une représentation binaire d'une géométrie (WKB ou EWKB) et crée une instance de type geography. Cette fonction assure le rôle de Geometry Factory en SQL.

Si le SRID n'est pas spécifié, la valeur 4326 est prise (WGS 84 long lat).

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Although bytea rep contains single \, these need to be escaped when inserting into a table.

Code

```
SELECT ST_GeogFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↩
    \\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230 ↩
    C@');
```

Output

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



Voir aussi

ST_GeogFromText, ST_AsBinary

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — Retourne un objet ST_Geometry à partir de sa représentation binaire étendue (Extended Well-Known Binary representation, EWKB).

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

Description

Constructs a PostGIS geometry from an Extended Well-Known Binary (EWKB) representation.



Note
Le format EWKB n'est pas une norme OGC, mais un format spécifique à PostGIS incluant l'identifiant du système de référence des coordonnées (SRID)

Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

This example constructs `LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932)` in NAD83 longitude/latitude (SRID 4269) from its EWKB representation.

PostgreSQL escape string syntax (`E'...'`) always interprets backslash escapes. Each backslash therefore has to be doubled so that the bytea input parser receives the octal escapes.

Code

```
SELECT ST_GeomFromEWKB((
    E'\\001\\002\\000\\000\\040\\255\\020\\000\\000\\003\\000\\000\\000\\344\\112\\075' ||
    E'\\013\\102\\312\\121\\300\\156\\303\\050\\010\\036\\041\\105\\100\\047\\277\\105' ||
    E'\\047\\113\\312\\121\\300\\366\\173\\142\\235\\052\\041\\105\\100\\225\\174\\354' ||
    E'\\056\\120\\312\\121\\300\\160\\231\\323\\145\\061\\041\\105\\100'
)::bytea);
```

Output

```
0102000020 ↩
    AD10000003000000E44A3D0B42CA51C06EC328081E21454027BF45274BCA51C0F67B629D2A214540957CEC2E50CA51C07099D36531214540
```

Figure



With `standard_conforming_strings` enabled (the default), ordinary string literals preserve backslashes. The same EWKB can therefore be written with a single backslash before each octal byte value.

Code

```
SELECT ST_GeomFromEWKB((
    '\001\002\000\000\040\255\020\000\000\003\000\000\000\344\112\075' ||
    '\013\102\312\121\300\156\303\050\010\036\041\105\100\047\277\105' ||
    '\047\113\312\121\300\366\173\142\235\052\041\105\100\225\174\354' ||
    '\056\120\312\121\300\160\231\323\145\061\041\105\100'
)::bytea);
```

Output

```
0102000020 ↩
    AD10000003000000E44A3D0B42CA51C06EC328081E21454027BF45274BCA51C0F67B629D2A214540957CEC2E50CA51C0
```

Voir aussi

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — Retourne un objet de type geometry à partir de sa représentation binaire Well-Know Binary (WKB) et d'un SRID optionnel.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Description

`ST_GeomFromWKB` prend en paramètre une représentation binaire d'une géométrie (WKB ou EWKB) et un SRID optionnel (`SRID`) et crée une instance de type geometry. Cette fonction assure le rôle de Geometry Factory en SQL. Alias pour ST_WKBToSQL.

Si le SRID n'est pas précisé, la valeur 0 (indéfini) est prise par défaut.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - le paramètre optionnel est issu des tests de conformité

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.41

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Although bytea rep contains single \, these need to be escaped when inserting into a table. Unless standard_conforming_strings is set to on.

Code

```
SELECT ST_GeomFromWKB(E'\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q ↩
    \\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230 ↩
    C@', 4326);
```

Output

```
SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



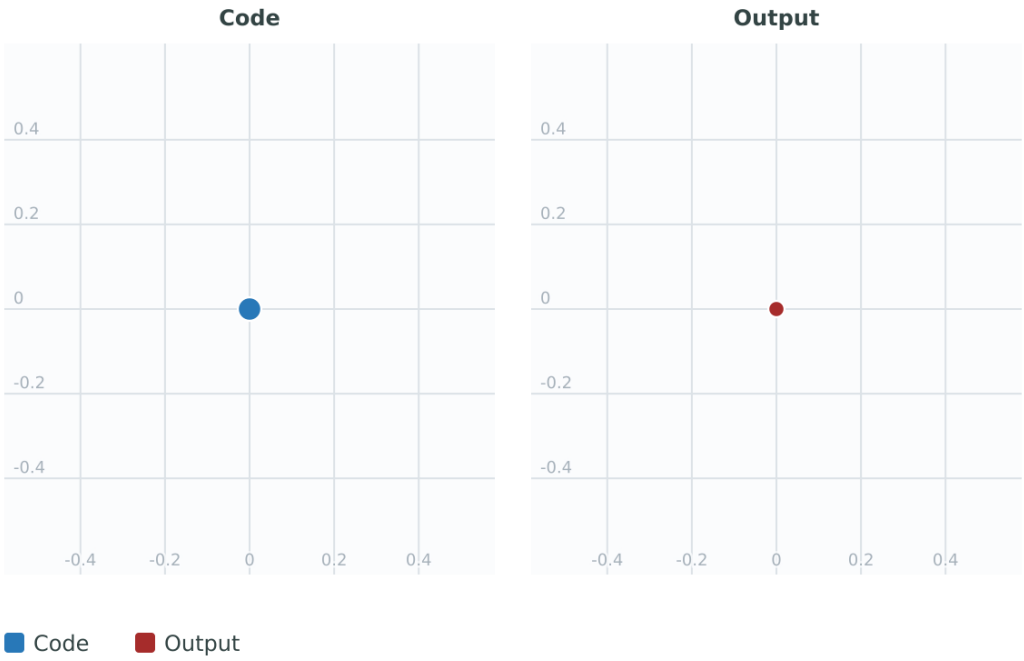
Code

```
SELECT ST_GeomFromWKB(ST_AsEWKB('POINT(2 5)'::geometry)
  );
```

Output

```
POINT(2 5)
```

Figure



The PostgreSQL `decode` function converts a hexadecimal WKB string to the bytea value accepted by `ST_GeomFromWKB`.

Code

```
SELECT ST_AsEWKT(ST_GeomFromWKB(
    decode('0101000000e5d022dbf93e2e40dbf97e6abc743540', 'hex'),
    4326
)) AS geom;
```

Output

```
SRID=4326;POINT(15.123 21.456)
```

Voir aussi

ST_WKBToSQL, ST_AsBinary, ST_AsEWKT, ST_GeomFromEWKB

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — Creates a LINESTRING from a WKB representation, optionally using a given SRID.

Synopsis

geometry **ST_LineFromWKB**(bytea WKB);
geometry **ST_LineFromWKB**(bytea WKB, integer srid);

Description

The ST_LineFromWKB function takes a Well-Known Binary (WKB) representation of a geometry and an optional Spatial Reference System ID (SRID), and creates a LINESTRING geometry. It is a type-specific geometry factory in SQL.

If an SRID is not specified, it defaults to 0. NULL is returned if the input bytea does not represent a LINESTRING geometry.



Note
OGC SPEC 3.2.6.2 - SRID optionnel issu des tests de conformité.



Note
If you know all your geometries are LINESTRINGs, it is more efficient to use ST_GeomFromWKB directly. This function calls ST_GeomFromWKB and adds validation that the result is a LINESTRING.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2



Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.2.9

Exemples

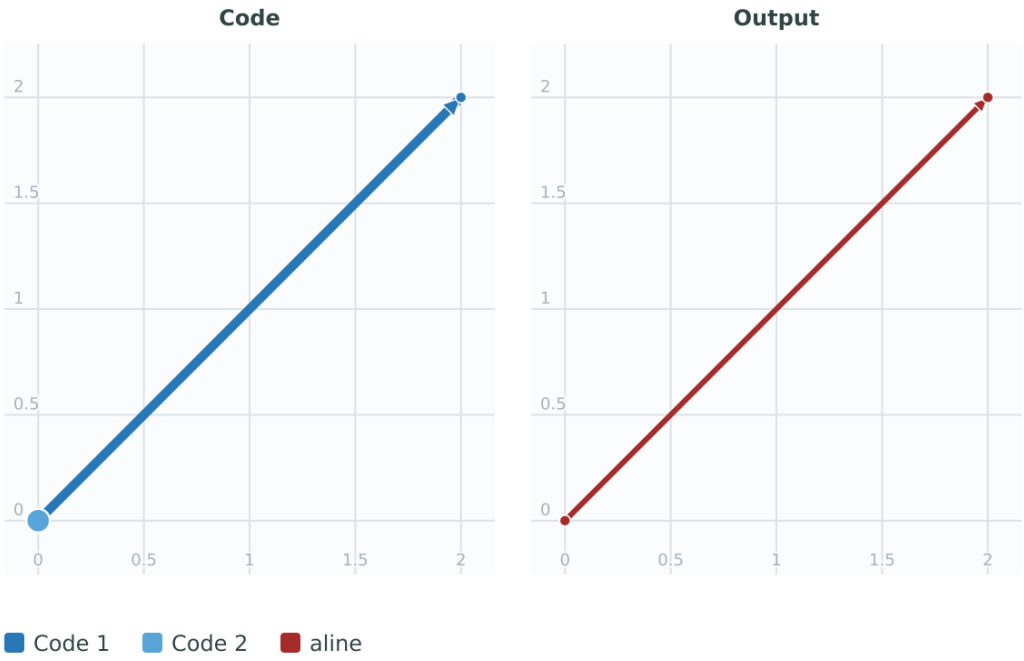
Code

```
SELECT ST_LineFromWKB(
    ST_AsBinary(ST_GeomFromText('LINESTRING(1 2,3 4)'))
) AS aline,
ST_LineFromWKB(
    ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
```

Output

```
aline                 | null_return
----------------------------------
LINESTRING(1 2,3 4)   | t
```

Figure



Voir aussi

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — Creates a `LINESTRING` from a WKB representation, optionally using a given SRID.

Synopsis

geometry **ST_LinestringFromWKB**(bytea WKB);
geometry **ST_LinestringFromWKB**(bytea WKB, integer srid);

Description

The `ST_LinestringFromWKB` function takes a Well-Known Binary (WKB) representation of a geometry and an optional Spatial Reference System ID (`SRID`), and creates a `LINESTRING` geometry. It is an alias for ST_LineFromWKB.

If an SRID is not specified, it defaults to 0. `NULL` is returned if the input `bytea` does not represent a `LINESTRING` geometry.



Note
OGC SPEC 3.2.6.2 - SRID optionnel issu des tests de conformité.

Note
If you know all your geometries are `LINESTRING`s, it is more efficient to use ST_GeomFromWKB directly. This function calls ST_GeomFromWKB and adds validation that the result is a `LINESTRING`.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 7.2.9

Exemples

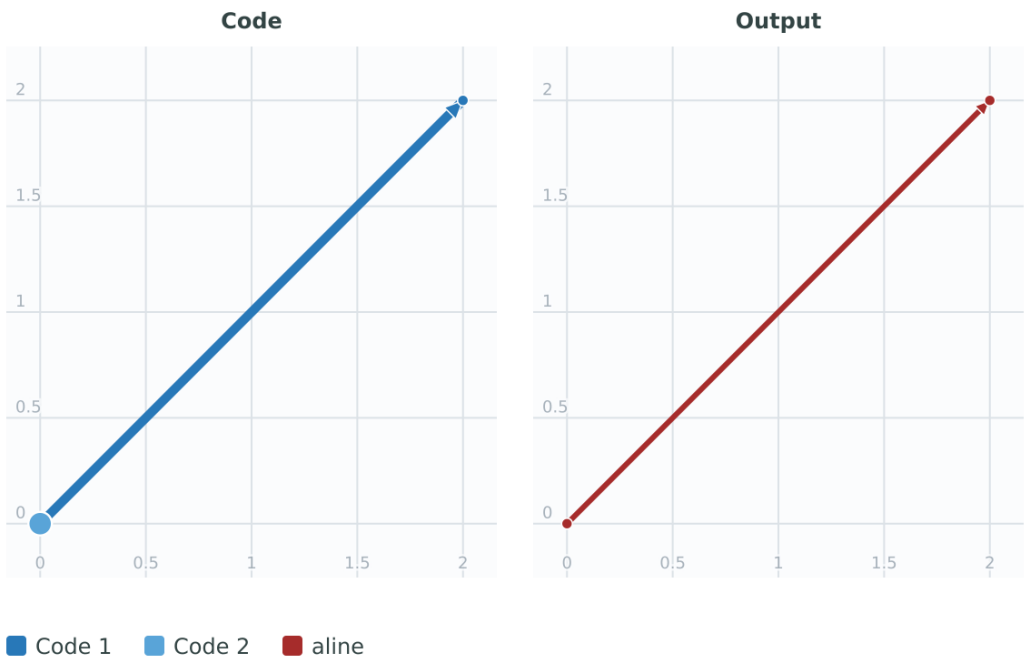
Code

```
SELECT ST_LinestringFromWKB(
    ST_AsBinary(ST_GeomFromText('LINESTRING(1 2,3 4)'))
) AS aline,
ST_LinestringFromWKB(
    ST_AsBinary(ST_GeomFromText('POINT(1 2)'))
) IS NULL AS null_return;
```

Output

```
aline                | null_return
-------------------------------------
LINESTRING(1 2,3 4)  | t
```

Figure



Voir aussi

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — Construit une géométrie depuis la représentation binaire WKB et le SRID donné

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Description

`ST_PointFromWKB` prend en paramètre une représentation binaire d'une géométrie et un SRID (`SRID`) et crée une instance du bon type géométrique, en l'occurence une `POINT`. Cette fonction assure le rôle de Geometry Factory en SQL.

Si le SRID n'est pas précisé, la valeur 0 est prise par défaut. `NULL` est retourné si le paramètre `bytea` donné ne représente pas une géométrie `POINT`.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 6.1.9

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

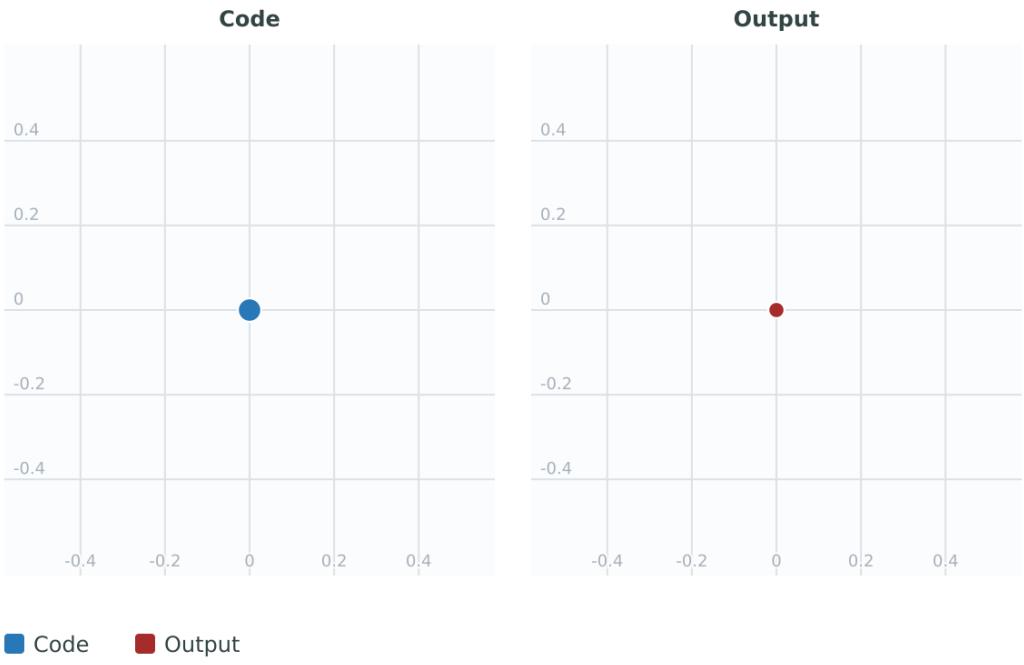
Code

```
SELECT ST_PointFromWKB(ST_AsEWKB('POINT(2 5)'::geometry));
```

Output

```
POINT(2 5)
```

Figure



Code

```
SELECT ST_PointFromWKB(ST_AsEWKB('LINESTRING(2 5,2 6)'::geometry));
```

Output

```
null
```

Voir aussi

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Returns a geometry from Well-Known Binary (WKB). SQL/MM alias for ST_GeomFromWKB without an SRID argument.

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

Description

This function is the SQL/MM alias for ST_GeomFromWKB with no SRID argument.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.36

Voir aussi

ST_GeomFromWKB

7.8.3 Autres formats

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — Retourne une BOX2D à partir d'une chaîne GeoHash.

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Description

Retourne une BOX2D à partir d'une chaîne GeoHash.

Si aucune `precision` n'est spécifiée, ST_Box2dFromGeoHash retourne un BOX2D basé sur la précision complète de la chaîne GeoHash d'entrée.

Si `precision` est spécifié, ST_Box2dFromGeoHash utilise autant de caractère du GeoHash pour créer la BOX2D. Une précision plus basse retourne des BOXD2 plus grandes, et une valeur plus haute améliore la précision.

Disponibilité : 2.1.0

Exemples

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0');
```

Output

```
BOX(-115.172816 36.114646,-115.172816 36.114646)
```

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 0);
```

Output

```
BOX(-180 -90,180 90)
```

Code

```
SELECT ST_Box2dFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);
```

Output

```
BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)
```

Voir aussi

ST_GeoHash, ST_GeomFromGeoHash, ST_PointFromGeoHash

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — Retourne une geometry depuis une chaîne GeoHash.

Synopsis

geometry **ST_GeomFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Description

Retourne une Geometry à partir d'une chaîne GeoHash. La géométrie sera un polygone représentant les limites du GeoHash.

Si aucune `precision` n'est spécifiée, ST_GeomFromGeoHash retourne un polygone basé sur la précision complète de la chaîne GeoHash en entrée.

Si `precision` est spécifié, ST_GeomFromGeoHash utilise autant de caractère du GeoHash pour créer le polygone.

Disponibilité : 2.1.0

Exemples

The cells are returned together so their relative sizes remain visible in a shared coordinate frame.

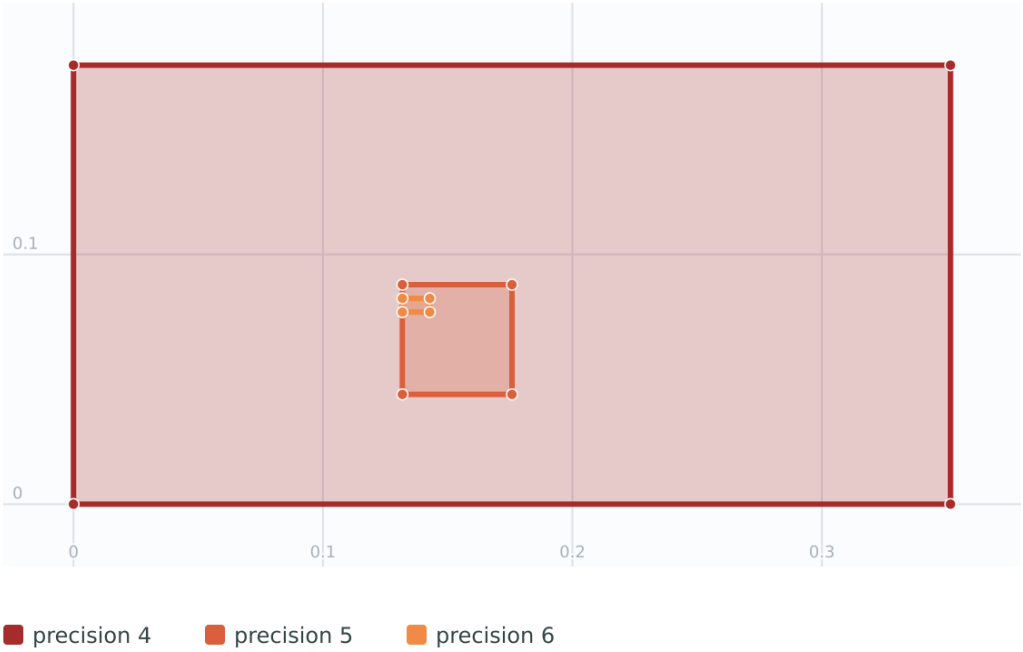
Code

```
SELECT ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4) AS "precision 4", ST_GeomFromGeoHash( ↩
    '9qqj7nmxncgyy4d0dbxqz0', 5) AS "precision 5", ST_GeomFromGeoHash('9 ↩
    qqj7nmxncgyy4d0dbxqz0', 6) AS "precision 6";
```

Output

```
-[ RECORD 1 ]----------
precision 4 | POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375 ↩
     36.03515625,-115.3125 36.03515625))
precision 5 | POLYGON((-115.1806640625 36.0791015625,-115.1806640625 36.123046875,-115.13671875 ↩
     36.123046875,-115.13671875 36.0791015625,-115.1806640625 36.0791015625))
precision 6 | POLYGON((-115.1806640625 36.112060546875,-115.1806640625 36.1175537109375,-115.169677734375 ↩
     36.1175537109375,-115.169677734375 36.112060546875,-115.1806640625 36.112060546875))
```

Figure



Voir aussi

ST_GeoHash,ST_Box2dFromGeoHash, ST_PointFromGeoHash

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry

Synopsis

geometry **ST_GeomFromGML**(text geomgml);
geometry **ST_GeomFromGML**(text geomgml, integer srid);

Description

Construit un objet PostGIS ST_Geometry à partir d'une représentation GML OGC.

La fonction ST_GeomFromGML fonctionne uniquement avec le fragment GML représentant la géométrie. Elle renvoie une error si un document GML complet est utilisé.

version OGC GML supportée :

— GML 3.2.1 Namespace

— GML 3.1.1 Simple Features profile SF-2 (with GML 3.1.0 and 3.0.0 backward compatibility)
— GML 2.1.2

OGC GML standards, cf : http://www.opengeospatial.org/standards/gml :

Disponibilité : 1.5, nécessite libxml2 1.6+

Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques.

Amélioration : 2.0.0 paramètre optionnel de srid par défaut ajouté.

Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Le format GML supporte des objets de dimensions différentes (2D et 3D dans la même MultiGeometry par exemple). PostGIS ne supportant pas cela, la fonction convertit toute la géometrie en 2D si une seule coordonnée Z manque.

Le format GML supporte des objets ayant des SRID différents dans la même MultiGeometry. PostGIS ne supportant pas cela, ST_GeomFromGML reprojète toutes les sous géométries dans le SRS du noeud racine. Si aucun attribut srsName n'est disponible pour le noeud racine GML, la fonction renvoie une erreur.

La fonction ST_GeomFromGML n'impose pas d'utiliser un espace de noms GML explicite. Pour les usages courants, il peut être ignoré. Il est en revanche nécessaire en cas d'utilisation de la fonctionnalité XLink dans le GML.

Exemples

A single geometry with srsName.

Code

```
SELECT ST_AsEWKT(ST_GeomFromGML($$
    <gml:LineString xmlns:gml="http://www.opengis.net/gml"
            srsName="EPSG:4269">
        <gml:coordinates>
    -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
        </gml:coordinates>
    </gml:LineString>
$$));
```

Output

```
SRID=4269;LINESTRING(-71.16028 42.258729,-71.160837 42.259112,-71.161143 42.25932)
```

Figure



XLink usage.

Code

```
        SELECT ST_AsEWKT(ST_GeomFromGML($$
            <gml:LineString xmlns:gml="http://www.opengis.net/gml"
                    xmlns:xlink="http://www.w3.org/1999/xlink"
                    srsName="EPSG:4269">
        <gml:pointProperty>
            <gml:Point gml:id="p1"
><gml:pos
>-71.16028 42.258729</gml:pos
></gml:Point>
        </gml:pointProperty>
        <gml:pos
>-71.160837 42.259112</gml:pos>
        <gml:pos
>-71.161143 42.258729</gml:pos>
                <gml:pointProperty>
                    <gml:Point xlink:type="simple" xlink:href="#p1"/>
                </gml:pointProperty>
            </gml:LineString>
        $$));
```

Output

```
SRID=4269;LINESTRING(-71.16028 42.258729,-71.160837 42.259112,-71.161143 42.258729,-71.16028 42.258729)
```

Figure



Polyhedral Surface.

Code

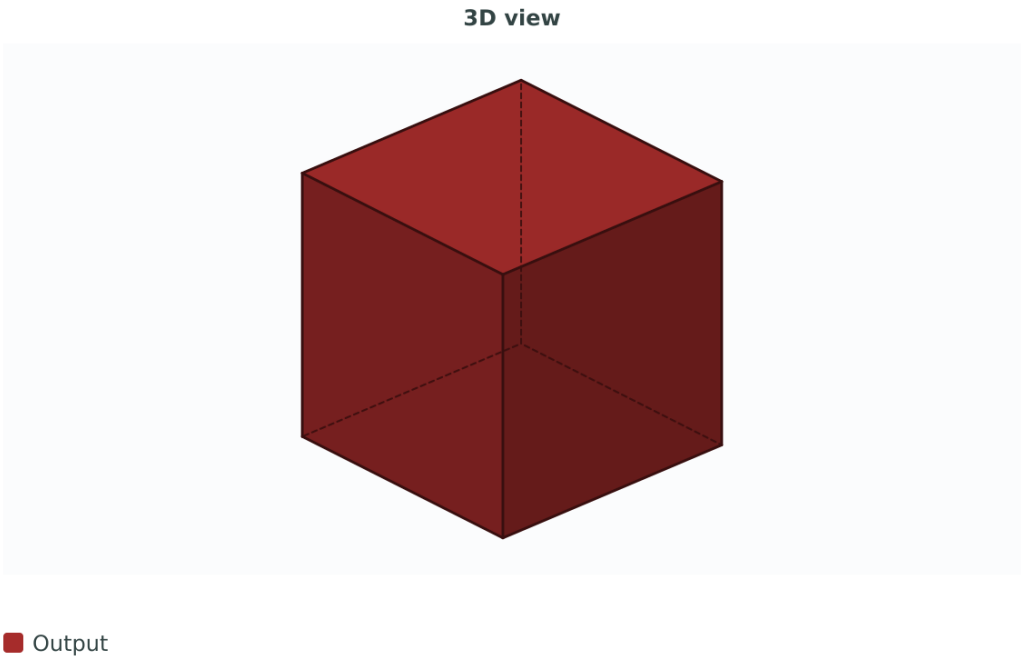
```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface xmlns:gml="http://www.opengis.net/gml">
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
```

```
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
    <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>'));
```

Output

```
POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))
```

Figure



Voir aussi

Section 2.2.3, ST_AsGML, ST_GMLToSQL

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — Prend en entrée une géométrie au format geojson et renvoie un objet Postgis de type geometry

Synopsis

geometry **ST_GeomFromGeoJSON**(text geomjson);
geometry **ST_GeomFromGeoJSON**(json geomjson);
geometry **ST_GeomFromGeoJSON**(jsonb geomjson);

Description

Construit un objet Postgis de type geometry à partir d'une représentation GeoJSON.

La fonction ST_GeomFromGeoJSON fonctionne uniquement avec le fragment JSON représentant la géométrie. Elle renvoie une erreur si un document JSON complet est utilisé.

Amélioré : 3.0.0 La géométrie parsée prend par défaut la valeur SRID=4326 si elle n'est pas spécifiée autrement.

Amélioration : 2.5.0 peut maintenant accepter json et jsonb comme entrées.

Disponibilité : 2.0.0 nécessite JSON-C >= 0.9



Note
Si JSON-C n'est pas disponible sur le système, une erreur est renvoyée. Pour activer JSON-C, lancer configure --with-jsondir=/path/to/json-c. Cf. Section 2.2.3 pour plus de détails.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

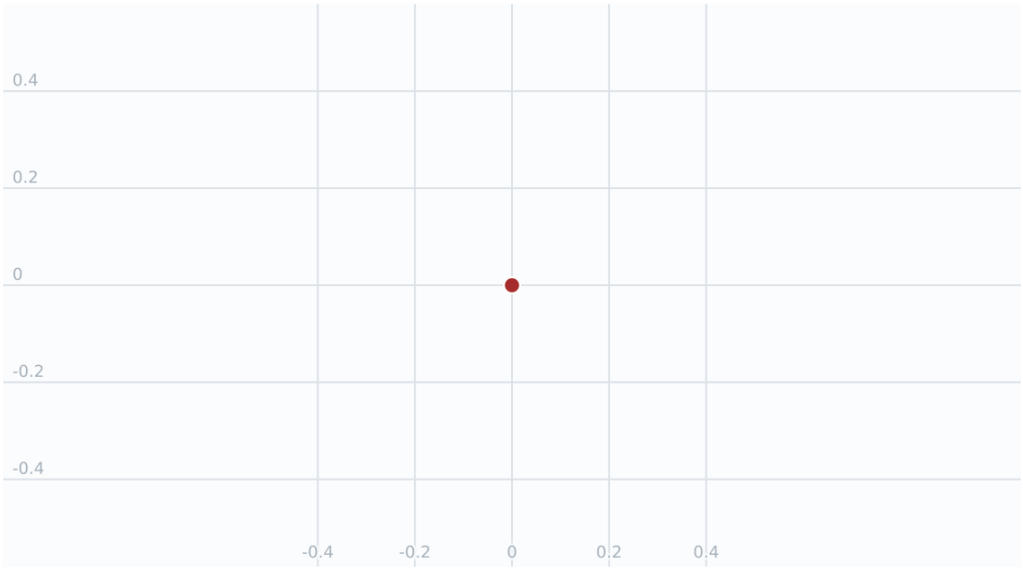
Code

```
SELECT ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}') AS wkt;
```

Output

```
POINT(-48.23456 20.12345)
```

Figure



Output

This example creates a 3D LineString.

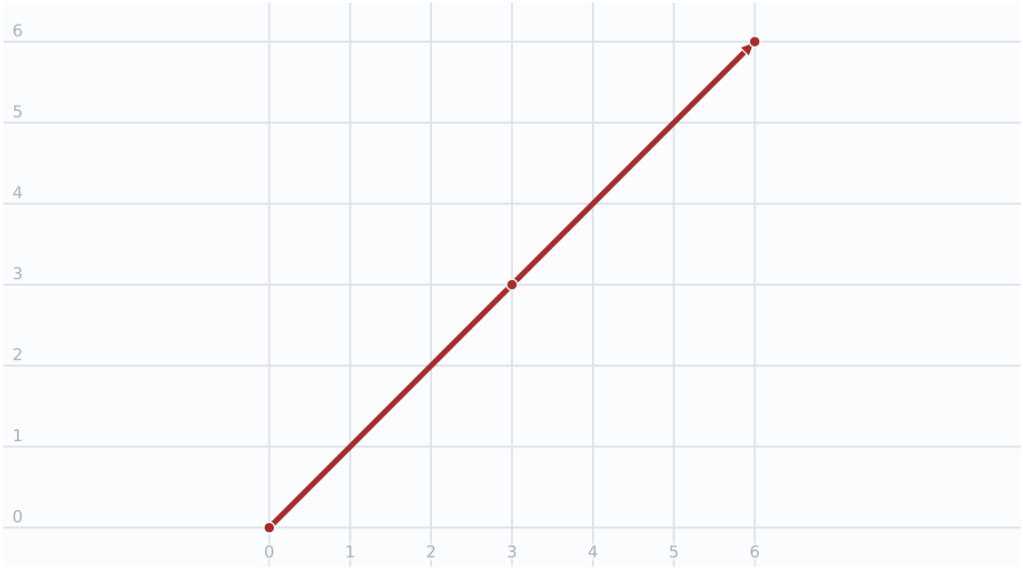
Code

```
SELECT ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],[4,5,6],[7,8,9]]}')  ↩
    AS wkt;
```

Output

```
LINESTRING Z (1 2 3,4 5 6,7 8 9)
```

Figure



Output

Voir aussi

ST_AsText, ST_AsGeoJSON, Section 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — Prend en entrée une géométrie au format KML et renvoie un objet Postgis de type geometry

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

Description

Constructs a PostGIS geometry from an OGC KML representation.

La fonction ST_GeomFromKML fonctionne uniquement avec le fragment KML représentant la géométrie. Elle renvoie une erreur si un document KML complet est utilisé.

versions OGC KML supportées :

— KML 2.2.0 Namespace

OGC KML standards, cf : http://www.opengeospatial.org/standards/kml :

Disponibilité : 1.5, nécessite libxml2 2.6+

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.



Note
ST_GeomFromKML does not support SQL/MM curve geometries.

Exemples

A single LineString geometry.

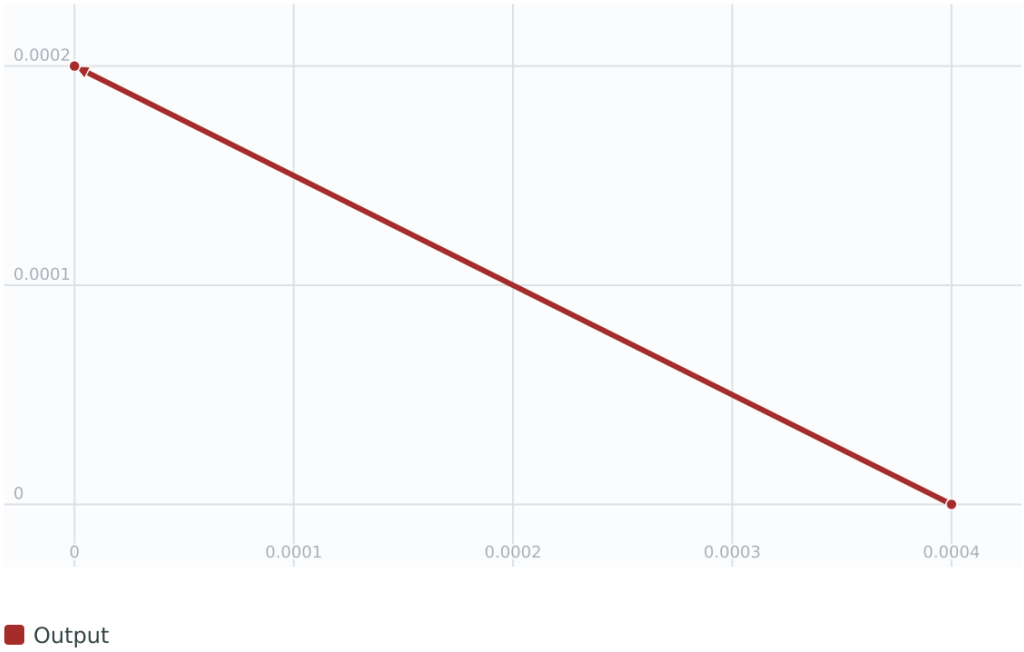
Code

```
        SELECT ST_AsEWKT(ST_GeomFromKML($$
            <LineString>
                <coordinates
>-71.1663, 42.2614
                    -71.1667, 42.2616</coordinates>
            </LineString>
        $$));
```

Output

```
SRID=4326;LINESTRING(-71.1663 42.2614,-71.1667 42.2616)
```

Figure



Voir aussi

Section 2.2.3, ST_AsKML

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — Crée une instance de geometry depuis une représentation de géométrie en TWKB ("Tiny Well-Known Binary").

Synopsis

geometry **ST_GeomFromTWKB**(bytea twkb);

Description

La fonction `ST_GeomFromTWKB` prend une représentation de géométrie TWKB ("Tiny Well-Known Binary") et crée une instance du type de géométrie approprié.

Exemples

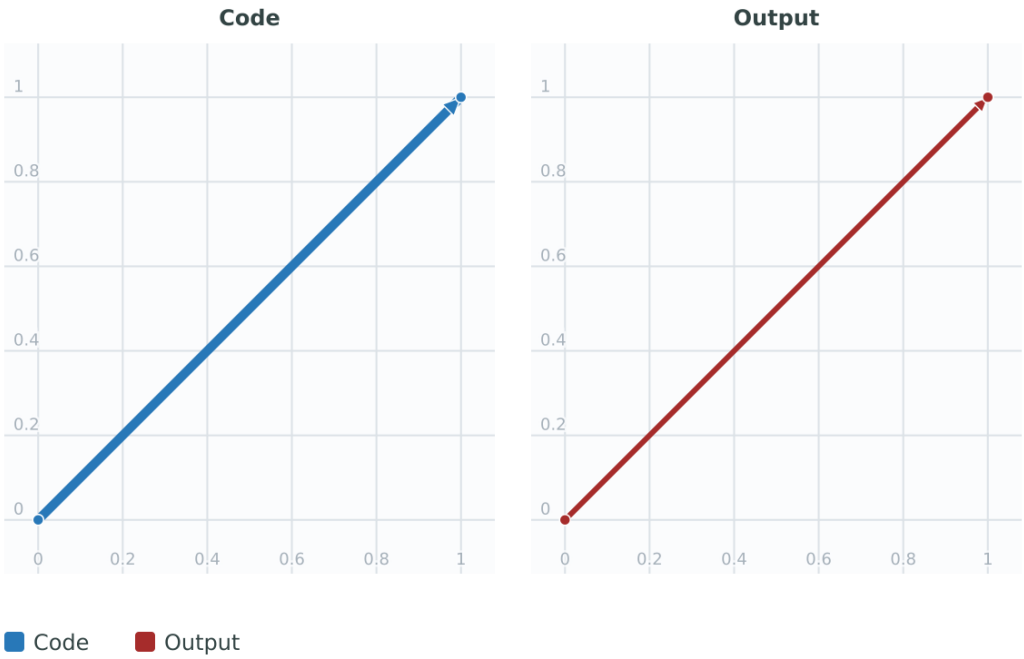
Code

```
SELECT ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34,127 35)'::geometry));
```

Output

```
LINESTRING(126 34, 127 35)
```

Figure



Code

```
SELECT ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105');
```

Output

```
LINESTRING(-113.98 39.198,-113.981 39.195)
```

Figure



Voir aussi

ST_AsTWKB

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — Returns a geometry from GML. Alias for ST_GeomFromGML.

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

Description

This function is an alias for ST_GeomFromGML and provides the same signatures.

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.50.

Disponibilité : 1.5, nécessite libxml2 1.6+

Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques.

Amélioration : 2.0.0 paramètre optionnel de srid par défaut ajouté.

Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced.

Voir aussi

Section 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — Crée une LineString depuis une polyligne encodée ("Encoded Polyline").

Synopsis

geometry **ST_LineFromEncodedPolyline**(text polyline, integer precision=5);

Description

Crée une LineString à partir d'une chaîne de polyligne encodée ("Encoded Polyline").

L'option `precision` spécifie le nombre de décimales qui seront préservées dans la polyligne encodée. La valeur doit être la même à l'encodage et au décodage, sinon les coordonnées seront incorrectes.

Voir http://developers.google.com/maps/documentation/utilities/polylinealgorithm

Disponibilité : 2.2.0

Exemples

This example creates a LineString from a polyline.

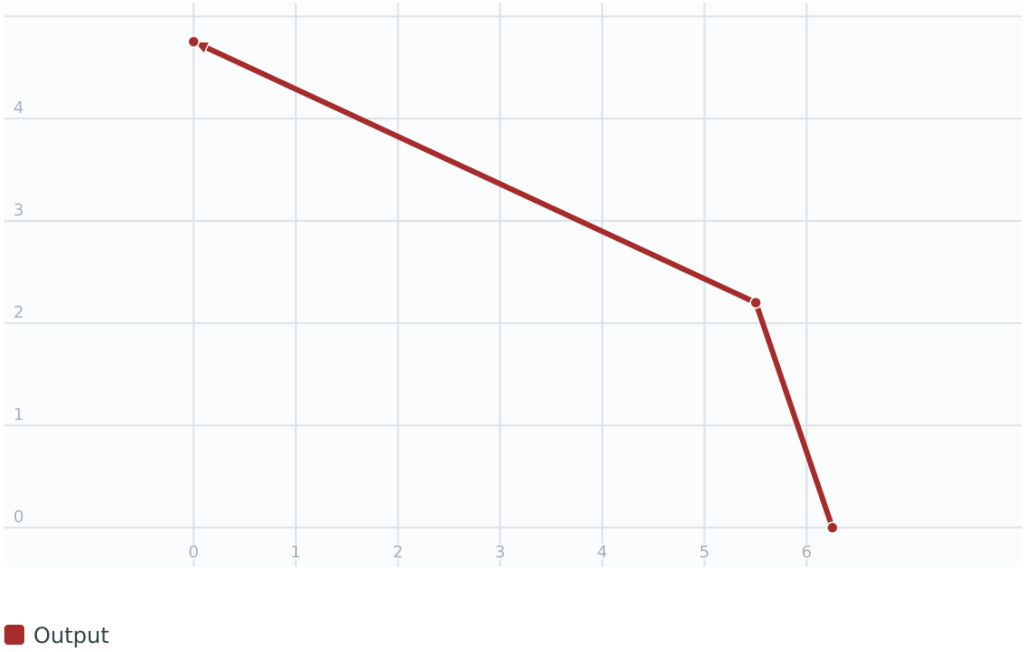
Code

```
SELECT ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@');
```

Output

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Figure



This example selects a different precision than the one used for polyline encoding.

Code

```
SELECT ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@', 6);
```

Output

```
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

Figure



Voir aussi

ST_AsEncodedPolyline

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — Retourne un point à partir d'une chaîne GeoHash.

Synopsis

point **ST_PointFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Description

Retourne une point à partir d'une chaîne GeoHash. Le point représente le centre du GeoHash.

Si aucune `precision` n'est spécifiée, ST_PointFromGeoHash retourne une un point basé sur la précision complète de la chaîne GeoHash en entrée.

Si `precision` est spécifié, ST_PointFromGeoHash utilise autant de caractère du GeoHash pour créer le point.

Disponibilité : 2.1.0

Exemples

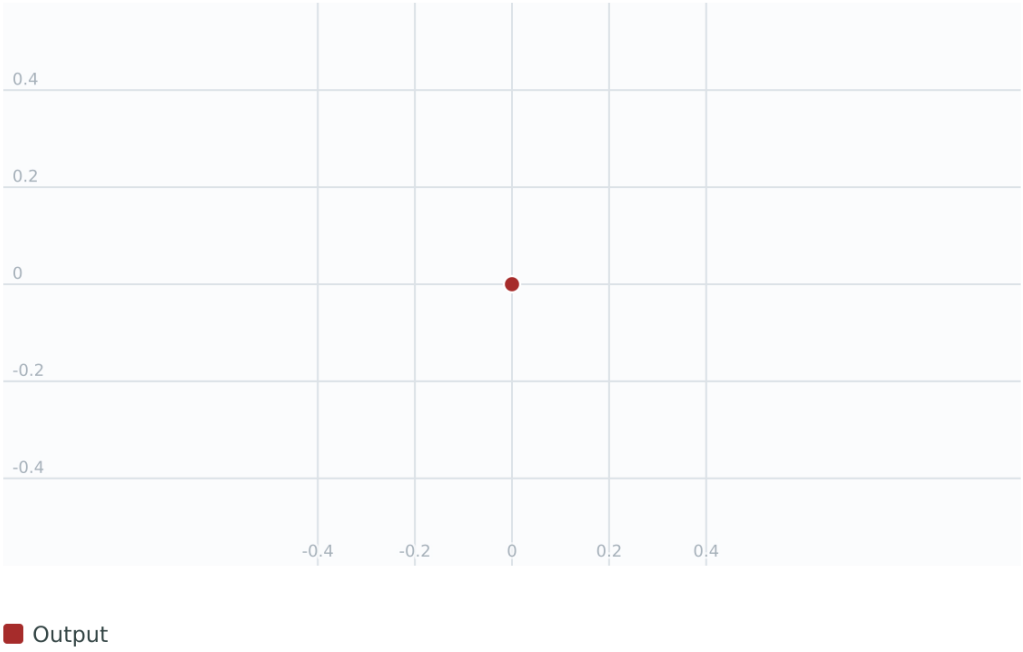
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0');
```

Output

```
POINT(-115.172816 36.114646)
```

Figure



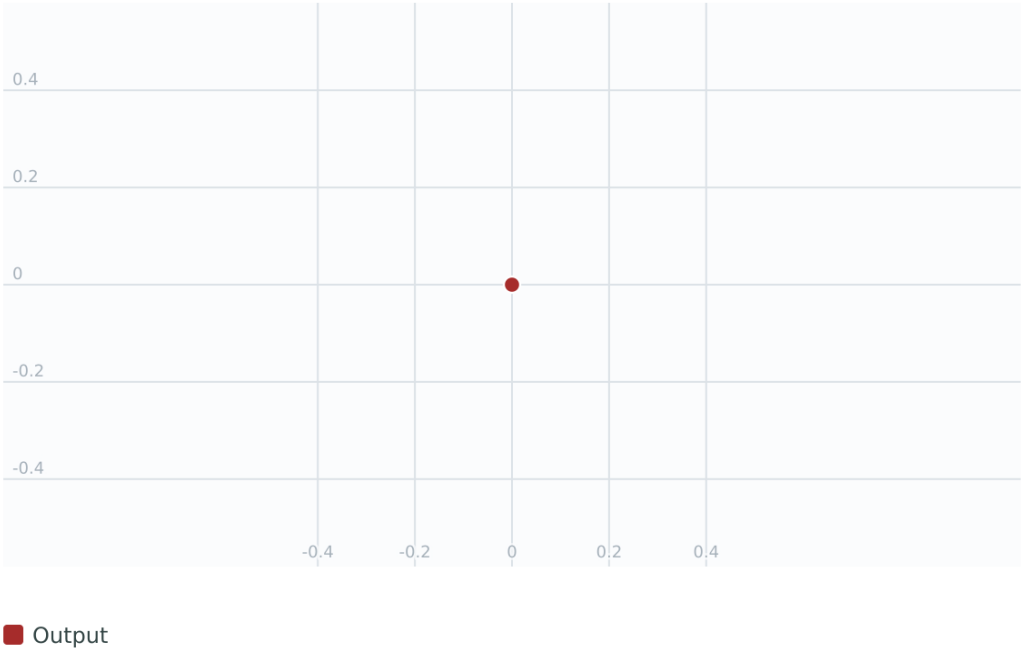
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 4);
```

Output

```
POINT(-115.13671875 36.123046875)
```

Figure



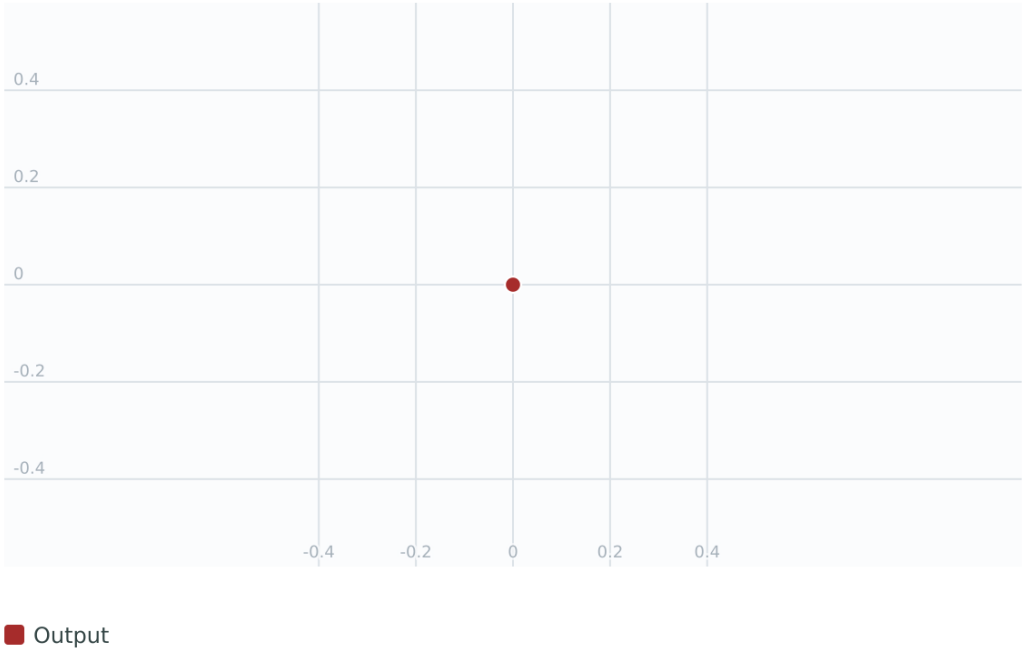
Code

```
SELECT ST_PointFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 10);
```

Output

```
POINT(-115.172815918922 36.1146435141563)
```

Figure



Voir aussi

ST_GeoHash, ST_Box2dFromGeoHash, ST_GeomFromGeoHash

7.8.3.10 ST_FromFlatGeobufToTable

ST_FromFlatGeobufToTable — Crée une table basée sur la structure des données FlatGeobuf.

Synopsis

void **ST_FromFlatGeobufToTable**(text schemaname, text tablename, bytea FlatGeobuf input data);

Description

Crée une table basée sur la structure des données FlatGeobuf. (http://flatgeobuf.org).

`schema` Nom du schéma.

`table` Nom de la table.

`data` Données FlatGeobuf en entrée.

Disponibilité : 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — Lit les données FlatGeobuf.

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

Description

Lit les données FlatGeobuf (http://flatgeobuf.org). REMARQUE : les chaînes binaires de PostgreSQL ne peuvent pas dépasser 1 Go.

`tabletype` référence à un type de table.

`data` données d'entrée FlatGeobuf.

Disponibilité : 3.2.0

7.9 Export de géométrie

7.9.1 Well-Known Text (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.

Synopsis

text **ST_AsEWKT**(geometry g1);
text **ST_AsEWKT**(geometry g1, integer maxdecimaldigits=15);
text **ST_AsEWKT**(geography g1);
text **ST_AsEWKT**(geography g1, integer maxdecimaldigits=15);

Description

Renvoie la représentation Well-Known Text de la géométrie préfixée par le SRID. L'argument facultatif *`maxdecimaldigits`* peut être utilisé pour réduire le nombre maximal de chiffres décimaux après la virgule flottante utilisés dans la sortie (valeur par défaut : 15).

Pour effectuer la conversion inverse de la représentation EWKT en géométrie PostGIS, utilisez ST_GeomFromEWKT.



AVERTISSEMENT
L'utilisation du paramètre *`maxdecimaldigits`* peut rendre la géométrie de sortie invalide. Pour éviter cela, utilisez d'abord ST_ReducePrecision avec une taille de grille appropriée.



Note
La spécification WKT ne comprend pas le SRID. Pour obtenir le format WKT de l'OGC, utilisez ST_AsText.

AVERTISSEMENT
Le format WKT ne maintient pas la précision. Pour éviter la troncature flottante, utilisez le format ST_AsBinary ou ST_AsEWKB pour le transport.

Amélioré : support du paramètre optionnel de précision dans la version 3.1.0.

Amélioration : la version 2.0.0 a introduit la prise en charge de la geography, des surfaces polyédriques, des triangles et des TIN.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

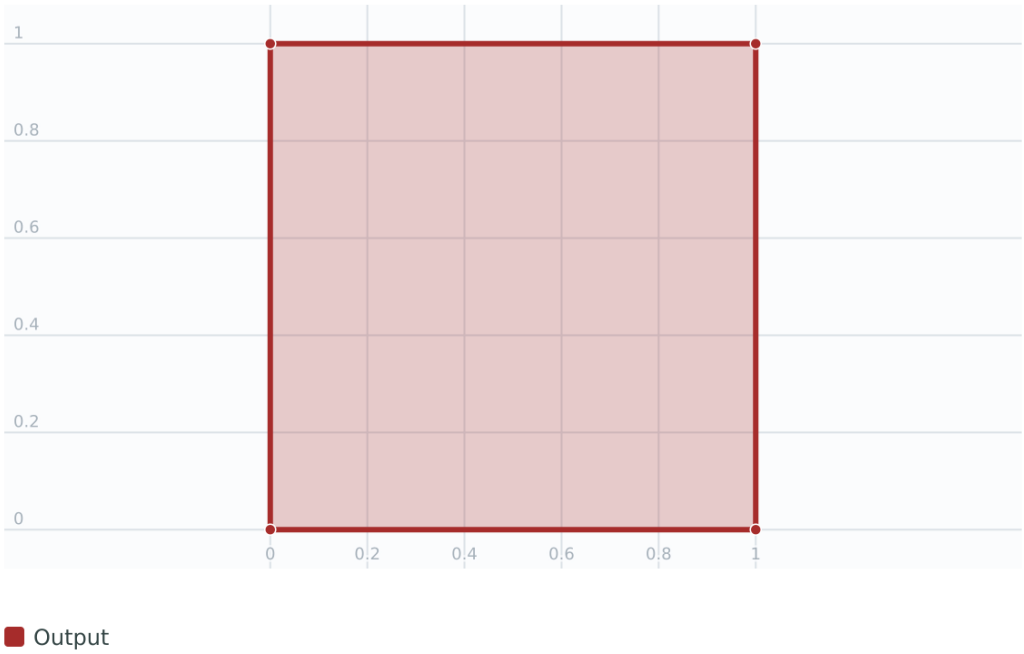
Code

```
SELECT ST_AsEWKT('0103000020 ↩
    e610000001000000050000000000000000000000000000000000000000000000000000000000000000000000000000000000f03f000000000
    '::geometry);
```

Output

```
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



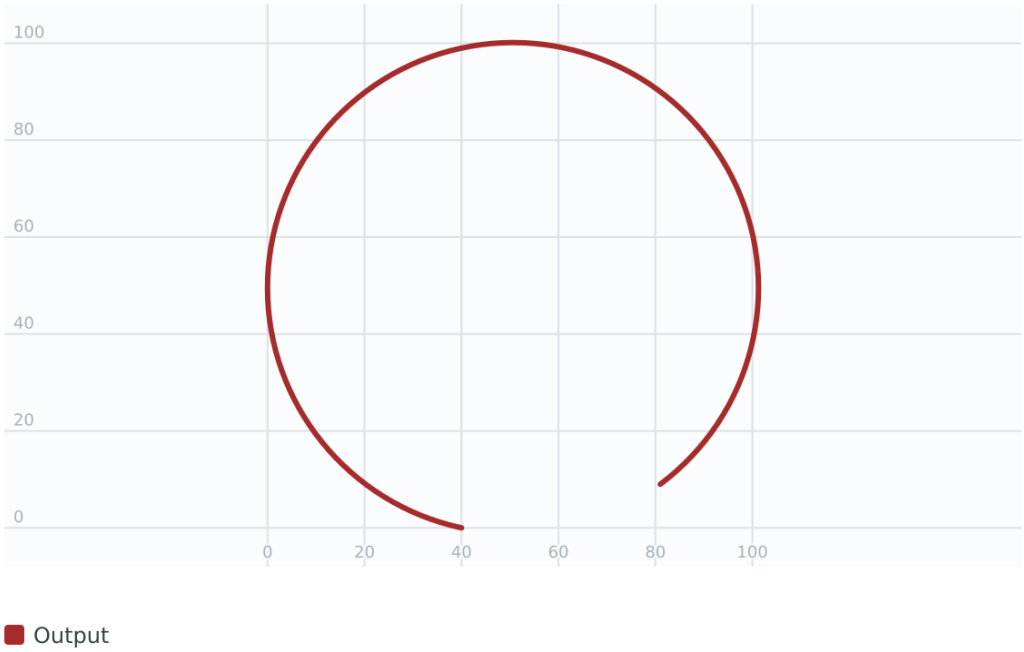
Code

```
SELECT ST_AsEWKT('0108000080030000000000000060 ↩
    E30A4100000000785C0241000000000000F03F0000000018E20A4100000000485F024100000000000000040000000000018
    ');
```

Output

```
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

Figure



Voir aussi

ST_AsBinary, ST_AsEWKB, ST_AsText, ST_GeomFromEWKT

7.9.1.2 ST_AsText

ST_AsText — Renvoie la représentation Well-Known Text (WKT) de la géométrie/geography sans métadonnées SRID.

Synopsis

text **ST_AsText**(geometry g1);
text **ST_AsText**(geometry g1, integer maxdecimaldigits = 15);
text **ST_AsText**(geography g1);
text **ST_AsText**(geography g1, integer maxdecimaldigits = 15);

Description

Renvoie la représentation OGC Well-Known Text (WKT) de la géométrie/geography. L'argument facultatif `maxdecimaldigits` peut être utilisé pour limiter le nombre de chiffres après la virgule dans les ordonnées de sortie (valeur par défaut : 15).

Pour effectuer la conversion inverse de la représentation WKT en géométrie PostGIS, utilisez ST_GeomFromText.



Note
La représentation WKT standard de l'OGC n'inclut pas le SRID. Pour inclure le SRID dans la représentation de sortie, utilisez la fonction PostGIS non standard ST_AsEWKT



AVERTISSEMENT
La représentation textuelle des nombres en WKT peut ne pas maintenir une précision totale en virgule flottante. Pour garantir une précision totale pour le stockage ou le transport des données, il est préférable d'utiliser le format Well-Known Binary (WKB) (voir ST_AsBinary et *maxdecimaldigits*).



AVERTISSEMENT
L'utilisation du paramètre *maxdecimaldigits* peut rendre la géométrie de sortie invalide. Pour éviter cela, utilisez d'abord ST_ReducePrecision avec une taille de grille appropriée.

Disponibilité : 1.5 - le support de la geography a été introduit.

Amélioration : 2.5 - introduction de la précision des paramètres optionnels.


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.25


Cette méthode prend en charge les types Circular String et Curve.

Exemples

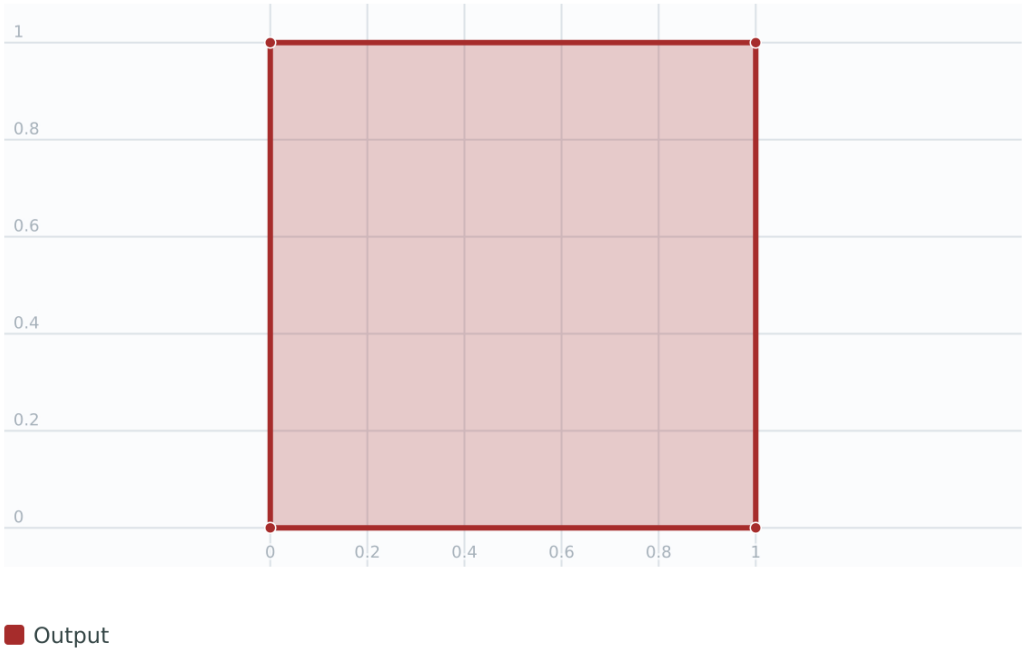
Code

```
SELECT ST_AsText(' ↩
    0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000 ↩
    f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000000000000000000000000000
    ');
```

Output

```
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



La sortie de précision complète est la valeur par défaut.

Code

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'::geometry);
```

Output

```
POINT(111.1111111 1.1111111)
```

L'argument *maxdecimaldigits* peut être utilisé pour limiter la précision de la sortie.

Code

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'::geometry, 2);
```

Output

```
POINT(111.11 1.11)
```

Voir aussi

ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.2 Well-Known Binary (WKB)

7.9.2.1 ST_AsBinary

ST_AsBinary — Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.

Synopsis

bytea **ST_AsBinary**(geometry g1);
bytea **ST_AsBinary**(geometry g1, text NDR_or_XDR);
bytea **ST_AsBinary**(geography g1);
bytea **ST_AsBinary**(geography g1, text NDR_or_XDR);

Description

Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

Le format WKB est utile pour lire les données géométriques de la base de données et maintenir une précision numérique totale. Cela permet d'éviter les arrondis de précision qui peuvent se produire avec les formats texte tels que WKT.

Pour effectuer la conversion inverse de la géométrie WKB en géométrie PostGIS, utilisez ST_GeomFromWKB.



Note
Le format WKB de l'OGC/ISO ne comprend pas le SRID. Pour obtenir le format EWKB qui inclut le SRID, utilisez ST_AsEWKB



Note
PostgreSQL outputs `bytea` values in hex encoding by default. If your GUI tools require the old behavior, then SET `bytea_output='escape'` in your database.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Amélioration : 2.0.0 le support pour des dimensions de coordonnées plus élevées a été introduit.

Amélioration : 2.0.0 le support pour spécifier endian avec geography a été introduit.

Disponibilité : 1.5.0 Le support de la géographie a été introduit.

Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as `ST_AsBinary('POINT(1 2)')` are no longer valid and you will get an `n ST_AsBinary(unknown) is not unique error`. Code like that needs to be changed to `ST_AsBinary('POINT(1 2)'::geometry);`. If that is not possible, then install `legacy.sql`.


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.37


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.


Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
\x0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000
000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000000000
000000000000000000000000
```

Code

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326), 'XDR');
```

Output

```
\x0000000003000000010000000500000000000000000000000000000000000000000000000000000000000000000003ff000
00000000003ff0000000000000003ff0000000000003ff0000000000000000000000000000000000000
000000000000000000000000
```

Use the PostgreSQL `encode` function to produce a hexadecimal WKB string without the `\x` prefix used for bytea display. Specifying `NDR` makes the byte order deterministic.

Code

```
SELECT encode(
    ST_AsBinary('POINT(15.123 21.456)'::geometry, 'NDR'),
    'hex'
) AS wkb_hex;
```

Output

```
0101000000e5d022dbf93e2e40dbf97e6abc743540
```

Voir aussi

ST_GeomFromWKB, ST_AsEWKB, ST_AsTWKB, ST_AsText,

7.9.2.2 ST_AsEWKB

ST_AsEWKB — Renvoie la représentation Extended Well-Known Binary (EWKB) de la géométrie avec les métadonnées SRID.

Synopsis

bytea **ST_AsEWKB**(geometry g1);
bytea **ST_AsEWKB**(geometry g1, text NDR_or_XDR);

Description

Returns the Extended Well-Known Binary (EWKB) representation of the geometry with SRID metadata. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding: either 'NDR' for little-endian; or 'XDR' for big-endian. Supplying unknown arguments will result in little-endian output.

Le format WKB est utile pour lire les données géométriques de la base de données et maintenir une précision numérique totale. Cela permet d'éviter les arrondis de précision qui peuvent se produire avec les formats texte tels que WKT.

Pour effectuer la conversion inverse de la géométrie EWKB en géométrie PostGIS, utilisez ST_GeomFromEWKB.



Note
Pour obtenir le format OGC/ISO WKB, utilisez ST_AsBinary. Notez que le format OGC/ISO WKB ne comprend pas le SRID.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Code

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
\x0103000020e6100000010000000500000000000000000000000000000000000000000000000000000000000000
0000000000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000
0000000000000000000000000000000000
```

Code

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326), 'XDR');
```

Output

```
\x002000000300010e60000000100000005000000000000000000000000000000000000000000000000000000000
003ff0000000000003ff0000000000003ff0000000000003ff000000000000000000000000000
0000000000000000000000000000000000
```

Voir aussi

ST_AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — Renvoie une géométrie au format HEXEWKB (en tant que texte) en utilisant l'encodage little-endian (NDR) ou big-endian (XDR).

Synopsis

text **ST_AsHEXEWKB**(geometry g1, text NDRorXDR);
text **ST_AsHEXEWKB**(geometry g1);

Description

Renvoie une géométrie au format HEXEWKB (en tant que texte) en utilisant l'encodage little-endian (NDR) ou big-endian (XDR). Si aucun encodage n'est spécifié, NDR est utilisé.

Disponibilité : 1.2.2

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
0103000020E6100000010000000500
0000000000000000000000000000000000000000
0000000000000000000000000000000000F03F
000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000000
```

Which gives the same answer as:

Code

```
SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326)::text;
```

7.9.3 Autres formats

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — Renvoie une polyligne encodée à partir d'une géométrie LineString.

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

Description

Returns the geometry in the Encoded Polyline Algorithm Format. The format stores latitude and longitude deltas as integers after multiplying coordinates by 10 raised to the requested `precision`.

The informal names *polyline5* and *polyline6* refer to the same encoding with precision 5 and 6. The precision must match when encoding and decoding; otherwise the coordinates are scaled incorrectly.

The input geometry must use SRID 4326, with longitude as X and latitude as Y. Use ST_Transform to transform geometries in other coordinate systems before encoding.

The OSRM API returns precision 5 for `geometries=polyline` and precision 6 for `geometries=polyline6`. Decode the resulting `route.geometry` with ST_LineFromEncodedPolyline using the matching precision.

Disponibilité : 2.2.0

Exemples

Encode the same WGS 84 LineString using precision 5 and 6.

Code

```
WITH route AS (
  SELECT ST_GeomFromEWKT(
    'SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)'
  ) AS geom
)
SELECT ST_AsEncodedPolyline(geom, 5) AS polyline5,
       ST_AsEncodedPolyline(geom, 6) AS polyline6
FROM route;
```

Output

```
polyline5          |           polyline6
---------------------------+-----------------------------------
 _p~iF~ps|U_ulLnnqC_mqNvxq`@ | _izlhA~rlgdF_{geC~ywl@_kwzCn`{nI
```

Decode an OSRM `route.geometry` returned with `geometries=polyline6`.

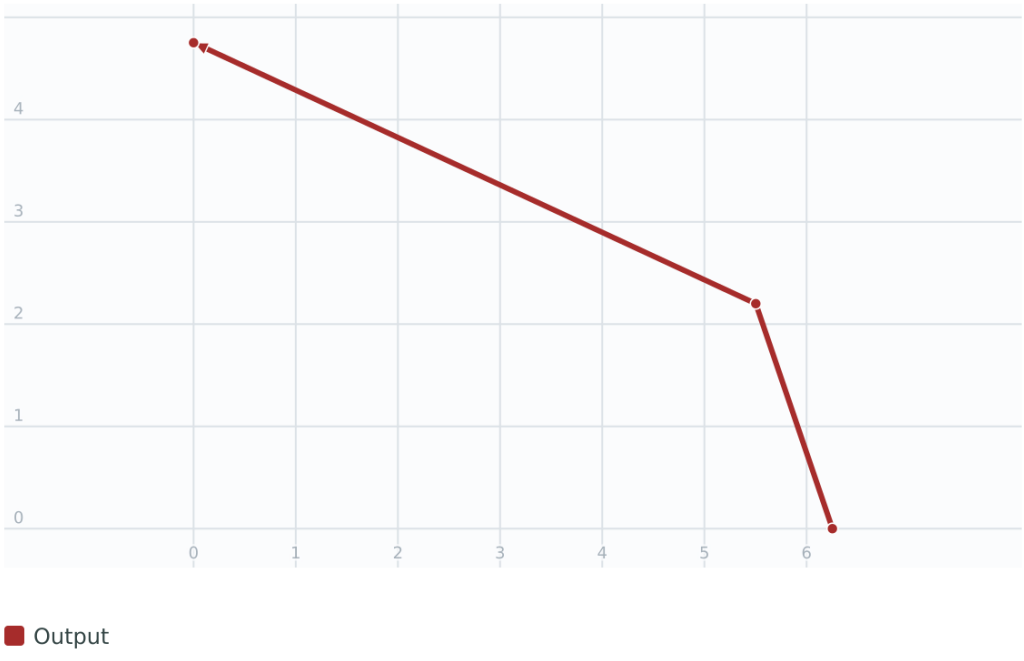
Code

```
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline(
  '_izlhA~rlgdF_{geC~ywl@_kwzCn`{nI',
  6
)) AS route;
```

Output

```
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)
```

Figure



The Google Maps JavaScript Geometry library decodes precision-5 strings. This browser example assumes that `map` is an initialized Google map.

Code

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=geometry"
></script>
<script>
  const encodedPath = "_p~iF~ps|U_ulLnnqC_mqNvxq`@";
  const flightPath = new google.maps.Polyline({
    path: google.maps.geometry.encoding.decodePath(encodedPath),
    map: map,
    strokeColor: "#0000CC",
    strokeOpacity: 1.0,
    strokeWeight: 4
  });
</script>
```

Voir aussi

ST_LineFromEncodedPolyline, ST_Transform

7.9.3.2 ST_AsFlatGeobuf

ST_AsFlatGeobuf — Renvoie une représentation FlatGeobuf d'un ensemble de lignes.

Synopsis

bytea **ST_AsFlatGeobuf**(anyelement set row);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index, text geom_name);

Description

Renvoie une représentation FlatGeobuf (http://flatgeobuf.org) d'un ensemble de lignes correspondant à une FeatureCollection. REMARQUE : les octets PostgreSQL ne peuvent pas dépasser 1 Go.

`row` données de ligne avec au moins une colonne de géométrie.

`index` basculer la création d'un index spatial. La valeur par défaut est false.

`geom_name` est le nom de la colonne de géométrie dans les données de la ligne. Si elle est NULL, elle prendra par défaut la première colonne de géométrie trouvée.

Disponibilité : 3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — Retourne une représentation Geobuf d'un ensemble de lignes.

Synopsis

bytea **ST_AsGeobuf**(anyelement set row);
bytea **ST_AsGeobuf**(anyelement row, text geom_name);

Description

Renvoie une représentation Geobuf (https://github.com/mapbox/geobuf) d'un ensemble de lignes correspondant à une Feature-Collection. Chaque géométrie en entrée est analysée afin de déterminer la précision maximale pour un stockage optimal. Notez que Geobuf dans sa forme actuelle ne peut pas être streamé, donc la sortie complète sera assemblée en mémoire.

`row` données de ligne avec au moins une colonne de géométrie.

`geom_name` est le nom de la colonne de géométrie dans les données de la ligne. Si elle est NULL, elle prendra par défaut la première colonne de géométrie trouvée.

Disponibilité : 2.4.0

Exemples

Code

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
FROM (SELECT 'POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry AS geom) AS q;
```

Output

```
GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

7.9.3.4 ST_AsGeoJSON

ST_AsGeoJSON — Renvoyer une géométrie ou un élément au format GeoJSON.

Synopsis

text **ST_AsGeoJSON**(record feature, text geom_column="", integer maxdecimaldigits=9, boolean pretty_bool=false, text id_column='')
text **ST_AsGeoJSON**(geometry geom, integer maxdecimaldigits=9, integer options=8);
text **ST_AsGeoJSON**(geography geog, integer maxdecimaldigits=9, integer options=0);

Description

Renvoie une géométrie sous forme d'objet GeoJSON "geometry", ou une ligne sous forme d'objet GeoJSON "feature".

Les représentations de géométrie et d'entités GeoJSON résultantes sont conformes aux spécifications GeoJSON RFC 7946, sauf lorsque les géométries analysées sont référencées avec un CRS autre que la longitude et la latitude WGS84 (EPSG:4326, urn:ogc:def:crs:OGC::CRS84) ; l'objet géométrique GeoJSON aura alors un identifiant SRID CRS court attaché par défaut. Les géométries 2D et 3D sont toutes deux supportées. GeoJSON ne supporte que les types de géométrie SFS 1.1 (pas de support pour les courbes par exemple).

Le paramètre `geom_column` est utilisé pour faire la distinction entre plusieurs colonnes géométriques. S'il est omis, la première colonne géométrique de l'enregistrement sera déterminée. Inversement, l'utilisation de ce paramètre permet d'économiser les recherches sur le type de colonne.

L'argument `maxdecimaldigits` peut être utilisé pour réduire le nombre maximum de décimales utilisées dans la sortie (par défaut 9). Si vous utilisez EPSG:4326 et que vous sortez la géométrie uniquement pour l'affichage, `maxdecimaldigits=6` peut être un bon choix pour de nombreuses cartes.



AVERTISSEMENT
L'utilisation du paramètre *`maxdecimaldigits`* peut rendre la géométrie de sortie invalide. Pour éviter cela, utilisez d'abord ST_ReducePrecision avec une taille de grille appropriée.

L'argument `options` peut être utilisé pour ajouter BBOX ou CRS dans la sortie GeoJSON :

— 0 : signifie aucune option
— 1 : GeoJSON BBOX
— 2 : GeoJSON Short CRS (e.g. EPSG:4326)
— 4 : GeoJSON Long CRS (e.g. urn:ogc:def:crs:EPSG::4326)
— 8 : GeoJSON Short CRS si pas EPSG:4326 (par défaut)

Le paramètre `id_column` est utilisé pour définir le membre "id" des caractéristiques GeoJSON renvoyées. Conformément à la RFC GeoJSON, ce paramètre DEVRAIT être utilisé chaque fois qu'un élément a un identifiant couramment utilisé, tel qu'une clé primaire. Si elle n'est pas spécifiée, les entités produites n'auront pas de membre "id" et toutes les colonnes autres que la géométrie, y compris les clés potentielles, se retrouveront dans le membre "properties" de l'entité.

When the input record includes duplicate column names, the resulting "properties" object will contain repeated keys. `ST_AsGeoJSON` raises a warning in that case because PostgreSQL jsonb keeps only the last value for duplicate keys.

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using ST_ForcePolygonCCW. The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using ST_Transform: `ST_Transform(geom, 4326 )`.

GeoJSON peut être testé et visualisé en ligne sur geojson.io et geojsonlint.com. Il est largement pris en charge par les frameworks de cartographie web :

— Exemple OpenLayers GeoJSON
— Exemple Leaflet GeoJSON
— Exemple Mapbox GL GeoJSON

Disponibilité : 1.3.4

Disponibilité : 1.5.0 Le support de la géographie a été introduit.

Modifié : 2.0.0 supporte les args par défaut et les args nommés.

Modifié : la version 3.0.0 prend en charge les enregistrements en tant que données d'entrée

Modifié : 3.0.0 SRID de sortie si ce n'est pas EPSG:4326.

Modifié : la version 3.5.0 permet de spécifier la colonne contenant l'identifiant de l'élément

Changed: 3.7.0 added warning about duplicate keys

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Générer une FeatureCollection :

Code

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*, id_column =
> 'id')::json)
    )
FROM ( VALUES (1, 'one', 'POINT(1 1)'::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
     ) as t(id, name, geom);
```

Output

```
{"type" : "FeatureCollection", "features" : [{"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "id ↩
    ": 1, "properties": {"name": "one"}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[2,2]}, "id": ↩
    2, "properties": {"name": "two"}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[3,3]}, "id": 3, ↩
    "properties": {"name": "three"}}]}
```

Génère une Feature :

Code

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

Output

```
{"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "id": 1, "properties": {"name": "one"}}
```

N'oubliez pas de transformer vos données en longitude et latitude WGS84 pour vous conformer à la spécification GeoJSON :

Code

```
SELECT ST_AsGeoJSON(ST_Transform(geom, 4326)) FROM fe_edges LIMIT 1;
```

Output

```
{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
```

Les géométries 3D sont prises en charge :

Code

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3,4 5 6)');
```

Output

```
{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

L'argument Options peut être utilisé pour ajouter BBOX et CRS dans la sortie GeoJSON :

Code

```
SELECT ST_AsGeoJSON(ST_SetSRID('POINT(1 1)'::geometry, 4326), 9, 4|1);
```

Output

```
{"type":"Point","crs":{"type":"name","properties":{"name":"urn:ogc:def:crs:EPSG::4326"}},"bbox ↩
    ":[1.000000000,1.000000000,1.000000000,1.000000000],"coordinates":[1,1]}
```

Voir aussi

ST_GeomFromGeoJSON, ST_ForcePolygonCCW, ST_Transform

7.9.3.5 ST_AsGML

ST_AsGML — Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.

Synopsis

text **ST_AsGML**(geometry geom, integer maxdecimaldigits=15, integer options=0);
text **ST_AsGML**(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text **ST_AsGML**(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
text **ST_AsGML**(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);

Description

Renvoie la géométrie sous la forme d'un élément GML (Geography Markup Language). Le paramètre de version, s'il est spécifié, peut être 2 ou 3. Si aucun paramètre de version n'est spécifié, la valeur par défaut est 2. L'argument `maxdecimaldigits` peut être utilisé pour réduire le nombre maximum de décimales utilisées dans la sortie (15 par défaut).



AVERTISSEMENT
L'utilisation du paramètre *`maxdecimaldigits`* peut rendre la géométrie de sortie invalide. Pour éviter cela, utilisez d'abord ST_ReducePrecision avec une taille de grille appropriée.

GML 2 fait référence à la version 2.1.2, GML 3 à la version 3.1.1

L'argument "options" est un champ de bits. Il peut être utilisé pour définir le type de sortie CRS dans la sortie GML, et pour déclarer les données comme lat/lon :

— 0 : GML Short CRS (e.g. EPSG:4326), valeur par défaut
— 1: GML Long CRS (e.g urn:ogc:def:crs:EPSG::4326)
— 2 : Pour GML 3 uniquement, supprimer l'attribut srsDimension de la sortie.
— 4 : Pour GML 3 uniquement, utilisez la balise <LineString> plutôt que la balise <Curve> pour les lignes.
— 16 : Déclare que les données sont des lat/lon (par exemple srid=4326). Par défaut, on suppose que les données sont planaires. Cette option n'est utile que pour la sortie GML 3.1.1, en ce qui concerne l'ordre des axes. Ainsi, si vous la définissez, les coordonnées seront échangées pour que l'ordre soit lat lon au lieu de lon lat dans la base de données.
— 32 : Sortie de la boîte de la géométrie (enveloppe).

L'argument 'namespace prefix' peut être utilisé pour spécifier un préfixe d'espace de noms personnalisé ou aucun préfixe (s'il est vide). Si l'argument est nul ou omis, le préfixe 'gml' est utilisé

Disponibilité : 1.3.2

Disponibilité : 1.5.0 Le support de la géographie a été introduit.

Amélioration : la prise en charge du préfixe 2.0.0 a été introduite. L'option 4 pour GML3 a été introduite pour permettre l'utilisation de LineString au lieu de Curve tag pour les lignes. La prise en charge GML3 des surfaces polyédriques et des TINS a été introduite. L'option 32 a été introduite pour produire la boîte.

Modifié : 2.0.0 utiliser les args nommés par défaut

Amélioration : 2.1.0 La prise en charge des identifiants a été introduite pour GML 3.



Note
Seule la version 3+ de ST_AsGML prend en charge les surfaces polyédriques et les TINS.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 17.2

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Version 2.

Code

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
<gml:Polygon srsName="EPSG:4326"
><gml:outerBoundaryIs
><gml:LinearRing
><gml:coordinates
>0,0 0,1 1,1 1,0 0,0</gml:coordinates
></gml:LinearRing
></gml:outerBoundaryIs
></gml:Polygon
>
```

Version 3.

Flip coordinates and output extended EPSG (16 | 1)

Code

```
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)', 4326), 5, 17);
```

Output

```
<gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos srsDimension="2"
>6.34535 5.23423</gml:pos
></gml:Point
>
```

Output the envelope (32).

Code

```
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
```

Output

```
<gml:Envelope srsName="EPSG:4326" srsDimension="2"
><gml:lowerCorner
>1 2</gml:lowerCorner
><gml:upperCorner
>10 20</gml:upperCorner
></gml:Envelope
>
```

Output the envelope (32), reverse (lat lon instead of lon lat) (16), long srs (1) = 32 | 16 | 1 = 49.

Code

```
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
```

Output

```
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326" srsDimension="2"
><gml:lowerCorner
>2 1</gml:lowerCorner
><gml:upperCorner
>20 10</gml:upperCorner
></gml:Envelope
>
```

This example uses a PolyhedralSurface.

Code

```
SELECT ST_AsGML(
3,
'POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )'::geometry);
```

Output

```
 <gml:PolyhedralSurface>
<gml:polygonPatches>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing>
           <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
        </gml:LinearRing>
    </gml:exterior>
   </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>
```

Voir aussi

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — Renvoyer la géométrie sous forme d'élément KML.

Synopsis

text **ST_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text **ST_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

Description

Renvoie la géométrie sous la forme d'un élément KML (Keyhole Markup Language). Le nombre maximal de décimales par défaut est de 15, la valeur par défaut du namespace est sans préfixe.



AVERTISSEMENT
L'utilisation du paramètre *maxdecimaldigits* peut rendre la géométrie de sortie invalide. Pour éviter cela, utilisez d'abord ST_ReducePrecision avec une taille de grille appropriée.

Disponibilité : 1.2.2 - les variantes ultérieures qui incluent le paramétrage de la version sont disponibles dans la version 1.3.2

Amélioré : 2.0.0 - Ajout d'un préfixe namespace, utilisation d'arguments par défaut et d'arguments nommés

Modifié : 3.0.0 - Suppression de la signature de la variante "versioned"



Note
Exige que PostGIS soit compilé avec le support Proj. Utilisez PostGIS_Full_Version pour confirmer que vous avez compilé le support Proj.



Note
La sortie AsKML ne fonctionnera pas avec les géométries qui n'ont pas de SRID


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

Output

```
<Polygon
><outerBoundaryIs
><LinearRing
><coordinates
>0,0 0,1 1,1 1,0 0,0</coordinates
></LinearRing
></outerBoundaryIs
></Polygon
>
```

This example outputs a 3D LineString.

Code

```
SELECT ST_AsKML('SRID=4326;LINESTRING(1 2 3,4 5 6)');
```

Output

```
<LineString
><coordinates
>1,2,3 4,5,6</coordinates
></LineString
>
```

Voir aussi

ST_AsSVG, ST_AsGML

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — Renvoie la représentation en degrés, minutes et secondes du point donné.

Synopsis

text **ST_AsLatLonText**(geometry pt, text format='');

Description

Renvoie la représentation en degrés, minutes et secondes du point.



Note
On suppose que le point se trouve dans une projection lat/lon. Les coordonnées X (lon) et Y (lat) sont normalisées dans la sortie à la plage "normale" (-180 à +180 pour lon, -90 à +90 pour lat).

Le paramètre text est une chaîne de caractères contenant le format du texte résultant, similaire à une chaîne de caractères de date. Les symboles autorisés sont "D" pour les degrés, "M" pour les minutes, "S" pour les secondes et "C" pour la direction cardinale (NSEW). Les jetons DMS peuvent être répétés pour indiquer la largeur et la précision souhaitées ("SSS.SSSS" signifie "1,0023").

Les lettres "M", "S" et "C" sont facultatives. Si "C" est omis, les degrés sont indiqués avec un signe "-" s'il s'agit du sud ou de l'ouest. Si "S" est omis, les minutes sont affichées sous forme décimale avec autant de chiffres de précision que vous le spécifiez. Si "M" est également omis, les degrés sont affichés sous forme décimale avec autant de chiffres de précision que vous le spécifiez.

Si la chaîne de caractères format est omise (ou de longueur nulle), un format par défaut sera utilisé.

Disponibilité : 2.0

Exemples

Format par défaut.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
```

Output

```
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Fournir un format (identique au format par défaut).

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M''S.SSS"C'));
```

Output

```
2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Les caractères autres que D, M, S, C et . ne sont pas pris en compte.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D degrees, M minutes, S seconds to ↩
    the C'));
```

Output

```
2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W
```

Les degrés signés au lieu des directions cardinales.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D\textdegree{}M''S.SSS"'));
```

Output

```
-2\textdegree{}19'29.928" -3\textdegree{}14'3.243"
```

Degrés décimaux.

Code

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
```

Output

```
2.3250 degrees S 3.2342 degrees W
```

To convert parsed degree, minute, and second components to decimal degrees, apply the cardinal-direction sign explicitly.

Code

```
WITH dms(label, degrees, minutes, seconds, hemisphere) AS (
  VALUES
    ('latitude', 43::numeric, 0::numeric, 50.60::numeric, 'S'),
    ('longitude', 147::numeric, 12::numeric, 18.20::numeric, 'E')
)
SELECT label,
       round(
         (CASE WHEN hemisphere IN ('S', 'W') THEN -1 ELSE 1 END)
         * (degrees + minutes / 60 + seconds / 3600),
         9
       ) AS decimal_degrees
FROM dms;
```

Output

```
   label    | decimal_degrees
------------+-----------------
 latitude   |   -43.014055556
 longitude  |   147.205055556
```

Les valeurs trop importantes sont normalisées.

Code

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
```

Output

```
72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — Renvoie la géométrie sous forme d'enregistrement MARC21/XML avec un champ de données géographiques (034).

Synopsis

text **ST_AsMARC21** (geometry geom , text format='hdddmmss');

Description

Cette fonction renvoie un enregistrement MARC21/XML avec Coded Cartographic Mathematical Data représentant la boîte de délimitation d'une géométrie donnée. Le paramètre `format` permet de coder les coordonnées dans les sous-champs `$d`,`$e`,`$f` et `$g` dans tous les formats pris en charge par la norme MARC21/XML. Les formats valides sont les suivants :

— la direction cardinale, en degrés, minutes et secondes (par défaut) : `hdddmmss`
— degrés décimaux avec la direction cardinale : `hddd.dddddd`
— degrés décimaux sans direction cardinale : `ddd.dddddd`
— minutes décimales avec la direction cardinale : `hdddmm.mmmm`
— minutes décimales sans direction cardinale : `dddmm.mmmm`
— secondes décimales avec la direction cardinale : `hdddmmss.sss`

Le signe décimal peut également être une virgule, par exemple `hdddmm,mmmm`.

La précision des formats décimaux peut être limitée par le nombre de caractères après le signe décimal, par exemple `hdddmm.mm` pour des minutes décimales avec une précision de deux décimales.

Cette fonction ignore les dimensions Z et M.

Prise en charge des versions LOC MARC21/XML :
— MARC21/XML 1.1
Disponibilité : 3.3.0



Note
Cette fonction ne prend pas en charge les géométries non lon/lat, car elles ne sont pas prises en charge par la norme MARC21/XML (Coded Cartographic Mathematical Data).



Note
La norme MARC21/XML ne prévoit aucun moyen d'annoter le système de référence spatiale pour les données mathématiques cartographiques codées, ce qui signifie que cette information sera perdue après la conversion au format MARC21/XML.

Exemples

Conversion d'un `POINT` en MARC21/XML au format hdddmmss (par défaut)

Code

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)'::geometry);
```

Output

```
                              st_asmarc21
-----------------------------------------------------
<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W0043015</subfield>
        <subfield code="e"
>W0043015</subfield>
        <subfield code="f"
>N0541512</subfield>
        <subfield code="g"
>N0541512</subfield>
    </datafield>
</record>
```

Conversion d'un `POLYGON` en MARC21/XML formaté en degrés décimaux

Code

```
SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625 ↩
    54.18172660239091,-4.56756591796875 54.196993557130355,-4.546623229980469 ↩
    54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,'hddd.dddd');

<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W004.5792</subfield>
        <subfield code="e"
>W004.5466</subfield>
        <subfield code="f"
>N054.1970</subfield>
        <subfield code="g"
>N054.1817</subfield>
    </datafield>
</record>
```

Conversion d'une `GEOMETRYCOLLECTION` en MARC21/XML en minutes décimales. L'ordre des géométries dans la sortie MARC21/XML correspond à leur ordre dans la collection.

Code

```
SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667 ↩
    52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1 52.65)),POINT ↩
    (-4.5 54.25))'::geometry,'hdddmm.mmmm');
```

Output

```
                                st_asmarc21
--------------------------------------------------
<record xmlns="http://www.loc.gov/MARC21/slim">
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>E01307.0000</subfield>
        <subfield code="e"
>E01331.0000</subfield>
        <subfield code="f"
>N05240.0000</subfield>
        <subfield code="g"
>N05224.0000</subfield>
    </datafield>
    <datafield tag="034" ind1="1" ind2=" ">
        <subfield code="a"
>a</subfield>
        <subfield code="d"
>W00430.0000</subfield>
        <subfield code="e"
>W00430.0000</subfield>
        <subfield code="f"
>N05415.0000</subfield>
        <subfield code="g"
>N05415.0000</subfield>
    </datafield>
</record>
```

Voir aussi

ST_GeomFromMARC21

7.9.3.9 ST_AsMVTGeom

ST_AsMVTGeom — Transforme une géométrie dans l'espace de coordonnées d'une tuile MVT.

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

Description

Transforme une géométrie dans l'espace de coordonnées d'une tuile MVT (Mapbox Vector Tile), en l'ajustant aux limites de la tuile si nécessaire. La géométrie doit être dans le système de coordonnées de la carte cible (en utilisant ST_Transform si nécessaire). Il s'agit généralement de Web Mercator (SRID:3857).

La fonction tente de préserver la validité de la géométrie et la corrige si nécessaire. Cela peut entraîner l'effondrement de la géométrie résultante à une dimension inférieure.

The function performs geometry simplification appropriate to the integer tile coordinate space. After transforming to tile coordinates it snaps coordinates to the integer grid, removes repeated points, and removes points that lie on straight lines. It does not perform general-purpose, scale-dependent simplification such as ST_Simplify. To reduce detail before tile conversion, simplify the geometry in map coordinates before calling `ST_AsMVTGeom`.

Les limites rectangulaires de la tuile dans l'espace de coordonnées de la carte cible doivent être fournies, afin que la géométrie puisse être transformée et coupée si nécessaire. Les limites peuvent être générées à l'aide de ST_TileEnvelope.

Cette fonction est utilisée pour convertir la géométrie dans l'espace de coordonnées de la tuile requis par ST_AsMVT.

`geom` est la géométrie à transformer, dans le système de coordonnées de la carte cible.

`bounds` est la limite rectangulaire de la tuile dans l'espace de coordonnées de la carte, sans tampon.

`extent` est la taille de l'étendue de la tuile dans l'espace de coordonnées de la tuile tel que défini par la spécification MVT. La valeur par défaut est 4096.

`buffer` est la taille du tampon dans l'espace de coordonnées de la tuile pour le découpage de la géométrie. La valeur par défaut est 256.

`clip_geom` est un booléen qui contrôle si les géométries sont découpées ou encodées telles quelles. La valeur par défaut est true.

Disponibilité : 2.4.0



Note
A partir de la version 3.0, Wagyu peut être choisi au moment de la configuration pour découper et valider les polygones MVT. Cette bibliothèque est plus rapide et produit des résultats plus corrects que la bibliothèque par défaut de GEOS, mais elle peut mettre de côté de petits polygones.

Exemples

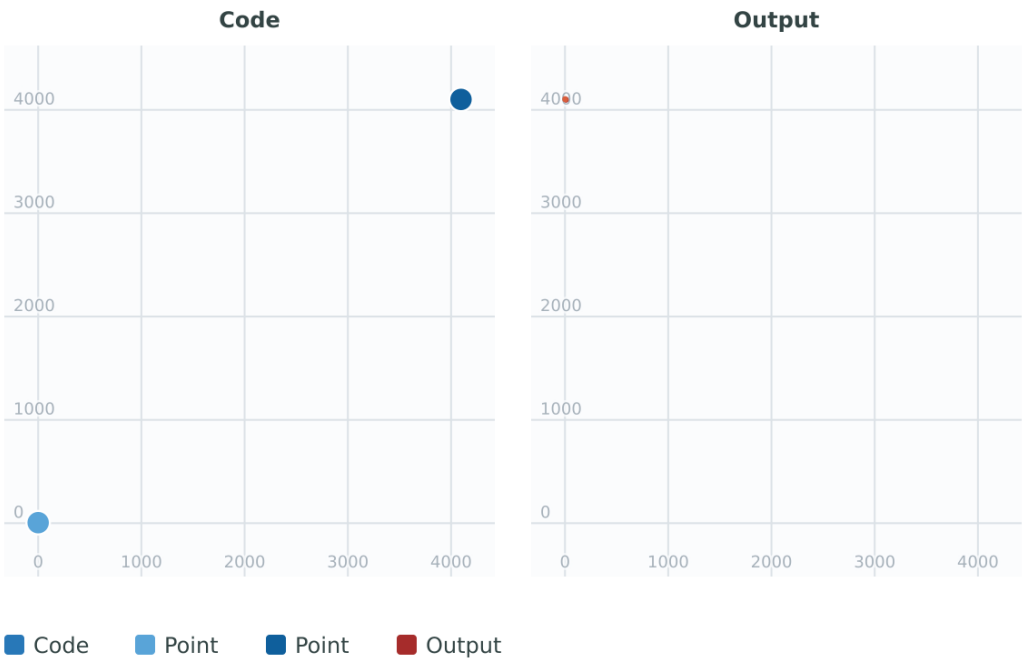
Code

```
SELECT ST_AsMVTGeom('POLYGON ((0 0,10 0,10 5,0 -5,0 0))',
    ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
    4096, 0, false);
```

Output

```
MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Figure



Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry. This assumes the data.geom column has srid of 4326.

Code

```
SELECT ST_AsMVTGeom(ST_Transform(geom, 3857 ),
            ST_TileEnvelope(12, 513, 412),
            extent =
```

```
> 4096,
            buffer =
> 64) AS geom
  FROM data
  WHERE geom && ST_Transform(ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096)), 4326)
```

Voir aussi

ST_AsMVT, ST_TileEnvelope, PostGIS_Wagyu_Version

7.9.3.10 ST_AsMVT

ST_AsMVT — Fonction d'agrégation renvoyant une représentation MVT d'un ensemble de lignes.

Synopsis

bytea **ST_AsMVT**(anyelement set row);
bytea **ST_AsMVT**(anyelement row, text name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name, text feature_id_name);

Description

Une fonction agrégée qui renvoie une représentation binaire Mapbox Vector Tile d'un ensemble de lignes correspondant à une couche de tuiles. Les lignes doivent contenir une colonne de géométrie qui sera encodée comme une géométrie d'élément. La géométrie doit être dans l'espace de coordonnées de la tuile et valide conformément à la spécification MVT. ST_AsMVTGeom peut être utilisé pour transformer la géométrie dans l'espace de coordonnées des tuiles. Les autres colonnes de la ligne sont encodées en tant qu'attributs de caractéristiques.

Le format Mapbox Vector Tile peut stocker des éléments avec différents ensembles d'attributs. Pour utiliser cette fonctionnalité, il convient de fournir une colonne JSONB dans les données de la ligne contenant des objets Json à un niveau de profondeur. Les clés et les valeurs des valeurs JSONB seront encodées en tant qu'attributs d'entités.

Les tuiles à couches multiples peuvent être créées en concaténant plusieurs appels à cette fonction en utilisant `||` ou `STRING_AGG`.



Important
Ne pas appeler avec un `GEOMETRYCOLLECTION` comme élément de la ligne. Cependant, vous pouvez utiliser ST_AsMVTGeom pour préparer une collection de géométrie à inclure.

`row` données de ligne avec au moins une colonne de géométrie.

`name` est le nom de la couche. La valeur par défaut est la chaîne de caractères "default".

`extent` est l'étendue de la tuile dans l'espace de l'écran tel que défini par la spécification. La valeur par défaut est 4096.

`geom_name` est le nom de la colonne géométrique dans les données de la ligne. La valeur par défaut est la première colonne géométrique. Notez que PostgreSQL, par défaut, plie automatiquement les identifiants non cités en minuscules, ce qui signifie qu'à moins que la colonne géométrique ne soit citée, par exemple `"MyMVTGeom"`, ce paramètre doit être fourni en minuscules.

`feature_id_name` est le nom de la colonne Feature ID dans les données de la ligne. S'il est NULL ou négatif, l'identifiant de la caractéristique n'est pas défini. La première colonne correspondant au nom et au type valide (smallint, integer, bigint) sera

utilisée comme Feature ID, et toute colonne suivante sera ajoutée en tant que propriété. Les propriétés JSON ne sont pas prises en charge.

Amélioration : 3.0 - ajout de la prise en charge du Feature ID.

Amélioration : 2.5.0 - ajout de la prise en charge des requêtes parallèles.

Disponibilité : 2.4.0

Exemples

Code

```
WITH mvtgeom AS
(
  SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom, name, description
  FROM points_of_interest
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```

Voir aussi

ST_AsMVTGeom, ST_TileEnvelope

7.9.3.11 ST_AsSVG

ST_AsSVG — Renvoie les données de chemin SVG pour une géométrie.

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

Description

Renvoie la géométrie sous forme de données de chemin SVG (Scalar Vector Graphics). Utilisez 1 comme deuxième argument pour que les données de chemin soient implémentées en termes de déplacements relatifs, la valeur par défaut (ou 0) utilise des déplacements absolus. Le troisième argument peut être utilisé pour réduire le nombre maximal de chiffres décimaux utilisés dans la sortie (15 par défaut). Les géométries ponctuelles seront rendues sous la forme cx/cy lorsque l'argument 'rel' est 0, x/y lorsque 'rel' est 1. Les géométries multipoints sont délimitées par des virgules (","), les géométries GeometryCollection sont délimitées par des points-virgules (";").

Pour travailler avec les graphiques PostGIS SVG, consultez la bibliothèque pg_svg qui fournit des fonctions plpgsql pour travailler avec les résultats de ST_AsSVG.

Amélioration : 3.4.0 pour prendre en charge tous les types de courbes

Modifié : 2.0.0 pour utiliser les args par défaut et supporter les args nommés

Disponibilité : 1.2.2. Disponibilité : 1.4.0 Modifié dans PostGIS 1.4.0 pour inclure la commande L dans le chemin absolu afin de se conformer à http://www.w3.org/TR/SVG/paths.html#PathDataBNF

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

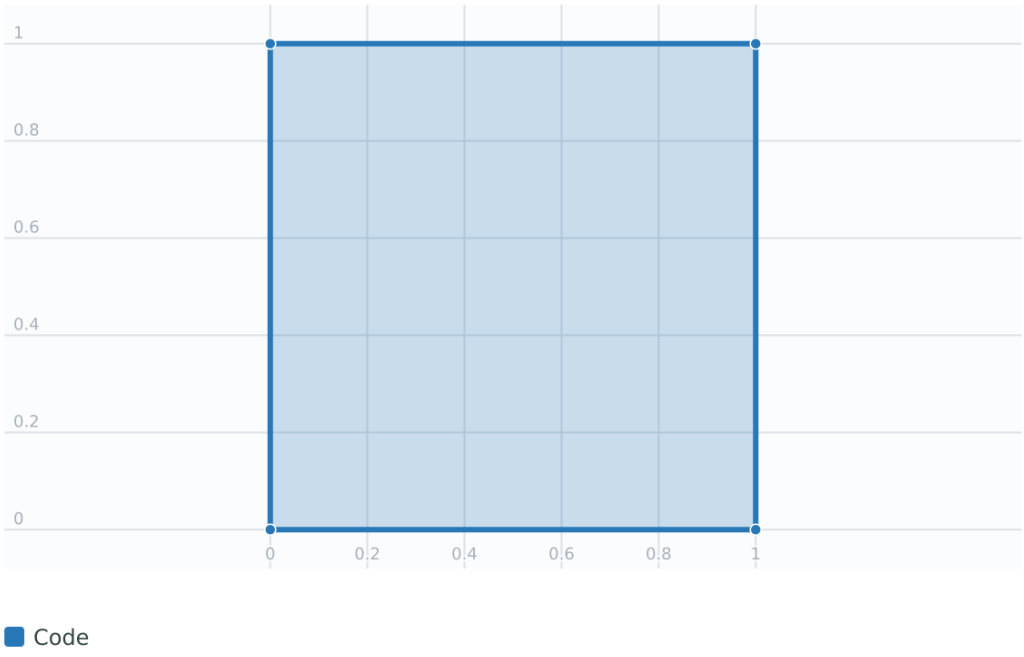
Code

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);
```

Output

```
M 0 0 L 0 -1 1 -1 1 0 Z
```

Figure



Circular string

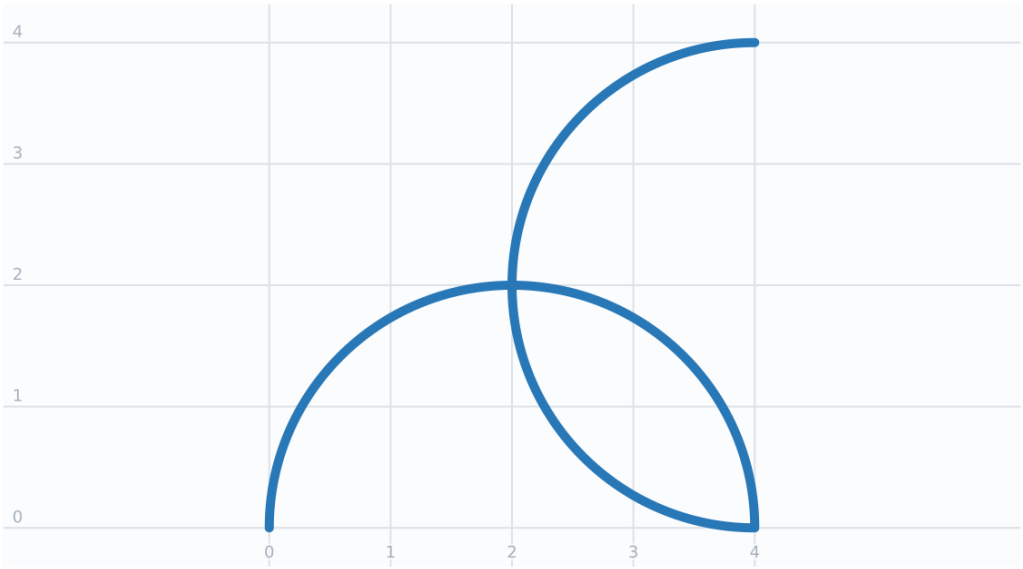
Code

```
SELECT ST_AsSVG('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)'::geometry);
```

Output

```
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

Figure



Code

Multi-curve

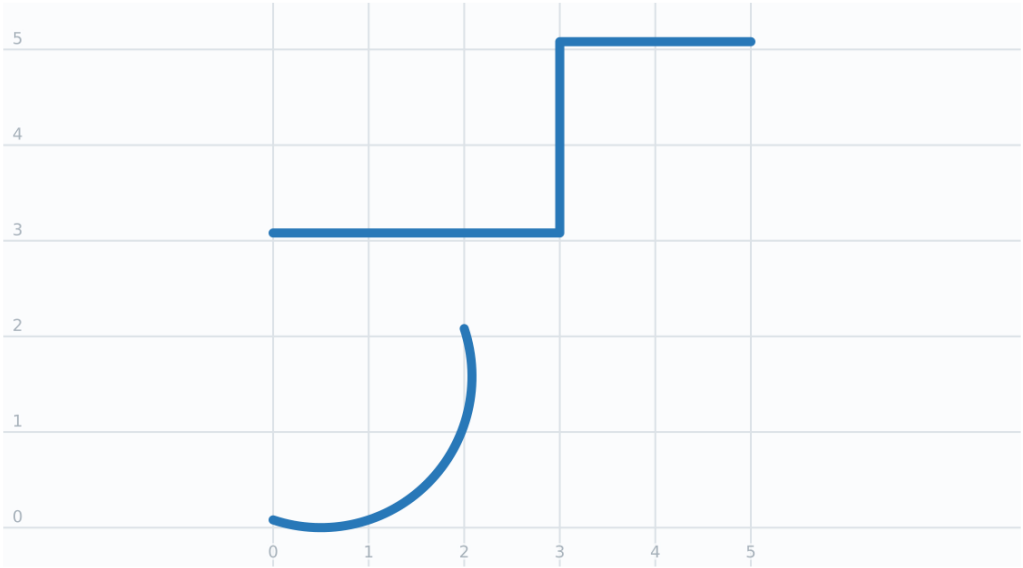
Code

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
```

Output

```
M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Figure



Code

Multi-surface

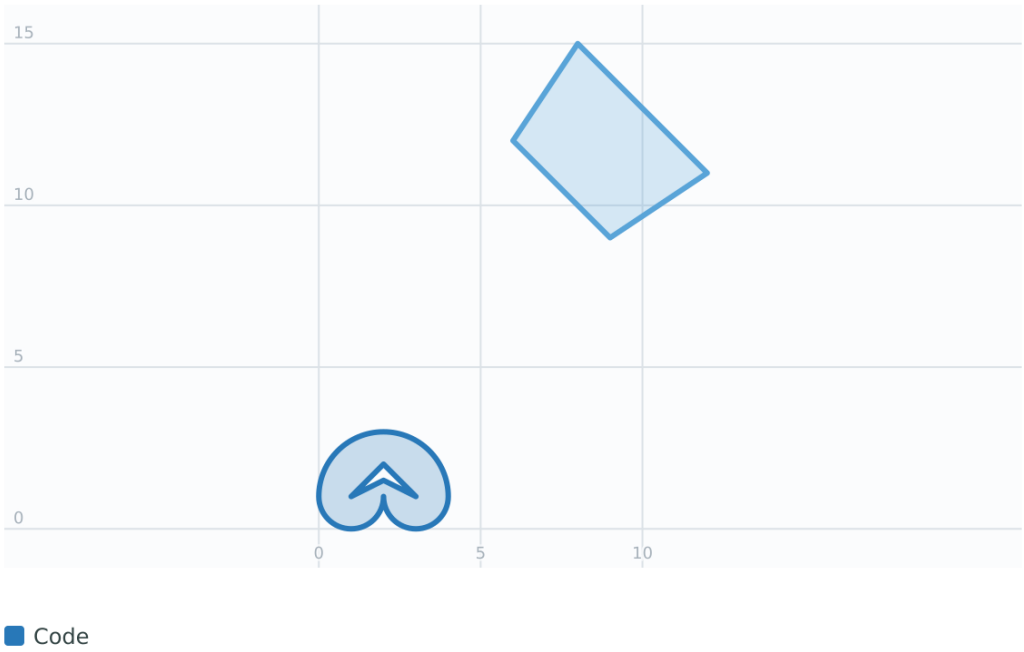
Code

```
SELECT ST_AsSVG('MULTISURFACE(
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),
    (-1 0,0 0.5,1 0,0 1,-1 0)),
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);
```

Output

```
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

Figure



7.9.3.12 ST_AsTWKB

ST_AsTWKB — Renvoie la géométrie sous forme de TWKB, diminutif de "Tiny Well-Known Binary"

Synopsis

bytea **ST_AsTWKB**(geometry geom, integer prec=0, integer prec_z=0, integer prec_m=0, boolean with_sizes=false, boolean with_boxes=false);
bytea **ST_AsTWKB**(geometry[] geom, bigint[] ids, integer prec=0, integer prec_z=0, integer prec_m=0, boolean with_sizes=false, boolean with_boxes=false);

Description

Renvoie la géométrie au format TWKB (Tiny Well-Known Binary). TWKB est un format binaire compressé dont l'objectif est de minimiser la taille de la sortie.

Les paramètres relatifs aux chiffres décimaux déterminent le degré de précision stocké dans la sortie. Par défaut, les valeurs sont arrondies à l'unité la plus proche avant l'encodage. Si vous souhaitez transférer plus de précision, augmentez le nombre. Par exemple, une valeur de 1 implique que le premier chiffre à droite du point décimal sera préservé.

Les paramètres "sizes" et "bounding boxes" déterminent si les informations optionnelles concernant la longueur encodée de l'objet et les limites de l'objet sont incluses dans la sortie. Par défaut, elles ne le sont pas. Ne les activez pas à moins que votre logiciel client n'en ait l'utilité, car ils ne font qu'utiliser de l'espace (et économiser de l'espace est l'objectif de TWKB).

La forme tableau de la fonction est utilisée pour convertir une collection de géométries et d'identifiants uniques en une collection TWKB qui préserve les identifiants. Cette fonction est utile pour les clients qui souhaitent décompresser une collection et accéder ensuite à d'autres informations sur les objets qu'elle contient. Vous pouvez créer les tableaux à l'aide de la fonction array_agg. Les autres paramètres sont les mêmes que pour la forme simple de la fonction.



Note
La spécification du format est disponible en ligne à l'adresse https://github.com/TWKB/Specification, et le code permettant de créer un client JavaScript est disponible à l'adresse https://github.com/TWKB/twkb.js.

Amélioration : 2.4.0 amélioration de la mémoire et de la vitesse.

Disponibilité : 2.2.0

Exemples

Code

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)'::geometry);
```

Output

```
\x02000202020808
```

Pour créer un objet TWKB agrégé comprenant des identifiants, il faut d'abord agréger les géométries et les objets souhaités en utilisant "array_agg()", puis appeler la fonction TWKB appropriée.

Code

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
```

Output

```
\x040402020400000202
```

Voir aussi

ST_GeomFromTWKB, ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.3.13 ST_AsX3D

ST_AsX3D — Renvoie une géométrie au format X3D xml node element : ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

Description

Renvoie une géométrie sous la forme d'un élément de nœud X3D xml formaté http://www.web3d.org/standards/number/19776-1. Si `maxdecimaldigits` (précision) n'est pas spécifié, la valeur par défaut est 15.



Note
Il existe plusieurs options pour traduire les géométries PostGIS en X3D, car les types de géométrie X3D ne correspondent pas directement aux types de géométrie PostGIS, et nous avons évité certains types X3D plus récents qui pourraient constituer de meilleures correspondances, car la plupart des outils de rendu ne les prennent pas en charge actuellement. Ce sont les correspondances que nous avons choisies. N'hésitez pas à poster un ticket de bug si vous avez des idées sur l'idée ou sur la façon dont nous pouvons permettre aux gens de dénoter leurs correspondances préférées.
Voici comment nous faisons actuellement correspondre les types 2D/3D de PostGIS aux types X3D

The 'options' argument is a bitfield that denotes whether to represent coordinates with the X3D GeoCoordinates geospatial node and whether to flip the x/y axis. By default, `ST_AsX3D` outputs in database form (long,lat or X,Y), but the X3D default of lat/lon, y/x may be preferred.

— 0 : X/Y dans l'ordre de la base de données (par exemple long/lat = X,Y est l'ordre standard de la base de données), valeur par défaut, et coordonnées non spatiales (juste un bon vieux Coordinate tag).

— 1 : Inverser X et Y. Si cette option est utilisée en conjonction avec l'option GeoCoordinate, la sortie sera par défaut "latitude_first" et les coordonnées seront également inversées.

— 2 : Sortie des coordonnées enGeoSpatial GeoCoordinates. Cette option génère une erreur si les géométries ne sont pas en WGS 84 long lat (srid : 4326). C'est actuellement le seul type de GeoCoordinate pris en charge. Référence aux spécifications X3D spécifiant un système de référence spatiale.. La sortie par défaut sera `GeoCoordinate geoSystem='"GD" "WE" "longitude_first"'`. Si vous préférez la sortie par défaut X3D de `GeoCoordinate geoSystem='"GD" "WE" "latitude_first"'`, utilisez `(2 + 1) = 3`

Type PostGIS	Type 2D X3D	Type 3D X3D
LINESTRING	LineSet (coordinates are promoted to 3D)	LineSet
MULTILINESTRING	IndexedLineSet (coordinates are promoted to 3D)	IndexedLineSet
MULTIPOINT	Polypoint2D	PointSet
POINT	produit les coordonnées délimitées par l'espace	produit les coordonnées délimitées par l'espace
(MULTI) POLYGON, POLYHEDRALSURFACE	IndexedFaceSet (coordinates are promoted to 3D; inner rings currently output as another faceset)	IndexedFaceSet (les anneaux intérieurs sont actuellement édités sous la forme d'un autre jeu de faces)
TIN	TriangleSet2D (Pas encore implémenté)	IndexedTriangleSet
CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, MULTISURFACE	linearized, then emitted with the linear geometry mapping	linearized, then emitted with the linear geometry mapping
NURBSCURVE	NurbsCurve with controlPoint, knot, weight, and order fields	NurbsCurve with controlPoint, knot, weight, and order fields



Note
La prise en charge de la géométrie 2D n'est pas encore terminée. Les anneaux intérieurs sont actuellement dessinés comme des polygones séparés. Nous y travaillons.

M coordinates are not represented in X3D output.

De nombreuses avancées ont lieu dans l'espace 3D, en particulier avec X3D Integration with HTML5

Il existe également une visionneuse X3D open source que vous pouvez utiliser pour visualiser les géométries rendues. Des binaires Wrl http://freewrl.sourceforge.net/ gratuits sont disponibles pour Mac, Linux et Windows. Utilisez le FreeWRL_Launcher pour visualiser les géométries.

Consultez également PostGIS minimalist X3D viewer qui utilise cette fonction et x3dDom html/js open source toolkit.

Disponibilité : 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Amélioration : 2.2.0 : Prise en charge des coordonnées géographiques et de l'inversion des axes (x/y, long/lat). Voir les options pour plus de détails.

Enhanced: 3.7.0: Support for curve geometries. NURBSCURVE is emitted as X3D NurbsCurve; other curve types are linearized before output.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Create a complete X3D document containing a cube that is viewable in FreeWRL and other X3D viewers.

Code

```
SELECT '<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ↩
    -3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
       <Appearance>
            <Material emissiveColor=''0 0 1''/>
       </Appearance
> ' ||
       ST_AsX3D(ST_GeomFromEWKT('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )')) ||
      '</Shape>
    </Transform>
  </Scene>
</X3D
>' As x3ddoc;
```

Output

```
    x3ddoc
    --------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D>
  <Scene>
    <Transform>
      <Shape>
       <Appearance>
            <Material emissiveColor='0 0 1'/>
       </Appearance>
       <IndexedFaceSet  coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 18 19 -1 20 21 22 23'>
            <Coordinate point='0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0  ↩
    1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1' />
       </IndexedFaceSet>
```

```
      </Shape>
    </Transform>
  </Scene>
</X3D>
```

PostGIS buildings.

Copiez et collez le résultat de cette requête dans x3d scene viewer et cliquez sur Show

Code

```
SELECT string_agg('<Shape
>' || ST_AsX3D(CG_Extrude(geom, 0,0, i*0.5)) ||
    '<Appearance>
          <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' ||  ↩
    (0.05*i)::text || ' 0 0.5"/>
        </Appearance>
    </Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```

The following preview shows the extruded geometry generated from a simplified version of the same source lettering. It is a geometry preview, not X3D markup; use the preceding `ST_AsX3D` query output in an X3D viewer to inspect the native X3D scene.

Code

```
SELECT
    CG_Extrude(
        ST_SimplifyPreserveTopology(ST_Letters('PostGIS'), 2),
        0, 0, 10
    ) AS buildings;
```

Figure



An Octagon elevated 3 Units and decimal precision of 6.

Code

```
SELECT ST_AsX3D(ST_Translate(ST_Force3D(ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
    3),
  6) As x3dfrag;
```

Output

```
x3dfrag
--------
<IndexedFaceSet  convex='false' coordIndex='0 1 2 3 4 5 6 7'
><Coordinate point='15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3 6.464466 13.535534 3 10 15 3 ↩
    13.535534 13.535534 3 ' /></IndexedFaceSet>
```

TIN.

Code

```
SELECT ST_AsX3D(ST_GeomFromEWKT('TIN (((
                0 0 0,
                0 0 1,
                0 1 0,
                0 0 0
            )), ((
                0 0 0,
                0 1 0,
                1 1 0,
                0 0 0
            ))
            )')) As x3dfrag;
```

Output

```
    x3dfrag
    --------
<IndexedTriangleSet  index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0'/></IndexedTriangleSet>
```

Closed multilinestring (the boundary of a polygon with holes).

Code

```
SELECT ST_AsX3D(ST_GeomFromEWKT('MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 ↩
    0 10,-12 16 10,0 24 10,16 16 10,20 0 10),
  (12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))')
) As x3dfrag;
```

Output

```
    x3dfrag
    --------
<IndexedLineSet  coordIndex='0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8'>
    <Coordinate point='20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16 16 10 12 0 10 8 8 10 0 12 10 ↩
      -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ' />
 </IndexedLineSet>
```

7.9.3.14 ST_GeoHash

ST_GeoHash — Retourne une représentation GeoHash de la géométrie.

Synopsis

text **ST_GeoHash**(geometry geom, integer maxchars=full_precision_of_point);

Description

Calcule une représentation GeoHash d'une géométrie. Un GeoHash encode un point géographique sous forme de texte qui peut être trié et recherché en fonction du préfixe. Un GeoHash plus court est une représentation moins précise d'un point. Il peut être considéré comme une boîte qui contient le point.

Les valeurs géométriques non ponctuelles dont l'étendue n'est pas nulle peuvent également être mises en correspondance avec des codes GeoHash. La précision du code dépend de l'étendue géographique de la géométrie.

Si `maxchars` n'est pas spécifié, le code GeoHash renvoyé correspond à la plus petite cellule contenant la géométrie d'entrée. Les points renvoient un GeoHash avec 20 caractères de précision (ce qui est suffisant pour contenir la double précision de l'entrée). D'autres types géométriques peuvent renvoyer un GeoHash avec moins de précision, en fonction de l'étendue de la géométrie. Les géométries plus grandes sont représentées avec moins de précision, les plus petites avec plus de précision. La boîte déterminée par le code GeoHash contient toujours l'élément d'entrée.

Si `maxchars` est spécifié, le code GeoHash renvoyé comporte au maximum ce nombre de caractères. Il correspond à une représentation (éventuellement) de moindre précision de la géométrie d'entrée. Pour les non-points, le point de départ du calcul est le centre de la boîte de délimitation de la géométrie.

Disponibilité : 1.4.0



Note
ST_GeoHash exige que la géométrie d'entrée soit en coordonnées géographiques (lon/lat).

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_GeoHash(ST_Point(-126, 48));
```

Output

```
c0w3hf1s70w3hf1s70w3
```

Code

```
SELECT ST_GeoHash(ST_Point(-126, 48), 5);
```

Output

```
c0w3h
```

This example shows that the GeoHash for a line containing a point is a prefix of the point code.

Code

```
SELECT ST_GeoHash('LINESTRING(-126 48,-126.1 48.1)'::geometry);
```

Output

```
c0w3
```

Voir aussi

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 Opérateurs

7.10.1 Opérateurs de Bounding Box

7.10.1.1 &&

&& — Renvoi VRAI si la boite englobante 2D de A intersecte la boite englobante 2D de B.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

Description

L'opérateur && renvoi VRAI si la boite englobante 2D de la géométrie A intersecte la boite englobante 2D de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Amélioration : 2.0.0 introduction du support des surfaces polyédriques.

Disponibilité : 1.5.0 le support de la géographie a été introduit.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

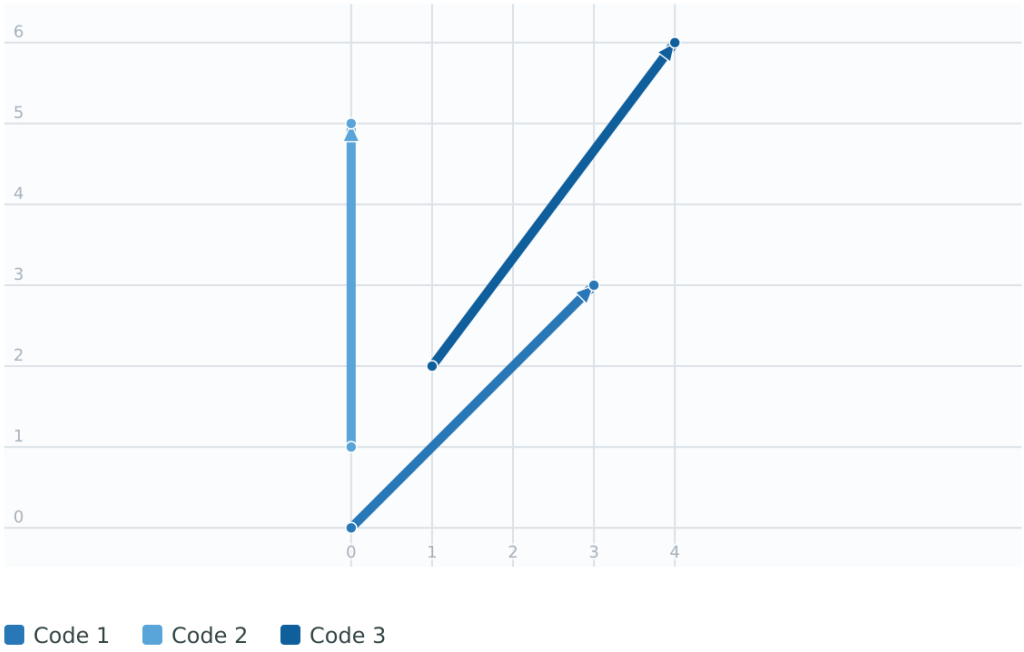
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps
FROM ( VALUES
    (1, 'LINESTRING(0 0, 3 3)'::geometry),
    (2, 'LINESTRING(0 1, 0 5)'::geometry)) AS tbl1,
( VALUES
    (3, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overlaps
---------+---------+----------
       1 |       3 | t
       2 |       3 | f
(2 rows)
```

Figure



Voir aussi

ST_Intersects, ST_Extent, |&>, &>, &<|, &<, ~, @

7.10.1.2 &&(geometry,box2df)

&&(geometry,box2df) — Renvoie `TRUE` si la boîte de délimitation 2D (en cache) d'une géométrie intersecte une boîte de délimitation 2D de précision flottante (BOX2DF).

Synopsis

boolean **&&**(geometry A , box2df B);

Description

L'opérateur `&&` renvoie `TRUE` si la boîte de délimitation 2D mise en cache de la géométrie A intersecte la boîte de délimitation 2D B, en utilisant la précision float. Cela signifie que si B est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D à précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Voir aussi

&&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.3 &&(box2df,geometry)

&&(box2df,geometry) — Renvoie TRUE si une boîte de délimitation 2D de précision flottante (BOX2DF) intersecte la boîte de délimitation 2D (mise en cache) d'une géométrie.

Synopsis

boolean **&&**(box2df A , geometry B);

Description

L'opérateur && renvoie TRUE si la boîte de délimitation 2D A intersecte la boîte de délimitation 2D mise en cache de la géométrie B, en utilisant la précision float. Cela signifie que si A est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D à précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Voir aussi

&&(geometry,box2df), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.4 &&(box2df,box2df)

&&(box2df,box2df) — Renvoie `TRUE` si deux boîtes de délimitation 2D à précision flottante (BOX2DF) se croisent.

Synopsis

boolean **&&**(box2df A , box2df B);

Description

L'opérateur `&&` renvoie `TRUE` si deux boîtes de délimitation 2D A et B se croisent, en utilisant la précision float. Cela signifie que si A (ou B) est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D de précision flottante (BOX2DF)



Note
Cet opérateur est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point ↩
    (3,3)) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.5 &&&

&&& — Renvoie `TRUE` si la boîte de délimitation n-D de A intersecte la boîte de délimitation n-D de B.

Synopsis

boolean **&&&**(geometry A , geometry B);

Description

L'opérateur `&&&` renvoie `TRUE` si la boîte de délimitation n-D de la géométrie A intersecte la boîte de délimitation n-D de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Disponibilité : 2.0.0

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

3D LineStrings.

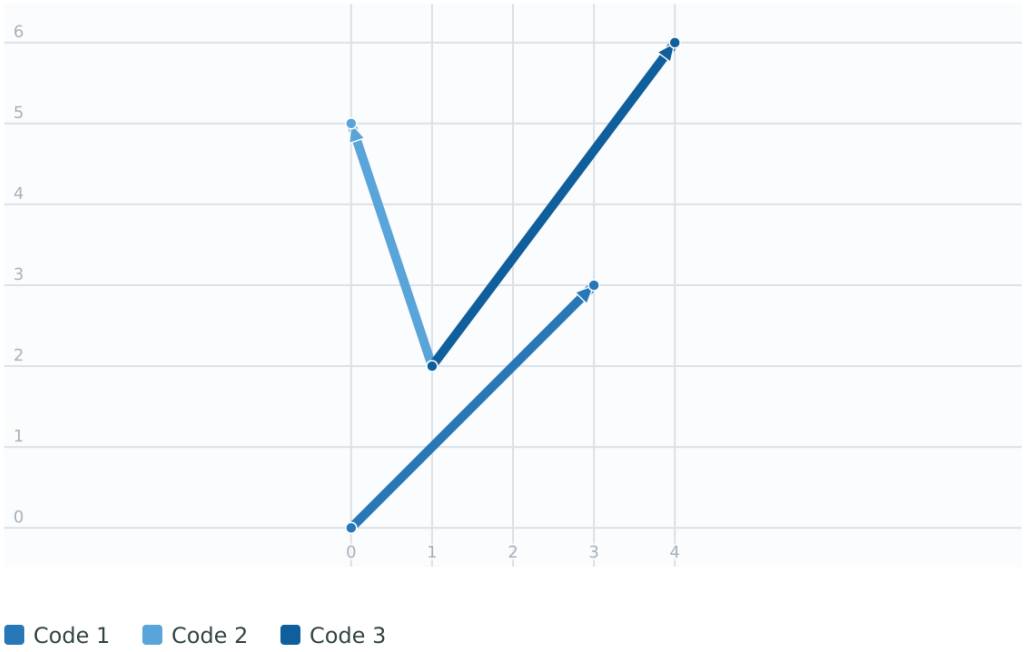
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3d,
                       tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
    (1, 'LINESTRING Z(0 0 1, 3 3 2)'::geometry),
    (2, 'LINESTRING Z(1 2 0, 0 5 -1)'::geometry)) AS tbl1,
( VALUES
    (3, 'LINESTRING Z(1 2 1, 4 6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overlaps_3d | overlaps_2d
---------+---------+-------------+-------------
       1 |       3 | t           | t
       2 |       3 | f           | t
```

Figure



3M LineStrings.

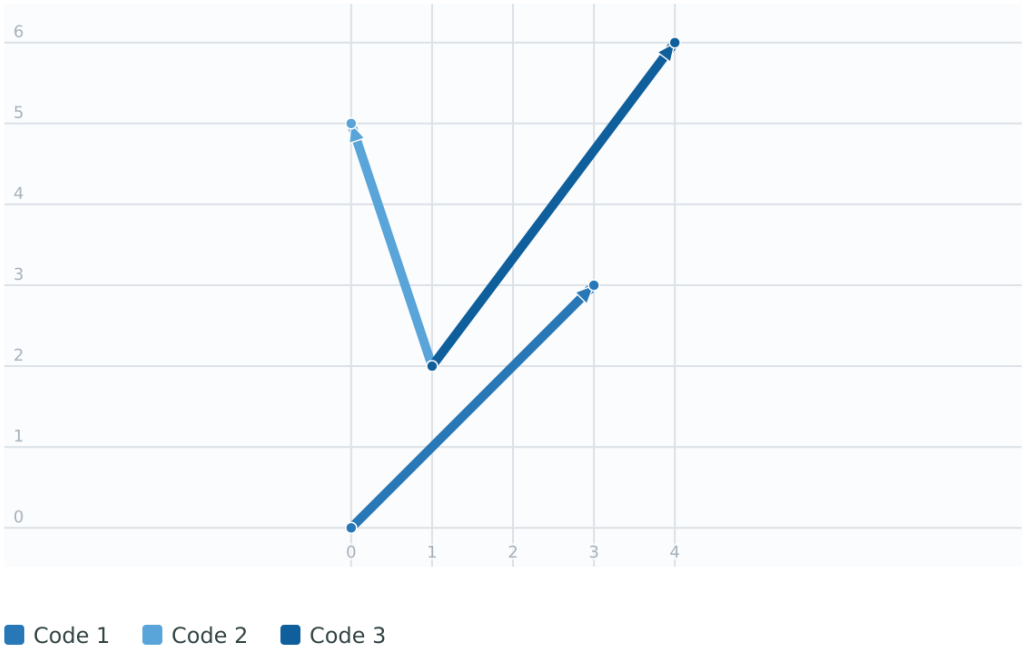
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps_3zm,
                       tbl1.column2 && tbl2.column2 AS overlaps_2d
FROM ( VALUES
    (1, 'LINESTRING M(0 0 1, 3 3 2)'::geometry),
    (2, 'LINESTRING M(1 2 0, 0 5 -1)'::geometry)) AS tbl1,
( VALUES
    (3, 'LINESTRING M(1 2 1, 4 6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overlaps_3zm | overlaps_2d
---------+---------+--------------+-------------
       1 |       3 | t            | t
       2 |       3 | f            | t
```

Figure



Voir aussi

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Renvoie `TRUE` si la boîte de délimitation n-D (en cache) d'une géométrie intersecte une boîte de délimitation de précision flottante n-D (GIDX).

Synopsis

boolean **&&&**(geometry A , gidx B);

Description

L'opérateur `&&&` renvoie `TRUE` si la boîte de délimitation n-D mise en cache de la géométrie A intersecte la boîte de délimitation n-D B, en utilisant la précision float. Cela signifie que si B est un box3d (double précision), il sera converti en interne en une boîte de délimitation 3D de précision flottante (GIDX)



Note
Cet opérateur est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS ↩
    overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Voir aussi

&&&(gidx,geometry), &&&(gidx,gidx)

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — Renvoie `TRUE` si une boîte de délimitation de précision flottante n-D (GIDX) intersecte la boîte de délimitation n-D (mise en cache) d'une géométrie.

Synopsis

boolean **&&&**(gidx A , geometry B);

Description

L'opérateur `&&&` renvoie `TRUE` si la boîte de délimitation n-D A intersecte la boîte de délimitation n-D mise en cache de la géométrie B, en utilisant la précision float. Cela signifie que si A est un box3d (à double précision), il sera converti en interne en une boîte de délimitation 3D à précision flottante (GIDX)



Note
Cet opérateur est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS ↩
    overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Voir aussi

&&&(geometry,gidx), &&&(gidx,gidx)

7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — Renvoie `TRUE` si deux boîtes de délimitation (GIDX) de précision flottante n-D se croisent.

Synopsis

boolean **&&&**(gidx A , gidx B);

Description

L'opérateur `&&&` renvoie `TRUE` si deux boîtes de délimitation n-D A et B se croisent, en utilisant la précision float. Cela signifie que si A (ou B) est un box3d (double précision), il sera converti en interne en une boîte de délimitation 3D de précision flottante (GIDX)



Note
Cet opérateur est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.


Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint ↩
    (1,1,1), ST_MakePoint(3,3,3)) AS overlaps;
```

Output

```
 overlaps
----------
 t
(1 row)
```

Voir aussi

&&&(geometry,gidx), &&&(gidx,geometry)

7.10.1.9 &<

&< — Renvoie TRUE si la boîte englobante de A chevauche ou est à gauche de celle de B.

Synopsis

boolean **&<**(geometry A , geometry B);

Description

L'opérateur &< renvoie TRUE si la boîte de délimitation de la géométrie A chevauche ou est à gauche de la boîte de délimitation de la géométrie B, ou plus exactement, chevauche ou n'est PAS à droite de la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

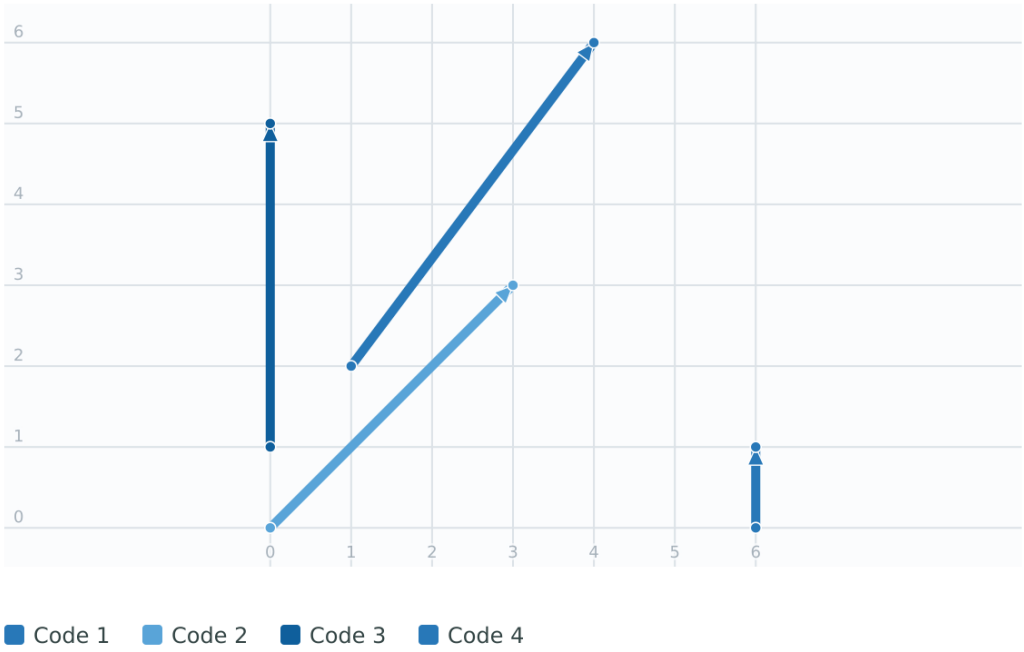
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft
FROM
  ( VALUES
    (1, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)'::geometry),
    (3, 'LINESTRING(0 1, 0 5)'::geometry),
    (4, 'LINESTRING(6 0, 6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overleft
---------+---------+----------
       1 |       2 | f
       1 |       3 | f
       1 |       4 | t
(3 rows)
```

Figure



Voir aussi

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — Renvoie `TRUE` si la boîte englobante de A chevauche ou est inférieure à celle de B.

Synopsis

boolean **&<|**(geometry A , geometry B);

Description

L'opérateur `&<|` renvoie `TRUE` si la boîte de délimitation de la géométrie A chevauche ou est en dessous de la boîte de délimitation de la géométrie B, ou plus exactement, chevauche ou n'est PAS au-dessus de la boîte de délimitation de la géométrie B.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

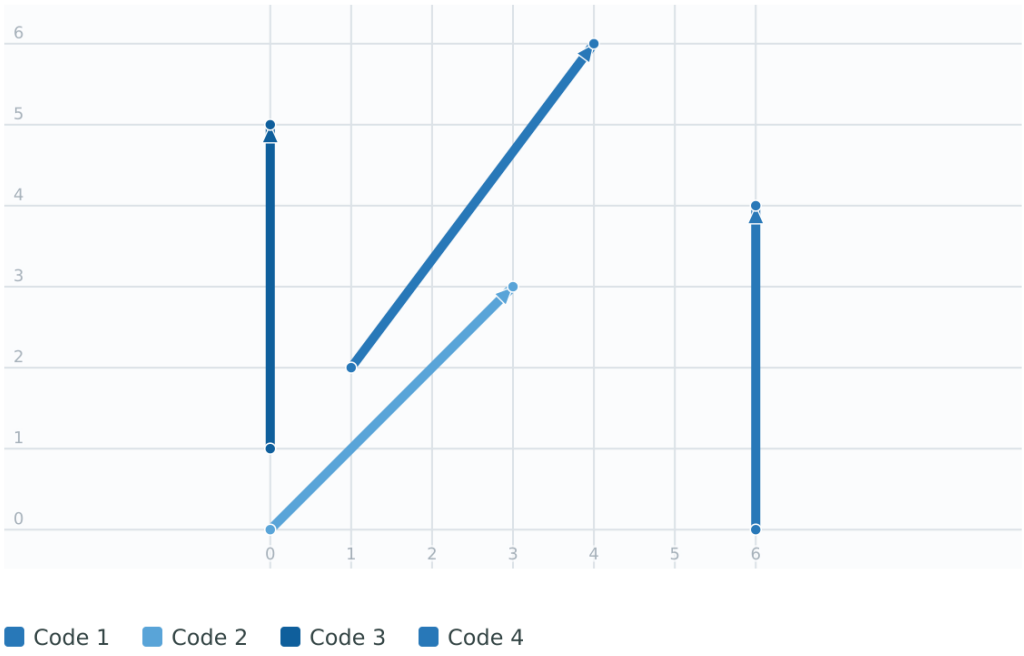
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &<| tbl2.column2 AS overbelow
FROM
  ( VALUES
    (1, 'LINESTRING(6 0, 6 4)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0, 3 3)'::geometry),
    (3, 'LINESTRING(0 1, 0 5)'::geometry),
    (4, 'LINESTRING(1 2, 4 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overbelow
---------+---------+-----------
       1 |       2 | f
       1 |       3 | t
       1 |       4 | t
(3 rows)
```

Figure



Voir aussi

&&, |&>, &>, &<

7.10.1.11 &>

&> — Renvoie `TRUE` si la boîte de délimitation de A chevauche ou est à droite de celle de B.

Synopsis

boolean **&>**(geometry A , geometry B);

Description

L'opérateur `&>` renvoie `TRUE` si la boîte de délimitation de la géométrie A chevauche ou est à droite de la boîte de délimitation de la géométrie B, ou plus exactement, chevauche ou n'est PAS à gauche de la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

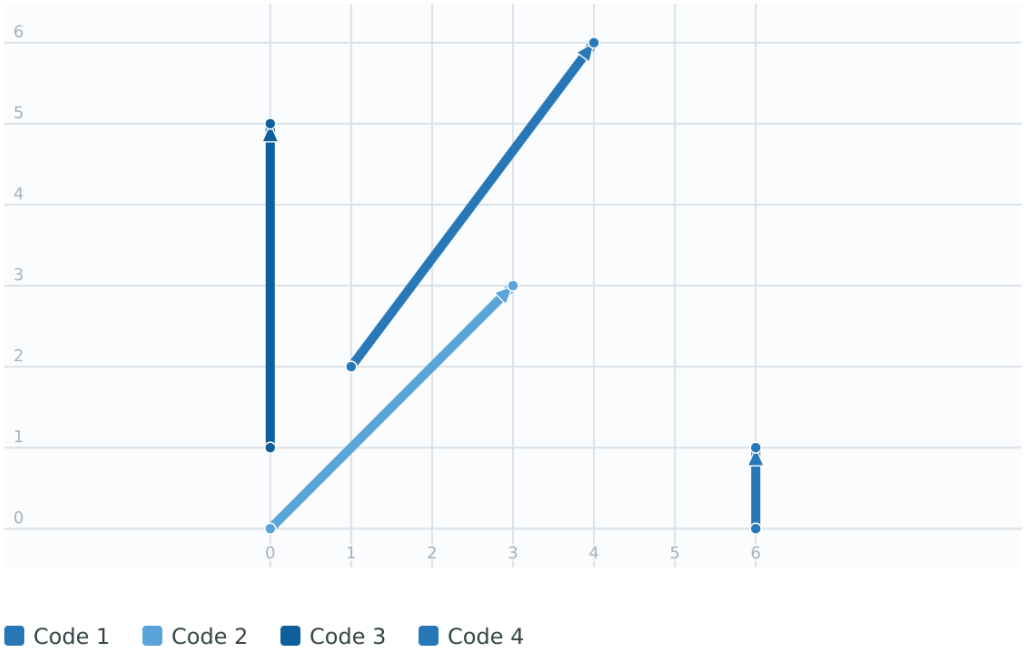
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 &
> tbl2.column2 AS overright
FROM
  ( VALUES
    (1, 'LINESTRING(1 2,4 6)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0,3 3)'::geometry),
    (3, 'LINESTRING(0 1,0 5)'::geometry),
    (4, 'LINESTRING(6 0,6 1)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overright
---------+---------+-----------
       1 |       2 | t
       1 |       3 | t
       1 |       4 | f
(3 rows)
```

Figure



Voir aussi

&&, |&>, &<|, &<

7.10.1.12 <<

<< — Renvoie TRUE si la boîte de délimitation de A est strictement à gauche de celle de B.

Synopsis

boolean **<<**(geometry A , geometry B);

Description

L'opérateur << renvoie TRUE si la boîte de délimitation de la géométrie A est strictement à gauche de la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

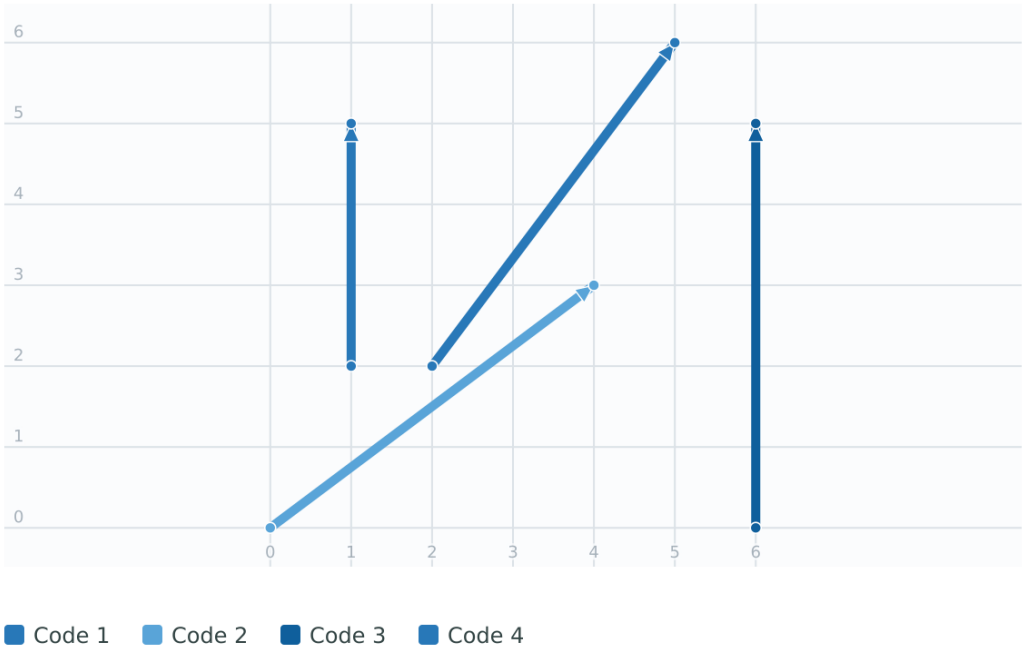
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
    (1, 'LINESTRING (1 2, 1 5)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 3)'::geometry),
    (3, 'LINESTRING (6 0, 6 5)'::geometry),
    (4, 'LINESTRING (2 2, 5 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | left
---------+---------+------
       1 |       2 | f
       1 |       3 | t
       1 |       4 | t
(3 rows)
```

Figure



Voir aussi

>>, |>>, <<|

7.10.1.13 <<|

<<| — Renvoie `TRUE` si la boîte de délimitation de A est strictement inférieure à celle de B.

Synopsis

boolean **<<|**(geometry A , geometry B);

Description

L'opérateur `<<|` renvoie `TRUE` si la boîte de délimitation de la géométrie A est strictement inférieure à la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

Code

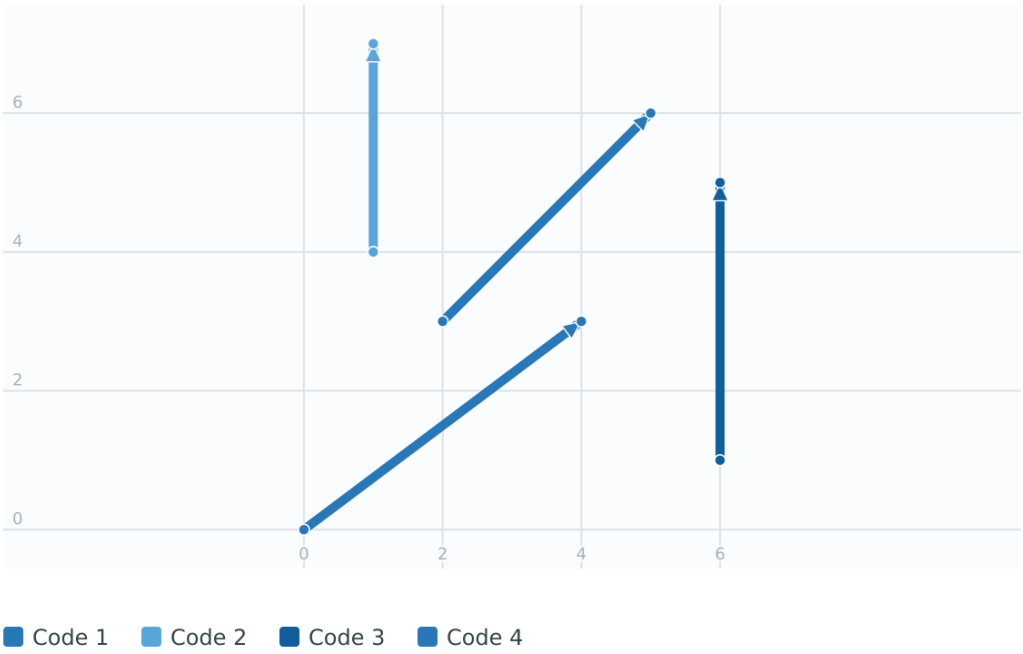
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
    (1, 'LINESTRING (0 0, 4 3)'::geometry)) AS tbl1,
```

```
  ( VALUES
    (2, 'LINESTRING (1 4, 1 7)'::geometry),
    (3, 'LINESTRING (6 1, 6 5)'::geometry),
    (4, 'LINESTRING (2 3, 5 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | below
---------+---------+-------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Voir aussi

<<, >>, |>>

7.10.1.14 =

= — Renvoie `TRUE` si les coordonnées et l'ordre des coordonnées de la géométrie/géographie A sont les mêmes que les coordonnées et l'ordre des coordonnées de la géométrie/géographie B.

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

Description

L'opérateur = renvoie `TRUE` si les coordonnées et l'ordre des coordonnées de la géométrie/géographie A sont les mêmes que les coordonnées et l'ordre des coordonnées de la géométrie/géographie B. PostgreSQL utilise les opérateurs =, <, et > définis pour les géométries pour effectuer des classements internes et des comparaisons de géométries (c'est-à-dire dans une clause GROUP BY ou ORDER BY).



Note
Seules les géométries/géographies qui sont exactement égales à tous égards, avec les mêmes coordonnées, dans le même ordre, sont considérées comme égales par cet opérateur. Pour une "égalité spatiale", qui ignore des choses comme l'ordre des coordonnées, et peut détecter des caractéristiques qui couvrent la même zone spatiale avec des représentations différentes, utilisez ST_OrderingEquals ou ST_Equals



Attention
Cet opérande n'utilisera PAS les index qui peuvent être disponibles sur les géométries. Pour un test d'égalité exact assisté par index, combinez = avec &&.

Modifié : 2.4.0, dans les versions précédentes, il s'agissait d'une égalité de boîte de délimitation et non d'une égalité géométrique. Si vous avez besoin d'une égalité de boîte de délimitation, utilisez ~= à la place.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

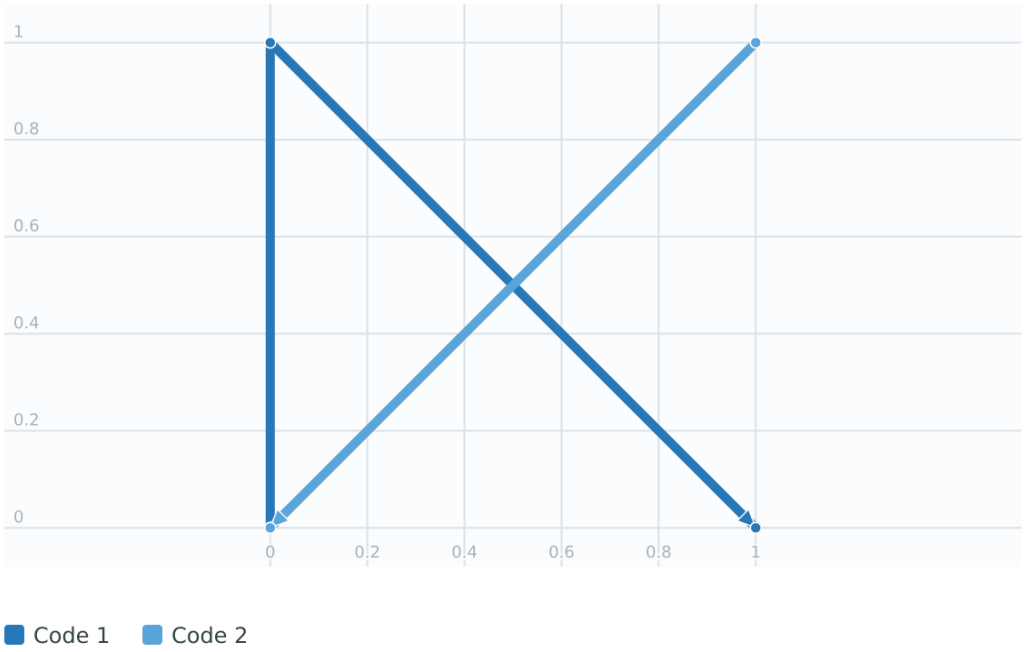
Code

```
SELECT 'LINESTRING(0 0,0 1,1 0)'::geometry = 'LINESTRING(1 1,0 0)'::geometry;
```

Output

```
 ?column?
----------
 f
(1 row)
```

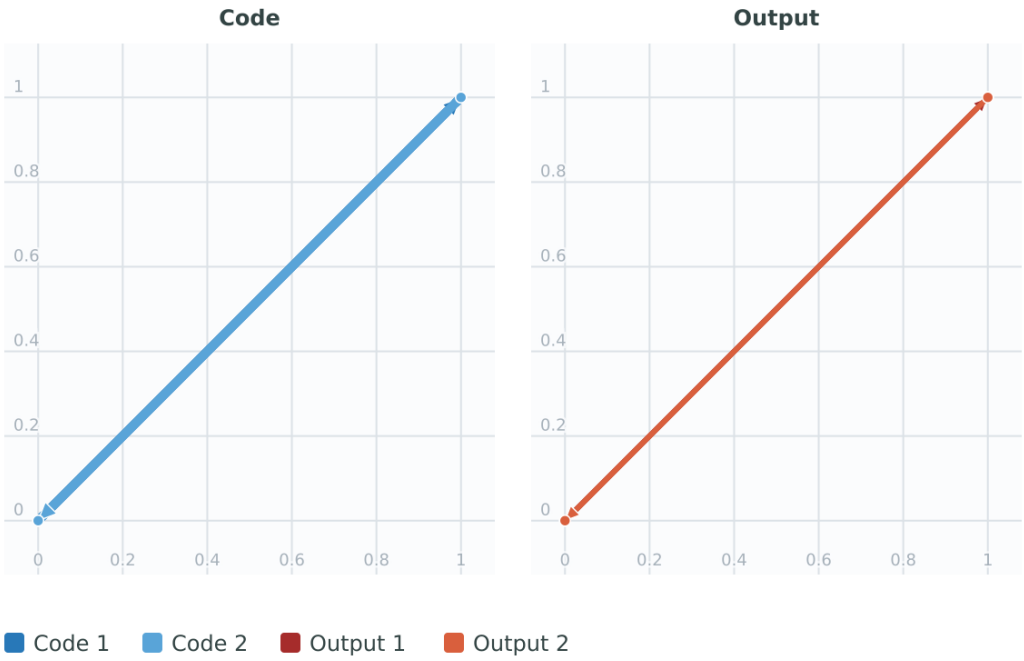
Figure



Code

```
SELECT column1
FROM ( VALUES
    ('LINESTRING(0 0, 1 1)'::geometry),
    ('LINESTRING(1 1, 0 0)'::geometry)) AS foo;
```

Figure

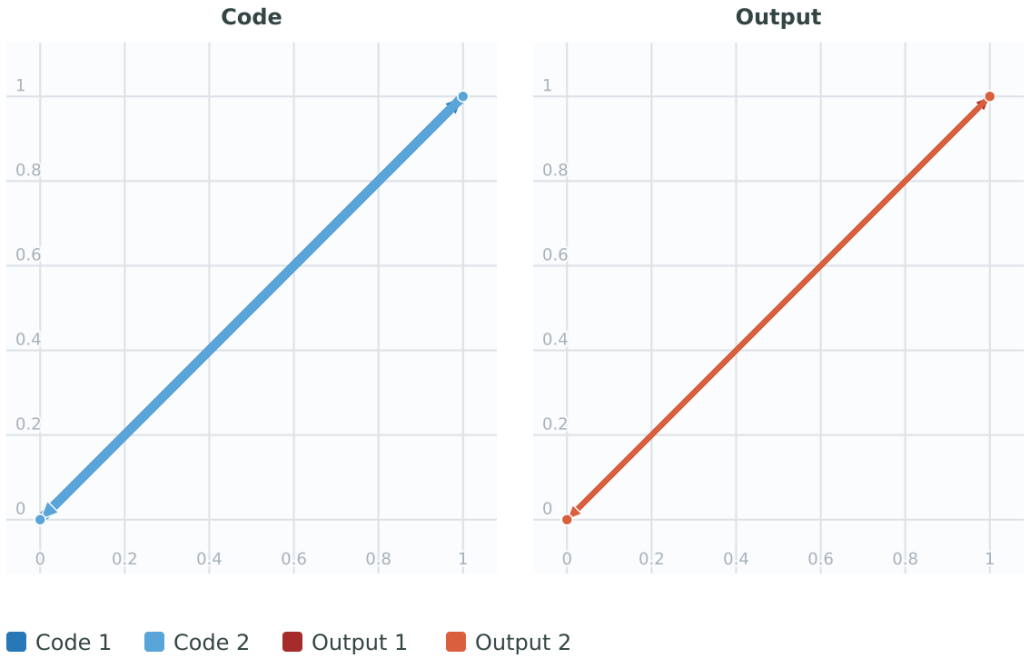


Note: the GROUP BY uses the "=" to compare for geometry equivalency.

Code

```
SELECT column1
FROM ( VALUES
    ('LINESTRING(0 0, 1 1)'::geometry),
    ('LINESTRING(1 1, 0 0)'::geometry)) AS foo
GROUP BY column1;
```

Figure



In versions prior to 2.0, this used to return true.

Code

```
SELECT 'POINT(1707296.37 4820536.77)'::geometry =
   'POINT(1707296.27 4820536.87)'::geometry As pt_intersect;
```

Output

```
 pt_intersect
--------------
 f
(1 row)
```

Voir aussi

ST_Equals, ST_OrderingEquals, ~=

7.10.1.15 >>

>> — Renvoie `TRUE` si la boîte de délimitation de A est strictement à droite de celle de B.

Synopsis

boolean **>>**(geometry A , geometry B);

Description

L'opérateur >> renvoie `TRUE` si la boîte de délimitation de la géométrie A est strictement à droite de la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

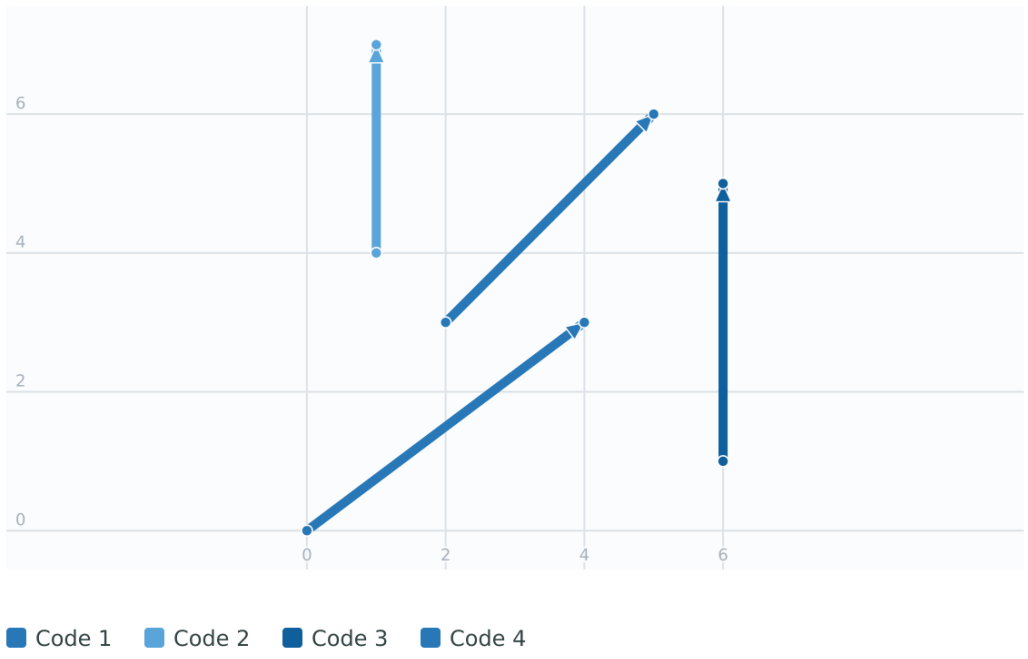
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2
>
> tbl2.column2 AS right
FROM
  ( VALUES
    (1, 'LINESTRING (2 3,5 6)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (1 4,1 7)'::geometry),
    (3, 'LINESTRING (6 1,6 5)'::geometry),
    (4, 'LINESTRING (0 0,4 3)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | right
---------+---------+-------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Voir aussi

<<, |>>, <<|

7.10.1.16 @

@ — Renvoie TRUE si la boîte de délimitation de A est contenue par celle de B.

Synopsis

boolean @(geometry A , geometry B);

Description

L'opérateur @ renvoie TRUE si la boîte de délimitation de la géométrie A est complètement contenue par la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

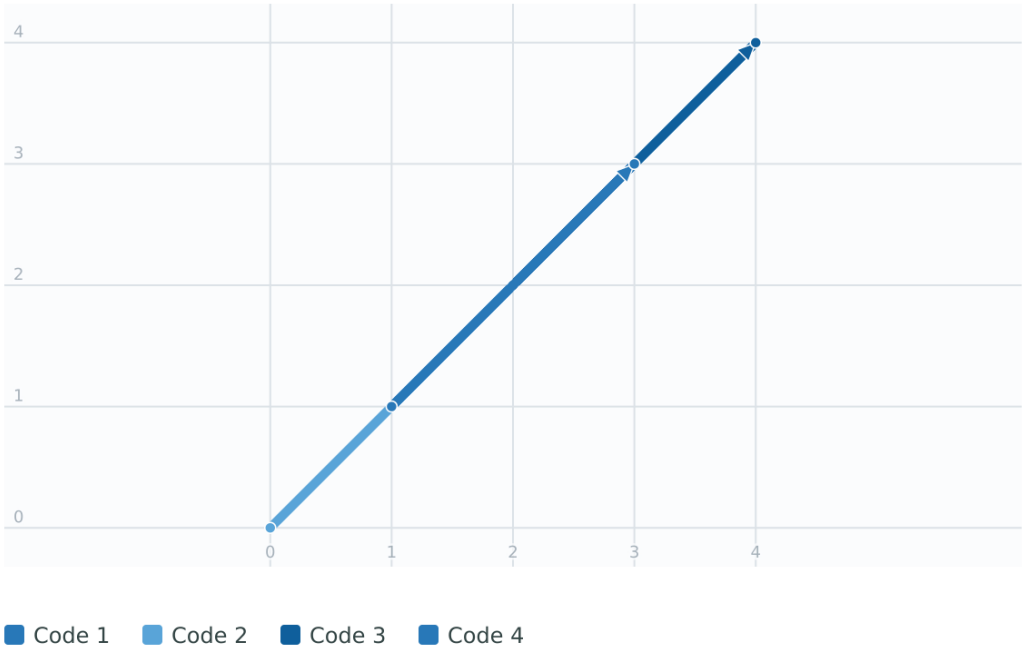
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
    (1, 'LINESTRING (1 1, 3 3)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0, 4 4)'::geometry),
    (3, 'LINESTRING (2 2, 4 4)'::geometry),
    (4, 'LINESTRING (1 1, 3 3)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | contained
---------+---------+-----------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | t
(3 rows)
```

Figure



Voir aussi

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Renvoie `TRUE` si la boîte de délimitation 2D d'une géométrie est contenue dans une boîte de délimitation 2D à précision flottante (BOX2DF).

Synopsis

boolean @(geometry A , box2df B);

Description

L'opérateur `@` renvoie `TRUE` si la boîte de délimitation 2D de la géométrie A est contenue dans la boîte de délimitation 2D B, en utilisant la précision float. Cela signifie que si B est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D de précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_Buffer('POINT(2 2)'::geometry, 1) @ ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5))  ↩
    AS is_contained;
```

Output

```
t
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Renvoie `TRUE` si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans la boîte de délimitation 2D d'une géométrie.

Synopsis

boolean @(box2df A , geometry B);

Description

L'opérateur `@` renvoie `TRUE` si la boîte de délimitation 2D A est contenue dans la boîte de délimitation 2D de la géométrie B, en utilisant la précision float. Cela signifie que si B est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D de précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) @ ST_Buffer('POINT(1 1)'::geometry, 10)  ↩
    AS is_contained;
```

Output

```
t
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Renvoie `TRUE` si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans une autre boîte de délimitation de précision flottante 2D.

Synopsis

boolean @(box2df A , box2df B);

Description

L'opérateur `@` renvoie `TRUE` si la boîte de délimitation 2D A est contenue dans la boîte de délimitation 2D B, en utilisant la précision float. Cela signifie que si A (ou B) est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D à précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_MakeBox2D(ST_Point(2, 2), ST_Point(3, 3)) @ ST_MakeBox2D(ST_Point(0, 0), ST_Point ↩
    (5, 5)) AS is_contained;
```

Output

```
t
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry)

7.10.1.20 |&>

|&> — Renvoie `TRUE` si la boîte de délimitation de A chevauche ou est au-dessus de celle de B.

Synopsis

boolean **|&>**(geometry A , geometry B);

Description

L'opérateur `|&>` renvoie `TRUE` si la boîte de délimitation de la géométrie A chevauche ou est au-dessus de la boîte de délimitation de la géométrie B, ou plus exactement, chevauche ou n'est PAS au-dessous de la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

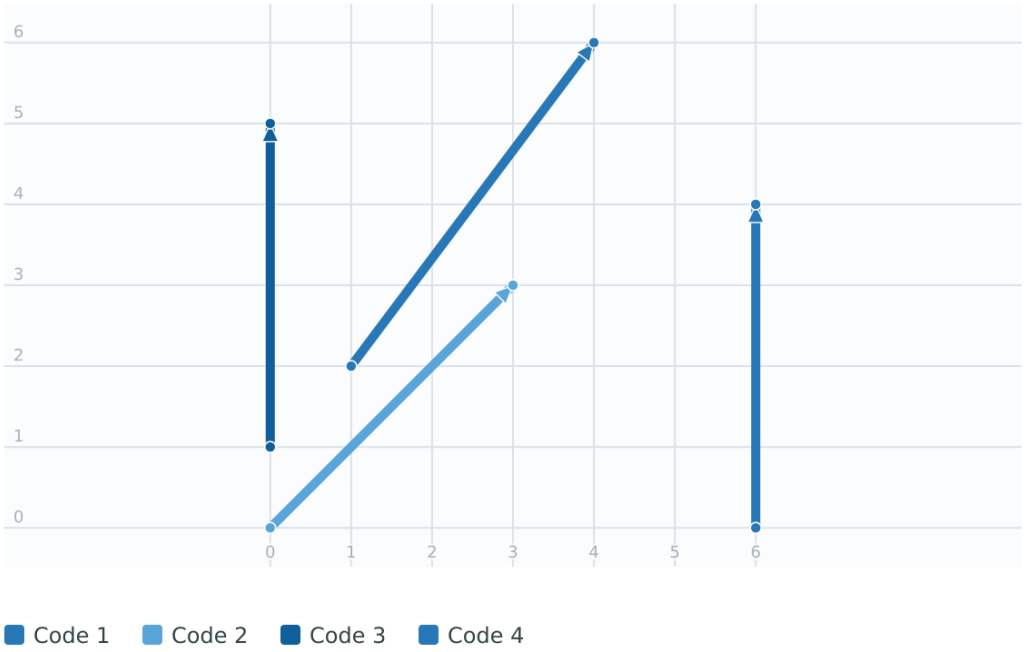
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&
> tbl2.column2 AS overabove
FROM
  ( VALUES
    (1, 'LINESTRING(6 0,6 4)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING(0 0,3 3)'::geometry),
    (3, 'LINESTRING(0 1,0 5)'::geometry),
    (4, 'LINESTRING(1 2,4 6)'::geometry)) AS tbl2;
```

Output

```
 column1 | column1 | overabove
---------+---------+-----------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Voir aussi

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — Renvoie `TRUE` si la boîte de délimitation de A est strictement au-dessus de celle de B.

Synopsis

boolean **|>>**(geometry A , geometry B);

Description

L'opérateur `|>>` renvoie `TRUE` si la boîte de délimitation de la géométrie A est strictement au-dessus de la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

Code

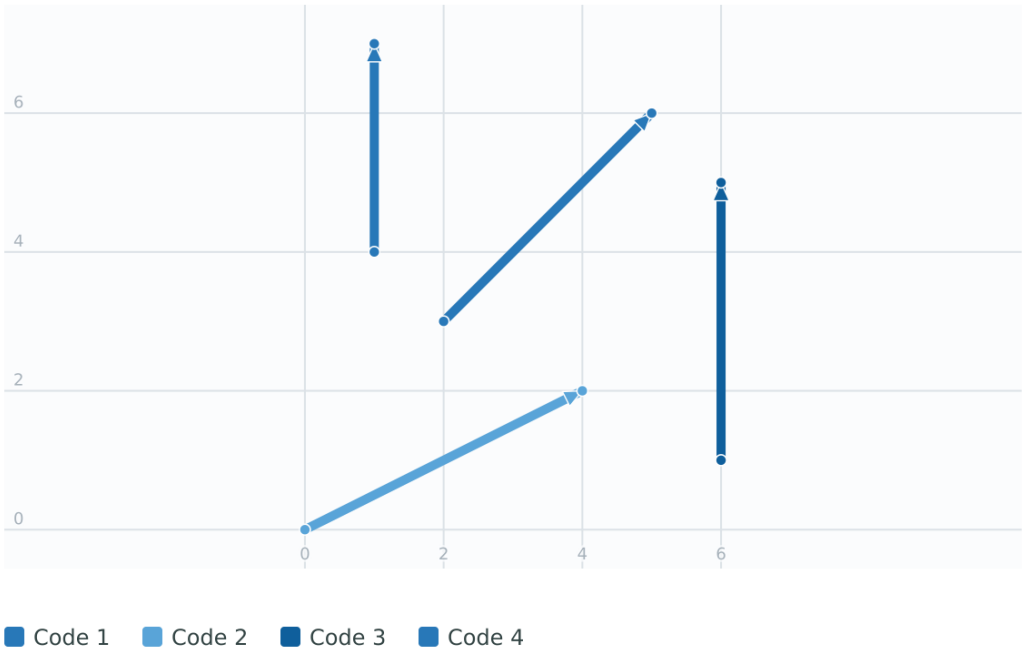
```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
    (1, 'LINESTRING (1 4,1 7)'::geometry)) AS tbl1,
```

```
  ( VALUES
    (2, 'LINESTRING (0 0,4 2)'::geometry),
    (3, 'LINESTRING (6 1,6 5)'::geometry),
    (4, 'LINESTRING (2 3,5 6)'::geometry)) AS tbl2;
```

Output

```
column1 | column1 | above
---------+---------+-------
       1 |       2 | t
       1 |       3 | f
       1 |       4 | f
(3 rows)
```

Figure



Voir aussi

<<, >>, <<|

7.10.1.22 ~

~ — Renvoie `TRUE` si la boîte de délimitation de A contient celle de B.

Synopsis

boolean ~(geometry A , geometry B);

Description

L'opérateur ~ renvoie `TRUE` si la boîte de délimitation de la géométrie A contient complètement la boîte de délimitation de la géométrie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

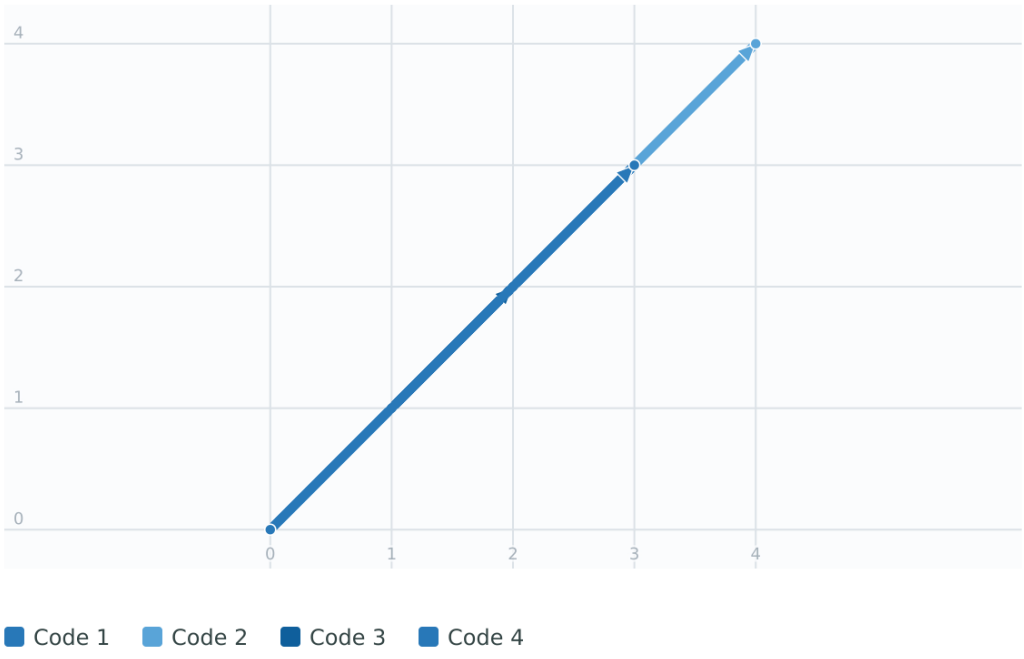
Code

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
    (1, 'LINESTRING (0 0,3 3)'::geometry)) AS tbl1,
  ( VALUES
    (2, 'LINESTRING (0 0,4 4)'::geometry),
    (3, 'LINESTRING (1 1,2 2)'::geometry),
    (4, 'LINESTRING (0 0,3 3)'::geometry)) AS tbl2;
```

Output

```
column1 | column1 | contains
---------+---------+----------
       1 |       2 | f
       1 |       3 | t
       1 |       4 | t
(3 rows)
```

Figure



Voir aussi

@, &&

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Renvoie TRUE si la boîte de délimitation 2D d'une géométrie contient une boîte de délimitation de précision flottante 2D (GIDX).

Synopsis

boolean ~(geometry A , box2df B);

Description

L'opérateur ~ renvoie TRUE si la boîte de délimitation 2D d'une géométrie A contient la boîte de délimitation 2D B, en utilisant la précision float. Cela signifie que si B est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D de précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_Buffer('POINT(1 1)'::geometry, 10) ~ ST_MakeBox2D(ST_Point(0, 0), ST_Point(2, 2))  ↩
    AS contains;
```

Output

```
t
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient la boîte de délimitation 2D d'une géométrie.

Synopsis

boolean ~(box2df A , geometry B);

Description

L'opérateur ~ renvoie TRUE si la boîte de délimitation 2D A contient la boîte de délimitation de la géométrie B, en utilisant la précision float. Cela signifie que si A est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D de précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.


Cette méthode prend en charge les types Circular String et Curve.


Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ~ ST_Buffer('POINT(2 2)'::geometry, 1)  ↩
    AS contains;
```

Output

```
t
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.25 ~(box2df,box2df)

~(box2df,box2df) — Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient une autre boîte de délimitation de précision flottante 2D (BOX2DF).

Synopsis

boolean ~(box2df A , box2df B);

Description

L'opérateur ~ renvoie TRUE si la boîte de délimitation 2D A contient la boîte de délimitation 2D B, en utilisant la précision float. Cela signifie que si A est un box2d (double précision), il sera converti en interne en une boîte de délimitation 2D de précision flottante (BOX2DF)



Note
Cet opérande est destiné à être utilisé en interne par les index BRIN, plus que par les utilisateurs.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Code

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(5, 5)) ~ ST_MakeBox2D(ST_Point(2, 2), ST_Point ↩
    (3, 3)) AS contains;
```

Output

```
t
```

Voir aussi

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.26 ~=

~= — Renvoie TRUE si la boîte de délimitation de A est la même que celle de B.

Synopsis

boolean **~=**(geometry A , geometry B);

Description

L'opérateur ~= renvoie TRUE si la boîte de délimitation de la géométrie/géographie A est la même que la boîte de délimitation de la géométrie/géographie B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Disponibilité : 1.5.0 comportement changé

Cette fonction prend en charge les surfaces Polyhedral.



AVERTISSEMENT
Cet opérateur a changé de comportement dans PostGIS 1.5, passant de la vérification de l'égalité géométrique réelle à la vérification de l'égalité du rectangle de délimitation. Pour compliquer les choses, le comportement de votre base de données dépend également du type de mise à niveau (hard ou soft) que vous avez effectué. Pour savoir quel est le comportement de votre base de données, vous pouvez exécuter la requête ci-dessous. Pour vérifier l'égalité réelle, utilisez ST_OrderingEquals ou ST_Equals.

Exemples

Code

```
SELECT 'LINESTRING(0 0,1 1)'::geometry ~= 'LINESTRING(0 1,1 0)'::geometry AS equality;
```

Output

```
t
```

Figure



Voir aussi

ST_Equals, ST_OrderingEquals, =

7.10.2 Opérateurs de distance

7.10.2.1 <->

<-> — Renvoie la distance en 2D entre A et B.

Synopsis

double precision **<->**(geometry A , geometry B);
double precision **<->**(geography A , geography B);

Description

The <-> operator returns the 2D distance between two geometries. Used in the "ORDER BY" clause provides index-assisted nearest-neighbor result sets with true KNN distance search for geometries, and distance on the sphere for geographies.



Note
Cet opérande utilise les index 2D GiST qui peuvent être disponibles sur les géométries. Il est différent des autres opérateurs qui utilisent des index spatiaux en ce sens que l'index spatial n'est utilisé que lorsque l'opérateur est dans la clause ORDER BY.



Note
L'index n'intervient que si l'une des géométries est une constante (pas dans une sous-requête/cte). Par exemple, 'SRID=3005;POINT(1011102 450541)'::geometry au lieu de a.geom

Reportez-vous à l'atelier PostGIS : La recherche du plus proche voisin pour un exemple détaillé.

Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography. Note for geography KNN is based on sphere rather than spheroid.

Changed: 2.2.0 -- Old hybrid-syntax workarounds may be slower once true KNN is available. See examples below.

Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types.

Exemples

Code

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d, edabbr, vaabbr
FROM va2005
ORDER BY d limit 10;
```

Output

```
d          | edabbr | vaabbr
-------------------+--------+--------
                 0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Puis la réponse brute KNN :

Code

```
SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
FROM va2005
ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry limit 10;
```

Output

```
        d          | edabbr | vaabbr
-------------------+--------+--------
                 0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
```

```
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Si vous exécutez "EXPLAIN ANALYZE" sur les deux requêtes, vous constaterez une amélioration des performances pour la seconde.

When a plan cannot produce exact distances directly, use a hybrid query to find the true nearest neighbors. First a CTE query using the index-assisted KNN, then an exact query to get correct ordering:

Code

```
WITH index_query AS (
  SELECT ST_Distance(geom, 'SRID=3005;POINT(1011102 450541)'::geometry) as d,edabbr, vaabbr
    FROM va2005
  ORDER BY geom <-> 'SRID=3005;POINT(1011102 450541)'::geometry LIMIT 100)
  SELECT *
    FROM index_query
  ORDER BY d limit 10;
```

Output

```
        d         | edabbr | vaabbr
------------------+--------+--------
                0 | ALQ    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Voir aussi

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — Renvoie la distance entre les trajectoires A et B à leur point d'approche le plus proche.

Synopsis

double precision **|=|**(geometry A , geometry B);

Description

The |=| operator returns the 3D distance between two trajectories (See ST_IsValidTrajectory). This is the same as ST_DistanceCPA but as an operator it can be used for doing nearest neighbor searches using an N-dimensional index.



Note
Cet opérande utilisera les index ND GiST qui peuvent être disponibles sur les géométries. Il est différent des autres opérateurs qui utilisent des index spatiaux en ce sens que l'index spatial n'est utilisé que lorsque l'opérateur est dans la clause ORDER BY.



Note
L'index n'intervient que si l'une des géométries est une constante (pas dans une sous-requête/cte). Par exemple, 'SRID=3005;LINESTRINGM(0 0 0,0 0 1)'::geometry au lieu de a.geom

Availability: 2.2.0.

Exemples

This example orders stored trajectories by their distance from a literal query trajectory.

Code

```
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350, 300, 0), ST_MakePointM(-410, 490, 0)) ↩
    , 10, 20)'

SELECT track_id, dist FROM (
  SELECT track_id,
    ST_DistanceCPA(tr, :qt) dist
  FROM trajectories
  ORDER BY tr |=| :qt
  LIMIT 5
) foo;
```

Output

```
track_id        dist
----------+------------------
      395 | 0.576496831518066
      380 |  5.06797130410151
      390 |  7.72262293958322
      385 |   9.8004461358071
      405 |  10.9534397988433
(5 rows)
```

Voir aussi

ST_DistanceCPA, ST_ClosestPointOfApproach, ST_IsValidTrajectory

7.10.2.3 <#>

<#> — Renvoie la distance 2D entre les boîtes de délimitation A et B.

Synopsis

double precision **<#>**(geometry A , geometry B);

Description

The <#> operator returns distance between two floating point bounding boxes, possibly reading them from a spatial index. Useful for doing nearest neighbor **approximate** distance ordering.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les géométries. Il est différent des autres opérateurs qui utilisent des index spatiaux en ce sens que l'index spatial n'est utilisé que lorsque l'opérateur est dans la clause ORDER BY.



Note
L'index n'intervient que si l'une des géométries est une constante, par exemple ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom) au lieu de g1.geom <#>.

Availability: 2.0.0.

Exemples

Code

```
SELECT *
FROM (
SELECT b.tlid, b.mtfcc,
    b.geom <#
> ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
        745787 2948499,745740 2948468,745712 2948438,
        745690 2948384,745677 2948319)', 2249) As b_dist,
        ST_Distance(b.geom, ST_GeomFromText('LINESTRING(746149 2948672,745954 2948576,
        745787 2948499,745740 2948468,745712 2948438,
        745690 2948384,745677 2948319)', 2249)) As act_dist
    FROM bos_roads As b
    ORDER BY b_dist, b.tlid
    LIMIT 100) As foo
    ORDER BY act_dist, tlid LIMIT 10;
```

Output

```
   tlid    | mtfcc |      b_dist      |     act_dist
-----------+-------+------------------+------------------
  85732027 | S1400 |                0 |                0
  85732029 | S1400 |                0 |                0
  85732031 | S1400 |                0 |                0
  85734335 | S1400 |                0 |                0
  85736037 | S1400 |                0 |                0
 624683742 | S1400 |                0 | 128.528874268666
  85719343 | S1400 | 260.839270432962 | 260.839270432962
  85741826 | S1400 | 164.759294123275 | 260.839270432962
  85732032 | S1400 |           277.75 | 311.830282365264
  85735592 | S1400 |           222.25 | 311.830282365264
(10 rows)
```

Voir aussi

ST_DWithin, ST_Distance, <->

7.10.2.4 <<->>

<<->> — Renvoie la distance n-D entre les géométries A et B ou les boîtes englobantes

Synopsis

double precision **<<->>**(geometry A , geometry B);

Description

L'opérateur <<−>> renvoie la distance n-D (euclidienne) entre les centroïdes des boîtes englobantes de deux géométries. Utile pour effectuer le classement par distance du plus proche voisin **approximate**.



Note
Cet opérande utilise les index GiST n-D qui peuvent être disponibles sur les géométries. Il est différent des autres opérateurs qui utilisent des index spatiaux en ce sens que l'index spatial n'est utilisé que lorsque l'opérateur est dans la clause ORDER BY.



Note
L'index n'intervient que si l'une des géométries est une constante (pas dans une sous-requête/cte). Par exemple, 'SRID=3005;POINT(1011102 450541)'::geometry au lieu de a.geom

Availability: 2.2.0.

Voir aussi

<->

7.11 Relations spatiales

7.11.1 Relations topologiques

7.11.1.1 ST_3DIntersects

ST_3DIntersects — Teste si deux géométries se croisent dans l'espace en 3D - uniquement pour les points, les lignes, les polygones, les surfaces polyédriques (aire)

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

Description

Overlaps, Touches, Within impliquent tous une intersection spatiale. Si l'un des éléments susmentionnés renvoie un résultat positif, les géométries se recoupent également dans l'espace. Disjoint implique faux pour l'intersection spatiale.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries.



Note
En raison des défaillances de la robustesse flottante, les géométries ne se croisent pas toujours comme on s'y attendrait après le traitement géométrique. Par exemple, le point le plus proche d'une géométrie sur une ligne peut ne pas se trouver sur la ligne. Pour ce type de problèmes, lorsqu'une distance d'un centimètre est considérée comme une intersection, utilisez ST_3DDWithin.

Modifié : 3.0.0 SFCGAL backend supprimé, GEOS backend supporte les TINs.

Disponibilité : 2.0.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1

Exemples de géométrie

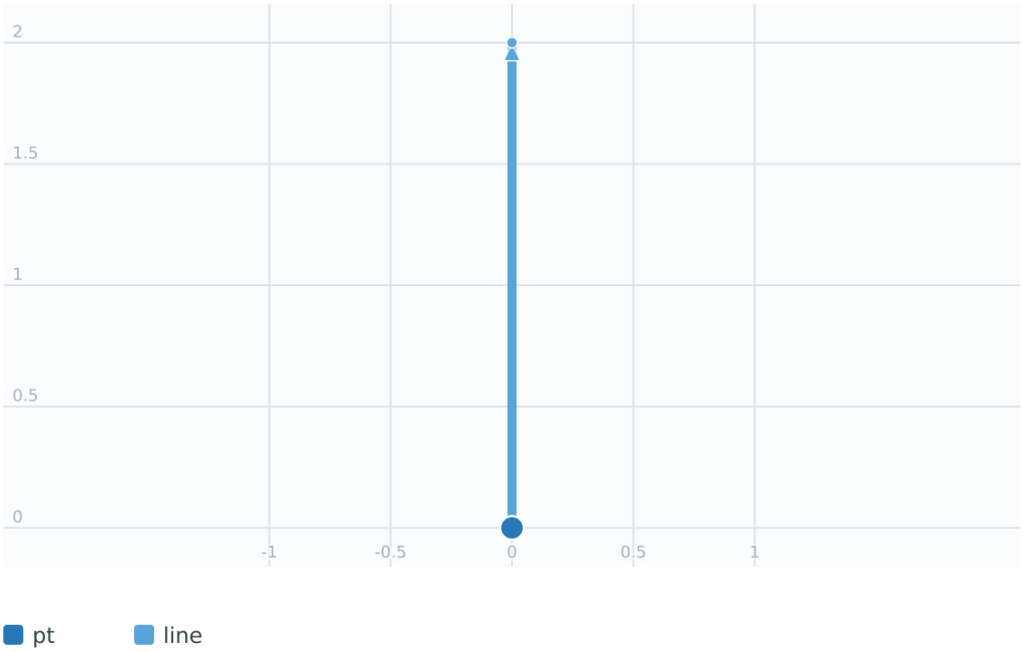
Code

```
SELECT ST_3DIntersects(pt, line), ST_Intersects(pt, line)
FROM (SELECT 'POINT(0 0 2)'::geometry As pt, 'LINESTRING (0 0 1,0 2 3)'::geometry As line)  ↩
    As foo;
```

Output

```
st_3dintersects | st_intersects
----------------+---------------
 f              | t
(1 row)
```

Figure



Exemples TIN

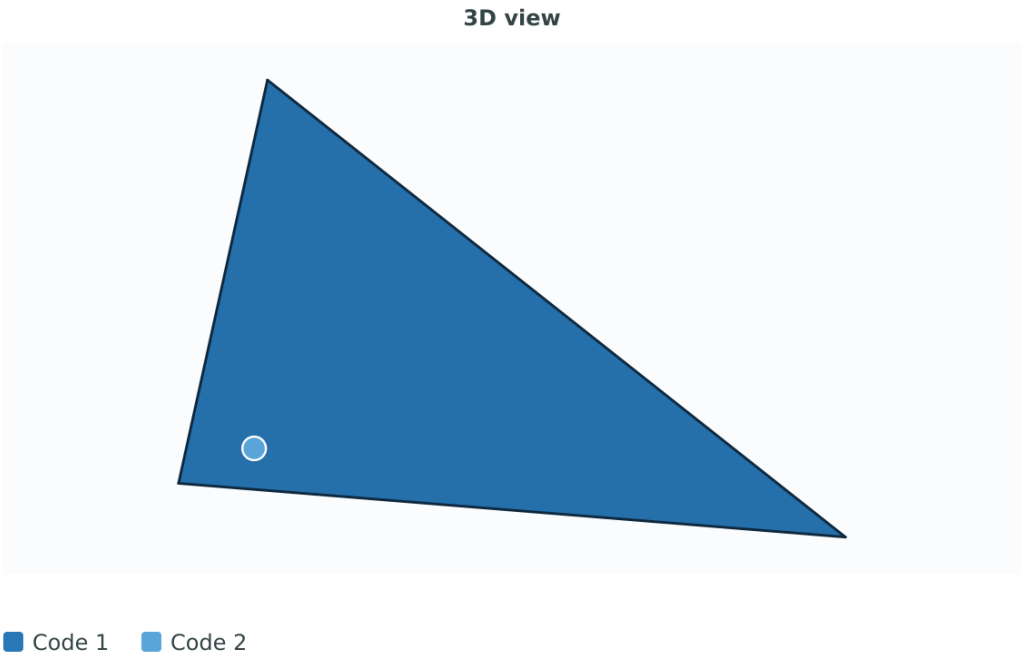
Code

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ↩
    geometry);
```

Output

```
t
```

Figure



Voir aussi

ST_3DDWithin, ST_Intersects

7.11.1.2 ST_Contains

ST_Contains — Tests si chaque point de B est situé dans A, et que leurs intérieurs ont un point commun

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

Description

Renvoie TRUE si la géométrie A contient la géométrie B. A contient B si et seulement si tous les points de B se trouvent à l'intérieur (c'est-à-dire à l'intérieur ou à la limite) de A (ou, de manière équivalente, si aucun point de B ne se trouve à l'extérieur de A), et si les intérieurs de A et de B ont au moins un point en commun.

En termes mathématiques : $ST_Contains(A, B) \Leftrightarrow (A \bigcap B = B) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$.

La relation contains est réflexive : toute géométrie se contient elle-même. (En revanche, dans le prédicat ST_ContainsProperly, une géométrie ne se contient*pas* elle-même correctement). La relation est antisymétrique : si `ST_Contains(A,B) = true` et `ST_Contains(B,A) = true`, alors les deux géométries doivent être topologiquement égales (`ST_Equals(A,B) = true`).

ST_Contains est le contraire de ST_Within. Ainsi, `ST_Contains(A,B) = ST_Within(B,A)`.



Note
Comme les intérieurs doivent avoir un point commun, une subtilité de la définition est que les polygones et les lignes ne contiennent *pas* de lignes et de points se trouvant entièrement dans leur limite. Pour plus de détails, voir Subtleties of OGC Covers, Contains, Within. Le prédicat ST_Covers fournit une relation plus inclusive.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter l'utilisation d'un index, utilisez la fonction `_ST_Contains`.

Effectué par le module GEOS

Amélioré : 2.3.0 Amélioration du court-circuit PIP étendu à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`



Important
N'utilisez pas cette fonction avec des géométries non valides. Vous obtiendrez des résultats inattendus.

NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - identique à within(geometry B, geometry A)

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.31

Exemples

`ST_Contains` renvoie `TRUE` dans les situations suivantes :

A LineString contains a MultiPoint whose points lie in its interior or boundary.

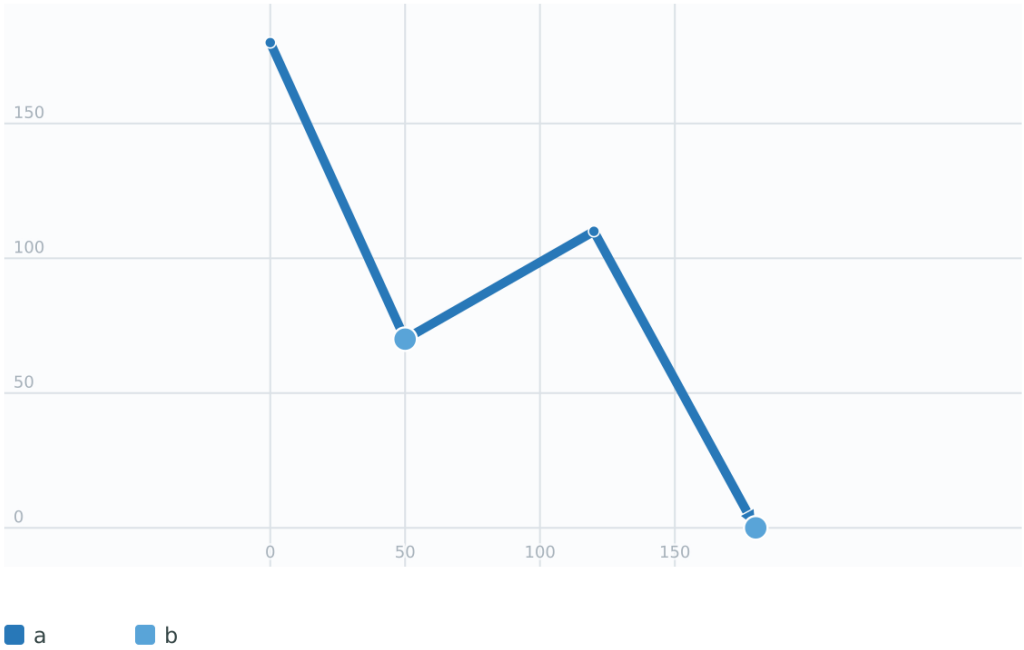
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'MULTIPOINT (60 80,190 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon contains a Point in its interior.

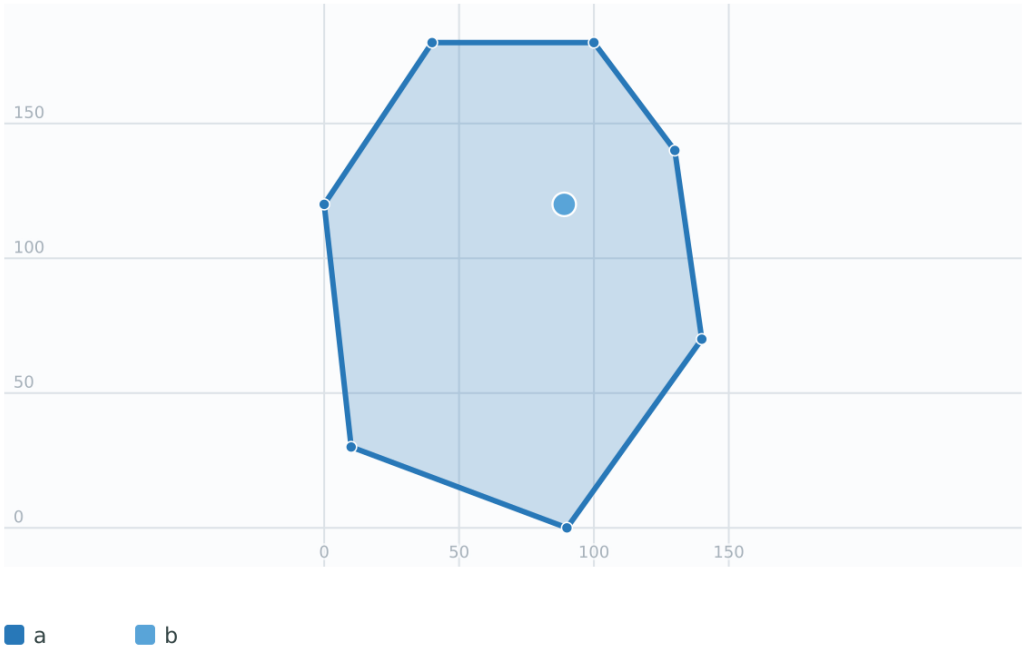
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'POINT (99 130)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon contains a LineString with interior points inside the Polygon.

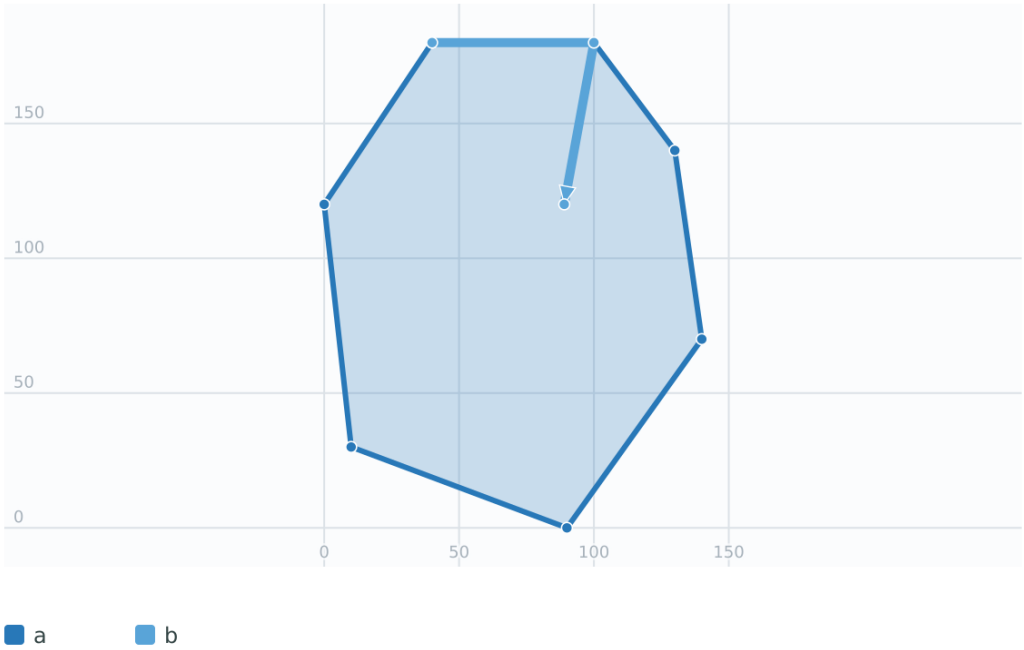
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (50 190,110 190,99 130)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon contains another Polygon in its interior.

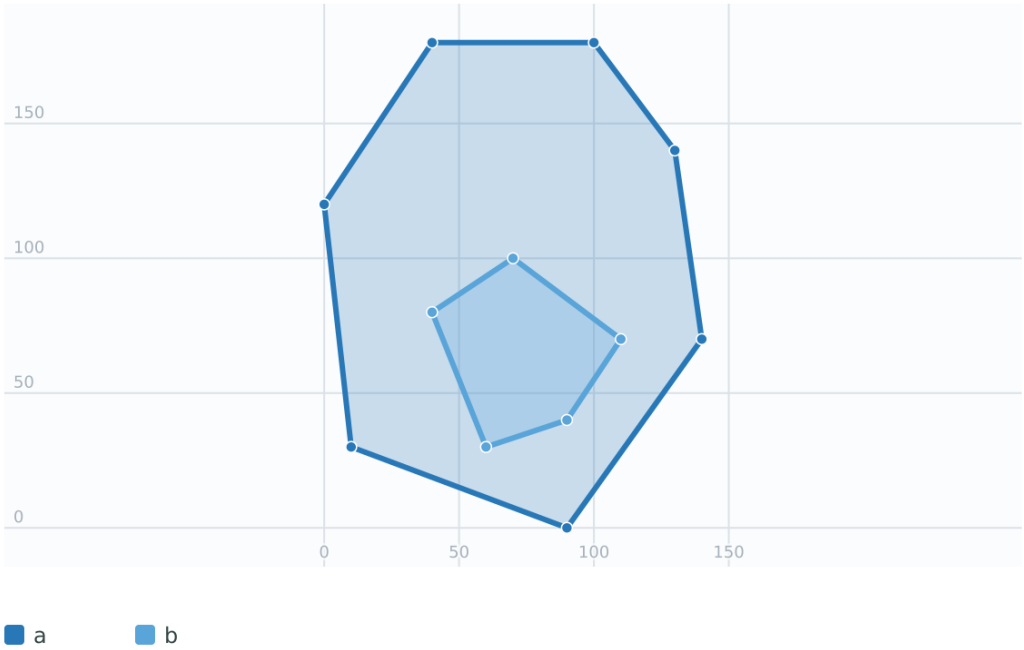
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'POLYGON ((70 40,100 50,120 80,80 110,50 90,70 40))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



La `ST_Contains` renvoie `FALSE` dans les situations suivantes :

A Polygon does not contain a MultiPoint when one point lies in a hole.

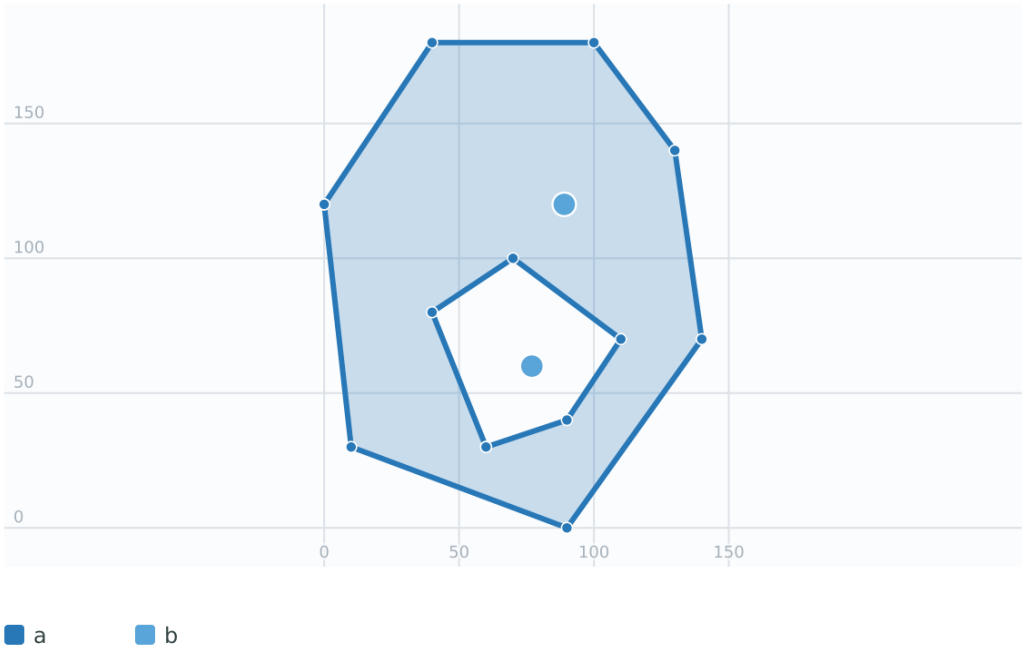
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130),(70 40,100 50,120 ↩
    80,80 110,50 90,70 40))'::geometry AS a,
  'MULTIPOINT (99 130,87 70)'::geometry AS b
) AS example;
```

Output

```
f
```

Figure



A Polygon does not contain a LineString extending outside it.

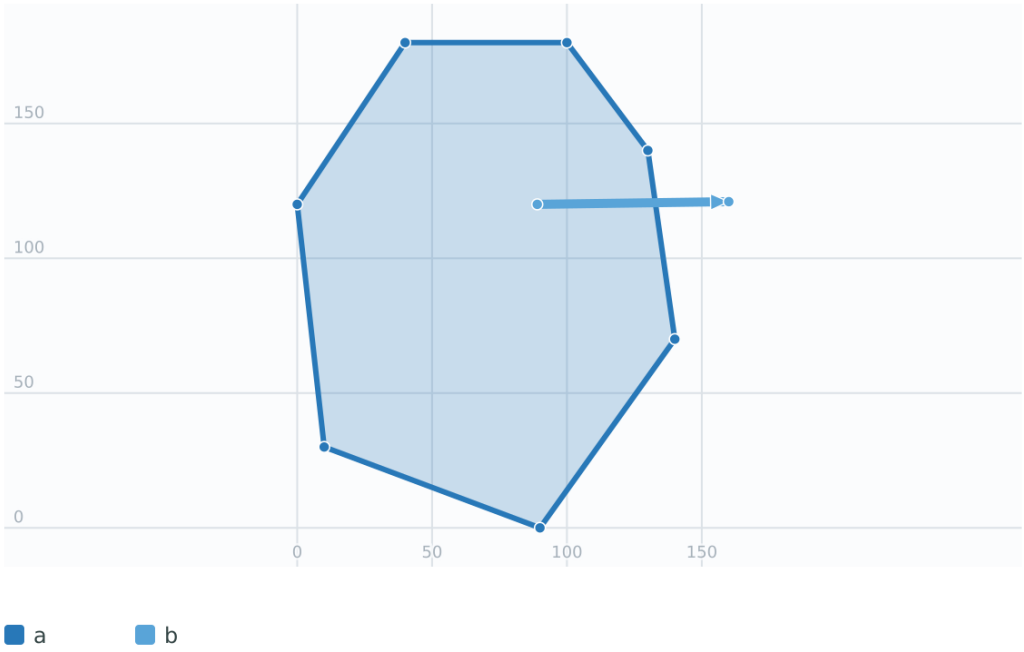
Code

```
SELECT ST_Contains(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (99 130,170 131)'::geometry AS b
) AS example;
```

Output

```
f
```

Figure



En raison de la condition d'intersection intérieure, la `ST_Contains` renvoie `FALSE` dans les situations suivantes (alors que la `ST_Covers` renvoie `TRUE`) :

A LineString does not contain a Point lying only on its boundary.

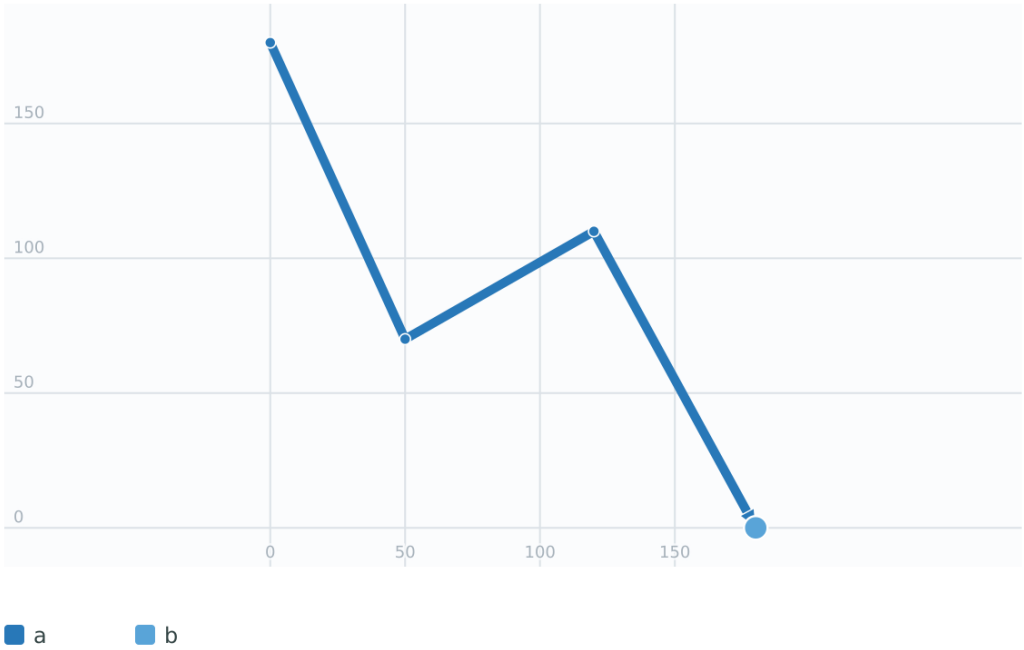
Code

```
SELECT ST_Contains(a, b), ST_Covers(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'POINT (190 10)'::geometry AS b
) AS example;
```

Output

```
f | t
```

Figure



A Polygon does not contain a LineString lying entirely on its boundary.

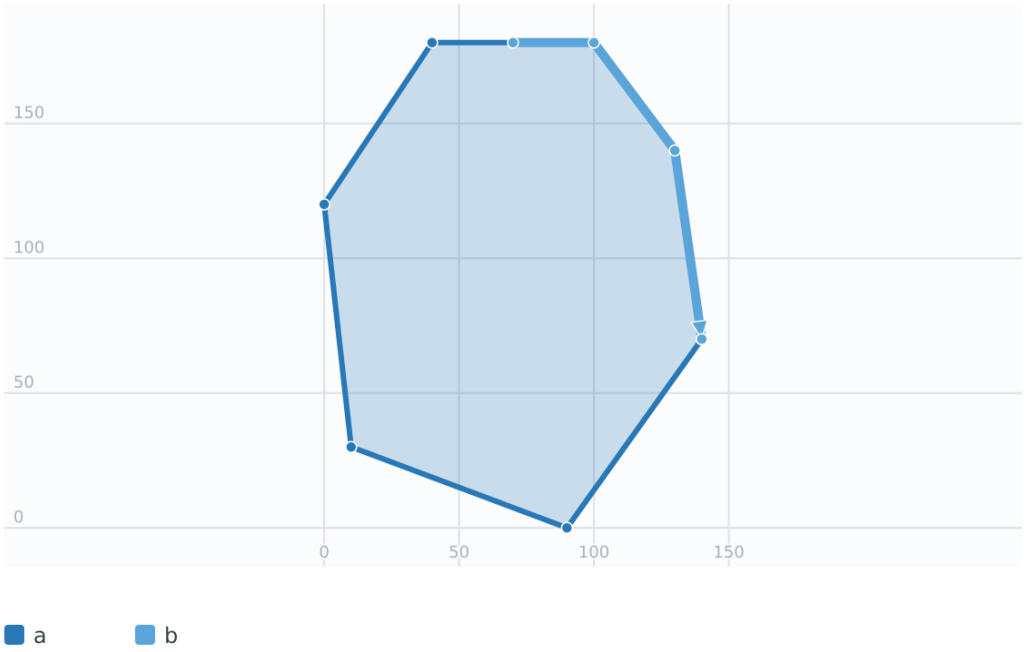
Code

```
SELECT ST_Contains(a, b), ST_Covers(a, b)
FROM (SELECT
  'POLYGON ((10 130,50 190,110 190,140 150,150 80,100 10,20 40,10 130))'::geometry AS a,
  'LINESTRING (80 190,110 190,140 150,150 80)'::geometry AS b
) AS example;
```

Output

```
f | t
```

Figure



This example tests a circle within another circle.

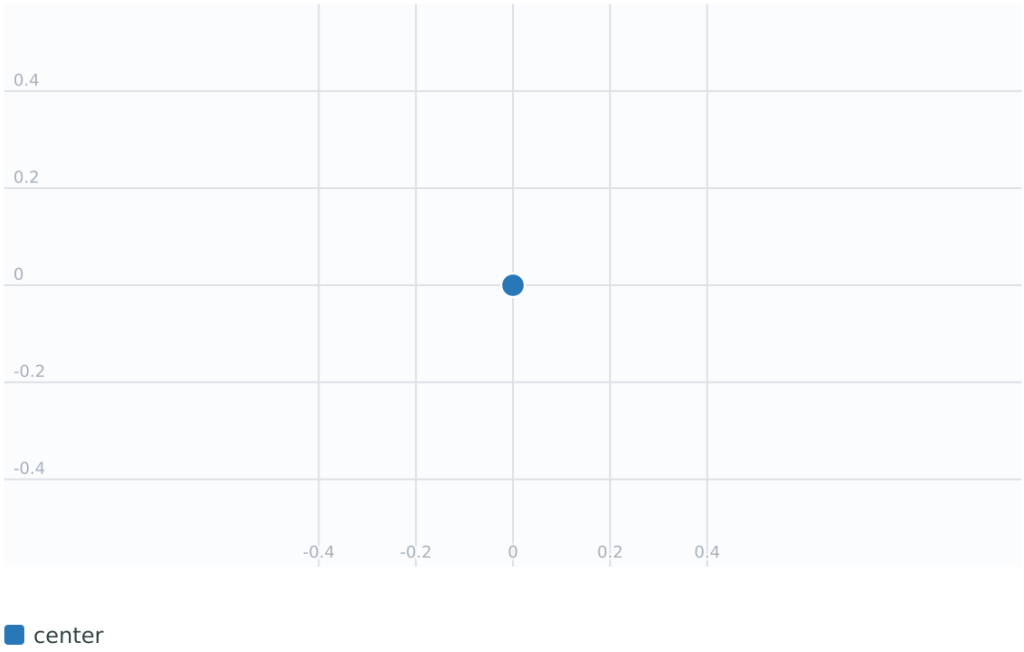
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_Contains(smallc, bigc) AS smallcontainsbig,
  ST_Contains(bigc, smallc) AS bigcontainssmall,
  ST_Contains(bigc, ST_Union(smallc, bigc)) AS bigcontainsunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) AS bigisunion,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) AS bigcoversexterior,
  ST_Contains(bigc, ST_ExteriorRing(bigc)) AS bigcontainsexterior
FROM circles;
```

Output

```
-[ RECORD 1 ]-----
smallcontainsbig   | f
bigcontainssmall   | t
bigcontainsunion   | t
bigisunion         | t
bigcoversexterior  | t
bigcontainsexterior | f
```

Figure



This example demonstrates the difference between contains and contains-properly predicates.

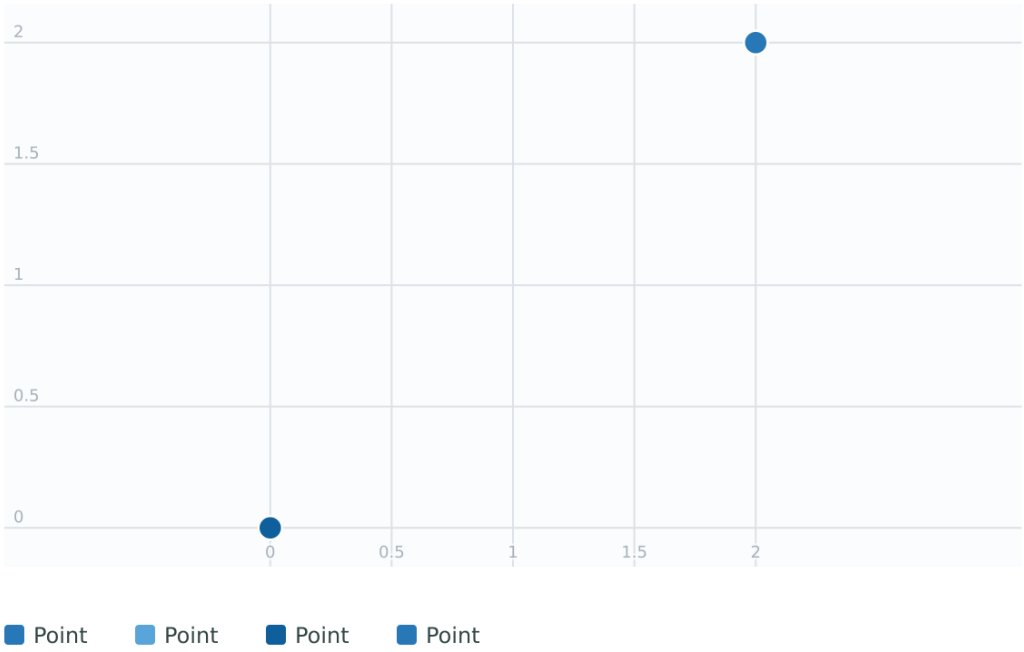
Code

```
SELECT
  ST_GeometryType(geom_a) AS geomtype,
  ST_Contains(geom_a, geom_a) AS acontainsa,
  ST_ContainsProperly(geom_a, geom_a) AS acontainspropa,
  ST_Contains(geom_a, ST_Boundary(geom_a)) AS acontainsba,
  ST_ContainsProperly(geom_a, ST_Boundary(geom_a)) AS acontainspropba
FROM (VALUES
    (ST_Buffer(ST_Point(1, 1), 5, 1)),
    (ST_MakeLine(ST_Point(1, 1), ST_Point(-1, -1))),
    (ST_Point(1, 1))
  ) AS foo(geom_a);
```

Output

```
geomtype     | acontainsa | acontainspropa | acontainsba | acontainspropba
-------------+------------+----------------+-------------+-----------------
ST_Polygon   | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point     | t          | t              | f           | f
```

Figure



Voir aussi

ST_Boundary, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Equals, ST_Within

7.11.1.3 ST_ContainsProperly

ST_ContainsProperly — Tests si chaque point de B se trouve à l'intérieur de A

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

Description

Retourne `vrai` si chaque point de B se trouve à l'intérieur de A (ou, de façon équivalente, si aucun point de B ne se trouve à la limite ou à l'extérieur de A).

En termes mathématiques : *ST_ContainsProperly(A, B)* $\Leftrightarrow$ *Int(A)* $\bigcap$ *B = B*

A contient B correctement si la matrice d'intersection DE-9IM pour les deux géométries correspond à [T**FF*FF*]

A ne se contient pas proprement, mais se contient.

Ce prédicat peut être utilisé pour calculer les intersections d'un ensemble de géométries avec une grande géométrie polygonale. L'intersection étant une opération assez lente, il peut être plus efficace d'utiliser containsProperly pour filtrer les géométries de test qui se trouvent entièrement à l'intérieur de la zone. Dans ce cas, on sait a priori que l'intersection correspond exactement à la géométrie d'essai originale.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter l'utilisation d'un index, utilisez la fonction `_ST_ContainsProperly`.



Note
L'avantage de ce prédicat par rapport à ST_Contains et ST_Intersects est qu'il peut être calculé plus efficacement, sans qu'il soit nécessaire de calculer la topologie en des points individuels.

Effectué par le module GEOS.

Disponibilité : 1.4.0

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`



Important
N'utilisez pas cette fonction avec des géométries non valides. Vous obtiendrez des résultats inattendus.

Exemples

This example tests a circle within another circle.

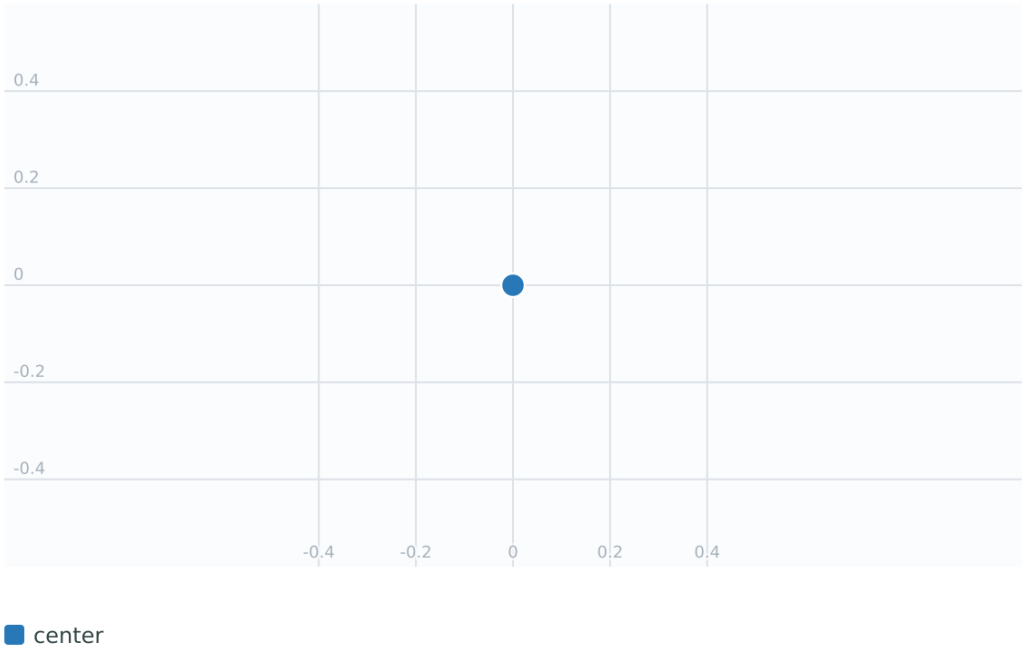
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_ContainsProperly(smallc, bigc) AS smallcontainspropbig,
  ST_ContainsProperly(bigc, smallc) AS bigcontainspropsmall,
  ST_ContainsProperly(bigc, ST_Union(smallc, bigc)) AS bigcontainspropunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) AS bigisunion,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) AS bigcoversexterior,
  ST_ContainsProperly(bigc, ST_ExteriorRing(bigc)) AS bigcontainsexterior
FROM circles;
```

Output

```
-[ RECORD 1 ]-----
smallcontainspropbig   | f
bigcontainspropsmall   | t
bigcontainspropunion   | f
bigisunion             | t
bigcoversexterior      | t
bigcontainsexterior    | f
```

Figure



This example demonstrates the difference between contains and contains-properly predicates.

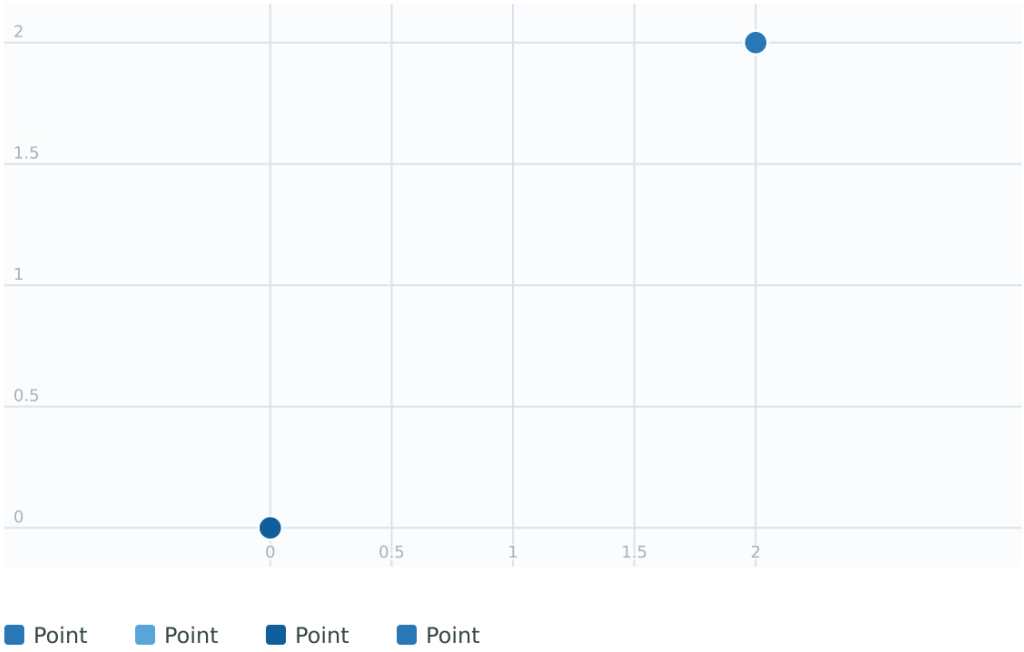
Code

```
WITH input(geom) AS (VALUES
  (ST_Buffer(ST_Point(1, 1), 5, 1)),
  (ST_MakeLine(ST_Point(1, 1), ST_Point(-1, -1))),
  (ST_Point(1, 1))
)
SELECT ST_GeometryType(geom) AS geomtype,
       ST_Contains(geom, geom) AS acontainsa,
       ST_ContainsProperly(geom, geom) AS acontainspropa,
       ST_Contains(geom, ST_Boundary(geom)) AS acontainsba,
       ST_ContainsProperly(geom, ST_Boundary(geom)) AS acontainspropba
FROM input;
```

Output

```
geomtype     | acontainsa | acontainspropa | acontainsba | acontainspropba
-------------+------------+----------------+-------------+-----------------
ST_Polygon   | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point     | t          | t              | f           | f
```

Figure



Voir aussi

ST_GeometryType, ST_Boundary, ST_Contains, ST_Covers, ST_CoveredBy, ST_Equals, ST_Relate, ST_Within

7.11.1.4 ST_CoveredBy

ST_CoveredBy — Tests si chaque point de A se trouve dans B

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

Description

Renvoie `true` si chaque point de la geometry/geography A se trouve à l'intérieur (c'est-à-dire coupe l'intérieur ou la limite) de la geometry/geography B. De manière équivalente, teste qu'aucun point de A ne se trouve à l'extérieur (dans l'extérieur) de B.

En termes mathématiques : *ST_CoveredBy(A, B)* $\Leftrightarrow A \cap B = A$

ST_CoveredBy est le contraire de ST_Covers. Ainsi, `ST_CoveredBy(A,B) = ST_Covers(B,A)`.

En général, cette fonction devrait être utilisée à la place de ST_Within, car elle a une définition plus simple qui n'a pas la particularité que "les limites ne sont pas à l'intérieur de leur géométrie".



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter l'utilisation d'un index, utilisez la fonction `_ST_CoveredBy`.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

Important
N'utilisez pas cette fonction avec des géométries non valides. Vous obtiendrez des résultats inattendus.

Effectué par le module GEOS

Disponibilité : 1.2.2

NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

Il ne s'agit pas d'une norme de l'OGC, mais Oracle l'a également.

Exemples

This example tests a circle covered by another circle.

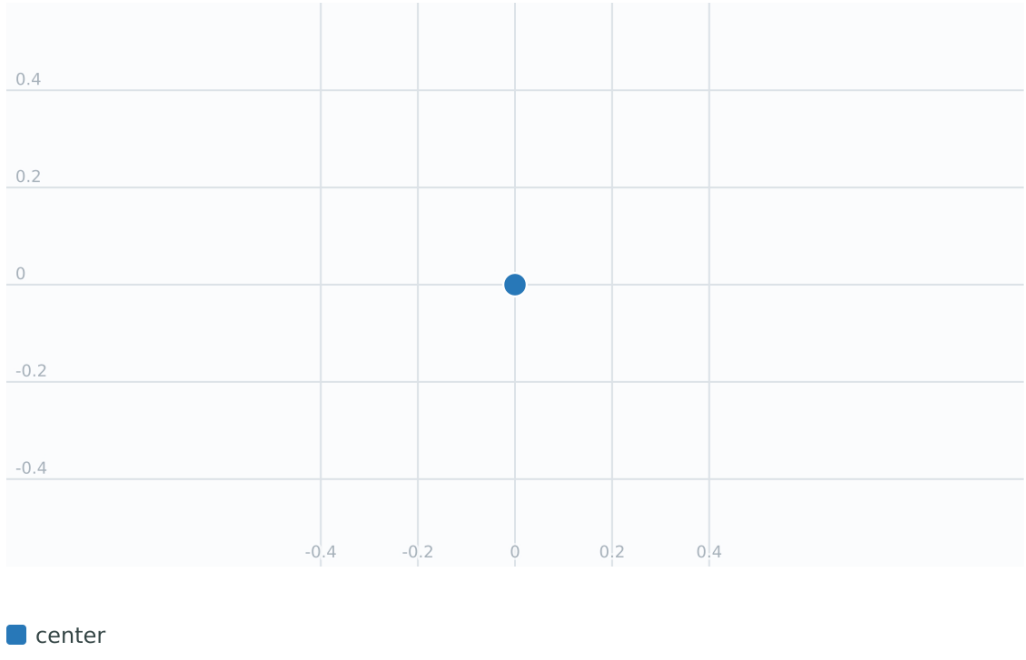
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_CoveredBy(smallc, smallc) AS smallinsmall,
  ST_CoveredBy(smallc, bigc) AS smallcoveredbybig,
  ST_CoveredBy(ST_ExteriorRing(bigc), bigc) AS exteriorcoveredbybig,
  ST_Within(ST_ExteriorRing(bigc), bigc) AS exteriorwithinbig
FROM circles;
```

Output

```
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exteriorwithinbig
--------------+-------------------+----------------------+-------------------
 t            | t                 | t                    | f
(1 row)
```

Figure



Voir aussi

ST_Contains, ST_Covers, ST_ExteriorRing, ST_Within

7.11.1.5 ST_Covers

ST_Covers — Tests si chaque point de B est situé dans A

Synopsis

boolean **ST_Covers**(geometry geomA, geometry geomB);
boolean **ST_Covers**(geography geogpolyA, geography geogpointB);

Description

Renvoie `true` si chaque point de la geometry/geography B se trouve à l'intérieur (c'est-à-dire qu'il coupe l'intérieur ou la limite) de la geometry/geography A. De façon équivalente, teste qu'aucun point de B ne se trouve à l'extérieur (dans l'extérieur) de A.

En termes mathématiques : *ST_Covers(A, B)* $\Leftrightarrow A \bigcap B = B$

ST_Covers est le contraire de ST_CoveredBy. Ainsi, `ST_Covers(A,B) = ST_CoveredBy(B,A)`.

En général, cette fonction devrait être utilisée à la place de ST_Contains, car elle a une définition plus simple qui n'a pas la particularité que "les géométries ne contiennent pas leur frontière".



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter l'utilisation d'un index, utilisez la fonction `_ST_Covers`.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`



Important
N'utilisez pas cette fonction avec des géométries non valides. Vous obtiendrez des résultats inattendus.

Effectué par le module GEOS

Amélioration : 2.4.0 Ajout de la prise en charge des polygones dans les polygones et des lignes dans les polygones pour le type geography

Amélioration : 2.3.0 Amélioration du court-circuit PIP pour la géométrie étendue à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones.

Disponibilité : 1.5 - le support de la geography a été introduit.

Disponibilité : 1.2.2

NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

Il ne s'agit pas d'une norme de l'OGC, mais Oracle l'a également.

Exemples

Exemple géométrique

This example tests a circle covering another circle.

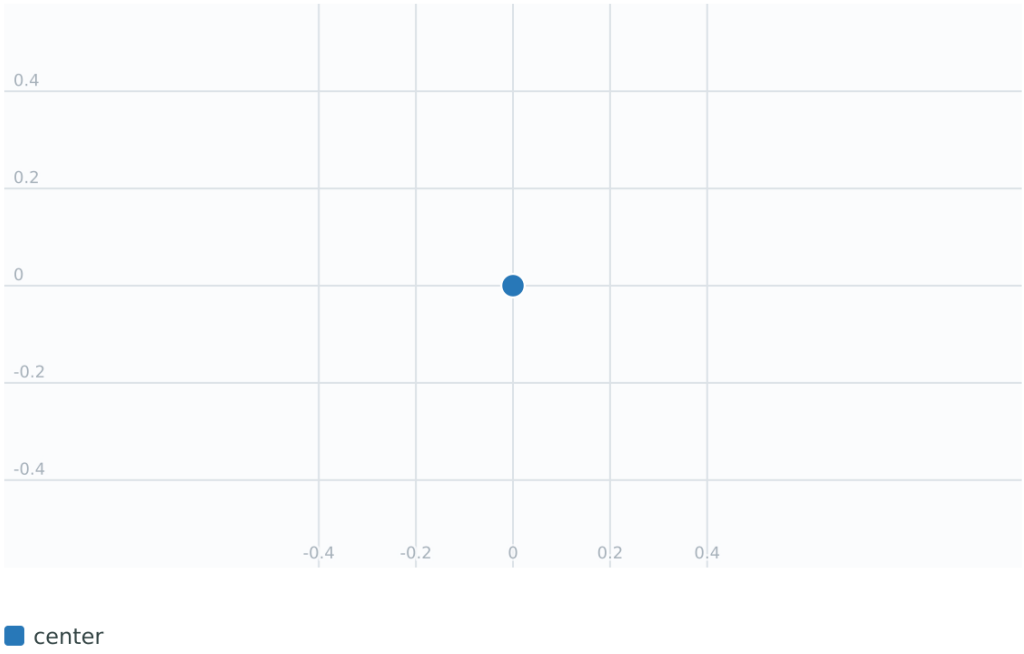
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 10) AS smallc,
         ST_Buffer(center, 20) AS bigc
  FROM (SELECT 'POINT(1 2)'::geometry AS center) AS p
)
SELECT
  ST_Covers(smallc, smallc) AS smallinsmall,
  ST_Covers(smallc, bigc) AS smallcoversbig,
  ST_Covers(bigc, ST_ExteriorRing(bigc)) AS bigcoversexterior,
  ST_Contains(bigc, ST_ExteriorRing(bigc)) AS bigcontainsexterior
FROM circles;
```

Output

```
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-------------+----------------+-------------------+--------------------
 t           | f              | t                 | f
(1 row)
```

Figure



Geography example

This example compares a point with a 300-meter buffer against a point and against a 10-meter buffer around that point.

Code

```
SELECT ST_Covers(geog_poly, geog_pt) As poly_covers_pt,
  ST_Covers(ST_Buffer(geog_pt, 10), geog_pt) As buff_10m_covers_cent
  FROM (SELECT ST_Buffer('SRID=4326;POINT(-99.327 31.4821)'::geography, 300) As geog_poly,
        'SRID=4326;POINT(-99.33 31.483)'::geography As geog_pt ) As foo;
```

Output

```
poly_covers_pt | buff_10m_covers_cent
---------------+--------------------
 f             | t
```

Voir aussi

ST_Contains, ST_CoveredBy, ST_Within

7.11.1.6 ST_Crosses

ST_Crosses — Teste si deux géométries ont en commun certains points intérieurs, mais pas tous

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Description

Compare deux objets géométriques et renvoie `true` si leur intersection "se croise spatialement", c'est-à-dire que les géométries ont certains points intérieurs en commun, mais pas tous. L'intersection des intérieurs des géométries doit être non vide et doit avoir une dimension inférieure à la dimension maximale des deux géométries d'entrée, et l'intersection des deux géométries ne doit pas être égale à l'une ou l'autre géométrie. Sinon, il renvoie `false`. La relation entre les croix est symétrique et irréflexive.

En termes mathématiques : $ST_Crosses(A, B) \Leftrightarrow (dim(\ Int(A) \bigcap Int(B)\ ) < max(\ dim(\ Int(A)\ ), dim(\ Int(B)\ )\ )) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$

Les géométries se croisent si leur matrice d'intersection DE-9IM correspond :
— `T*T******` pour les situations point/ligne, point/zone et ligne/zone
— `T*****T**` pour les situations Ligne/Point, Zone/Point et Zone/Ligne
— `0********` pour les situations ligne/ligne
— le résultat est `false` pour les situations Point/Point et Area/Area



Note
La spécification OpenGIS Simple Features définit ce prédicat uniquement pour les situations Point/Ligne, Point/Zone, Ligne/Ligne et Ligne/Zone. Le STC / GEOS étend la définition pour qu'elle s'applique également aux situations Ligne/-Point, Zone/Point et Zone/Ligne. Cela rend la relation symétrique.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.29

Exemples

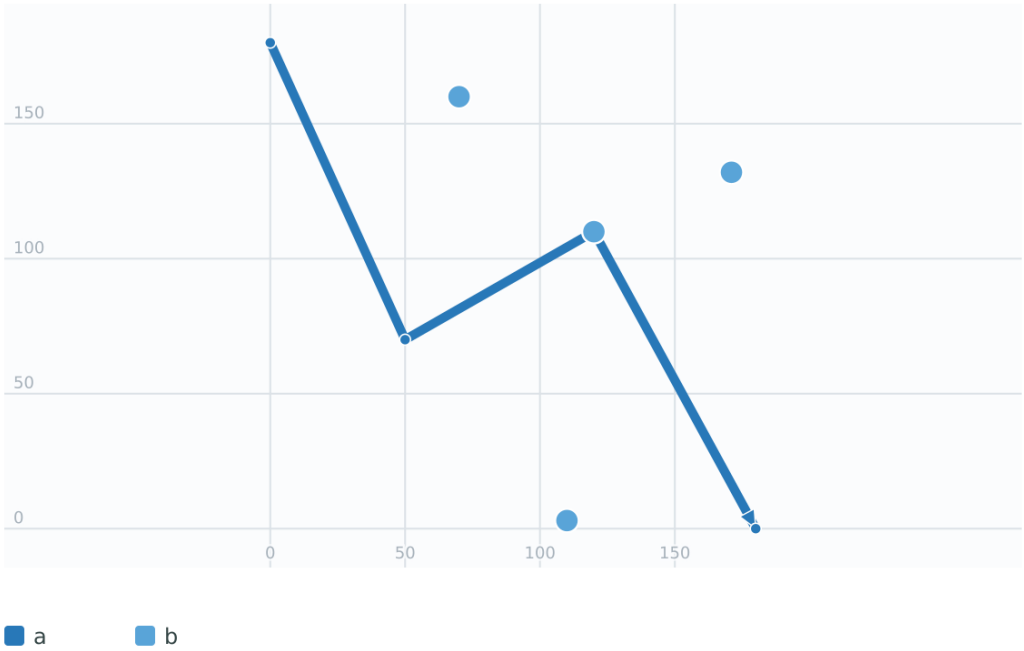
Les situations suivantes renvoient toutes `true`.

A LineString crosses a MultiPoint when some, but not all, points lie on the line.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'MULTIPOINT (80 170,120 13,130 120,181 142)'::geometry AS b
) AS example;
```

Figure

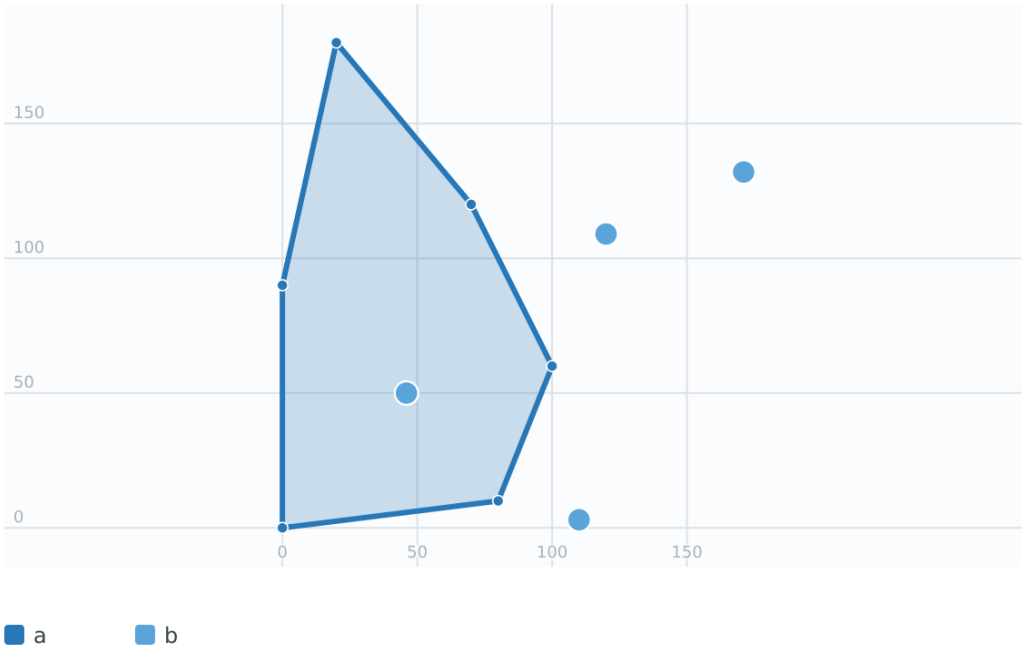


A Polygon crosses a MultiPoint when some, but not all, points lie in its interior.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'POLYGON ((30 190,10 100,10 10,90 20,110 70,80 130,30 190))'::geometry AS a,
  'MULTIPOINT (56 60,120 13,130 119,181 142)'::geometry AS b
) AS example;
```

Figure

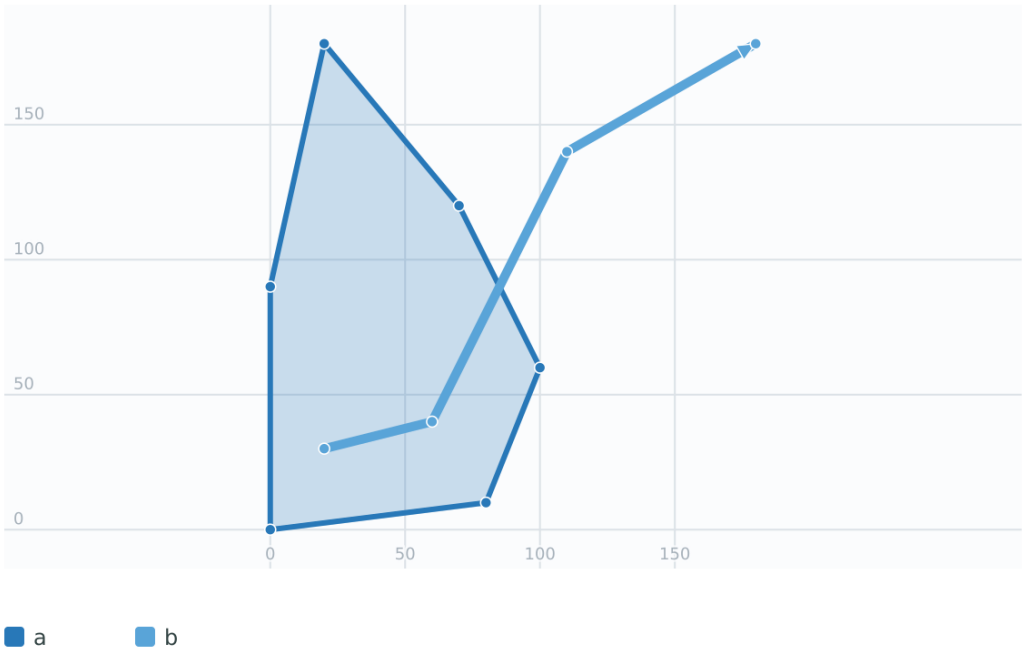


A LineString crosses a Polygon when part of the line lies inside and part outside.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'POLYGON ((30 190,10 100,10 10,90 20,110 70,80 130,30 190))'::geometry AS a,
  'LINESTRING (30 40,70 50,120 150,190 190)'::geometry AS b
) AS example;
```

Figure

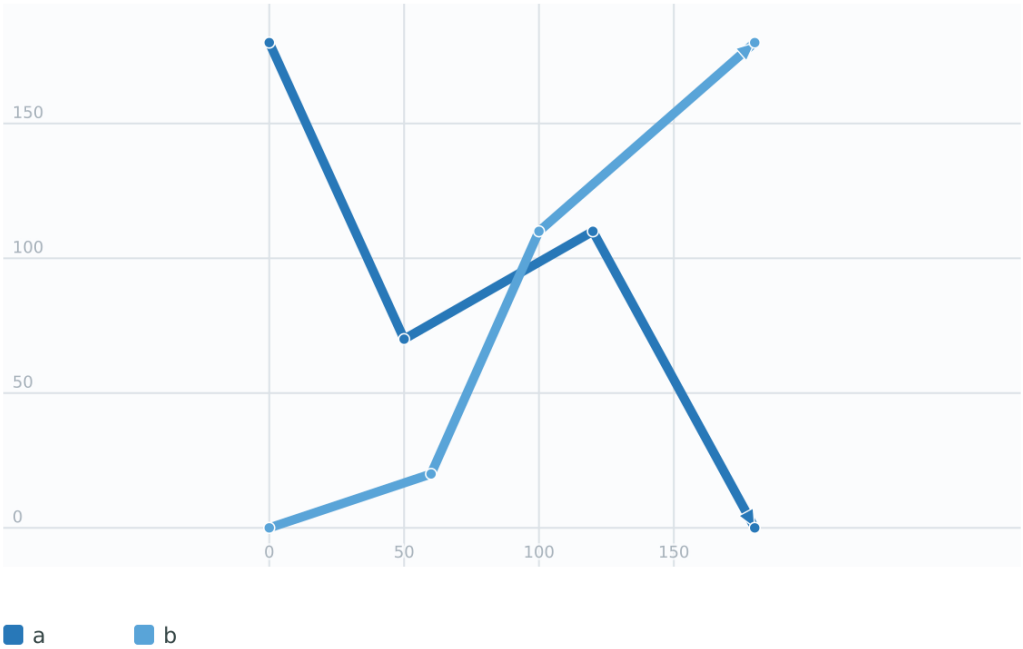


Two LineStrings cross when they share an interior point.

Code

```
SELECT ST_Crosses(a, b)
FROM (SELECT
  'LINESTRING (10 190,60 80,130 120,190 10)'::geometry AS a,
  'LINESTRING (10 10,70 30,110 120,190 190)'::geometry AS b
) AS example;
```

Figure



Prenons le cas d'un utilisateur qui dispose de deux tables : une table de routes et une table d'autoroutes.

Code

```
CREATE TABLE roads (
  id serial NOT NULL,
  geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↩
    road_id)
);
```

Code

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_gem geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ↩
    road_id)
);
```

Pour obtenir une liste des routes qui traversent une autoroute, utilisez une requête similaire à :

Code

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

Voir aussi

ST_Contains, ST_Overlaps

7.11.1.7 ST_Disjoint

ST_Disjoint — Teste si deux géométries n'ont pas de points communs

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

Description

Renvoie `true` si deux géométries sont disjointes. Les géométries sont disjointes si elles n'ont aucun point en commun.

Si une autre relation spatiale est vraie pour une paire de géométries, celles-ci ne sont pas disjointes. La disjonction implique que ST_Intersects est fausse.

En termes mathématiques : $ST_Disjoint(A, B) \Leftrightarrow A \bigcap B = \varnothing$

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

Effectué par le module GEOS



Note
Cet appel de fonction n'utilise pas d'index. Un prédicat négatif ST_Intersects peut être utilisé comme une alternative plus performante qui utilise des index : `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`



Note
NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')

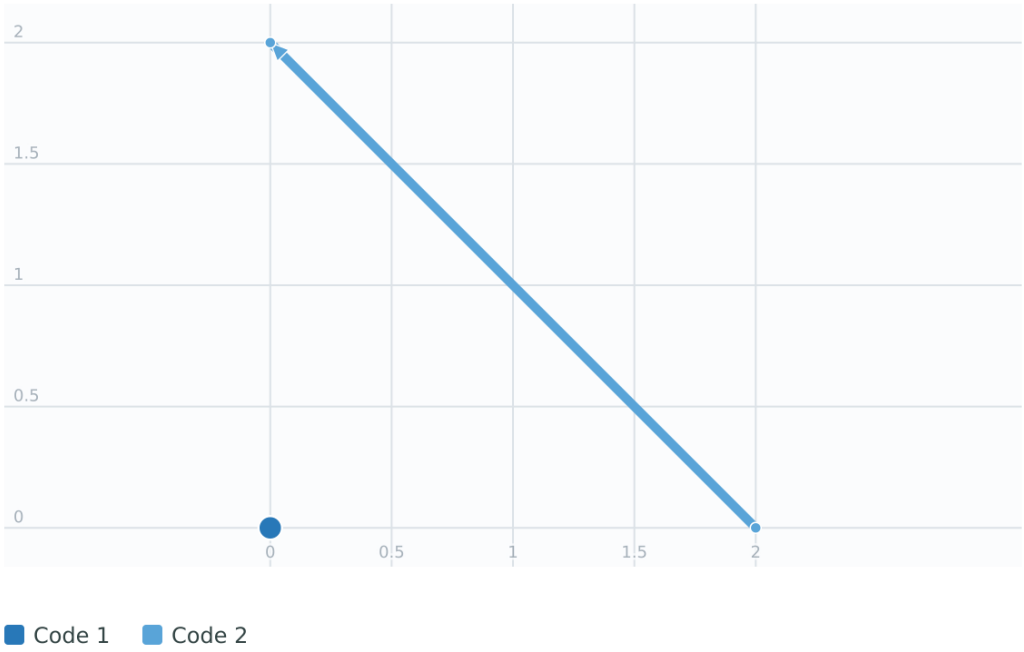
Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.26

Exemples

Code

```
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

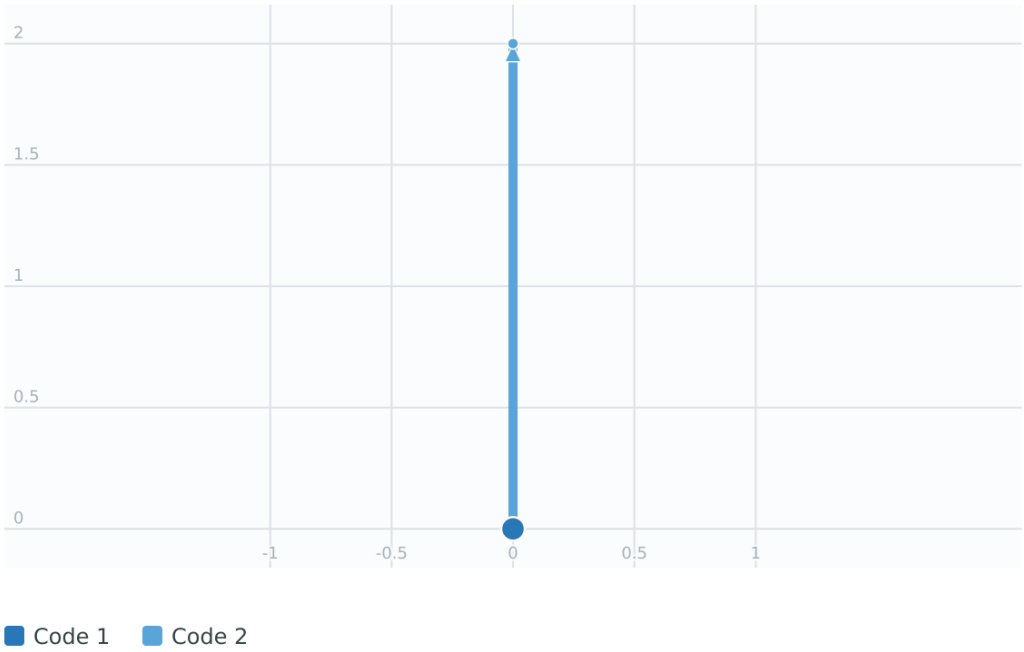
Figure



Code

```
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Figure



Voir aussi

ST_Intersects

7.11.1.8 ST_Equals

ST_Equals — Teste si deux géométries comprennent le même ensemble de points

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

Description

Renvoie `true` si les géométries données sont "topologiquement égales". Utilisez ceci pour une "meilleure" réponse que "=". L'égalité topologique signifie que les géométries ont la même dimension et que leurs ensembles de points occupent le même espace. Cela signifie que l'ordre des sommets peut être différent dans des géométries topologiquement égales. Pour vérifier que l'ordre des points est cohérent, utilisez ST_OrderingEquals (il convient de noter que ST_OrderingEquals est un peu plus strict que la simple vérification de l'ordre des points).

En termes mathématiques : *ST_Equals(A, B)* ⇔ *A = B*

La relation suivante est valable : *ST_Equals(A, B)* ⇔ *ST_Within(A,B)* ∧ *ST_Within(B,A)*

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.24

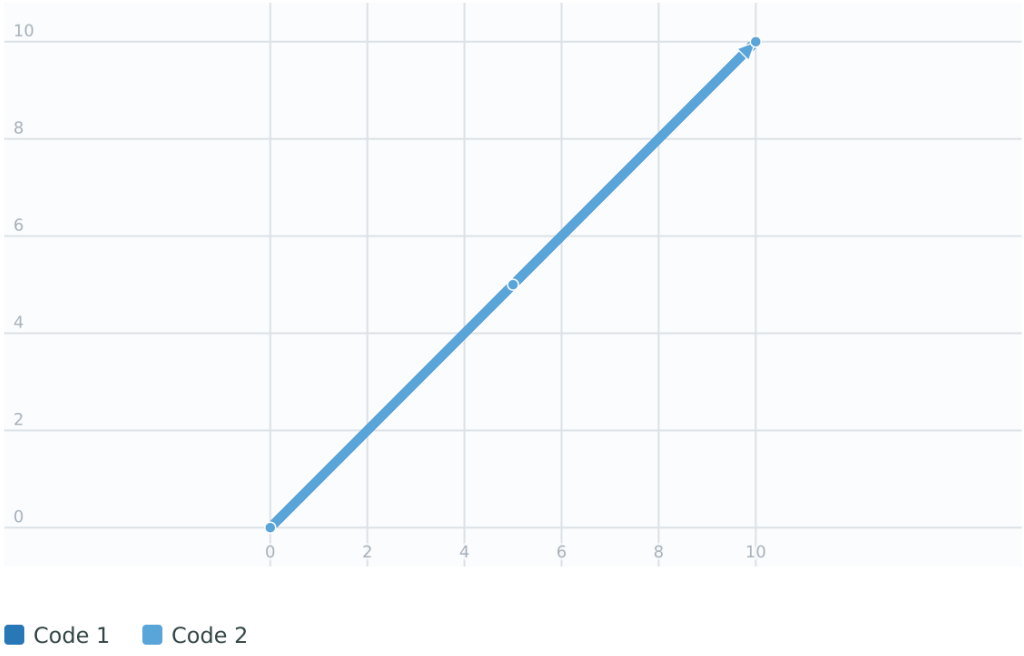
Modifié : 2.2.0 Retourne vrai même pour les géométries invalides si elles sont binairement égales

Exemples

Code

```
SELECT ST_Equals('LINESTRING(0 0,10 10)',
'LINESTRING(0 0,5 5,10 10)');
```

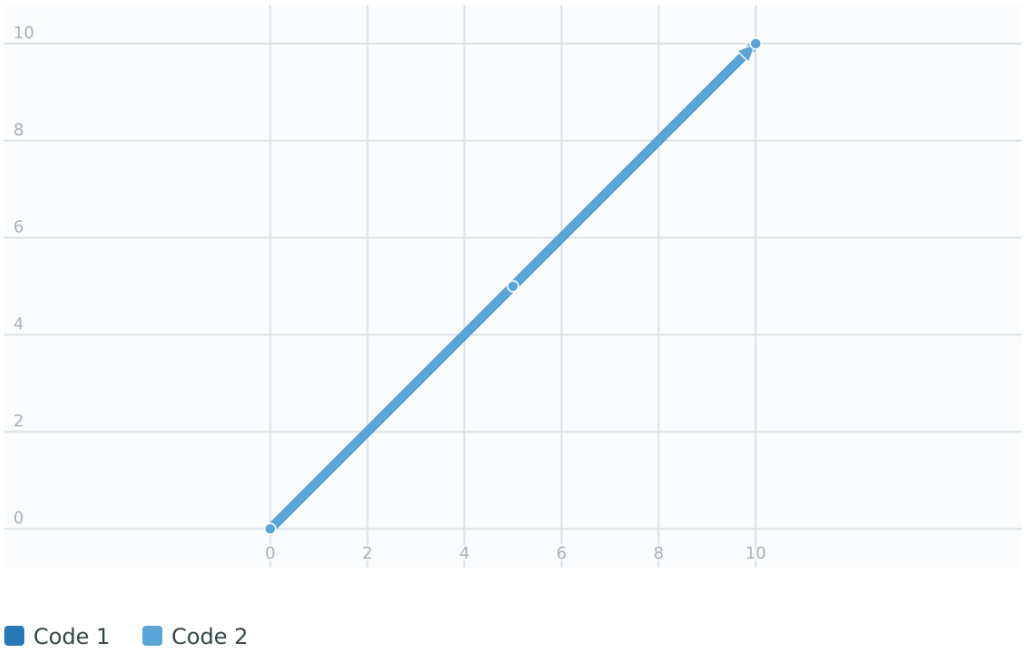
Figure



Code

```
SELECT ST_Equals(ST_Reverse('LINESTRING(0 0,10 10)'),
'LINESTRING(0 0,5 5,10 10)');
```

Figure



Voir aussi

ST_IsValid, ST_OrderingEquals, ST_Reverse, ST_Within

7.11.1.9 ST_Intersects

ST_Intersects — Teste si deux géométries se croisent (elles ont au moins un point en commun)

Synopsis

boolean **ST_Intersects**(geometry geomA , geometry geomB);
boolean **ST_Intersects**(geography geogA , geography geogB);

Description

Renvoie `true` si deux géométries se croisent. Les géométries se croisent si elles ont un point commun.

Pour les objets de type geography, une tolérance de distance de 0,00001 mètre est utilisée (les points très proches sont donc considérés comme se croisant).

En termes mathématiques : *ST_Intersects(A, B)* $\Leftrightarrow A \bigcap B \neq \varnothing$

Les géométries se croisent si leur matrice d'intersection DE-9IM correspond à l'un des éléments suivants :

— `T********`
— `*T*******`
— `***T*****`

— ****T****

L'intersection spatiale est impliquée par tous les autres tests de relations spatiales, à l'exception de ST_Disjoint, qui teste que les géométries ne se croisent PAS.



Note

Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries.

Modifié : 3.0.0 La version SFCGAL a été supprimée et la prise en charge native des TINS 2D a été ajoutée.

Amélioré : 2.5.0 Supporte GEOMETRYCOLLECTION.

Amélioré : 2.3.0 Amélioration du court-circuit PIP étendu à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones.

Effectuée par le module GEOS (pour geometry), la geography est native

Disponibilité : la version 1.5 a introduit la prise en charge du type geography.



Note

Pour geography, cette fonction a une tolérance de distance d'environ 0,00001 mètre et utilise la sphère plutôt que le calcul du sphéroïde.



Note

NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2))

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.27

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples de géométrie

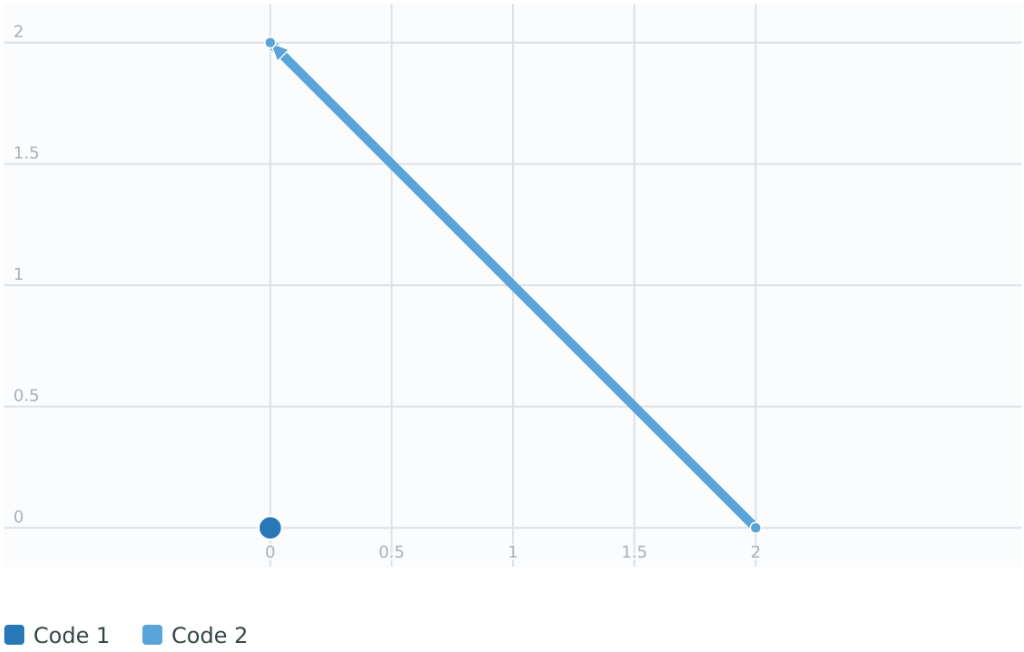
Code

```
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
f
```

Figure



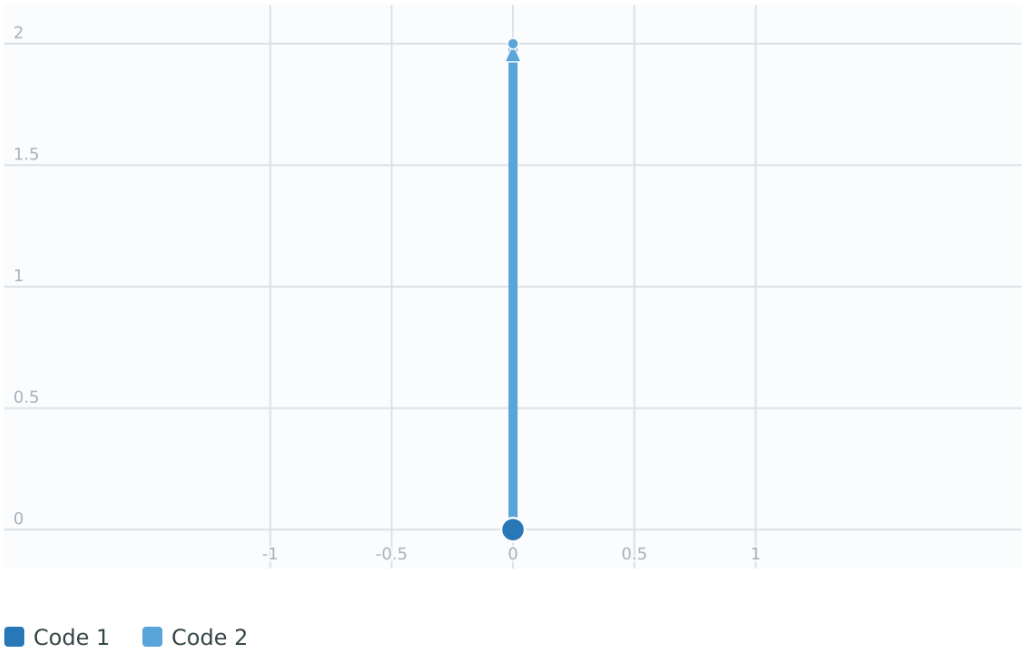
Code

```
SELECT ST_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Output

```
t
```

Figure



This example looks up rows in a table; for faster lookup, the geometry column should have a GiST index.

Code

```
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707 ↩
    52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
```

Output

```
id | name
----+-------
  2 | Minsk
(1 row)
```

Exemples géographiques

Code

```
SELECT ST_Intersects('SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'::geography,
'SRID=4326;POINT(-43.23456 72.4567772)'::geography
);
```

Output

```
t
```

Voir aussi

&&, ST_3DIntersects, ST_Disjoint

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — Renvoie un nombre indiquant le comportement de croisement de deux LineStrings

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

Description

En ayant 2 lignes en entrée, renvoie un nombre entier entre -3 et 3 indiquant le type de croisement qui existe entre elles. 0 indique qu'il n'y a pas de croisement. Cette fonction n'est prise en charge que pour les `LINESTRING`s.

Le numéro de croisement a la signification suivante :

— 0 : LIGNE SANS CROISEMENT
— -1 : LIGNE CROISÉE À GAUCHE
— -1 : LIGNE CROISÉE À DROITE
— -2 : LIGNE MULTICROISEMENT EXTRÉMITÉ GAUCHE
— -2 : LIGNE MULTICROISEMENT EXTRÉMITÉ DROITE
— -3 : LIGNE MULTICROISEMENT FIN IDENTIQUE AU PREMIER GAUCHE
— -3 : LIGNE MULTICROISEMENT FIN IDENTIQUE AU PREMIER DROITE

Disponibilité : 1.4

Exemples

Exemple: LIGNE CROISEE A GAUCHE et LIGNE CROISEE A DROITE

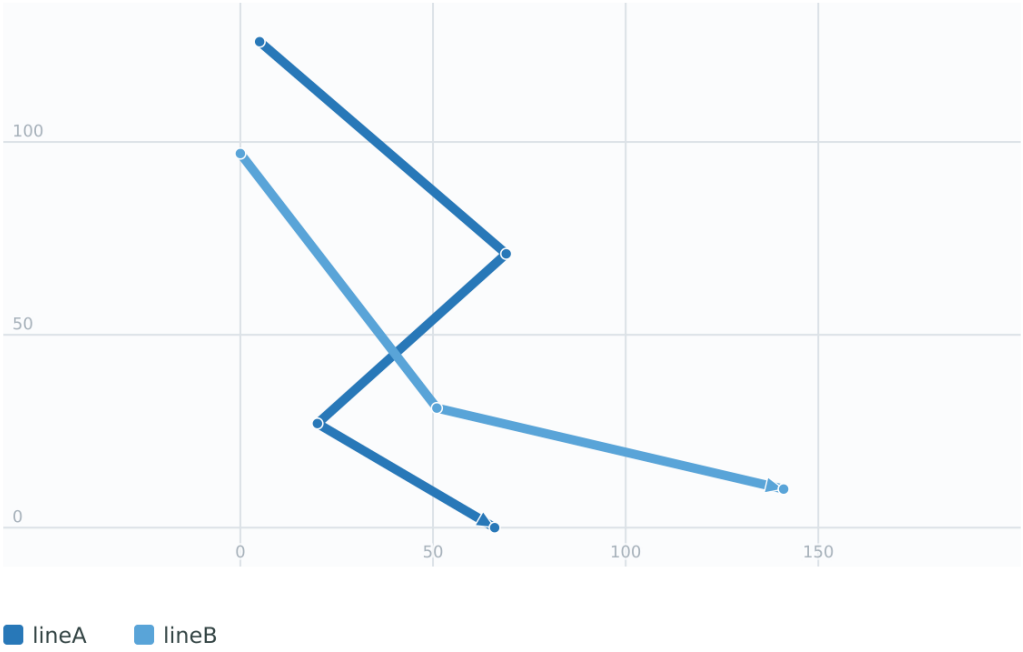
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  'LINESTRING(25 169,89 114,40 70,86 43)'::geometry As lineA,
  'LINESTRING (20 140,71 74,161 53)'::geometry As lineB
  ) As foo;
```

Output

```
A_cross_B | B_cross_A
----------+----------
       -1 |         1
```

Figure



Exemple: LIGNE MULTICROISEMENT FIN IDENTIQUE AU PREMIER GAUCHE et LIGNE MULTICROISEMENT FIN IDENTIQUE AU PREMIER DROITE

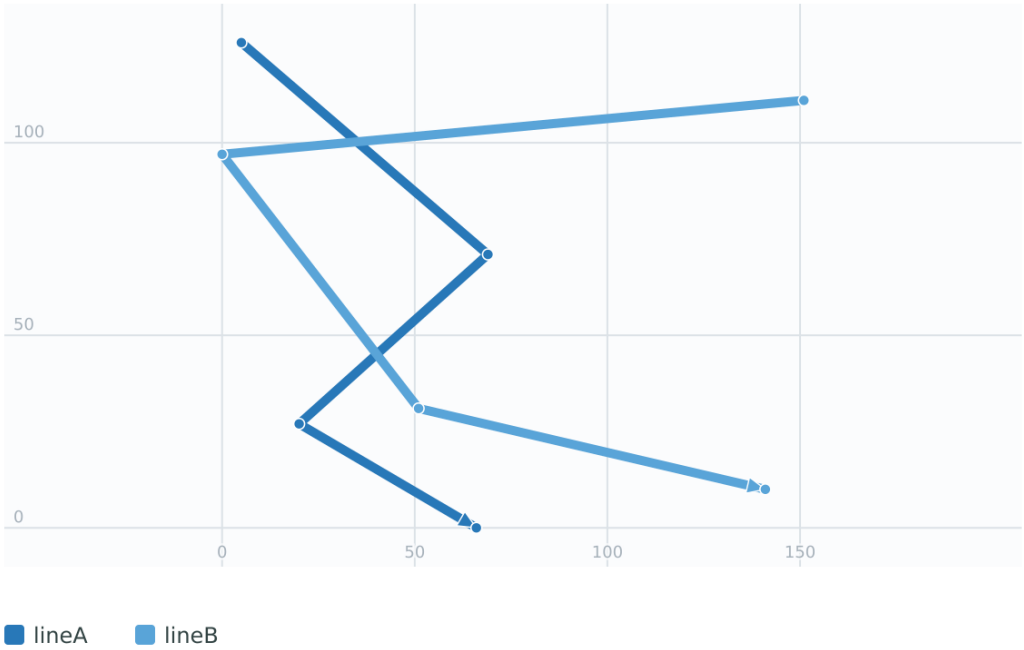
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 'LINESTRING(25 169,89 114,40 70,86 43)'::geometry As lineA,
 'LINESTRING(171 154,20 140,71 74,161 53)'::geometry As lineB
  ) As foo;
```

Output

```
A_cross_B | B_cross_A
----------+----------
        3 |        -3
```

Figure



Exemple: LIGNE MULTICROISEMENT EXTRÉMITÉ GAUCHE et LIGNE MULTICROISEMENT EXTRÉMITÉ DROITE

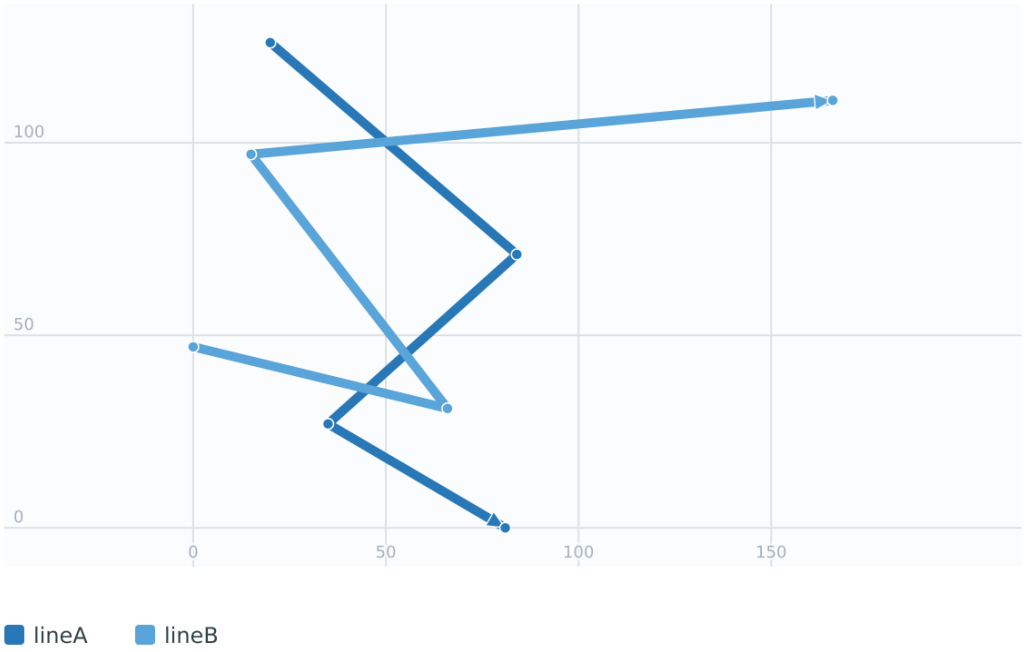
Code

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  'LINESTRING(25 169,89 114,40 70,86 43)'::geometry As lineA,
  'LINESTRING(5 90,71 74,20 140,171 154)'::geometry As lineB
  ) As foo;
```

Output

```
A_cross_B | B_cross_A
----------+----------
       -2 |         2
```

Figure



Exemple: Trouve toutes les rues qui se croisent

Code

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
  FROM streets s1 CROSS JOIN streets s2
        ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

Voir aussi

ST_Crosses

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — Teste si deux géométries représentent la même géométrie et ont des points dans le même ordre directionnel

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

Description

ST_OrderingEquals compare deux géométries et renvoie t (TRUE) si les géométries sont égales et si les coordonnées sont dans le même ordre ; sinon, il renvoie f (FALSE).

Note
Cette fonction est mise en œuvre conformément à la spécification SQL d'ArcSDE plutôt que SQL-MM. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.43

Exemples

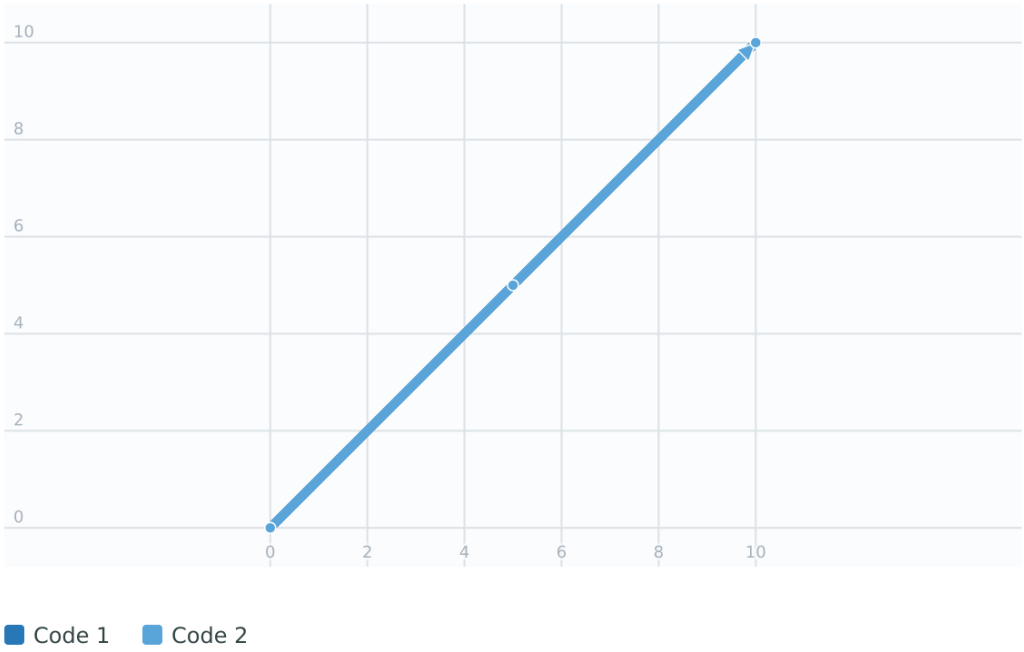
Code

```
SELECT ST_OrderingEquals('LINESTRING(0 0,10 10)',
'LINESTRING(0 0,5 5,10 10)');
```

Output

```
f
```

Figure



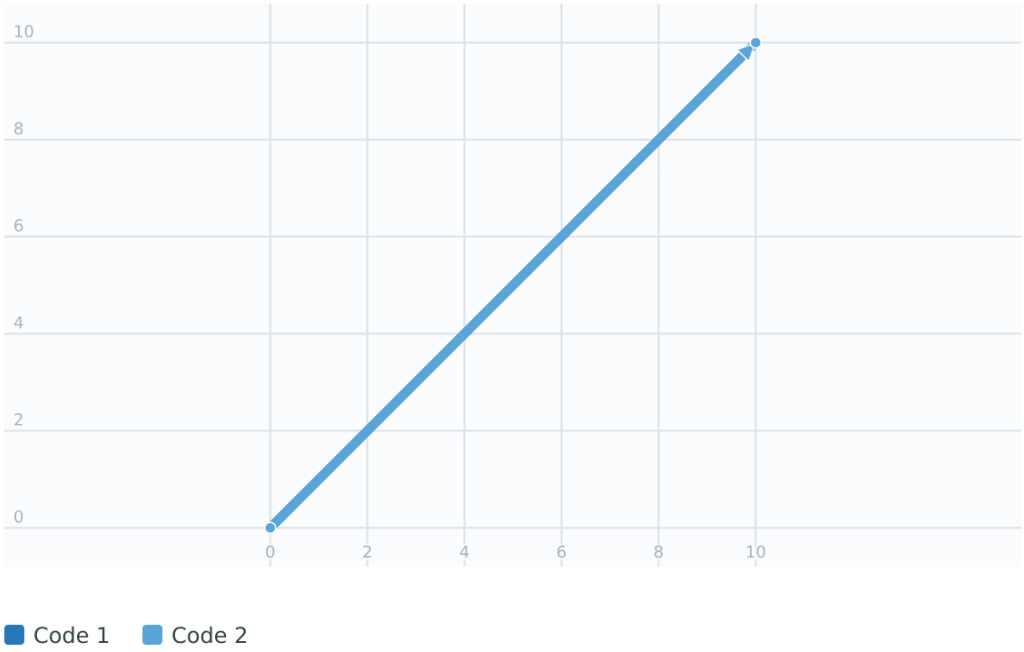
Code

```
SELECT ST_OrderingEquals('LINESTRING(0 0,10 10)',
'LINESTRING(0 0,10 10)');
```

Output

```
t
```

Figure



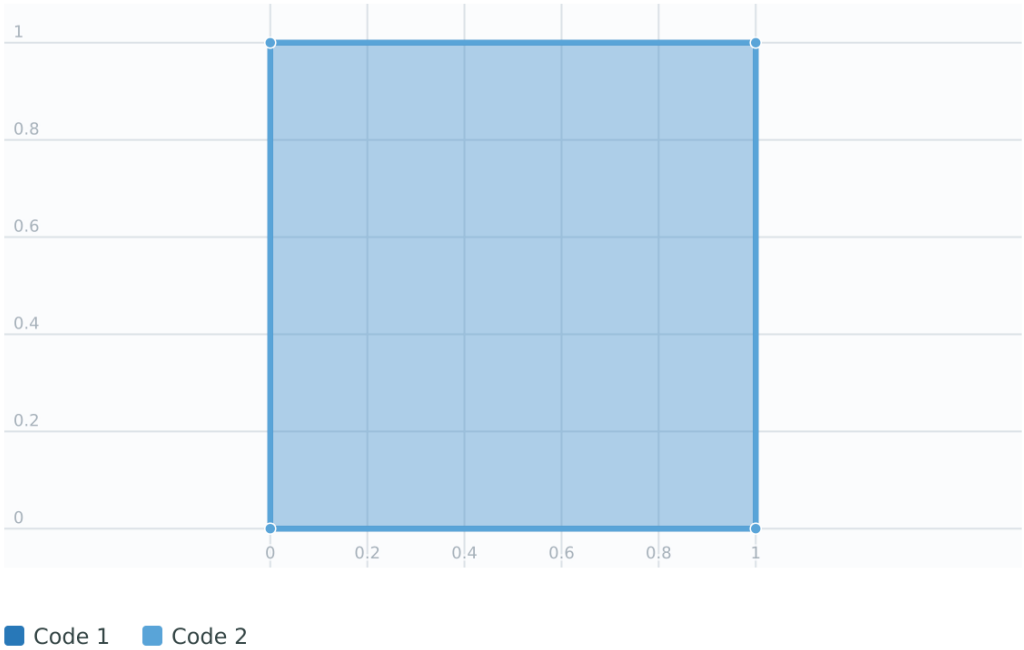
Code

```
SELECT ST_OrderingEquals('POLYGON((0 0,0 1,1 1,1 0,0 0))',
'POLYGON((0 0,1 0,1 1,0 1,0 0))');
```

Output

```
f
```

Figure



Voir aussi

&&, ST_Equals, ST_Reverse

7.11.1.12 ST_Overlaps

ST_Overlaps — Teste si deux géométries ont la même dimension et se croisent, mais si chacune a au moins un point qui n'est pas dans l'autre

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

Description

Renvoie TRUE si les géométries A et B se "chevauchent spatialement". Deux géométries se chevauchent si elles ont la même dimension, si leurs intérieurs se croisent dans cette dimension et si chacune a au moins un point à l'intérieur de l'autre (ou, de manière équivalente, si aucune des deux ne recouvre l'autre). La relation de chevauchement est symétrique et irréflexive.

En termes mathématiques : $ST_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \bigcap Int(B))). \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter l'utilisation d'un index, utilisez la fonction `_ST_Overlaps`.

Effectué par le module GEOS

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.32

Exemples

`ST_Overlaps` renvoie `TRUE` dans les situations suivantes :

Two MultiPoints overlap when they share some, but not all, points.

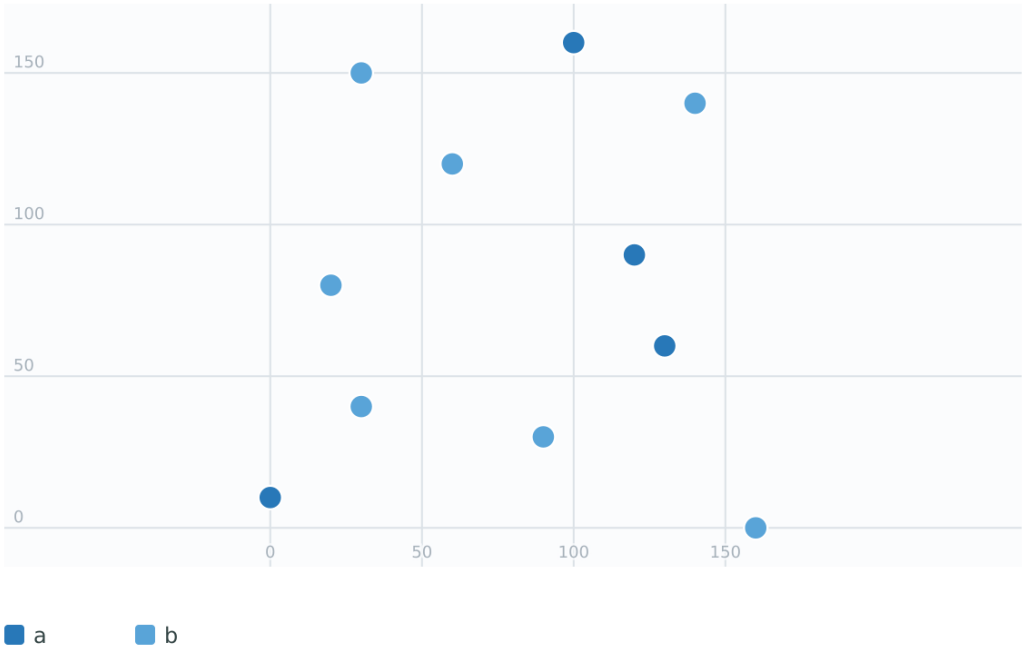
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'MULTIPOINT ((20 20),(40 90),(80 130),(140 100),(110 40),(120 170),(150 70))'::geometry AS ↩
    a,
  'MULTIPOINT ((180 10),(40 90),(80 130),(160 150),(110 40),(50 50),(50 160))'::geometry AS  ↩
    b
) AS example;
```

Output

```
t
```

Figure



Two LineStrings overlap along part of their paths.

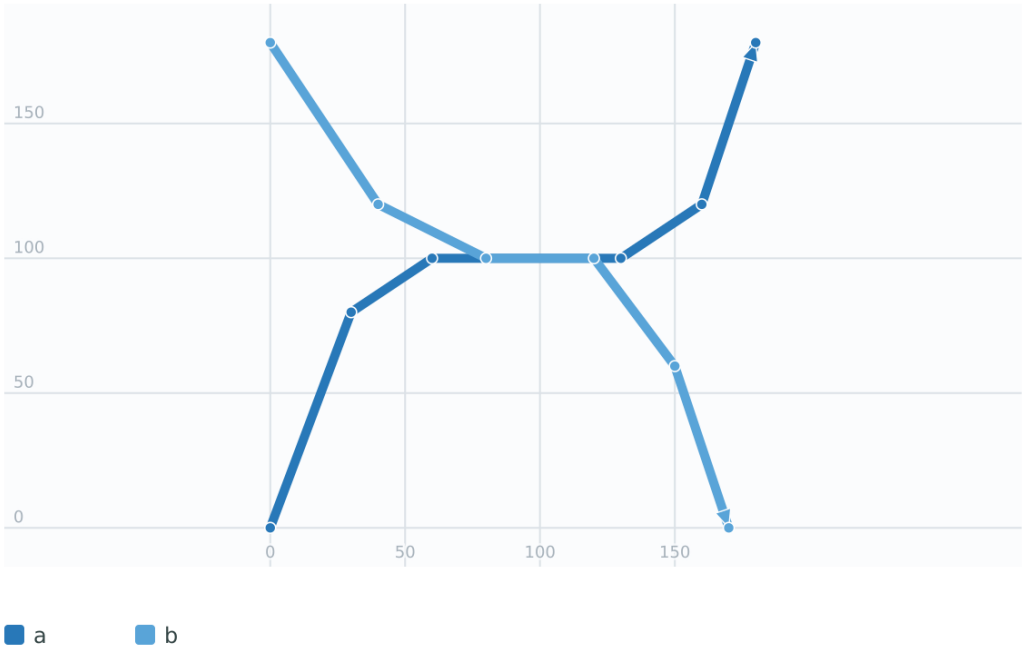
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'LINESTRING (10 10,40 90,70 110,140 110,170 130,190 190)'::geometry AS a,
  'LINESTRING (10 190,50 130,90 110,130 110,160 70,180 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two Polygons overlap where their interiors intersect and neither contains the other.

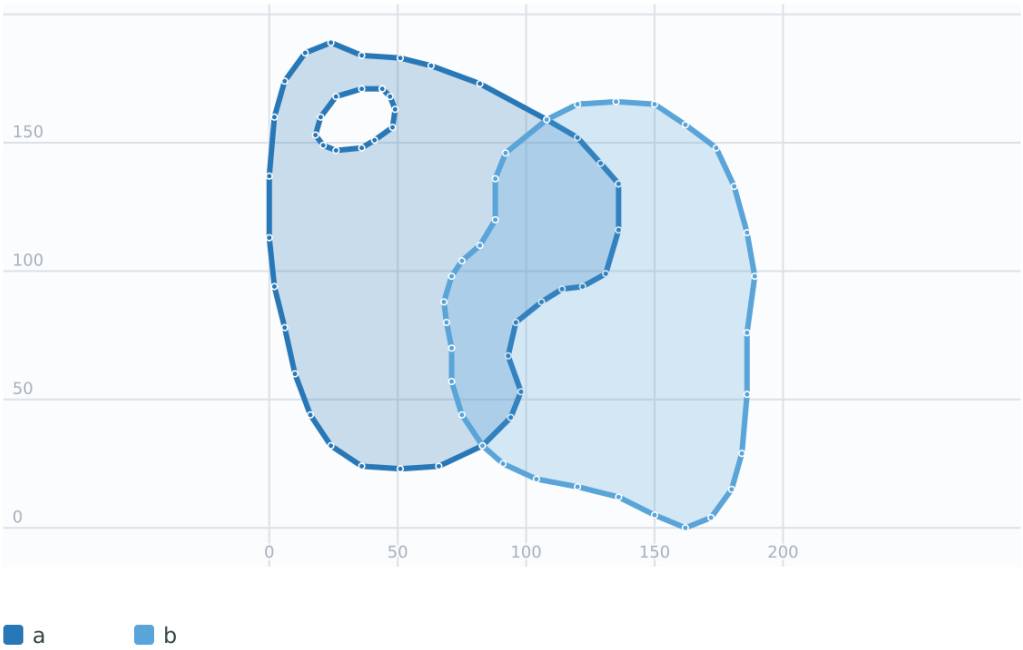
Code

```
SELECT ST_Overlaps(a, b)
FROM (SELECT
  'POLYGON ((140 140,140 122,135 105,126 100,118 99,110 94,100 86,97 73,102 59,98 49,87 ↩
    38,70 30,55 29,40 30,28 38,20 50,14 66,10 84,6 100,4 119,4 143,6 166,10 180,18 191,28 ↩
    195,40 190,55 189,67 186,86 179,112 165,124 158,133 148,140 140),(48 177,40 177,30 ↩
    174,24 166,22 159,25 155,30 153,40 154,45 157,52 162,53 169,51 174,48 177))'::geometry ↩
    AS a,
  'POLYGON ((75 63,79 50,87 38,95 31,108 25,124 22,140 18,154 11,166 6,176 10,184 21,188 ↩
    35,190 58,190 82,193 104,190 121,185 139,178 154,166 163,154 171,139 172,124 171,112 ↩
    165,96 152,92 142,92 126,86 116,79 110,75 104,72 94,73 86,75 76,75 63))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Un point sur une LineString est contenu, mais comme il a une dimension inférieure, il ne se chevauche pas et ne se croise pas.

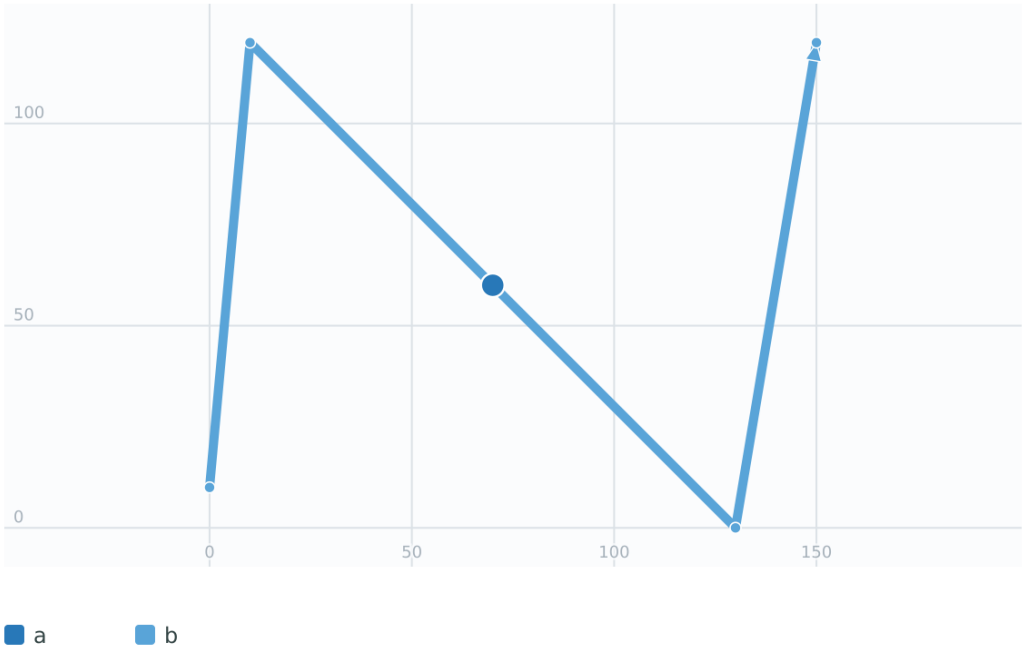
Code

```
SELECT ST_Overlaps(a, b) AS overlaps, ST_Crosses(a, b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(b, a) AS b_contains_a
FROM (SELECT 'POINT (100 100)'::geometry As a,
             'LINESTRING (30 50,40 160,160 40,180 160)'::geometry  AS b) AS t
```

Output

```
overlaps | crosses | intersects | b_contains_a
---------+---------+------------+--------------
f        | f       | t          | t
```

Figure



Une chaîne de lignes qui recouvre partiellement un polygone l'intersecte et le traverse, mais ne se chevauche pas car elle a des dimensions différentes.

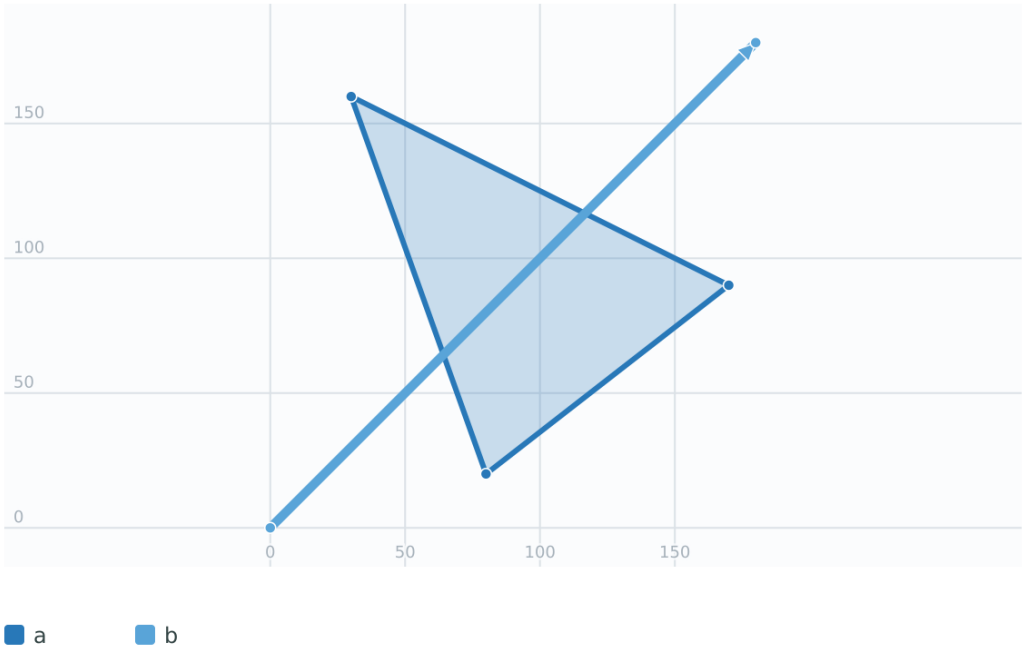
Code

```
SELECT ST_Overlaps(a, b) AS overlaps, ST_Crosses(a, b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(a, b) AS contains
FROM (SELECT 'POLYGON ((40 170,90 30,180 100,40 170))'::geometry AS a,
             'LINESTRING(10 10,190 190)'::geometry AS b) AS t;
```

Output

```
overlap | crosses | intersects | contains
---------+---------+------------+--------------
 f       | t       | t          | f
```

Figure



Deux polygones qui se croisent mais dont aucun n'est contenu par l'autre se chevauchent, mais ne se croisent pas car leur intersection a la même dimension.

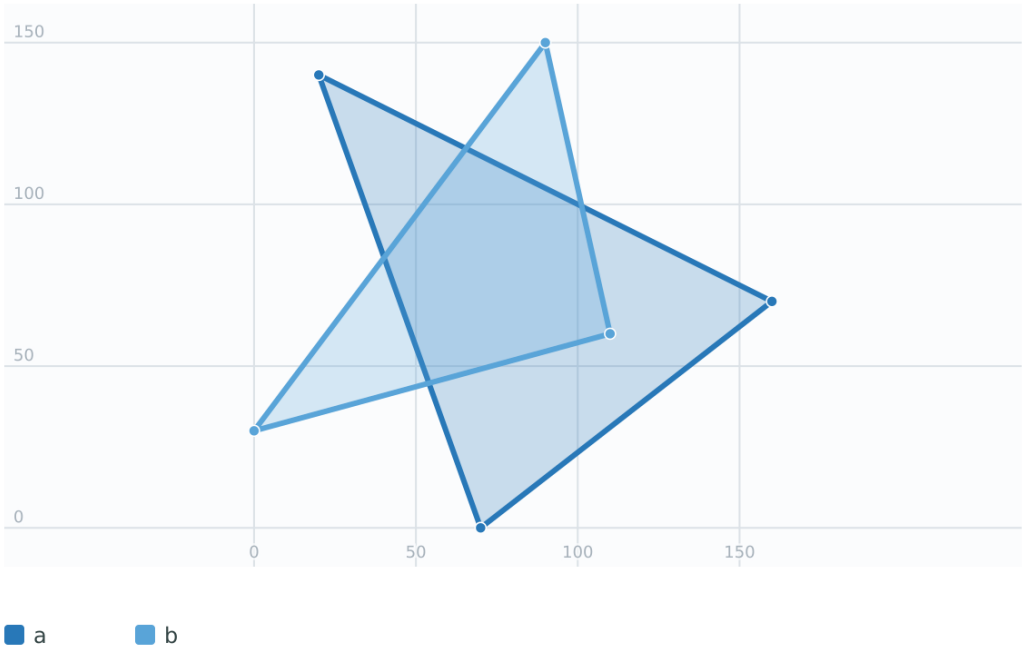
Code

```
SELECT ST_Overlaps(a, b) AS overlaps, ST_Crosses(a, b) AS crosses,
       ST_Intersects(a, b) AS intersects, ST_Contains(b, a) AS b_contains_a,
       ST_Dimension(a) AS dim_a, ST_Dimension(b) AS dim_b,
       ST_Dimension(ST_Intersection(a, b)) AS dim_int
FROM (SELECT 'POLYGON ((40 170,90 30,180 100,40 170))'::geometry AS a,
             'POLYGON ((110 180,20 60,130 90,110 180))'::geometry AS b) As t;
```

Output

```
overlaps | crosses | intersects | b_contains_a | dim_a | dim_b | dim_int
---------+---------+------------+--------------+-------+-------+----------
 t       | f       | t          | f            |     2 |     2 |       2
```

Figure



Voir aussi

ST_Contains, ST_Crosses, ST_Dimension, ST_Intersects

7.11.1.13 ST_Relate

ST_Relate — Teste si deux géométries ont une relation topologique correspondant à un modèle de matrice d'intersection, ou calcule leur matrice d'intersection

Synopsis

boolean **ST_Relate**(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text **ST_Relate**(geometry geomA, geometry geomB);
text **ST_Relate**(geometry geomA, geometry geomB, integer boundaryNodeRule);

Description

Ces fonctions permettent de tester et d'évaluer la relation spatiale (topologique) entre deux géométries, telle que définie par le Dimensionally Extended 9-Intersection Model (DE-9IM).

Le DE-9IM est une matrice à 9 éléments indiquant la dimension des intersections entre l'intérieur, la frontière et l'extérieur de deux géométries. Elle est représentée par une chaîne de texte de 9 caractères utilisant les symboles "F", "0", "1", "2" (par exemple, `'FF1FF0102'`).

Un type spécifique de relation spatiale peut être testé en faisant correspondre la matrice d'intersection à un *motif de matrice d'intersection*. Les motifs peuvent inclure les symboles supplémentaires "T" (signifiant "l'intersection est non vide") et "*" (signifiant "n'importe quelle valeur"). Les relations spatiales communes sont fournies par les fonctions nommées ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, et ST_Within. L'utilisation d'un modèle explicite permet de tester plusieurs conditions d'intersection, de croisement, etc. en une seule étape. Elle permet également de tester des relations spatiales qui n'ont pas de fonction de relation spatiale nommée. Par exemple, la relation "Interior-Intersects" possède le motif DE-9IM `T********`, qui n'est évalué par aucun prédicat nommé.

Pour plus d'informations, voir Section 5.1.

Variante 1: Teste si deux géométries sont spatialement liées selon le `intersectionMatrixPattern` donné.



Note
Contrairement à la plupart des prédicats de relations spatiales nommées, ce prédicat n'inclut PAS automatiquement un appel d'index. La raison en est que certaines relations sont vraies pour des géométries qui ne s'intersectent PAS (par exemple Disjoint). Si vous utilisez un modèle de relation qui nécessite une intersection, incluez l'appel à l'index &&.



Note
Il est préférable d'utiliser une fonction de relation nommée si elle est disponible, car elle utilise automatiquement un index spatial lorsqu'il existe. En outre, elles peuvent mettre en œuvre des optimisations de performance qui ne sont pas disponibles avec l'évaluation de la relation complète.

Variante 2: Renvoie la chaîne matricielle DE-9IM pour la relation spatiale entre les deux géométries d'entrée. La chaîne matricielle peut être testée pour vérifier si elle correspond à un modèle DE-9IM en utilisant ST_RelateMatch.

Variante 3: Comme la variante 2, mais permet de spécifier une **Boundary Node Rule**. Une boundary node rule permet de contrôler plus finement si les extrémités des multilignes sont considérées comme se situant à l'intérieur ou à la limite du DE-9IM. Les valeurs de `boundaryNodeRule` sont les suivantes :

— `1` : **OGC-Mod2** - les extrémités des lignes sont dans la frontière si elles apparaissent un nombre impair de fois. C'est la règle définie par la norme SFS de l'OGC, et c'est la valeur par défaut de la `ST_Relate`.
— `2` : **Endpoint** - tous les points d'extrémité sont dans la frontière.
— `3` : **MultivalentEndpoint** - les points d'extrémité sont dans la frontière s'ils apparaissent plus d'une fois. En d'autres termes, la frontière est constituée de tous les points d'extrémité "attachés" ou "internes" (mais pas des points d'extrémité "non attachés/externes").
— `4` : **MonovalentEndpoint** - les points d'extrémité sont dans la frontière s'ils n'apparaissent qu'une seule fois. En d'autres termes, la frontière est constituée de tous les points d'extrémité "non attachés" ou "extérieurs".

Cette fonction ne figure pas dans la spécification de l'OGC, mais elle est implicite. voir s2.1.13.2


Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3


Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.25

Effectué par le module GEOS

Amélioration : 2.0.0 - ajout de la prise en charge de la spécification de boundary node rule.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

Exemples

Utilisation de la fonction booléenne pour tester les relations spatiales.

Code

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '0FFFFF212');
```

Output

```
t
```

Code

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
```

Output

```
t
```

Test d'un modèle de relation spatiale personnalisé comme condition de requête, avec `&&` pour permettre l'utilisation d'un index spatial.

Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

Code

```
SELECT c.* , p.name As poly_name
    FROM polys AS p
    INNER JOIN compounds As c
          ON c.geom && p.geom
             AND ST_Relate(p.geom, c.geom, 'T********');
```

Calcul de la matrice d'intersection pour les relations spatiales.

Code

```
SELECT ST_Relate('POINT(1 2)',
ST_Buffer('POINT(1 2)', 2));
```

Output

```
0FFFFF212
```

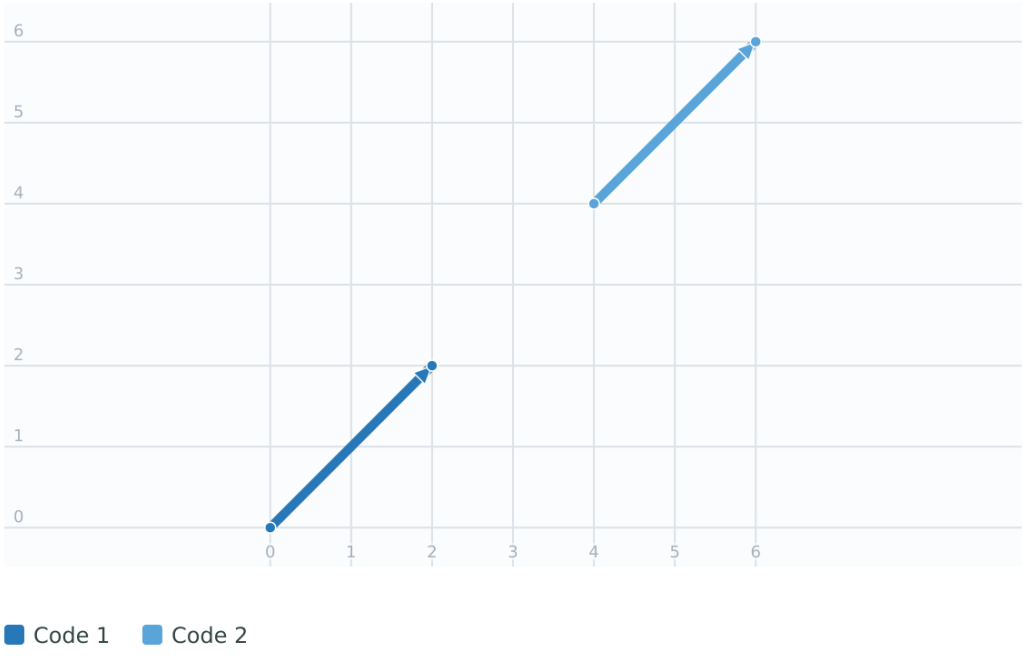
Code

```
SELECT ST_Relate('LINESTRING(1 2,3 4)',
'LINESTRING(5 6,7 8)' );
```

Output

```
FF1FF0102
```

Figure



Utilisation de différentes Boundary Node Rules pour calculer la relation spatiale entre une LineString et une MultiLineString avec une extrémité dupliquée `(3 3)` :

— En utilisant la règle **OGC-Mod2** (1), l'extrémité dupliquée se trouve dans l' **intérieur** de la MultiLineString, de sorte que l'entrée de la matrice DE-9IM [aB:bI] est `0` et [aB:bB] est `F`.

— En utilisant la règle **Endpoint** (2), l'extrémité dupliquée se trouve dans la **limite** de la MultiLineString, de sorte que l'entrée de la matrice DE-9IM [aB:bI] est `F` et [aB:bB] est `0`.

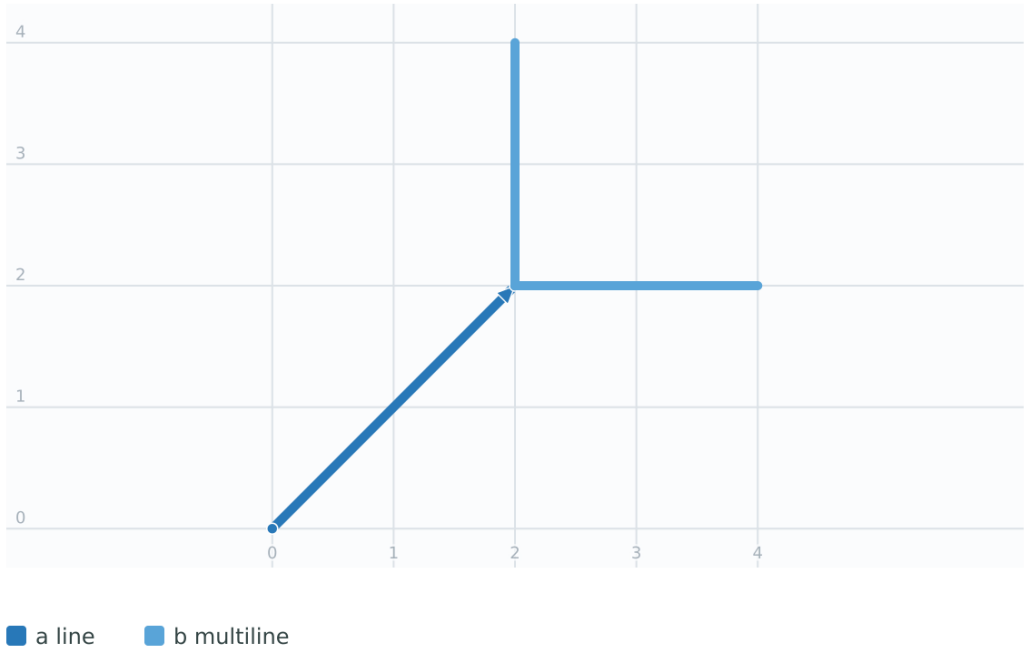
Code

```
WITH data AS (SELECT
  'LINESTRING(1 1,3 3)'::geometry AS a_line,
  'MULTILINESTRING((3 3,3 5),(3 3,5 3))'::geometry AS b_multiline
)
SELECT ST_Relate(a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate(a_line, b_multiline, 2) AS bnr_endpoint
    FROM data;
```

Output

```
bnr_mod2  | bnr_endpoint
----------+--------------
 FF10F0102 | FF1F00102
```

Figure



Voir aussi

Section 5.1, ST_RelateMatch, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within

7.11.1.14 ST_RelateMatch

ST_RelateMatch — Teste si une matrice d'intersection DE-9IM correspond à un modèle de matrice d'intersection

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

Description

Teste si une Dimensionally Extended 9-Intersection Model (DE-9IM) `intersectionMatrix` satisfait un `intersectionMatrixP`
Les valeurs de la matrice d'intersection peuvent être calculées par ST_Relate.

Pour plus d'informations, voir Section 5.1.

Effectué par le module GEOS

Disponibilité : 2.0.0

Exemples

Code

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
```

Output

```
t
```

Modèles de relations spatiales communes mis en correspondance avec les valeurs de la matrice d'intersection, pour une ligne dans différentes positions par rapport à un polygone

Code

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
    FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
                 ( 'Overlaps', 'T*T***T**' ),
                 ( 'Within', 'T*F**F***' ),
                 ( 'Disjoint', 'FF*FF****' )) AS pat(name, val)
    CROSS JOIN
        (VALUES  ('non-intersecting', 'FF1FF0212'),
                 ('overlapping', '1010F0212'),
                 ('inside', '1FF0FF212')) AS mat(name, val);
```

Output

```
relationship |  pattern  |     position     |  matrix   | match
-------------+-----------+------------------+-----------+-------
 Equality    | T1FF1FFF1 | non-intersecting | FF1FF0212 | f
 Equality    | T1FF1FFF1 | overlapping      | 1010F0212 | f
 Equality    | T1FF1FFF1 | inside           | 1FF0FF212 | f
 Overlaps    | T*T***T** | non-intersecting | FF1FF0212 | f
 Overlaps    | T*T***T** | overlapping      | 1010F0212 | t
 Overlaps    | T*T***T** | inside           | 1FF0FF212 | f
 Within      | T*F**F*** | non-intersecting | FF1FF0212 | f
 Within      | T*F**F*** | overlapping      | 1010F0212 | f
 Within      | T*F**F*** | inside           | 1FF0FF212 | t
 Disjoint    | FF*FF**** | non-intersecting | FF1FF0212 | t
 Disjoint    | FF*FF**** | overlapping      | 1010F0212 | f
 Disjoint    | FF*FF**** | inside           | 1FF0FF212 | f
```

Voir aussi

Section 5.1, ST_Relate

7.11.1.15 ST_Touches

ST_Touches — Teste si deux géométries ont au moins un point en commun, mais que leurs intérieurs ne se croisent pas

Synopsis

boolean **ST_Touches**(geometry A, geometry B);

Description

Retourne `TRUE` si A et B se croisent, mais que leurs intérieurs ne se croisent pas. De manière équivalente, A et B ont au moins un point en commun, et les points communs se situent dans au moins une frontière. Pour les entrées point/point, la relation est toujours `FALSE`, puisque les points n'ont pas de frontière.

En termes mathématiques : $ST_Touches(A, B) \Leftrightarrow (Int(A) \bigcap Int(B) = \varnothing) \wedge (A \bigcap B \neq \varnothing)$.

Cette relation est valable si la matrice d'intersection DE-9IM pour les deux géométries correspond à l'une d'entre elles :

— FT*******
— F**T*****
— F***T****



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter d'utiliser un index, utilisez plutôt la `_ST_Touches`.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`

 Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.28

Exemples

Le prédicat `ST_Touches` renvoie `TRUE` dans les exemples suivants.

Two Polygons touching along a boundary segment.

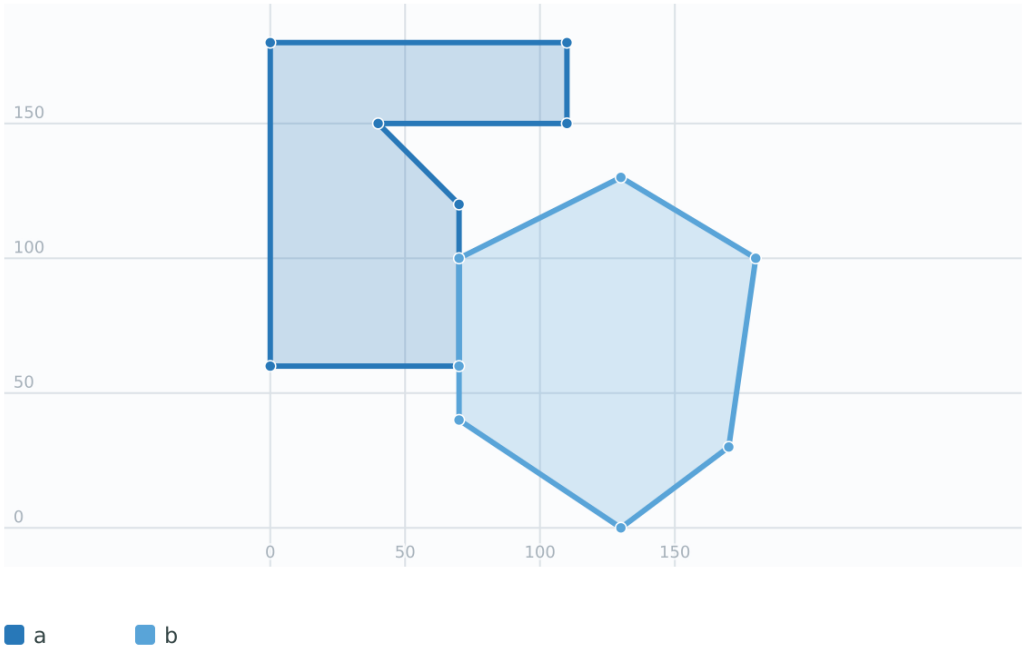
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS a,
  'POLYGON ((80 110,140 140,190 110,180 40,140 10,80 50,80 70,80 110))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two Polygons touching at a boundary point.

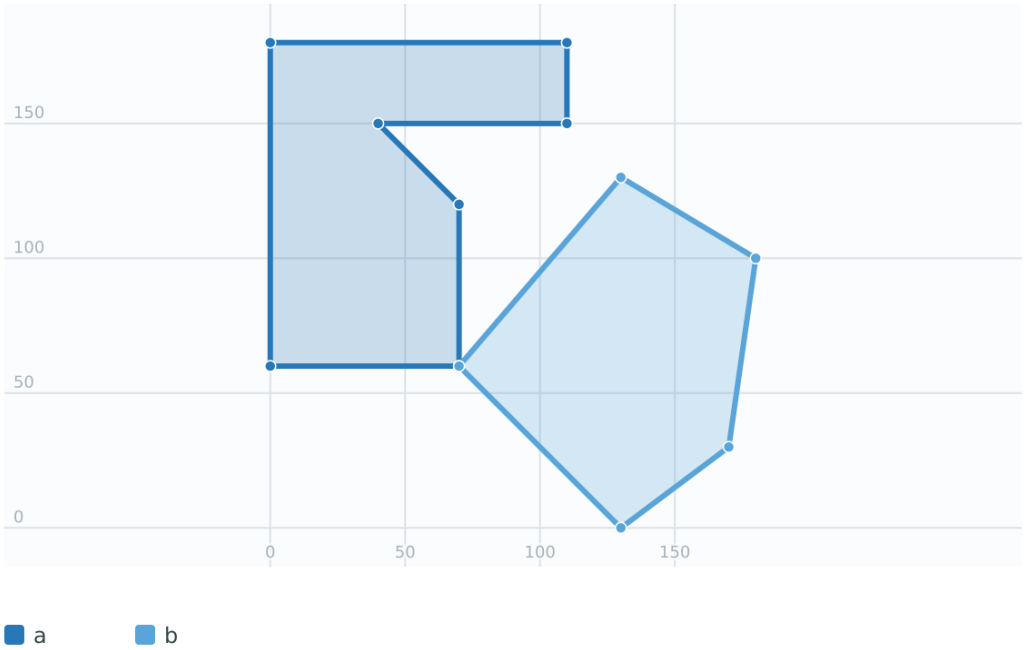
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS a,
  'POLYGON ((140 140,190 110,180 40,140 10,80 70,140 140))'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Polygon and a LineString touching at the line endpoint.

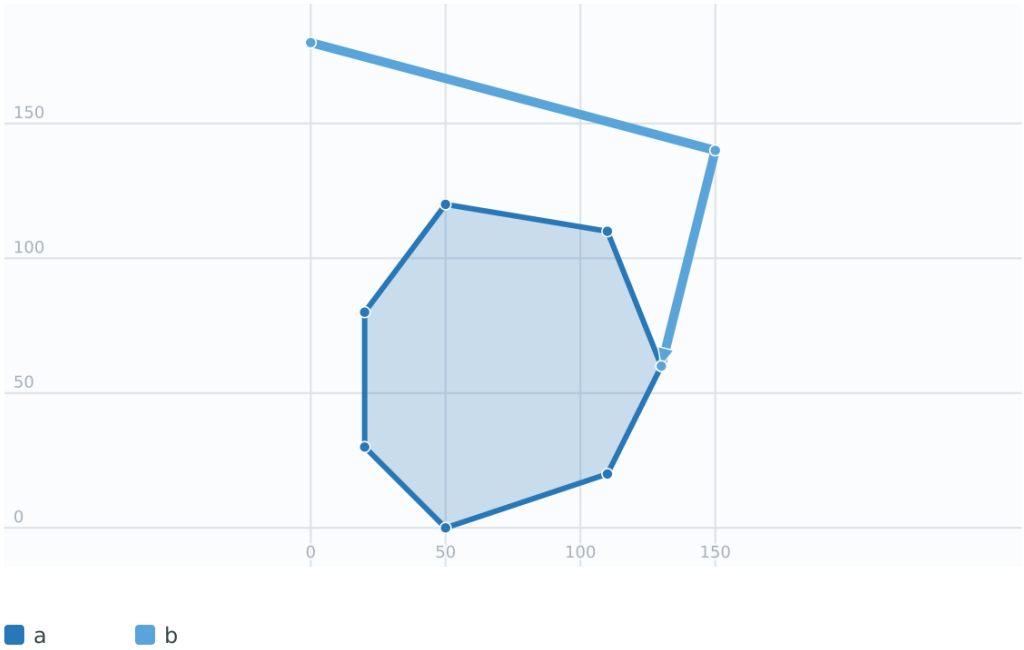
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((80 10,50 40,50 90,80 130,140 120,160 70,140 30,80 10))'::geometry AS a,
  'LINESTRING (30 190,180 150,160 70)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two LineStrings touching at their endpoints.

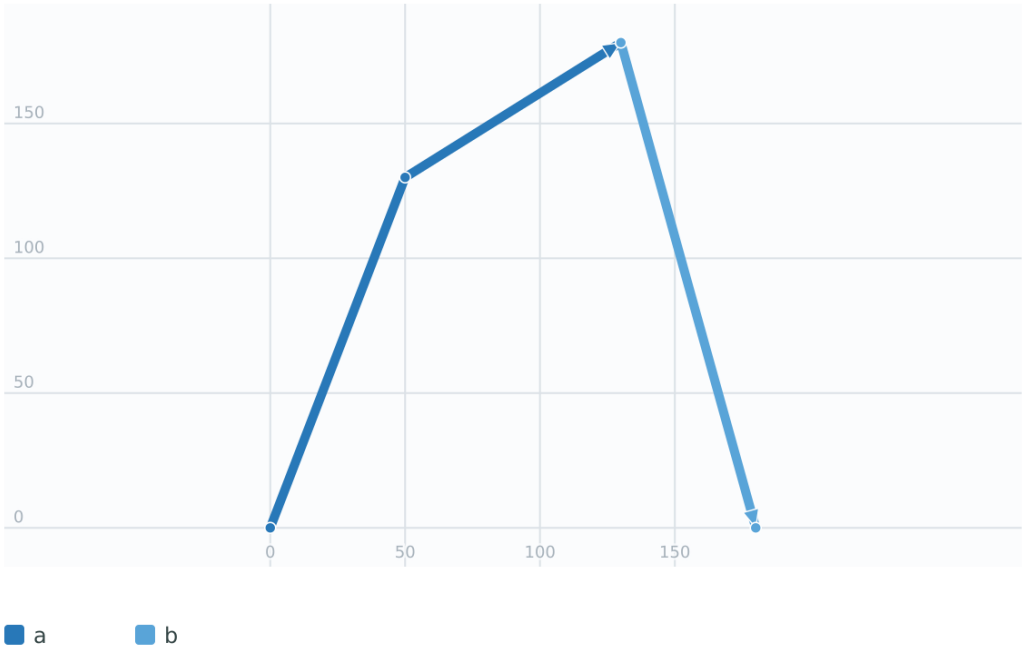
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'LINESTRING (10 10,60 140,140 190)'::geometry AS a,
  'LINESTRING (140 190,190 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



Two LineStrings touching where an endpoint meets the other line.

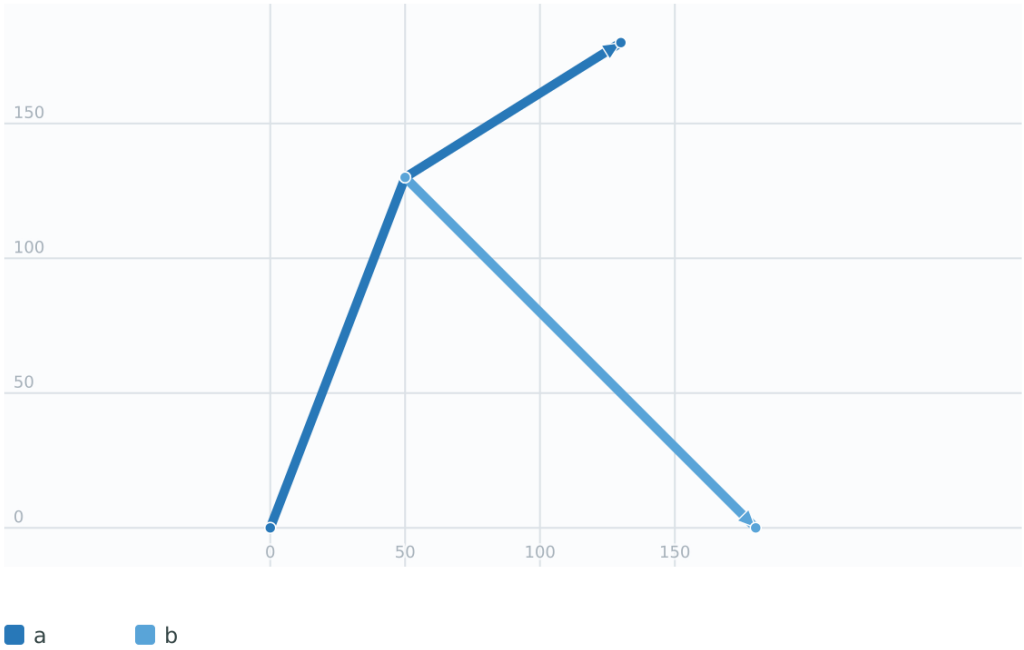
Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'LINESTRING (10 10,60 140,140 190)'::geometry AS a,
  'LINESTRING (60 140,190 10)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



A Point touching a Polygon boundary.

Code

```
SELECT ST_Touches(a, b)
FROM (SELECT
  'POLYGON ((30 30,10 110,40 190,120 170,180 180,170 100,130 10,30 30))'::geometry AS a,
  'POINT (170 100)'::geometry AS b
) AS example;
```

Output

```
t
```

Figure



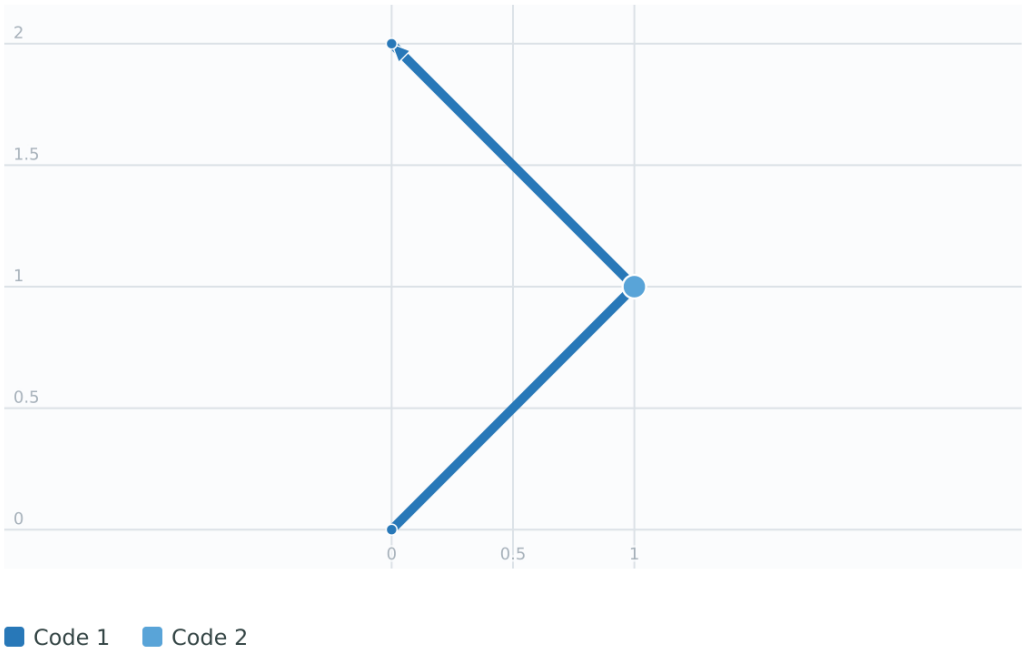
Code

```
SELECT ST_Touches('LINESTRING(0 0,1 1,0 2)'::geometry, 'POINT(1 1)'::geometry);
```

Output

```
f
```

Figure



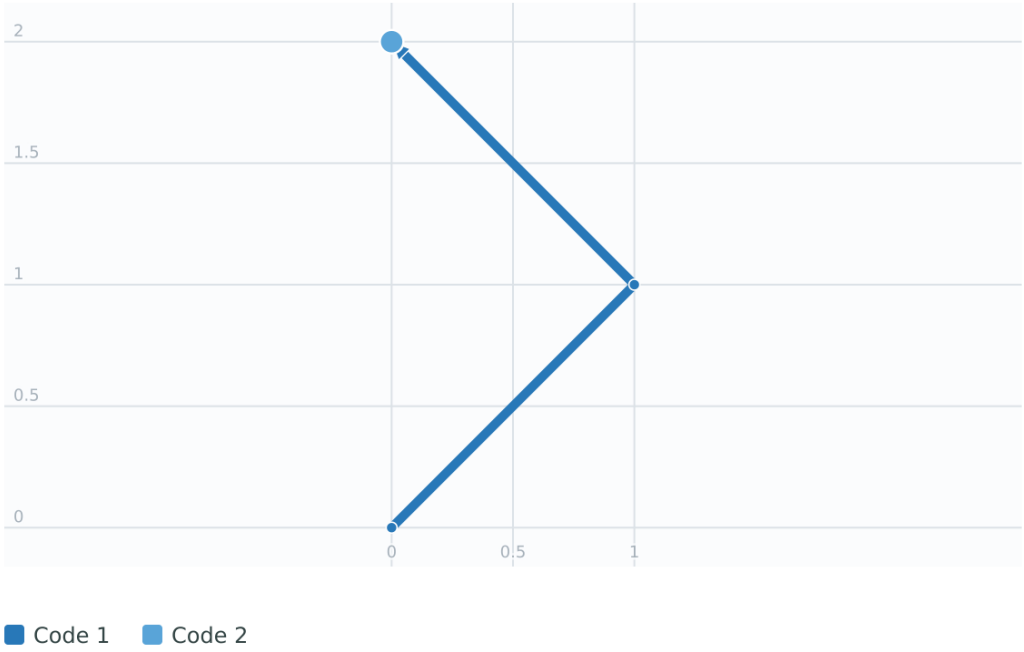
Code

```
SELECT ST_Touches('LINESTRING(0 0,1 1,0 2)'::geometry, 'POINT(0 2)'::geometry);
```

Output

```
t
```

Figure



7.11.1.16 ST_Within

ST_Within — Tests si chaque point de A se trouve dans B, et que leurs intérieurs ont un point commun

Synopsis

boolean **ST_Within**(geometry A, geometry B);

Description

Renvoie TRUE si la géométrie A est à l'intérieur de la géométrie B. A est à l'intérieur de B si et seulement si tous les points de A se trouvent à l'intérieur (c'est-à-dire à l'intérieur ou à la limite) de B (ou de manière équivalente, aucun point de A ne se trouve à l'extérieur de B), et si les intérieurs de A et de B ont au moins un point en commun.

Pour que cette fonction ait un sens, les géométries sources doivent toutes deux avoir la même projection de coordonnées et le même SRID.

En termes mathématiques : $ST_Within(A, B) \Leftrightarrow (A \bigcap B = A) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$.

La relation within est réflexive : toute géométrie est à l'intérieur d'elle-même. La relation est antisymétrique : si `ST_Within(A,B) = true` et `ST_Within(B,A) = true`, alors les deux géométries doivent être topologiquement égales (`ST_Equals(A,B) = true`).

ST_Within est le contraire de ST_Contains. Ainsi, `ST_Within(A,B) = ST_Contains(B,A)`.



Note
Comme les intérieurs doivent avoir un point commun, une subtilité de la définition est que les lignes et les points situés entièrement dans la limite des polygones ou des lignes ne sont *pas* à l'intérieur de la géométrie. Pour plus de détails, voir Subtleties of OGC Covers, Contains, Within. Le prédicat ST_CoveredBy fournit une relation plus inclusive.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries. Pour éviter l'utilisation d'un index, utilisez la fonction `_ST_Within`.

Effectué par le module GEOS

Amélioration : 2.3.0 Amélioration du court-circuit PIP pour la géométrie étendue à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones.

Amélioration : 3.0.0 a permis la prise en charge de `GEOMETRYCOLLECTION`



Important
N'utilisez pas cette fonction avec des géométries non valides. Vous obtiendrez des résultats inattendus.

NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T*F**F***')



Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.30

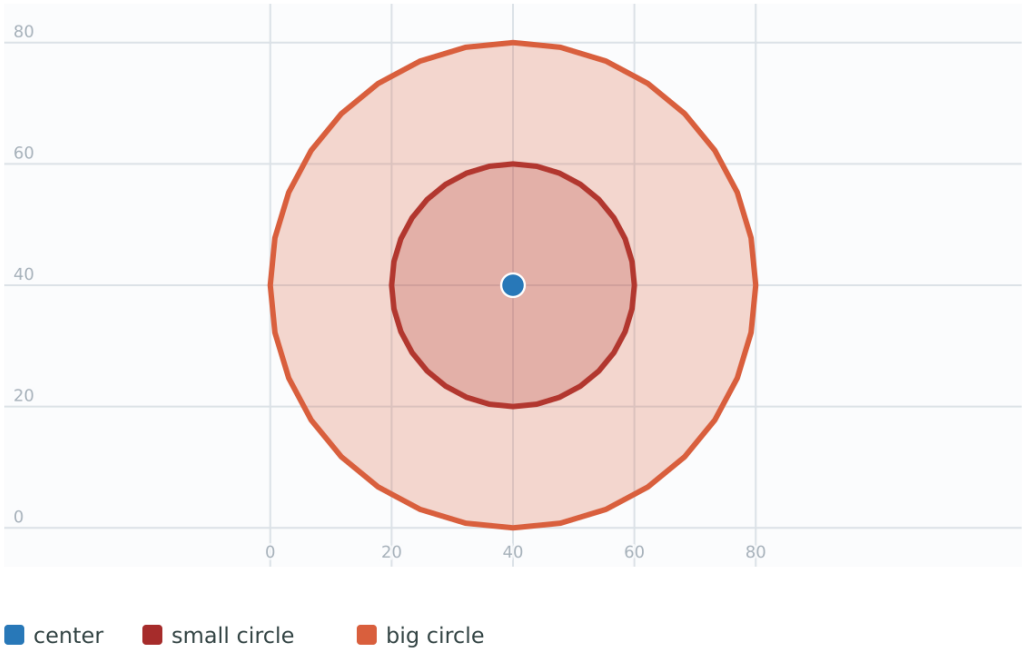
Exemples

This example defines two concentric circles once in a CTE so the same inputs can be rendered and then reused in related predicate checks.

Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 20) AS smallc,
         ST_Buffer(center, 40) AS bigc
  FROM (SELECT 'POINT(50 50)'::geometry AS center) AS p
)
SELECT smallc AS small_circle,
       bigc AS big_circle
FROM circles;
```

Figure



Using the same circles, `ST_Within` is reflexive on each circle, false from the larger circle into the smaller one, and true for `ST_Union` only when the union does not extend beyond the larger circle.

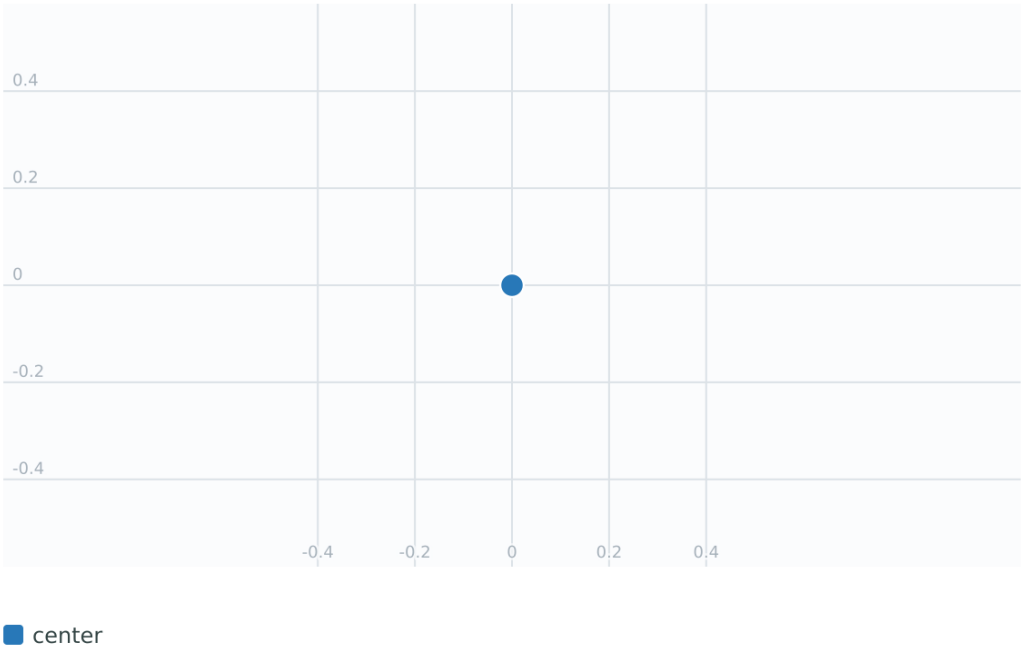
Code

```
WITH circles AS (
  SELECT ST_Buffer(center, 20) AS smallc,
         ST_Buffer(center, 40) AS bigc
  FROM (SELECT 'POINT(50 50)'::geometry AS center) AS p
)
SELECT
  ST_Within(smallc, smallc) AS smallinsmall,
  ST_Within(smallc, bigc) AS smallinbig,
  ST_Within(bigc, smallc) AS biginsmall,
  ST_Within(ST_Union(smallc, bigc), bigc) AS unioninbig,
  ST_Within(bigc, ST_Union(smallc, bigc)) AS biginunion,
  ST_Equals(bigc, ST_Union(smallc, bigc)) AS bigisunion
FROM circles;
```

Output

```
t | t | f | t | t | t
```

Figure



Voir aussi

ST_Contains, ST_CoveredBy, ST_Equals, ST_IsValid

7.11.2 Relations de distance

7.11.2.1 ST_3DDWithin

ST_3DDWithin — Teste si deux géométries 3D se trouvent à une distance 3D donnée

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

Description

Renvoie true si la distance 3D entre deux valeurs géométriques n'est pas supérieure à la distance `distance_of_srid`. La distance est spécifiée en unités définies par le système de référence spatial des géométries. Pour que cette fonction ait un sens, les géométries sources doivent se trouver dans le même système de coordonnées (avoir le même SRID).



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette méthode implémente la spécification SQL/MM. SQL-MM ?

Disponibilité : 2.0.0

Exemples

Compare the same transformed point and line in both 3D and 2D. The 3D test fails because the Z difference matters even though the 2D footprints are close enough.

Code

```
WITH sample AS (
  SELECT ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)'::geometry, 2163) AS input_point,
         ST_Transform(
           'SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)'::geometry,
           2163
         ) AS input_line
)
SELECT ST_3DDWithin(input_point, input_line, 126.8) AS within_dist_3d,
       ST_DWithin(input_point, input_line, 126.8) AS within_dist_2d,
       input_point AS input_point,
       input_line AS input_line
FROM sample;
```

Output

```
-[ RECORD 1 ↩
    ]--+-----------------------------------------------------------------------------------------------------------------------------------------------------------------
within_dist_3d | f
within_dist_2d | t
input_point    | POINT Z (-11774128.9 5237160.2 4)
input_line     | LINESTRING Z (-11774339.5 5243556.8 15,-11773706.7 5215256.3 20)
```

Voir aussi

ST_3DDFullyWithin, ST_DWithin, ST_DFullyWithin, ST_3DDistance, ST_Distance, ST_3DMaxDistance, ST_Transform

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — Teste si deux géométries 3D sont entièrement comprises dans une distance 3D donnée

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Renvoie true si les géométries 3D se trouvent à la distance spécifiée l'une de l'autre. La distance est spécifiée en unités définies par le système de référence spatiale des géométries. Pour que cette fonction ait un sens, les géométries sources doivent toutes deux avoir la même projection de coordonnées et le même SRID.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries.

Disponibilité : 2.0.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

This example compares distance-within predicates in 2D and 3D. The point is distance-within the line in 3D, but the full 3D line is not within the same distance of the point.

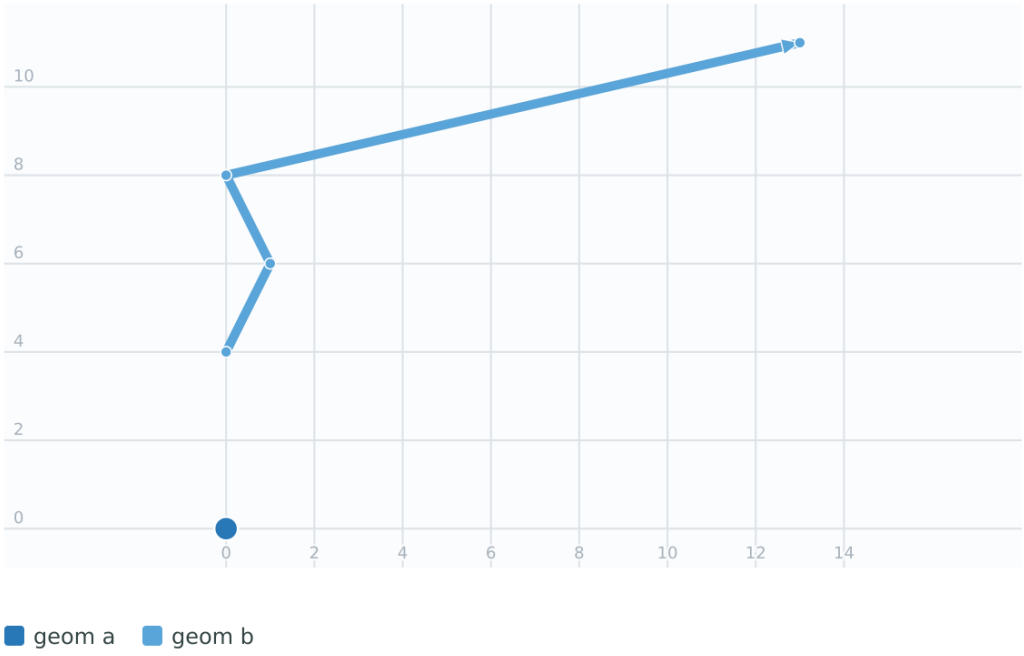
Code

```
WITH data AS (
  SELECT
    'POINT(1 1 2)'::geometry AS geom_a,
    'LINESTRING(1 5 2,2 7 20,1 9 100,14 12 3)'::geometry AS geom_b
)
SELECT
  ST_3DDFullyWithin(geom_a, geom_b, 10) AS d3dfullywithin10,
  ST_3DDWithin(geom_a, geom_b, 10) AS d3dwithin10,
  ST_DFullyWithin(geom_a, geom_b, 20) AS d2dfullywithin20,
  ST_3DDFullyWithin(geom_a, geom_b, 20) AS d3dfullywithin20
FROM data;
```

Output

```
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----------------+-------------+------------------+-----------------
 f               | t           | t                | f
```

Figure



Voir aussi

ST_3DDWithin, ST_DWithin, ST_DFullyWithin, ST_3DMaxDistance

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — Teste si une géométrie se trouve entièrement à une distance d'une autre géométrie

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Retourne vrai si `g2` se trouve entièrement à une distance `distance` de `g1`. Visuellement, la condition est vraie si `g2` est contenu dans un tampon de `distance` de `g1`. La distance est spécifiée en unités définies par le système de référence spatial des géométries.



Note
Cette fonction inclut une comparaison de la boîte englobante qui utilise tous les index disponibles sur les géométries.

Disponibilité : 1.5.0

Modifié : 3.5.0 : la logique implémentée utilise désormais un test de confinement dans un tampon, plutôt que l'algorithme ST_MaxDistance. Les résultats seront différents de ceux des versions précédentes, mais devraient être plus proches des attentes de l'utilisateur.

Exemples

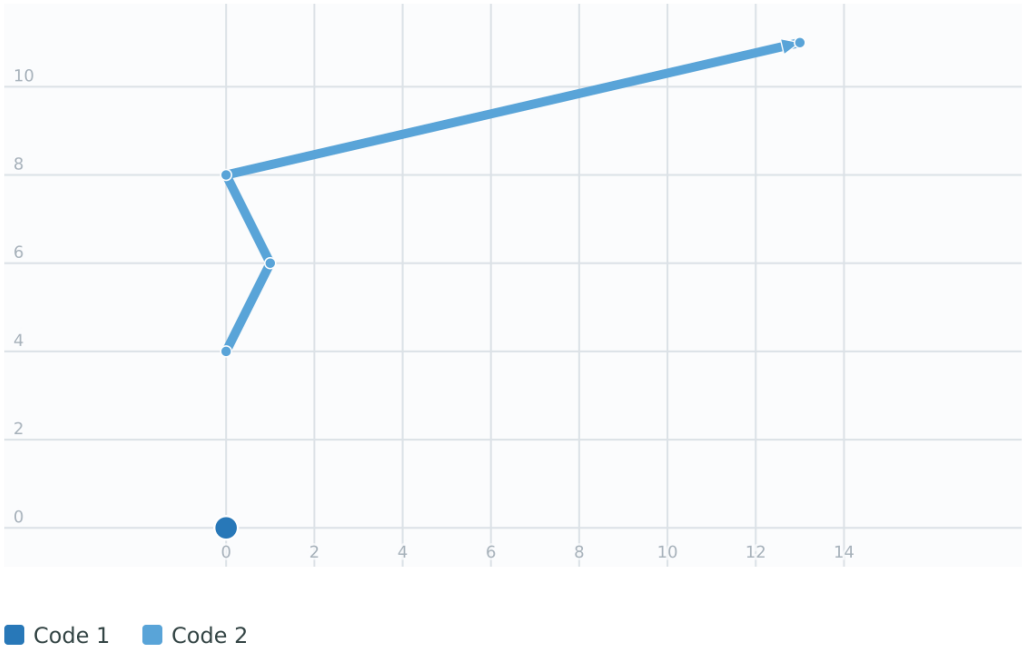
Code

```
SELECT
    ST_DFullyWithin(geom_a, geom_b, 10) AS DFullyWithin10,
    ST_DWithin(geom_a, geom_b, 10) AS DWithin10,
    ST_DFullyWithin(geom_a, geom_b, 20) AS DFullyWithin20
FROM (VALUES
    ('POINT(1 1)'::geometry, 'LINESTRING(1 5,2 7,1 9,14 12)'::geometry)
    ) AS v(geom_a, geom_b)
```

Output

```
dfullywithin10 | dwithin10 | dfullywithin20
---------------+-----------+---------------
 f             | t         | t
```

Figure



Voir aussi

ST_MaxDistance, ST_DWithin, ST_3DDWithin, ST_3DDFullyWithin

7.11.2.4 ST_DWithin

ST_DWithin — Teste si deux géométries se trouvent à une distance donnée

Synopsis

boolean **ST_DWithin**(geometry g1, geometry g2, double precision distance_of_srid);
boolean **ST_DWithin**(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid = true);

Description

Renvoie true si les géométries se trouvent à une distance donnée

Pour le type geometry : la distance est spécifiée en unités définies par le système de référence spatiale des géométries. Pour que cette fonction ait un sens, les géométries sources doivent se trouver dans le même système de coordonnées (avoir le même SRID).

Pour le type geography : les unités sont en mètres et la mesure de la distance est par défaut `use_spheroid = true`. Pour une évaluation plus rapide, utilisez `use_spheroid = false` pour mesurer sur la sphère.



Note
Utilisez ST_3DDWithin pour les géométries 3D.



Note
Cet appel de fonction inclut une comparaison de la boîte de délimitation qui utilise tous les index disponibles sur les géométries.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Disponibilité : la prise en charge du type geography a été introduite dans la version 1.5.0

Amélioration : la version 2.1.0 a amélioré la vitesse pour le type geography. Voir Making Geography faster pour plus de détails.

Amélioration : la prise en charge des géométries courbes a été introduite dans la version 2.1.0.

Avant la version 1.3, ST_Expand était couramment utilisé en conjonction avec && et ST_Distance pour tester la distance, et avant la version 1.3.4, cette fonction utilisait cette logique. À partir de la version 1.3.4, ST_DWithin utilise une fonction de distance de court-circuit plus rapide.

Exemples

This self-contained geometry example keeps the compared shapes visible while also returning the boolean results.

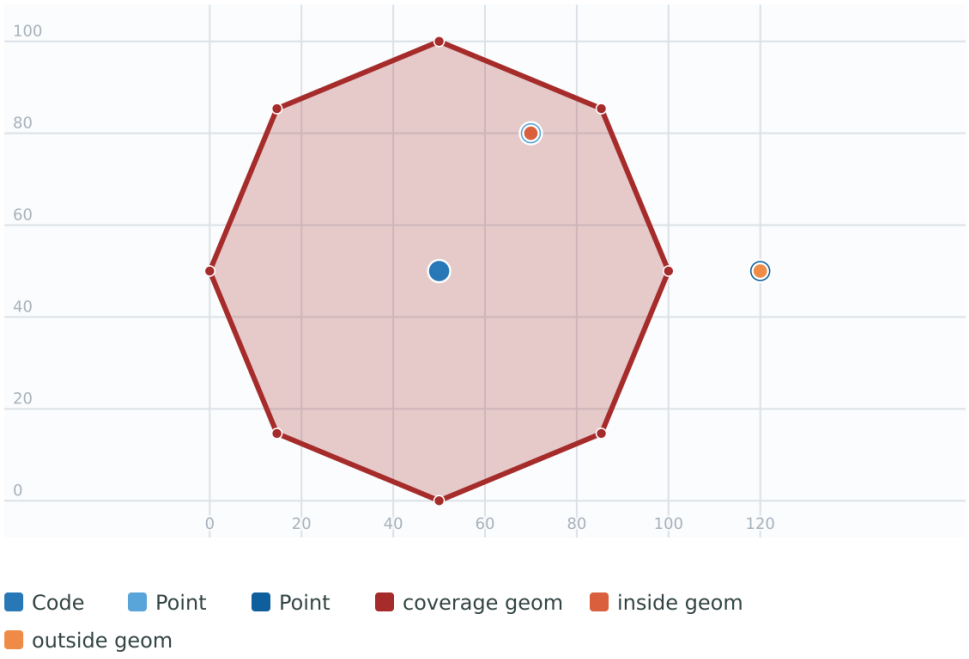
Code

```
WITH sample AS (
  SELECT ST_Buffer('POINT(100 90)'::geometry, 50, 'quad_segs=2') AS coverage,
         ST_Point(120, 120) AS inside_pt,
         ST_Point(170, 90) AS outside_pt
)
SELECT ST_DWithin(coverage, inside_pt, 0) AS inside_now,
       ST_DWithin(coverage, outside_pt, 10) AS outside_within_10,
       coverage AS coverage_geom,
       inside_pt AS inside_geom,
       outside_pt AS outside_geom
FROM sample;
```

Output

```
-[ RECORD 1 ]-----
inside_now        | t
outside_within_10 | f
coverage_geom     | POLYGON((150 90,135 55,100 40,65 55,50 90,65 125,100 140,135 125,150 90))
inside_geom       | POINT(120 120)
outside_geom      | POINT(170 90)
```

Figure



Find the nearest hospital to each school. That is within 3000 units of the school. We do an ST_DWithin search to utilize indexes to limit our search list. That the non-indexable ST_Distance needs to process. If the units of the spatial reference is meters then units would be meters.

Code

```
SELECT DISTINCT ON (s.gid) s.gid, s.school_name, s.geom, h.hospital_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  ORDER BY s.gid, ST_Distance(s.geom, h.geom);
```

The schools with no close hospitals. Find all schools with no hospital within 3000 units away from the school. Units are in units of spatial reference (for example meters, feet, or degrees).

Code

```
SELECT s.gid, s.school_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  WHERE h.gid IS NULL;
```

Find locations more than 3000 units away from their nearest hospital. Use `NOT EXISTS` so that a location is returned only when no nearby hospital exists.

Code

```
SELECT l.location_name
  FROM locations l
  WHERE NOT EXISTS (
    SELECT 1
    FROM hospitals h
    WHERE ST_DWithin(l.geom, h.geom, 3000)
  );
```

Find broadcasting towers that a receiver with limited range can receive. Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

Code

```
CREATE INDEX ON broadcasting_towers USING gist (geom);
CREATE INDEX ON broadcasting_towers USING gist (ST_Expand(geom, sending_range));
```

Query towers that a 4-kilometer receiver in Minsk Hackerspace can get. Note: two conditions are required because the shorter `LEAST(b.sending_range, 4000)` will not use an index.

Code

```
SELECT b.tower_id, b.geom
  FROM broadcasting_towers b
  WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
    AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

Voir aussi

ST_Distance, ST_3DDWithin

7.11.2.5 ST_PointInsideCircle

ST_PointInsideCircle — Teste si un point géométrique se trouve à l'intérieur d'un cercle défini par un centre et un rayon

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

Description

Retourne true si la géométrie est un point et se trouve à l'intérieur du cercle de centre `center_x`,`center_y` et de rayon `radius`.



AVERTISSEMENT
N'utilise pas les index spatiaux. Utilisez plutôt ST_DWithin.

Disponibilité : 1.2

Modifié : 2.2.0 Dans les versions précédentes, cette fonction était appelée ST_Point_Inside_Circle

Exemples

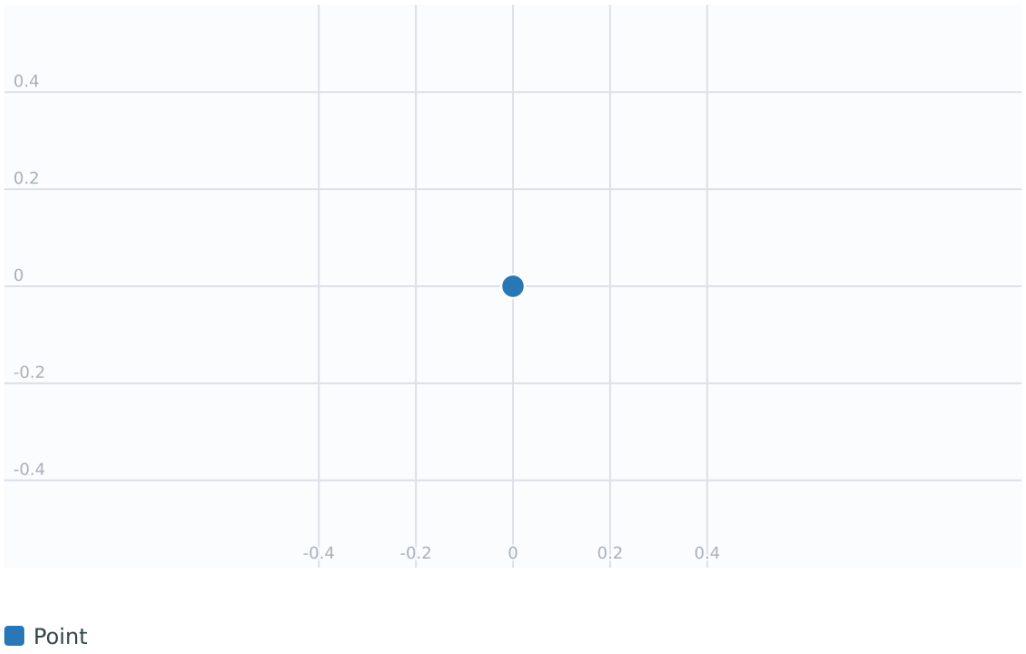
Code

```
SELECT ST_PointInsideCircle(ST_Point(1, 2), 0.5, 2, 3);
```

Output

```
t
```

Figure



Voir aussi

ST_DWithin

7.12 Fonctions de mesure

7.12.1 ST_Area

ST_Area — Renvoie l'aire d'une géométrie polygonale.

Synopsis

float **ST_Area**(geometry g1);
float **ST_Area**(geography geog, boolean use_spheroid = true);

Description

Renvoie l'aire d'une géométrie polygonale. Pour les types geometry, une surface cartésienne 2D (planaire) est calculée, avec des unités spécifiées par le SRID. Pour les types geography, la surface est déterminée par défaut sur un sphéroïde avec des unités en mètres carrés. Pour calculer la surface en utilisant le modèle sphérique, plus rapide mais moins précis, utilisez `ST_Area(geog,false`

Amélioration : 2.0.0 - la prise en charge des surfaces polyédriques 2D a été introduite.

Amélioration : 2.2.0 - mesure sur sphéroïde effectuée avec GeographicLib pour une meilleure précision et robustesse. Nécessite PROJ >= 4.9.0 pour profiter de la nouvelle fonctionnalité.

Modifié : 3.0.0 - ne dépend plus de SFCGAL.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 8.1.2, 9.5.3

Cette fonction prend en charge les surfaces Polyhedral.



Note
Pour les surfaces polyédriques, ne prend en charge que les surfaces polyédriques 2D (pas 2.5D). Pour les surfaces 2.5D, la réponse peut être différente de zéro, mais uniquement pour les faces qui se trouvent entièrement dans le plan XY.

Exemples

Retourne la superficie en pieds carrés d'un terrain du Massachusetts et multipliez par la conversion pour obtenir des mètres carrés. Notez que la surface est exprimée en pieds carrés, car EPSG:2249 correspond aux pieds de plan de l'État du Massachusetts

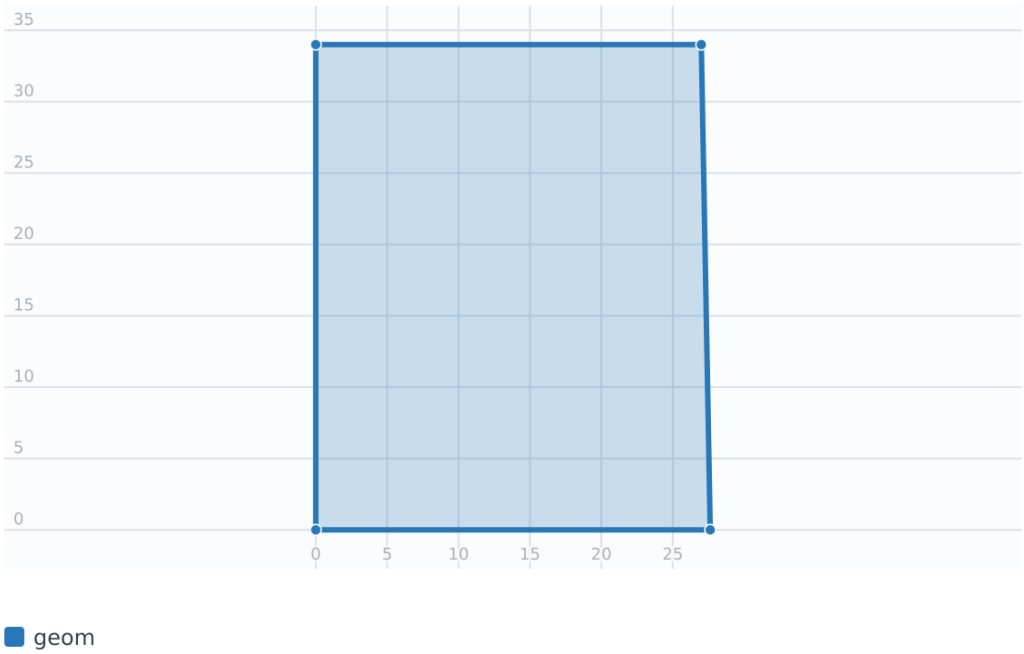
Code

```
SELECT ST_Area(geom) AS sqft,
       ST_Area(geom) * 0.3048 ^ 2 AS sqm
FROM (
  SELECT 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
          743265 2967450,743265.625 2967416,743238 2967416))'::geometry AS geom
) AS subquery;
```

Output

```
sqft   |     sqm
--------+-------------
928.625 | 86.27208552
```

Figure



Return area square feet and transform to Massachusetts state plane meters (EPSG:26986) to get square meters. Note this is in square feet because 2249 is Massachusetts State Plane Feet and transformed area is in square meters since EPSG:26986 is state plane Massachusetts meters.

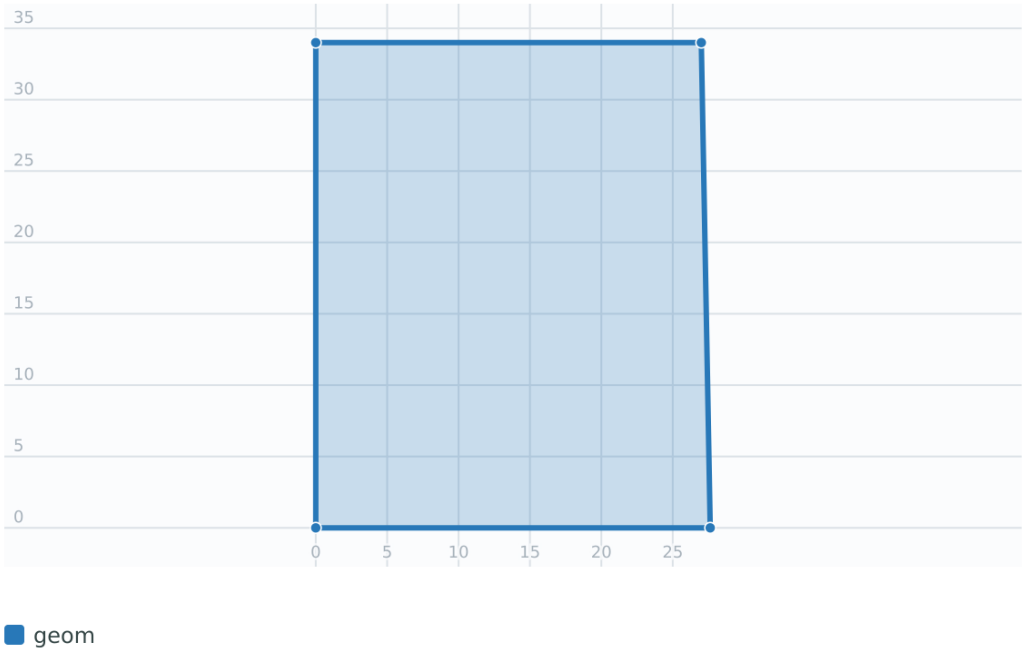
Code

```
SELECT ST_Area(geom) AS sqft,
       round(ST_Area(ST_Transform(geom, 26986))::numeric, 6) AS sqm
FROM (
  SELECT 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
          743265 2967450,743265.625 2967416,743238 2967416))'::geometry AS geom
) AS subquery;
```

Output

```
sqft    |    sqm
--------+-----------
928.625 | 86.272431
```

Figure



Retourne la surface en pieds carrés et en mètres carrés en utilisant le type de données geography. Notez que nous transformons notre geometry en geography (avant de pouvoir le faire, assurez-vous que votre géométrie est en WGS 84 long lat 4326). La geography se mesure toujours en mètres. Il s'agit d'une démonstration à des fins de comparaison. Normalement, votre table sera déjà stockée dans le type de données geography.

Code

```
SELECT round((ST_Area(geog) / 0.3048 ^ 2)::numeric, 2) AS sqft_spheroid,
       round((ST_Area(geog, false) / 0.3048 ^ 2)::numeric, 2) AS sqft_sphere,
       round(ST_Area(geog)::numeric, 2) AS sqm_spheroid
FROM (
  SELECT ST_Transform(
           'SRID=2249;POLYGON((743238 2967416,743238 2967450,
           743265 2967450,743265.625 2967416,743238 2967416))'::geometry,
           4326
         )::geography AS geog
) AS subquery;
```

Output

```
sqft_spheroid | sqft_sphere | sqm_spheroid
---------------+-------------+--------------
        928.68 |      926.61 |        86.28
```

Si vos données sont déjà dans le type geography :

Code

```
SELECT ST_Area(geog) / 0.3048 ^ 2 AS sqft,
               ST_Area(the_geog) AS sqm
        FROM somegeogtable;
```

Voir aussi

CG_3DArea, ST_GeomFromText, ST_GeographyFromText, ST_SetSRID, ST_Transform

7.12.2 ST_Azimuth

ST_Azimuth — Renvoie l'azimut basé sur le nord d'une ligne entre deux points.

Synopsis

float **ST_Azimuth**(geometry origin, geometry target);
float **ST_Azimuth**(geography origin, geography target);

Description

Renvoie l'azimut en radians du point cible par rapport au point d'origine, ou NULL si les deux points coïncident. L'angle d'azimut est un angle positif dans le sens des aiguilles d'une montre, référencé à partir de l'axe Y positif (geometry) ou du méridien Nord (geography) : North = 0; Northeast = $\pi/4$; East = $\pi/2$; Southeast = $3\pi/4$; South = π; Southwest $5\pi/4$; West = $3\pi/2$; Northwest = $7\pi/4$.

Pour le type geography, la solution azimutale est connue sous le nom de inverse geodesic problem.

L'azimut est un concept mathématique défini comme l'angle entre un vecteur de référence et un point, avec des unités angulaires en radians. La valeur du résultat en radians peut être convertie en degrés à l'aide de la fonction PostgreSQL `degrees()`.

For offsetting lines relative to their direction, use ST_OffsetCurve. For projecting a point by distance and bearing, use ST_Project.

Disponibilité : 1.1.0

Amélioration : la prise en charge du type geography a été introduite dans la version 2.0.0.

Amélioration : 2.2.0 mesure sur sphéroïde effectuée avec GeographicLib pour améliorer la précision et la robustesse. Nécessite PROJ >= 4.9.0 pour profiter de la nouvelle fonctionnalité.

Exemples

Geometry Azimut en degrés

Code

```
SELECT degrees(ST_Azimuth(ST_Point(25, 45), ST_Point(75, 100))) AS degA_B,
       degrees(ST_Azimuth(ST_Point(75, 100), ST_Point(25, 45) )) AS degB_A;
```

Output

```
dega_b       |     degb_a
-----------------+-----------------
42.27368900609371 | 222.27368900609372
```

Compass direction labels can be computed from the azimuth.

Code

```
WITH directions(deg) AS (VALUES
  (0), (45), (90), (135), (180), (225), (270), (315)
),
bearings AS (
  SELECT deg,
         degrees(ST_Azimuth(ST_Point(0, 0),
                            ST_MakePoint(sin(radians(deg)),
                                         cos(radians(deg))))) AS azimuth
  FROM directions
)
SELECT deg,
       round(azimuth::numeric, 1) AS azimuth,
```

```
       (ARRAY['N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW', 'N'])
         [floor((azimuth + 22.5) / 45)::int + 1] AS direction
FROM bearings
ORDER BY deg;
```

Output

```
deg | azimuth | direction
-----+---------+-----------
   0 |     0.0 | N
  45 |    45.0 | NE
  90 |    90.0 | E
 135 |   135.0 | SE
 180 |   180.0 | S
 225 |   225.0 | SW
 270 |   270.0 | W
 315 |   315.0 | NW
```

Azimuth from Point(25, 45) to Point(75, 100), shown against the positive Y axis (north).

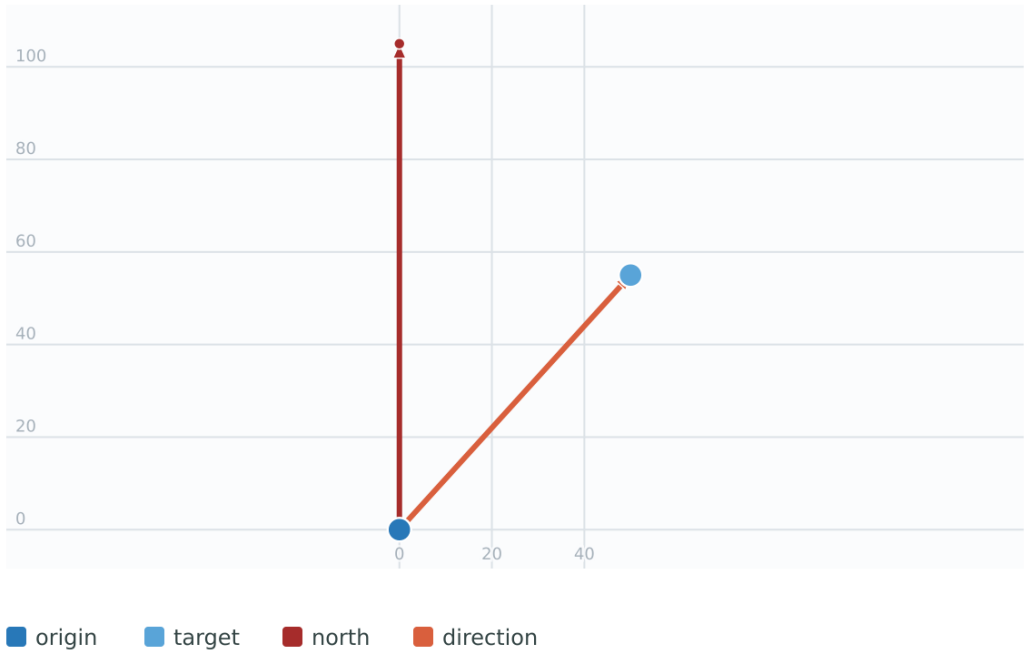
Code

```
WITH points AS (
  SELECT 'POINT(25 45)'::geometry AS origin,
         'POINT(75 100)'::geometry AS target
)
SELECT round(degrees(ST_Azimuth(origin, target))::numeric, 6) AS degrees,
       ST_MakeLine(origin, ST_Translate(origin, 0, 105)) AS north,
       ST_MakeLine(origin, target) AS direction
FROM points;
```

Output

```
42.273689 | LINESTRING(25 45,25 150) | LINESTRING(25 45,75 100)
```

Figure



Reverse azimuth from Point(75, 100) to Point(25, 45).

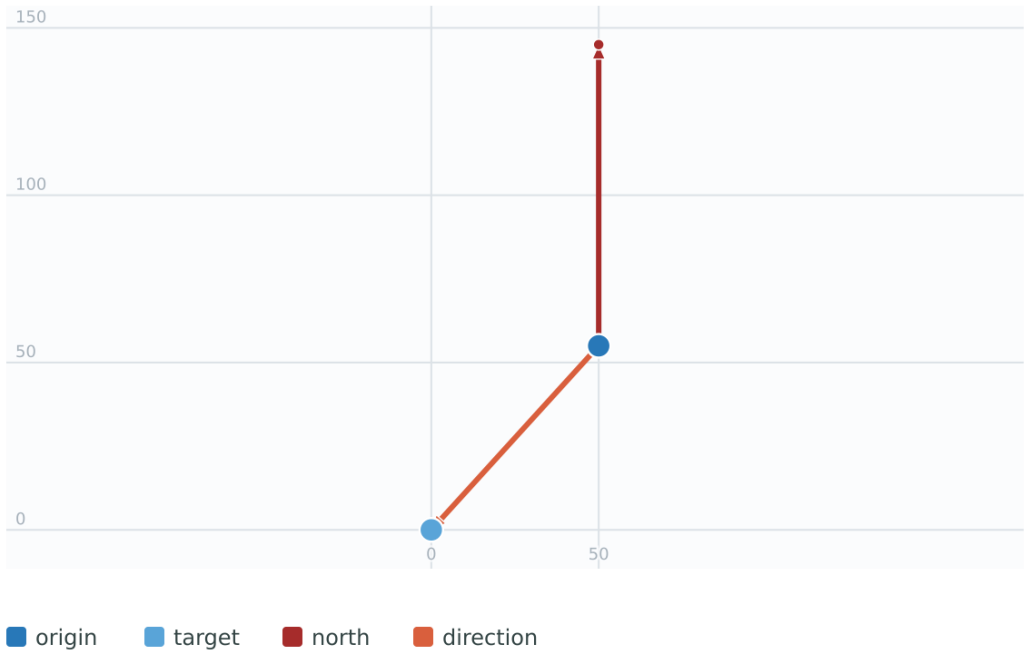
Code

```
WITH points AS (
  SELECT 'POINT(75 100)'::geometry AS origin,
         'POINT(25 45)'::geometry AS target
)
SELECT round(degrees(ST_Azimuth(origin, target))::numeric, 6) AS degrees,
       ST_MakeLine(origin, ST_Translate(origin, 0, 90)) AS north,
       ST_MakeLine(origin, target) AS direction
FROM points;
```

Output

```
222.273689 | LINESTRING(75 100,75 190) | LINESTRING(75 100,25 45)
```

Figure



Voir aussi

ST_Angle, ST_Point, ST_Translate, ST_Project, PostgreSQL Math Functions

7.12.3 ST_Angle

ST_Angle — Renvoie l'angle entre deux vecteurs définis par 3 ou 4 points, ou 2 lignes.

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

Description

Calcule l'angle dans le sens des aiguilles d'une montre entre deux vecteurs.

Variante 1: calcule l'angle formé par les points P1-P2-P3. Si un quatrième point est fourni, elle calcule l'angle formé par les points P1-P2 et P3-P4

Variante 2: calcule l'angle entre deux vecteurs S1-E1 et S2-E2, définis par les points de départ et d'arrivée des lignes d'entrée

Le résultat est un angle positif compris entre 0 et 2π radians. Le résultat en radians peut être converti en degrés à l'aide de la fonction PostgreSQL `degrees()`.

Notez que `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Disponibilité : 2.5.0

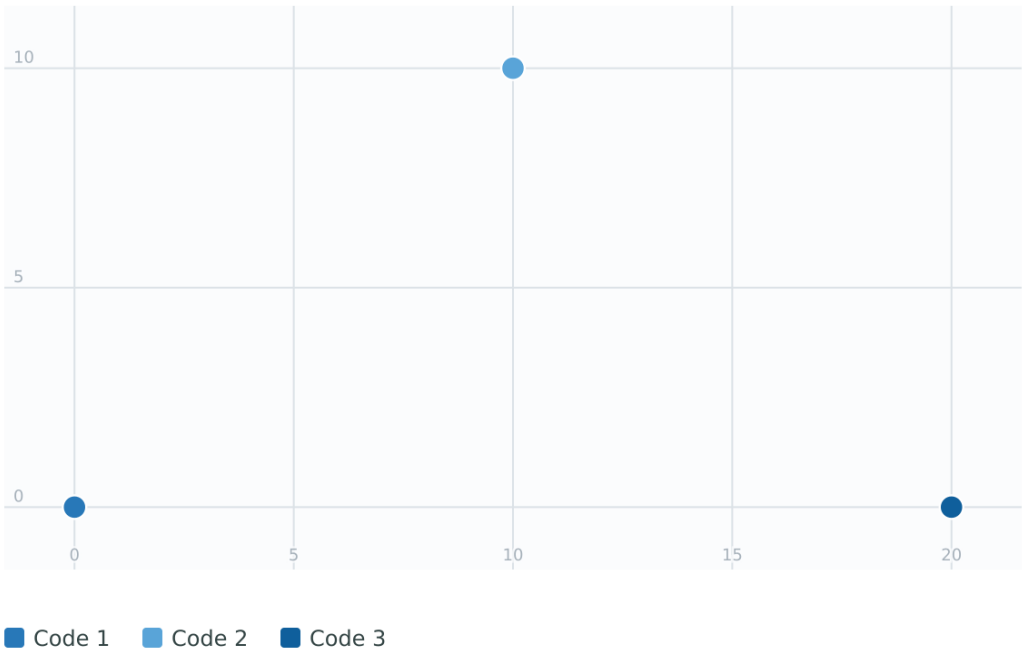
Exemples

Angle entre trois points

Code

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));
```

Figure

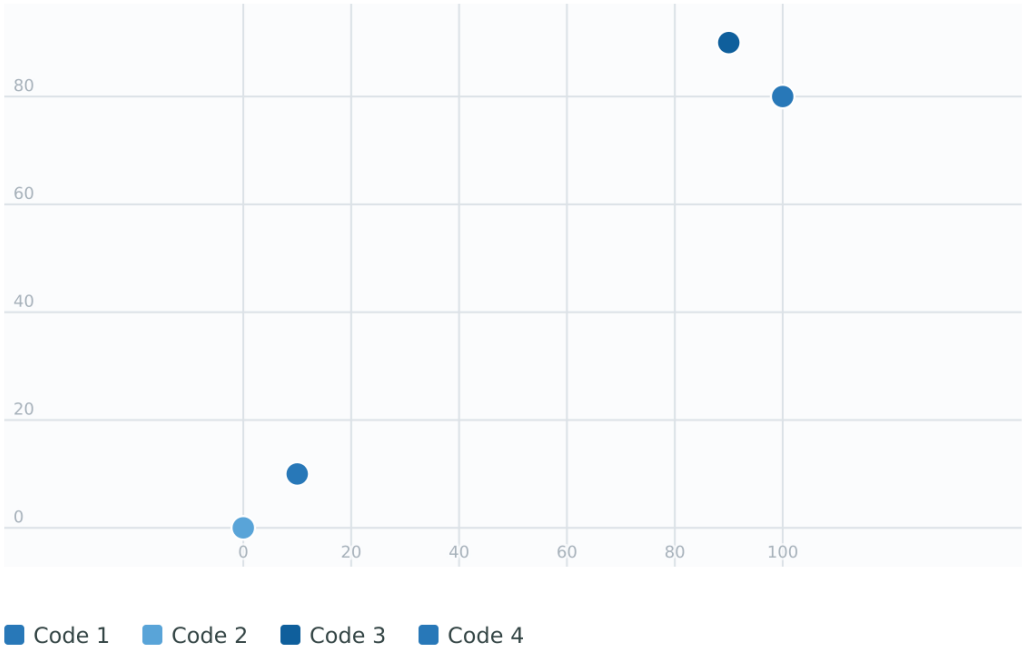


Angle entre les vecteurs définis par quatre points

Code

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)'));
```

Figure

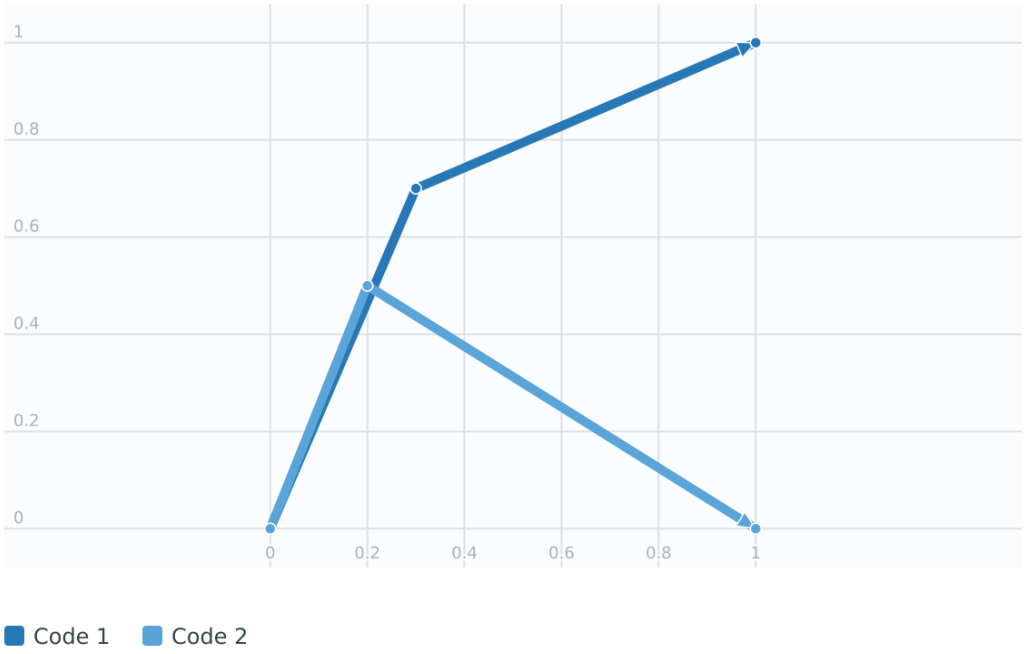


Angle entre les vecteurs définis par les points de départ et d'arrivée des lignes

Code

```
SELECT degrees(ST_Angle('LINESTRING(0 0,0.3 0.7,1 1)', 'LINESTRING(0 0,0.2 0.5,1 0)'));
```

Figure



Voir aussi

ST_Azimuth

7.12.4 ST_ClosestPoint

ST_ClosestPoint — Renvoie le point 2D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne la plus courte d'une géométrie à l'autre.

Synopsis

geometry **ST_ClosestPoint**(geometry geom1, geometry geom2);
geography **ST_ClosestPoint**(geography geom1, geography geom2, boolean use_spheroid = true);

Description

Renvoie le point bidimensionnel sur `geom1` qui est le plus proche de `geom2`. Il s'agit du premier point de la ligne la plus courte entre les géométries (telle que calculée par ST_ShortestLine).



Note
Si vous avez une géométrie 3D, vous préférerez peut-être utiliser ST_3DClosestPoint.

Amélioré : 3.4.0 - Prise en charge de la geography.

Disponibilité : 1.5.0

Exemples

Le point le plus proche d'un point et d'une LineString est le point lui-même. Le point le plus proche d'une LineString et d'un Point est un point de la ligne.

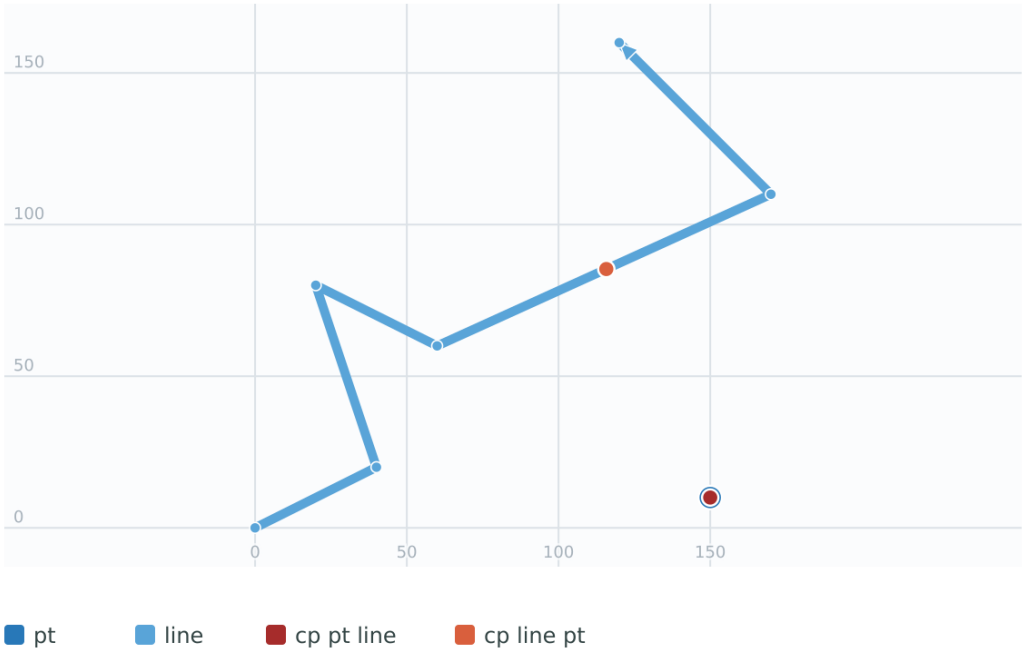
Code

```
SELECT ST_ClosestPoint(pt, line) AS cp_pt_line, ST_ClosestPoint(line, pt) AS cp_line_pt
    FROM (SELECT 'POINT (160 40)'::geometry AS pt,
                 'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)'::geometry AS line ) ↩
     AS t;
```

Output

```
cp_pt_line   |               cp_line_pt
---------------+-----------------------------------------
 POINT(160 40)  | POINT(125.75342465753425 115.34246575342466)
```

Figure

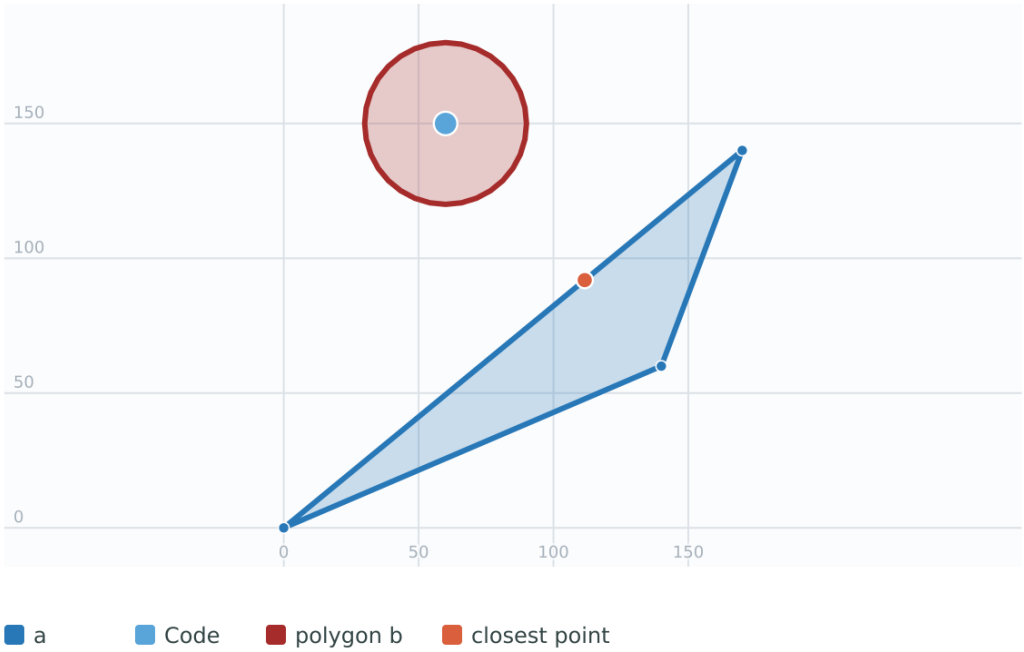


Find the closest point on polygon A to polygon B.

Code

```
WITH source AS (
  SELECT 'POLYGON ((190 150,20 10,160 70,190 150))'::geometry AS a,
         ST_Buffer('POINT(80 160)'::geometry, 30) AS b
)
SELECT b AS polygon_b,
       ST_ClosestPoint(a, b) AS closest_point
FROM source;
```

Figure



Voir aussi

ST_3DClosestPoint, ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — Renvoie le point 3D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne 3D la plus courte.

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

Description

Renvoie le point tridimensionnel de g1 le plus proche de g2. Il s'agit du premier point de la ligne 3D la plus courte. La longueur 3D de la ligne 3D la plus courte est la distance 3D.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Disponibilité : 2.0.0

Modifié : 2.2.0 - si 2 géométries 2D sont saisies, un point 2D est renvoyé (au lieu de l'ancien comportement supposant 0 pour Z manquant). Dans le cas de 2D et 3D, Z n'est plus supposé être 0 pour Z manquant.

Exemples

Compare the 3D closest point to the 2D closest point for a LineString and a Point.

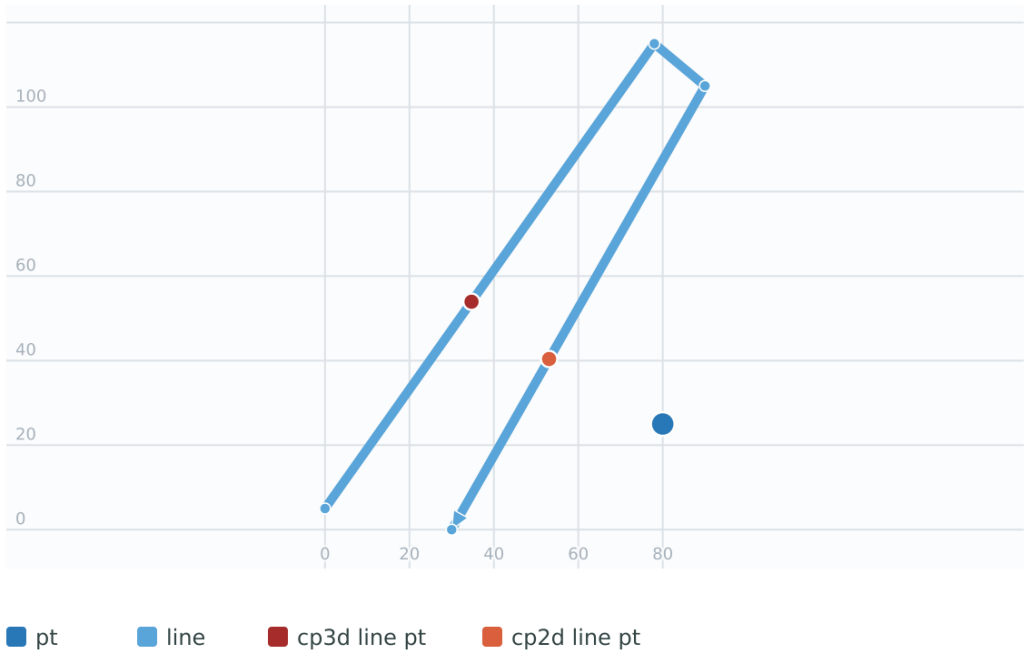
Code

```
WITH input AS (
  SELECT 'POINT(100 100 30)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)'::geometry AS line
)
SELECT ST_3DClosestPoint(line, pt) AS cp3d_line_pt,
       ST_ClosestPoint(line, pt) AS cp2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------
cp3d_line_pt   | POINT(54.69937988676193 128.93502291722837 11.547586950660556)
cp2d_line_pt   | POINT(73.07692307692307 115.38461538461539)
```

Figure



Compare the 3D closest point to the 2D closest point for a LineString and a MultiPoint.

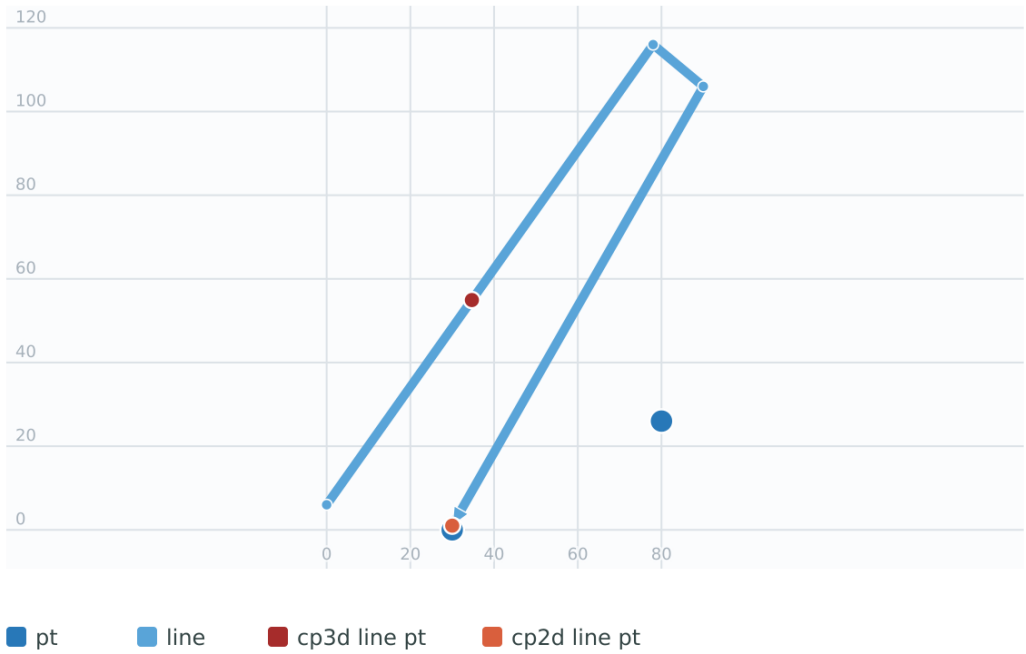
Code

```
WITH input AS (
  SELECT 'MULTIPOINT(100 100 30,50 74 1000)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 900)'::geometry AS line
)
SELECT ST_3DClosestPoint(line, pt) AS cp3d_line_pt,
       ST_ClosestPoint(line, pt) AS cp2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------
cp3d_line_pt   | POINT(54.69937988676193 128.93502291722837 11.547586950660556)
cp2d_line_pt   | POINT(50 75)
```

Figure



Compare the 3D closest point to the 2D closest point for a Polygon and a MultiLineString.

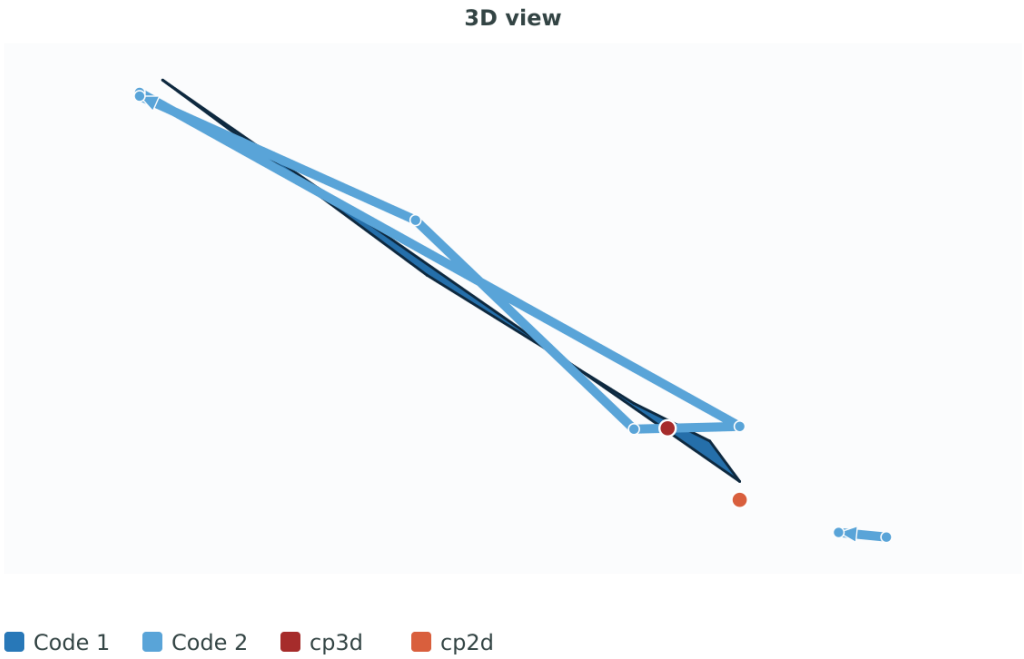
Code

```
WITH input AS (
  SELECT ST_GeomFromEWKT(
           'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))'
         ) AS poly,
         ST_GeomFromEWKT(
           'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),
                            (1 10 2,5 20 1))'
         ) AS mline
)
SELECT ST_3DClosestPoint(poly, mline) AS cp3d,
       ST_ClosestPoint(poly, mline) AS cp2d
FROM input;
```

Output

```
-[ RECORD 1 ]--+---------------------------------------
cp3d           | POINT(40.45454545454545 53.63636363636364 5)
cp2d           | POINT(20 40)
```

Figure



Voir aussi

ST_AsEWKT, ST_ClosestPoint, ST_3DDistance, ST_3DShortestLine

7.12.6 ST_Distance

ST_Distance — Renvoie la distance entre deux valeurs de geometry ou geography.

Synopsis

float **ST_Distance**(geometry g1, geometry g2);
float **ST_Distance**(geography geog1, geography geog2, boolean use_spheroid = true);

Description

Pour les types geometry, renvoie la distance cartésienne (planaire) minimale en 2D entre deux géométries, en unités projetées (unités de référence spatiales).

Pour les types geography, la valeur par défaut est la distance géodésique minimale entre deux géographies en mètres, calculée sur le sphéroïde déterminé par le SRID. Si `use_spheroid` est faux, un calcul sphérique plus rapide est utilisé.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.23

Cette méthode prend en charge les types Circular String et Curve.

Disponibilité : 1.5.0 La prise en charge du type geography a été introduite dans la version 1.5. Amélioration de la vitesse pour les géométries planaires afin de mieux gérer les géométries de grande taille ou à nombreux sommets

Amélioration : la version 2.1.0 a amélioré la vitesse pour le type geography. Voir Making Geography faster pour plus de détails.

Amélioration : 2.1.0 - la prise en charge des géométries courbes a été introduite.

Amélioration : 2.2.0 - mesure sur sphéroïde effectuée avec GeographicLib pour une meilleure précision et robustesse. Nécessite PROJ >= 4.9.0 pour profiter de la nouvelle fonctionnalité.

Modifié : 3.0.0 - ne dépend plus de SFCGAL.

Exemples de géométrie

Exemple de géométrie - unités en degrés planaires 4326 est le long lat WGS 84, les unités sont des degrés.

Code

```
SELECT ST_Distance(
'SRID=4326;POINT(-72.1235 42.3521)'::geometry,
'SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry );
```

Exemple de géométrie - unités en mètres (SRID : 3857, proportionnel aux pixels sur les cartes web populaires). Bien que la valeur soit erronée, les valeurs proches peuvent être comparées correctement, ce qui en fait un bon choix pour des algorithmes tels que KNN ou KMeans.

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 3857) );
```

Exemple de géométrie - unités en mètres (SRID : 3857 comme ci-dessus, mais corrigé par cos(lat) pour tenir compte de la distorsion)

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 3857)
        ) * cosd(42.3521);
```

Exemple de géométrie - unités en mètres (SRID : 26986 Massachusetts state plane meters) (plus précis pour le Massachusetts)

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 26986),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 26986) );
```

Exemple de géométrie - unités en mètres (SRID : 2163 US National Atlas Equal area) (moins précis)

Code

```
SELECT ST_Distance(
ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 2163),
ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geometry, 2163) );
```

Exemples géographiques

Identique à l'exemple du type geometry, mais avec des unités en mètres - utiliser la sphère pour un calcul légèrement plus rapide et moins précis.

Code

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
    'SRID=4326;POINT(-72.1235 42.3521)'::geography as gg1,
    'SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)'::geography as gg2
    ) As foo  ;
```

Output

```
spheroid_dist   |   sphere_dist
-----------------+-----------------
 123.802076746848 | 123.475736916397
```

Voir aussi

ST_3DDistance, ST_DWithin, ST_DistanceSphere, ST_DistanceSpheroid, ST_MaxDistance, ST_HausdorffDistance, ST_FrechetDistan
ST_Transform

7.12.7 ST_3DDistance

ST_3DDistance — Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

Description

Renvoie la distance cartésienne minimale tridimensionnelle entre deux géométries en unités projetées (unités de référence spatiales).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette méthode implémente la spécification SQL/MM. SQL-MM ISO/IEC 13249-3

Disponibilité : 2.0.0

Modifié : 2.2.0 - Dans le cas de la 2D et de la 3D, Z n'est plus considéré comme égal à 0 en cas de Z manquant.

Modifié : 3.0.0 - Version SFCGAL supprimée

Exemples

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

Code

```
SELECT ST_3DDistance(
            ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)':: ↩
    geometry, 2163)
            ) As dist_3d,
```

```
            ST_Distance(
                ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45,-72.123 42.1546)':: ↩
    geometry, 2163)
            ) As dist_2d;
```

Output

```
dist_3d      |     dist_2d
-----------------+-----------------
 127.295059324629 | 126.66425605671
```

Multilinestring and polygon both 3D and 2D distance. Same example as 3D closest point example.

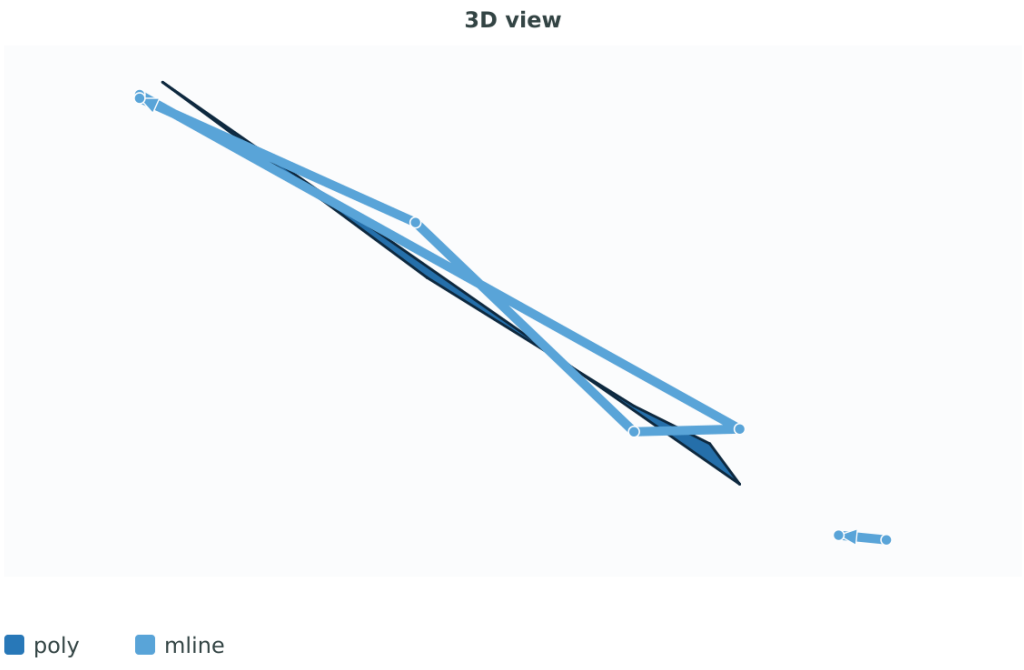
Code

```
SELECT ST_3DDistance(poly, mline) As dist3d,
    ST_Distance(poly, mline) As dist2d
        FROM (SELECT  'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))':: ↩
    geometry as poly,
                'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),(1 10 2,5  ↩
    20 1))'::geometry as mline) as foo;
```

Output

```
dist3d       | dist2d
------------------+--------
                0 |      0
```

Figure



Voir aussi

ST_Distance, ST_3DClosestPoint, ST_3DDWithin, ST_3DMaxDistance, ST_3DShortestLine, ST_Transform

7.12.8 ST_DistanceSphere

ST_DistanceSphere — Renvoie la distance minimale en mètres entre deux géométries lon/lat en utilisant un modèle de terre sphérique.

Synopsis

float **ST_DistanceSphere**(geometry geomlonlatA, geometry geomlonlatB, float8 radius=6371008);

Description

Renvoie la distance minimale en mètres entre deux points lon/lat. Utilise une terre sphérique et un rayon dérivé du sphéroïde défini par le SRID. Plus rapide que ST_DistanceSpheroid, mais moins précis. PostGIS Versions antérieures à la version 1.5 uniquement pour les points.

Disponibilité : 1.5 - la prise en charge d'autres types de géométrie que les points a été introduite. Les versions précédentes ne fonctionnaient qu'avec des points.

Modifié : 2.2.0 Dans les versions antérieures, cette fonction s'appelait ST_Distance_Sphere

Exemples

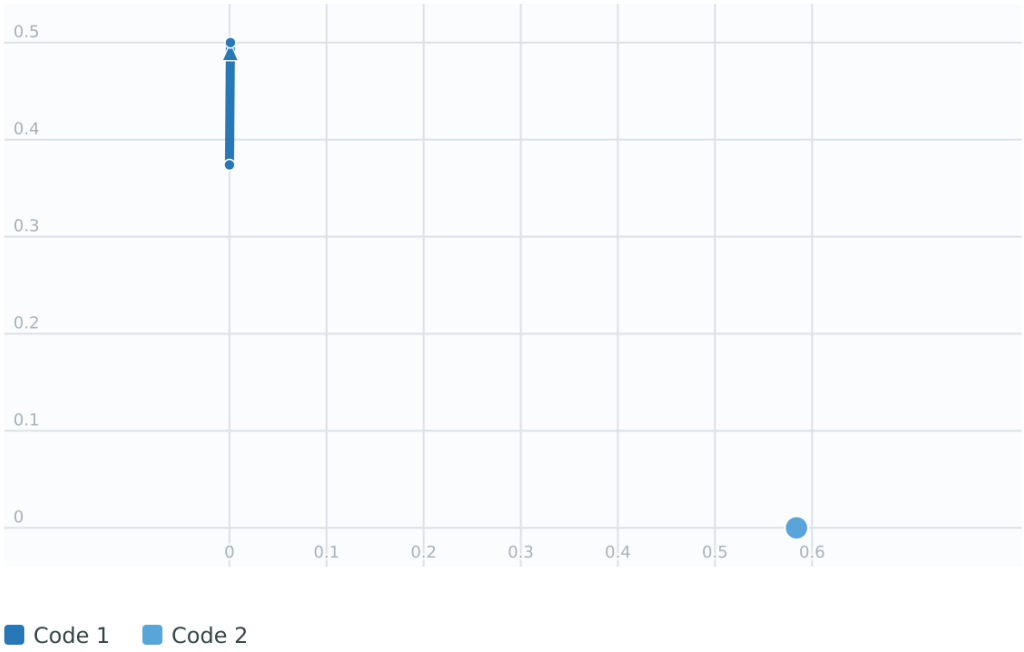
Code

```
WITH sample AS (
    SELECT
        ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) AS geom,
        ST_GeomFromText('POINT(-118 38)', 4326) AS pt
)
SELECT
    round(ST_DistanceSphere(ST_Centroid(geom), pt)::numeric, 2) AS dist_meters,
    round(ST_Distance(
        ST_Transform(ST_Centroid(geom), 32611),
        ST_Transform(pt, 32611)
    )::numeric, 2) AS dist_utm11_meters,
    round(ST_Distance(ST_Centroid(geom), pt)::numeric, 5) AS dist_degrees,
    round(ST_Distance(
        ST_Transform(geom, 32611),
        ST_Transform(pt, 32611)
    )::numeric, 2) AS min_dist_line_point_meters
FROM sample;
```

Output

```
dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
------------+-------------------+--------------+---------------------------
    70424.71 |          70438.00 |      0.72900 |                   65871.18
```

Figure



Voir aussi

ST_Distance, ST_DistanceSpheroid

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — Renvoie la distance minimale entre deux géométries lon/lat en utilisant un modèle de terre sphéroïdale.

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);

Description

Renvoie la distance minimale en mètres entre deux géométries lon/lat pour un sphéroïde donné. Voir l'explication des sphéroïdes donnée pour ST_LengthSpheroid.



Note
Cette fonction ne prend pas en compte le SRID de la géométrie. Elle suppose que les coordonnées de la géométrie sont basées sur le sphéroïde fourni.

Disponibilité : 1.5 - la prise en charge d'autres types de géométrie que les points a été introduite. Les versions précédentes ne fonctionnaient qu'avec des points.

Modifié : 2.2.0 Dans les versions précédentes, cette fonction était appelée ST_Distance_Sphéroïde

Exemples

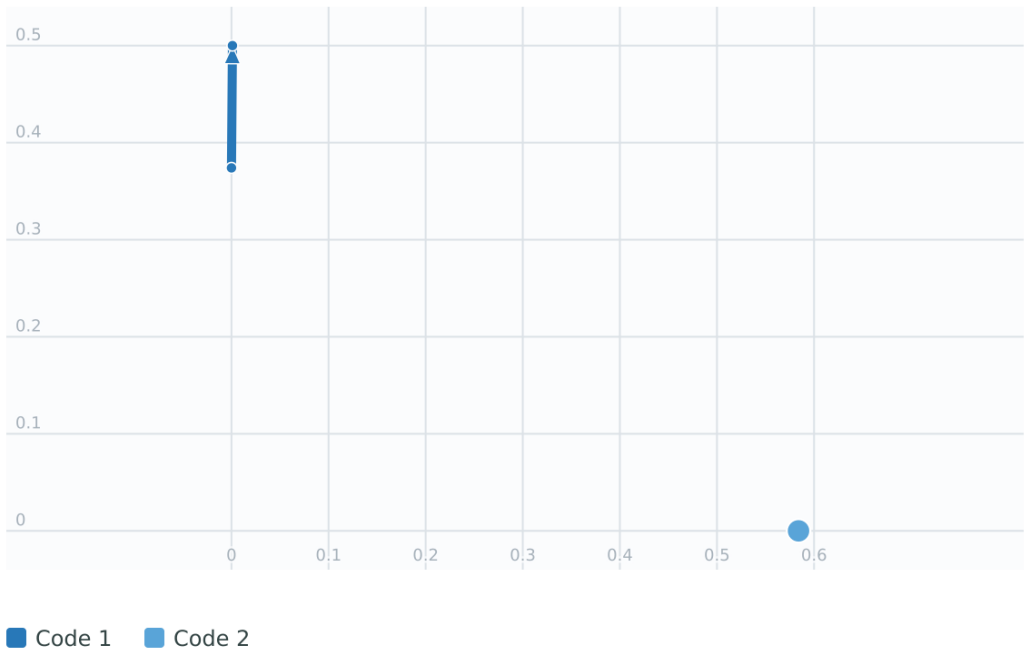
Code

```
WITH input AS (
  SELECT ST_Centroid(
           ST_GeomFromText(
             'LINESTRING(-118.584 38.374,-118.583 38.5)', 4326
           )
         ) AS source,
         ST_GeomFromText('POINT(-118 38)', 4326) AS target
)
SELECT round(
         ST_DistanceSpheroid(
           source,
           target,
           'SPHEROID["WGS 84",6378137,298.257223563]'
         )::numeric,
         2
       ) AS dist_meters_spheroid,
       round(ST_DistanceSphere(source, target)::numeric, 2)
         AS dist_meters_sphere,
       round(
         ST_Distance(
           ST_Transform(source, 32611),
           ST_Transform(target, 32611)
         )::numeric,
         2
       ) AS dist_utm11_meters
FROM input;
```

Output

```
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
---------------------+--------------------+-------------------
             70454.92 |           70424.71 |          70438.00
```

Figure



Voir aussi

ST_Distance, ST_DistanceSphere

7.12.10 ST_FrechetDistance

ST_FrechetDistance — Renvoie la distance de Fréchet entre deux géométries.

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

Description

Implémente l'algorithme de calcul de la distance de Fréchet limitée aux points discrets pour les deux géométries, basé sur Computing Discrete Fréchet Distance. La distance de Fréchet est une mesure de similarité entre les courbes qui tient compte de l'emplacement et de l'ordre des points le long des courbes. Elle est donc souvent meilleure que la distance de Hausdorff.

Lorsque le paramètre optionnel densifyFrac est spécifié, cette fonction effectue une densification des segments avant de calculer la distance de Fréchet discrète. Le paramètre densifyFrac définit la fraction par laquelle chaque segment doit être densifié. Chaque segment sera divisé en un certain nombre de sous-segments de longueur égale, dont la fraction de la longueur totale est la plus proche de la fraction donnée.

Les unités sont celles du système de référence spatiale des géométries.



Note
L'implémentation actuelle ne prend en charge que les sommets en tant qu'emplacements discrets. Elle pourrait être étendue pour permettre l'utilisation d'une densité arbitraire de points.



Note
Plus la valeur de densifyFrac spécifiée est petite, plus la distance de Fréchet est précise. Mais le temps de calcul et l'utilisation de la mémoire augmentent avec le carré du nombre de sous-segments.

Effectué par le module GEOS.

Disponibilité : 2.4.0 - nécessite GEOS >= 3.7.0

Exemples

Both lines are traversed from left to right. The first line is straight, while the second bows upward through its middle vertex. The point ordering matters, so keeping both inputs in the same direction makes the comparison clearer.

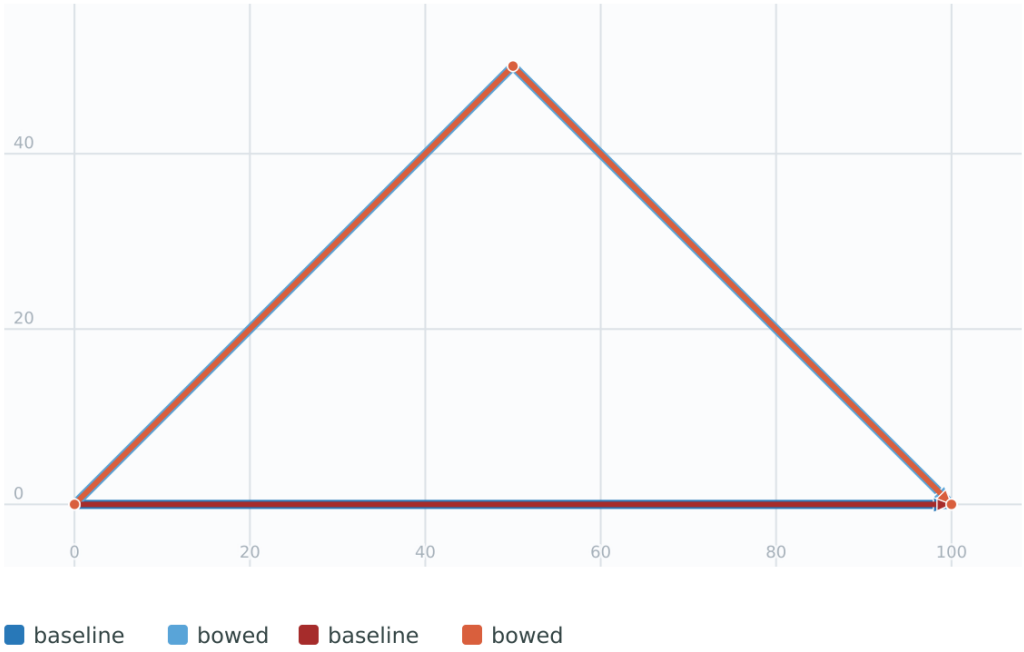
Code

```
WITH lines AS (
  SELECT 'LINESTRING (0 0,100 0)'::geometry AS baseline,
         'LINESTRING (0 0,50 50,100 0)'::geometry AS bowed
)
SELECT baseline AS baseline,
       bowed AS bowed,
       ST_FrechetDistance(baseline, bowed) AS discrete_distance,
       ST_FrechetDistance(baseline, bowed, 0.5) AS densified_distance
FROM lines;
```

Output

```
-[ RECORD 1 ]-----+---------------------------
baseline          | LINESTRING(0 0,100 0)
bowed             | LINESTRING(0 0,50 50,100 0)
discrete_distance | 70.7106781186548
densified_distance | 50
```

Figure



Voir aussi

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

ST_HausdorffDistance — Renvoie la distance de Hausdorff entre deux géométries.

Synopsis

float **ST_HausdorffDistance**(geometry g1, geometry g2);
float **ST_HausdorffDistance**(geometry g1, geometry g2, float densifyFrac);

Description

Renvoie la distance de Hausdorff entre deux géométries. La distance de Hausdorff est une mesure de la similarité ou de la dissemblance de deux géométries.

La fonction calcule en fait la "Discrete Hausdorff Distance". Il s'agit de la distance de Hausdorff calculée en des points discrets des géométries. Le paramètre `densifyFrac` peut être spécifié pour fournir une réponse plus précise en densifiant les segments avant de calculer la distance discrète de Hausdorff. Chaque segment est divisé en un certain nombre de sous-segments de longueur égale dont la fraction de la longueur du segment est la plus proche de la fraction donnée.

Les unités sont celles du système de référence spatiale des géométries.



Note
Cet algorithme n'est PAS équivalent à la distance standard de Hausdorff. Cependant, il calcule une approximation qui est correcte pour un large sous-ensemble de cas utiles. Un cas important est celui des lignes qui sont à peu près parallèles les unes aux autres et dont la longueur est à peu près égale. Il s'agit d'une métrique utile pour l'appariement des lignes.

Disponibilité : 1.5.0

Exemples

Hausdorff distance and minimum distance between two lines. The Hausdorff witness segment from the original illustration is retained and its length is checked against the computed distance.

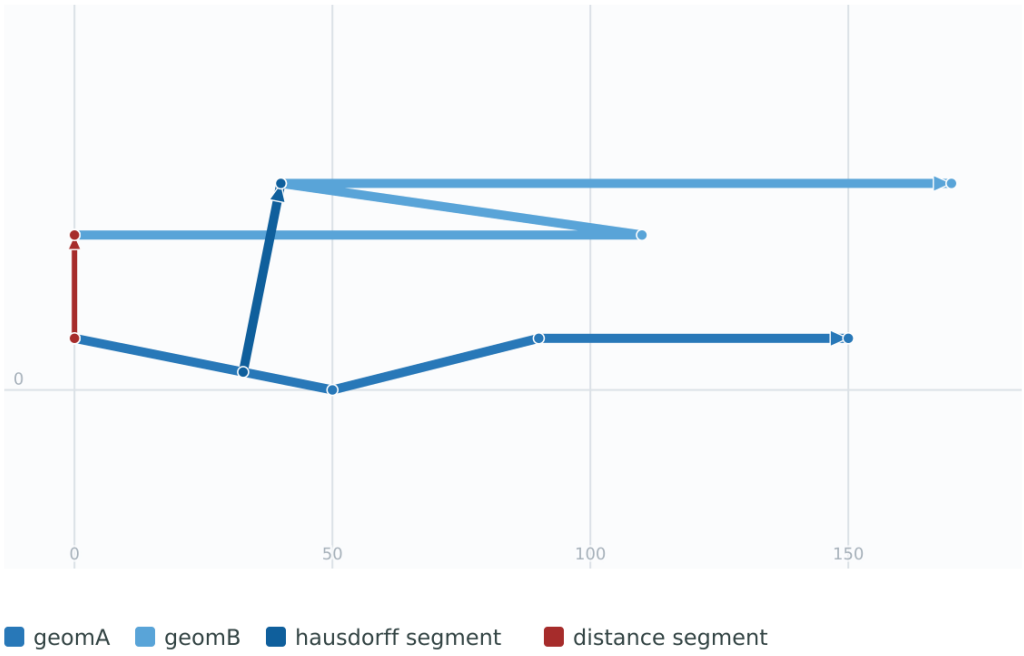
Code

```
WITH lines AS (
  SELECT 'LINESTRING (20 70,70 60,110 70,170 70)'::geometry AS geomA,
         'LINESTRING (20 90,130 90,60 100,190 100)'::geometry AS geomB,
         'LINESTRING (52.69230769230769 63.46153846153846,60 100)'::geometry AS ↩
    hausdorff_segment
)
SELECT round(ST_HausdorffDistance(geomA, geomB)::numeric, 6) AS hausdorff_distance,
       round(ST_Distance(geomA, geomB)::numeric, 6) AS distance,
       round(ST_Length(hausdorff_segment)::numeric, 6) AS witness_length,
       ST_ShortestLine(geomA, geomB) AS distance_segment
FROM lines;
```

Output

```
37.262066 | 20.000000 | 37.262066 | LINESTRING(20 70,20 90)
```

Figure

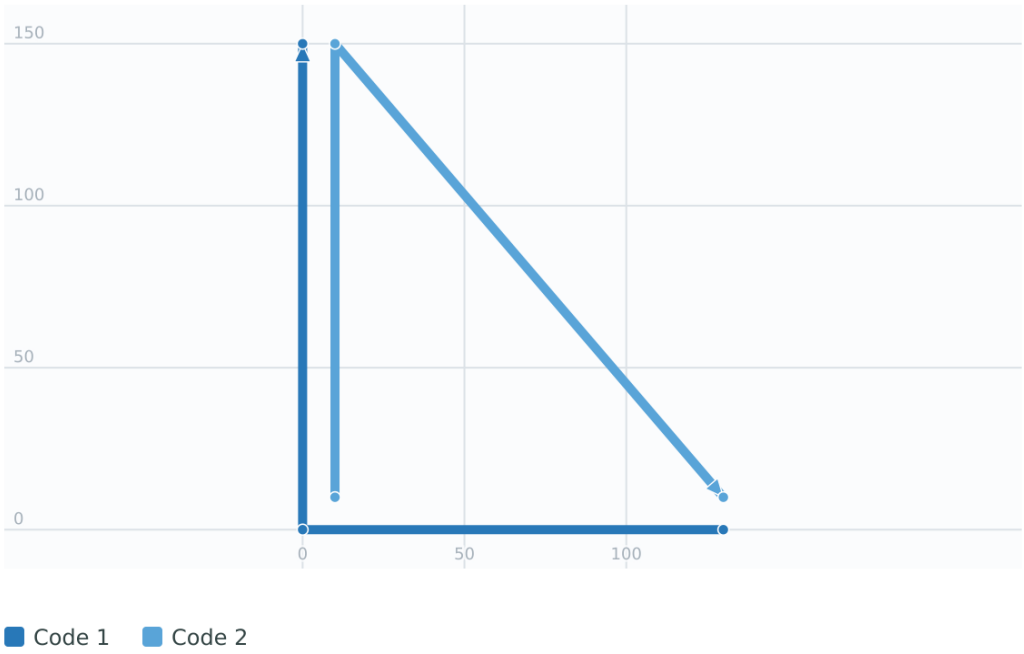


Exemple: Distance de Hausdorff avec densification.

Code

```
SELECT ST_HausdorffDistance(
            'LINESTRING (130 0,0 0,0 150)'::geometry,
            'LINESTRING (10 10,10 150,130 10)'::geometry,
            0.5);
```

Figure



Exemple: Pour chaque bâtiment, trouvez la parcelle qui le représente le mieux. Tout d'abord, nous exigeons que la parcelle intersecte la géométrie du bâtiment. `DISTINCT ON` garantit que chaque bâtiment ne sera listé qu'une seule fois. `ORDER BY .. ST_HausdorffDistance` sélectionne la parcelle qui est la plus similaire au bâtiment.

Code

```
SELECT DISTINCT ON (buildings.gid) buildings.gid, parcels.parcel_id
   FROM buildings
       INNER JOIN parcels
       ON ST_Intersects(buildings.geom, parcels.geom)
   ORDER BY buildings.gid, ST_HausdorffDistance(buildings.geom, parcels.geom);
```

Voir aussi

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — Renvoie la longueur 2D d'une géométrie linéaire.

Synopsis

float **ST_Length**(geometry a_2dlinestring);
float **ST_Length**(geography geog, boolean use_spheroid = true);

Description

Pour les types geometry : renvoie la longueur cartésienne 2D de la géométrie s'il s'agit d'une LineString, MultiLineString, ST_Curve, ST_MultiCurve. Pour les géométries aréolaires, 0 est renvoyé ; utilisez ST_Perimeter à la place. Les unités de longueur sont déterminées par le système de référence spatial de la géométrie.

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. The spheroid is specified by the SRID; when no SRID is provided, WGS84 is used. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

Actuellement, pour la géométrie, il s'agit d'un alias de ST_Length2D, mais cela pourrait changer pour prendre en charge des dimensions plus élevées.

Modifié : 2.0.0 Rupture -- dans les versions précédentes, appliquer ceci à un MULTI/POLYGONE de type geography donnait le périmètre du POLYGONE/MULTIPOLYGONE. Dans la version 2.0.0, cette fonction a été modifiée pour retourner 0 afin d'être en ligne avec le comportement de la géométrie. Veuillez utiliser ST_Perimeter si vous souhaitez obtenir le périmètre d'un polygone



Note
Pour geography, le calcul utilise par défaut un modèle sphéroïdal. Pour utiliser le calcul sphérique, plus rapide mais moins précis, utilisez ST_Length(gg,false) ;

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 7.1.2, 9.3.4

Disponibilité : 1.5.0 La prise en charge du type geography a été introduite dans la version 1.5.

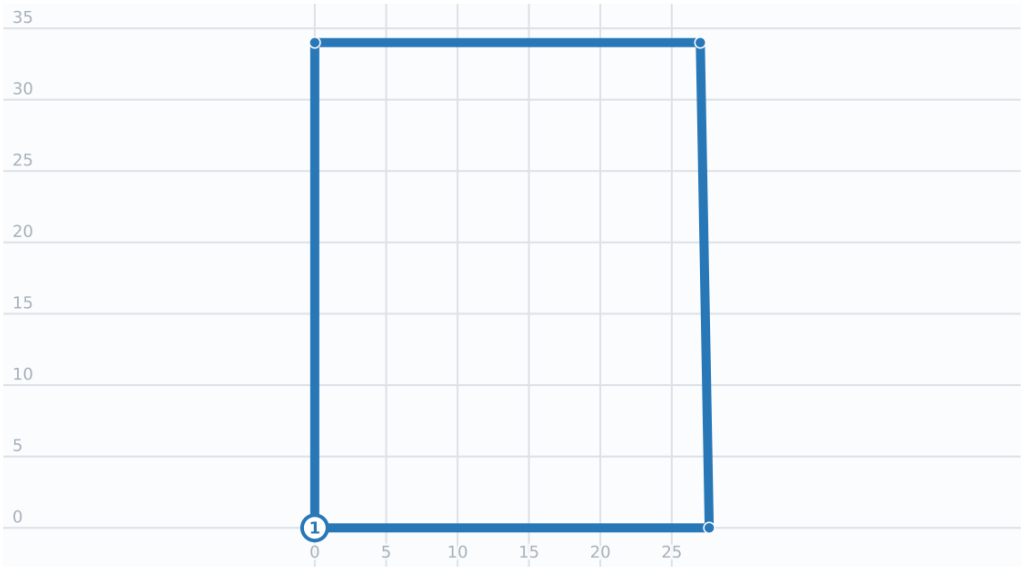
Exemples de géométrie

Renvoie la longueur en pieds de la ligne. Notez que cette longueur est exprimée en pieds car EPSG:2249 correspond aux pieds du plan de l'État du Massachusetts

Code

```
SELECT ST_Length(ST_GeomFromText(
'LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',
2249));
```

Figure



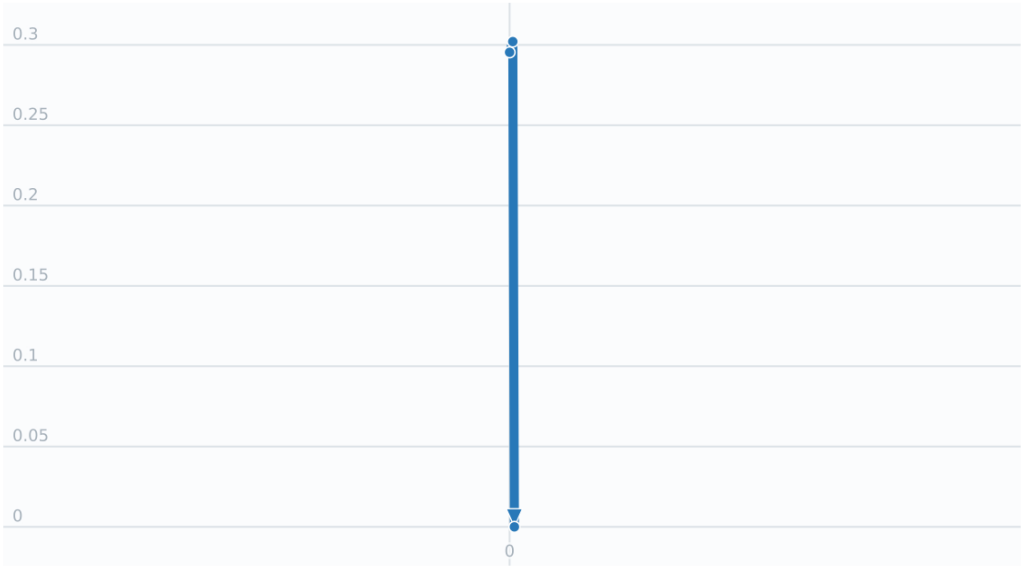
Code

This example transforms a WGS 84 LineString to Massachusetts State Plane meters.

Code

```
SELECT ST_Length(ST_Transform(
        'SRID=4326;LINESTRING(-72.1260 42.45,-72.1240 42.45666,-72.123 42.1546)'::geometry,
            26986
    )
);
```

Figure



Code

Exemples géographiques

Retourne la longueur d'une ligne de type geography avec le SRID WGS 84

This example uses the default spheroid calculation.

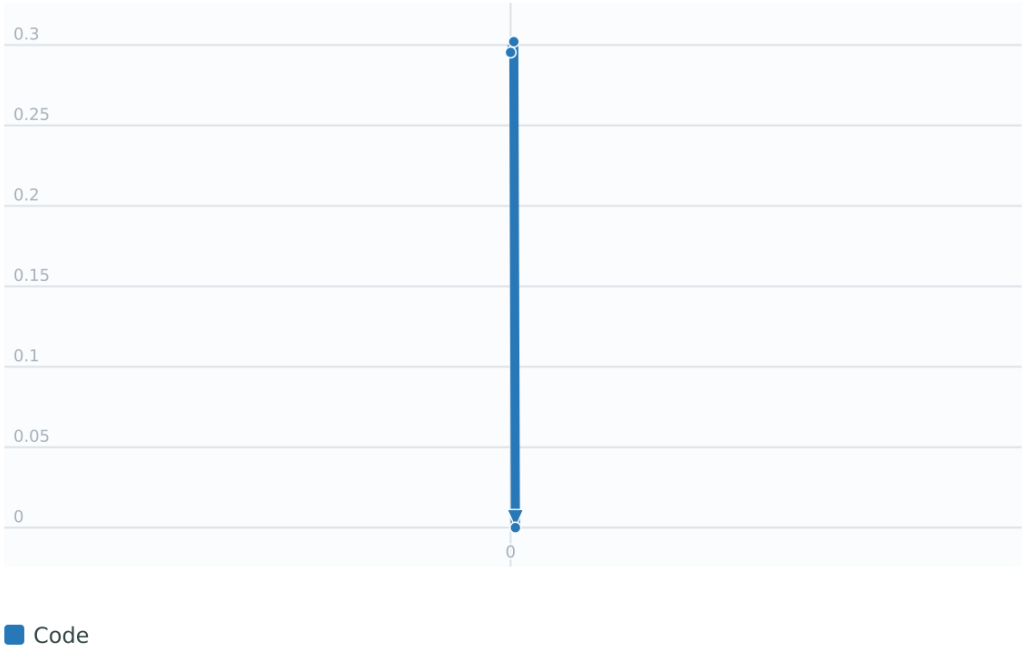
Code

```
SELECT ST_Length(the_geog) As length_spheroid, ST_Length(the_geog, false) As length_sphere
FROM (SELECT ST_GeographyFromText('SRID=4326;LINESTRING(-72.1260 42.45,-72.1240 ↩
    42.45666,-72.123 42.1546)') As the_geog)
 As foo;
```

Output

```
length_spheroid  |  length_sphere
-----------------+-----------------
34310.5703627284 | 34346.20609607416
```

Figure



Voir aussi

ST_GeographyFromText, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.13 ST_Length2D

ST_Length2D — Renvoie la longueur 2D d'une géométrie linéaire. Alias de `ST_Length`

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

Description

Renvoie la longueur 2D de la géométrie s'il s'agit d'une ligne ou d'une multiligne. C'est un alias de `ST_Length`

Voir aussi

ST_Length, ST_3DLength

7.12.14 ST_3DLength

ST_3DLength — Renvoie la longueur 3D d'une géométrie linéaire.

Synopsis

float **ST_3DLength**(geometry a_3dlinestring);

Description

Renvoie la longueur tridimensionnelle ou bidimensionnelle de la géométrie s'il s'agit d'une LineString ou d'une MultiLineString. Pour les lignes à 2 dimensions, la longueur à 2 dimensions est renvoyée (comme ST_Length et ST_Length2D)

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 7.1, 10.3

Modifié : 2.0.0 Dans les versions précédentes, cette fonction était appelée ST_Length3D

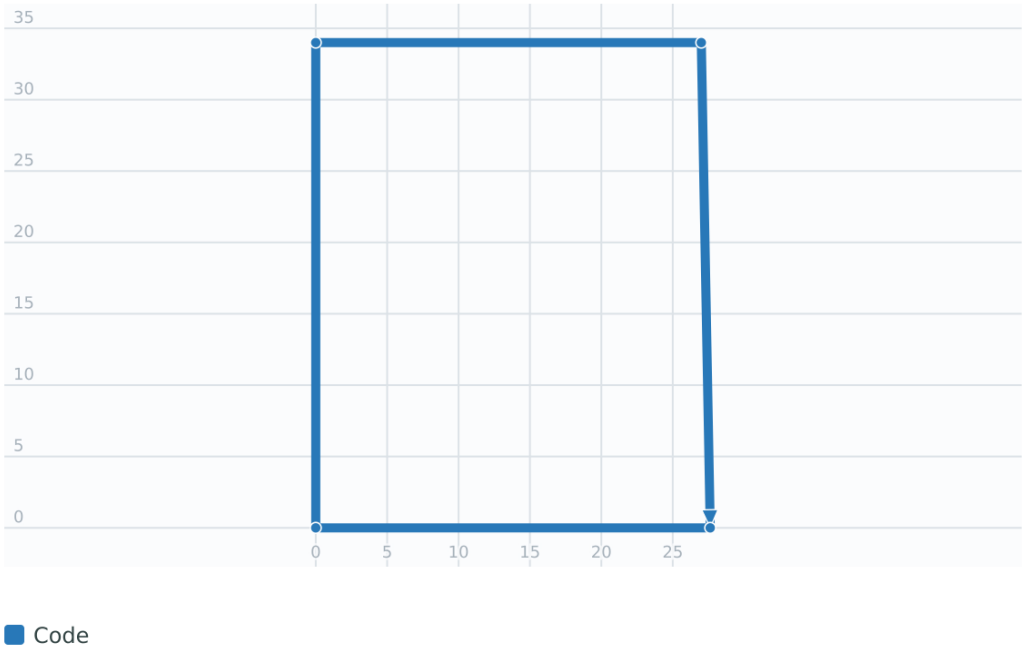
Exemples

Longueur de retour en pieds pour un câble 3D. Notez que cette longueur est exprimée en pieds car EPSG:2249 correspond aux pieds du plan de l'État du Massachusetts

Code

```
SELECT ST_3DLength(ST_GeomFromText(
'LINESTRING(743238 2967416 1,743238 2967450 1,743265 2967450 3,
743265.625 2967416 3,743238 2967416 3)',
2249));
```

Figure



Voir aussi

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — Renvoie la longueur/périmètre 2D ou 3D d'une géométrie lon/lat sur un sphéroïde.

Synopsis

float **ST_LengthSpheroid**(geometry a_geometry, spheroid a_spheroid);

Description

Calculates the length or perimeter of a geometry on a spheroid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The spheroid is specified by a text value as follows:

```
SPHEROID[<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]
```

Par exemple :

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Disponibilité : 1.2.2

Modifié : 2.2.0 Dans les versions précédentes, cette fonction s'appelait ST_Length_Spheroid et avait l'alias ST_3DLength_Spheroid

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Measure a table geometry directly on a spheroid when the coordinates are stored in longitude and latitude.

Code

```
SELECT ST_LengthSpheroid(
    geometry_column,
    'SPHEROID["GRS_1980",6378137,298.257222101]'
    )
FROM geometry_table;
```

This example measures a 2D MultiLineString and each component separately on the same spheroid.

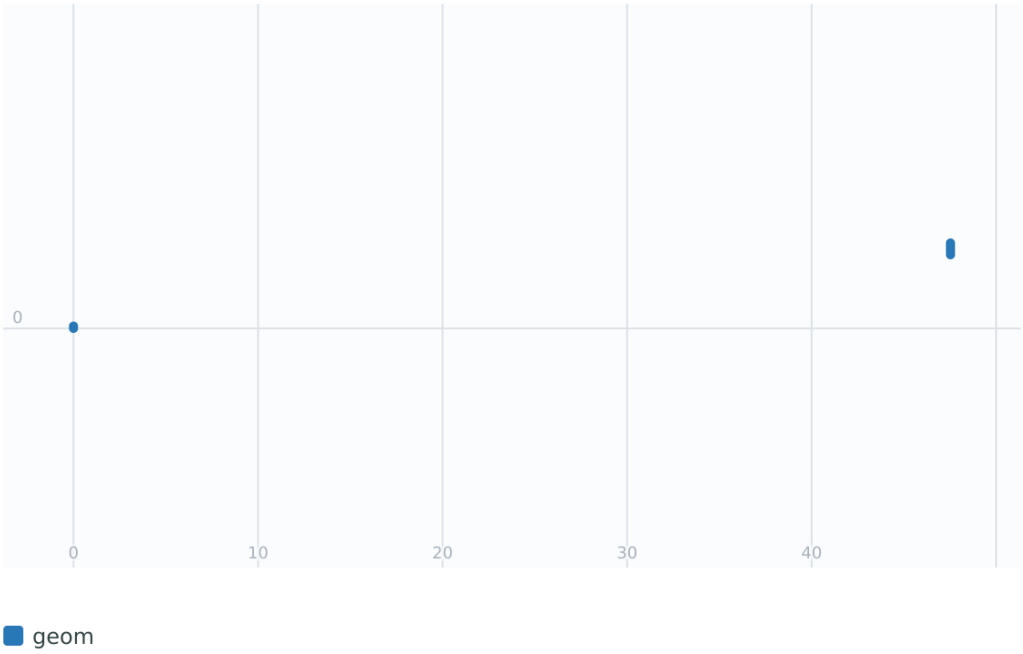
Code

```
WITH sample AS (
  SELECT 'MULTILINESTRING((-118.584 38.374,-118.583 38.5),
          (-71.05957 42.3589,-71.061 43))'::geometry AS geom,
         CAST('SPHEROID["GRS_1980",6378137,298.257222101]' AS spheroid) AS sph_m
)
SELECT ST_LengthSpheroid(geom, sph_m) AS tot_len,
       ST_LengthSpheroid(ST_GeometryN(geom, 1), sph_m) AS len_line1,
       ST_LengthSpheroid(ST_GeometryN(geom, 2), sph_m) AS len_line2
FROM sample;
```

Output

```
tot_len      |    len_line1     |    len_line2
-----------------+------------------+------------------
 85204.52077117894 | 13986.87252824332 | 71217.64824293563
```

Figure



This example uses 3D input; Z contributes to the measured result.

Code

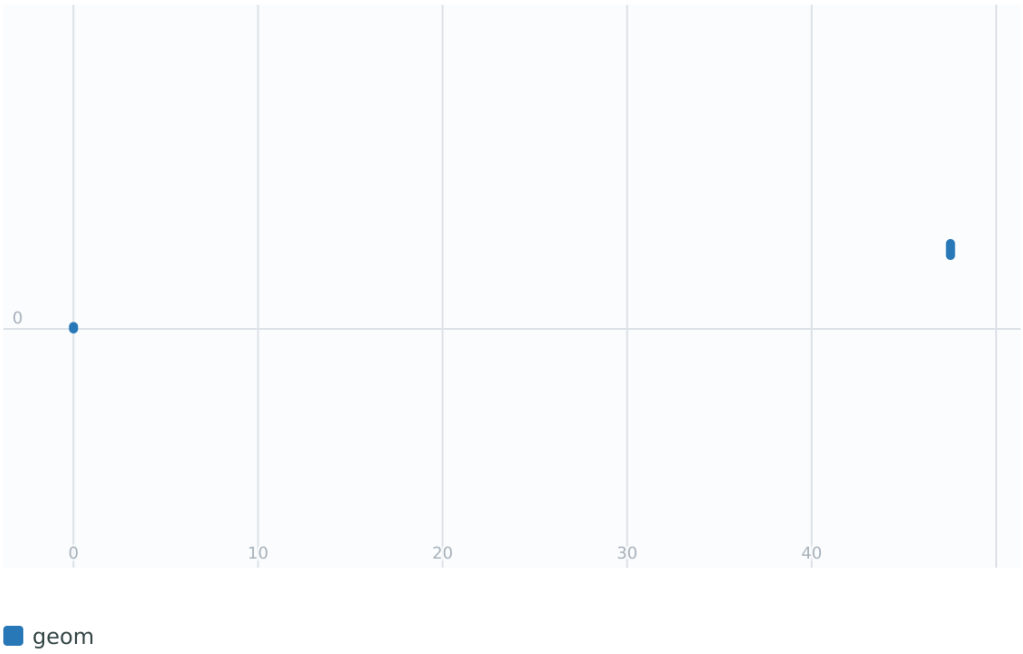
```
WITH sample AS (
  SELECT 'MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30),
          (-71.05957 42.3589 75,-71.061 43 90))'::geometry AS geom,
         CAST('SPHEROID["GRS_1980",6378137,298.257222101]' AS spheroid) AS sph_m
)
```

```
SELECT ST_LengthSpheroid(geom, sph_m) AS tot_len,
       ST_LengthSpheroid(ST_GeometryN(geom, 1), sph_m) AS len_line1,
       ST_LengthSpheroid(ST_GeometryN(geom, 2), sph_m) AS len_line2
FROM sample;
```

Output

```
tot_len      |    len_line1    |    len_line2
-----------------+-----------------+-----------------
 85204.52592562366 | 13986.876103023424 | 71217.64982260024
```

Figure



Voir aussi

ST_GeometryN, ST_Length

7.12.16 ST_LongestLine

ST_LongestLine — Renvoie la ligne 2D la plus longue entre deux géométries.

Synopsis

geometry **ST_LongestLine**(geometry g);
geometry **ST_LongestLine**(geometry g1, geometry g2);

Description

Renvoie la plus longue ligne bidimensionnelle entre les points de deux géométries. La ligne renvoyée commence sur `g1` et se termine sur `g2`.

La ligne la plus longue se trouve toujours entre deux sommets. La fonction renvoie la première ligne la plus longue si plusieurs sont trouvées. La longueur de la ligne est égale à la distance renvoyée par ST_MaxDistance.

If only one geometry is provided, or g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by ST_MinimumBoundingCircle.

Disponibilité : 1.5.0

Enhanced: 3.7.0 - support for a single geometry input.

Exemples

Longest line between a Point and a LineString.

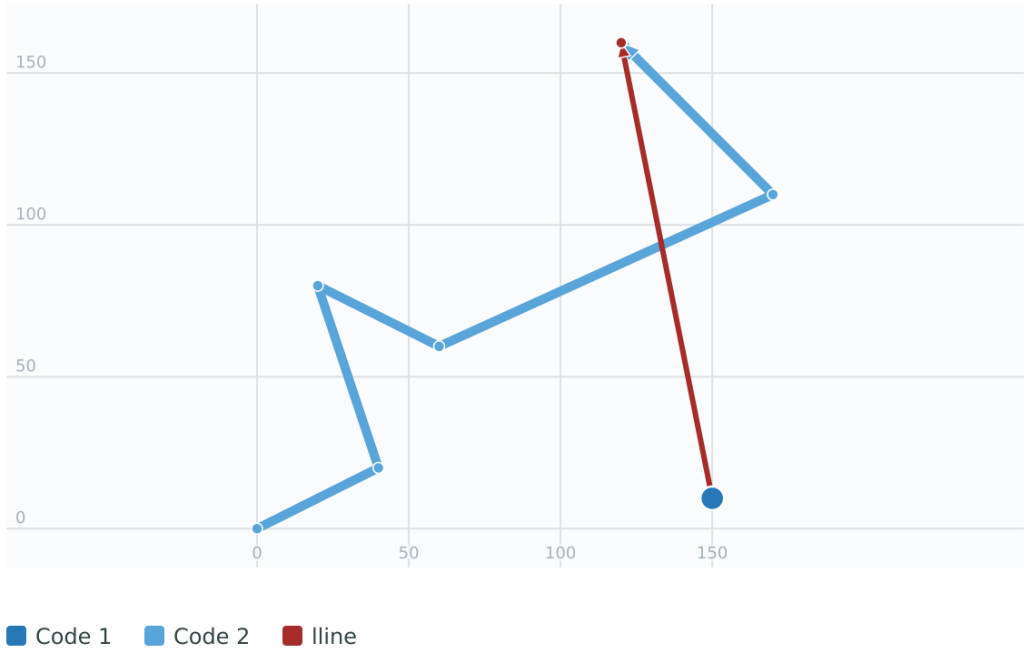
Code

```
SELECT ST_LongestLine(
        'POINT (160 40)',
        'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)' ) AS lline;
```

Output

```
LINESTRING(160 40,130 190)
```

Figure



Longest line between two polygons.

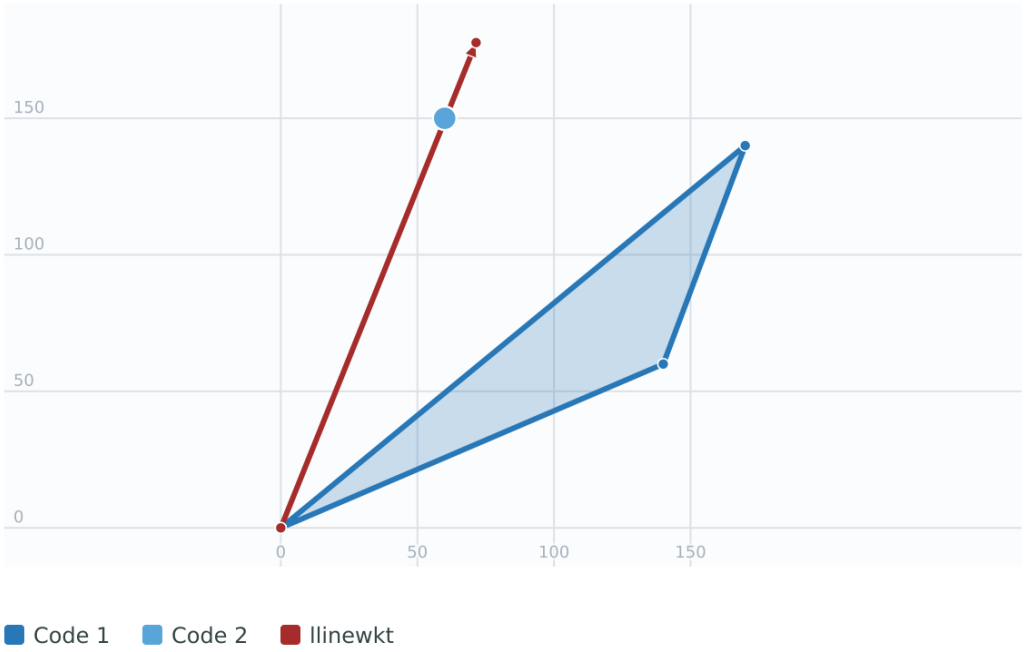
Code

```
SELECT ST_LongestLine(
        'POLYGON ((190 150,20 10,160 70,190 150))',
        ST_Buffer('POINT(80 160)', 30)
            ) AS llinewkt;
```

Output

```
LINESTRING(20 10,91.4805029709527 187.7163859753386)
```

Figure



Ligne la plus longue traversant une seule géométrie. La longueur de la ligne est égale à la distance maximale. Les extrémités de la ligne sont situées sur le cercle de délimitation minimal.

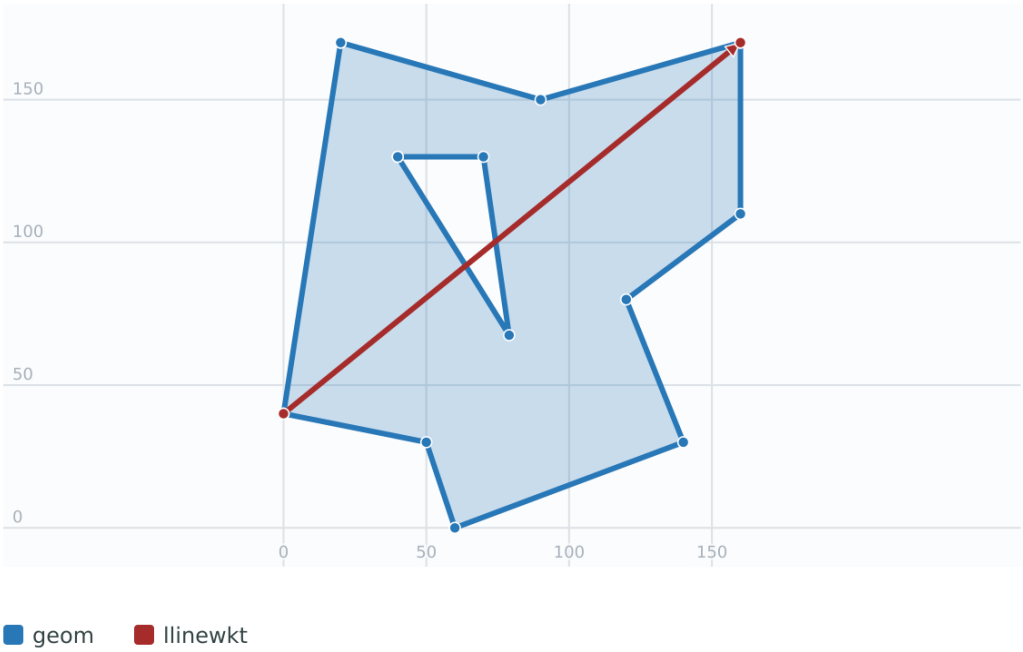
Code

```
SELECT ST_LongestLine(geom) AS llinewkt,
                  ST_MaxDistance(geom) AS max_dist,
                  ST_Length(ST_LongestLine(geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180,110 160,180 180,180 120,140 90,160 40,80 10,70 40,20 50,40 ↩
    180),
              (60 140,99 77.5,90 140,60 140))'::geometry AS geom) AS t;
```

Output

```
llinewkt          |      max_dist      |       lenll
--------------------------+--------------------+--------------------
 LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
```

Figure



Voir aussi

ST_MaxDistance, ST_ShortestLine, ST_3DLongestLine, ST_MinimumBoundingCircle

7.12.17 ST_3DLongestLine

ST_3DLongestLine — Renvoie la ligne 3D la plus longue entre deux géométries

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

Description

Renvoie la ligne tridimensionnelle la plus longue entre deux géométries. La fonction renvoie la première ligne la plus longue s'il y en a plusieurs. La ligne retournée commence en g1 et se termine en g2. La longueur 3D de la ligne est égale à la distance renvoyée par ST_3DMaxDistance.

Disponibilité : 2.0.0

Modifié : 2.2.0 - si 2 géométries 2D sont saisies, un point 2D est renvoyé (au lieu de l'ancien comportement supposant 0 pour Z manquant). Dans le cas de 2D et 3D, Z n'est plus supposé être 0 pour Z manquant.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Longest line between a LineString and a Point, comparing the 3D and 2D results.

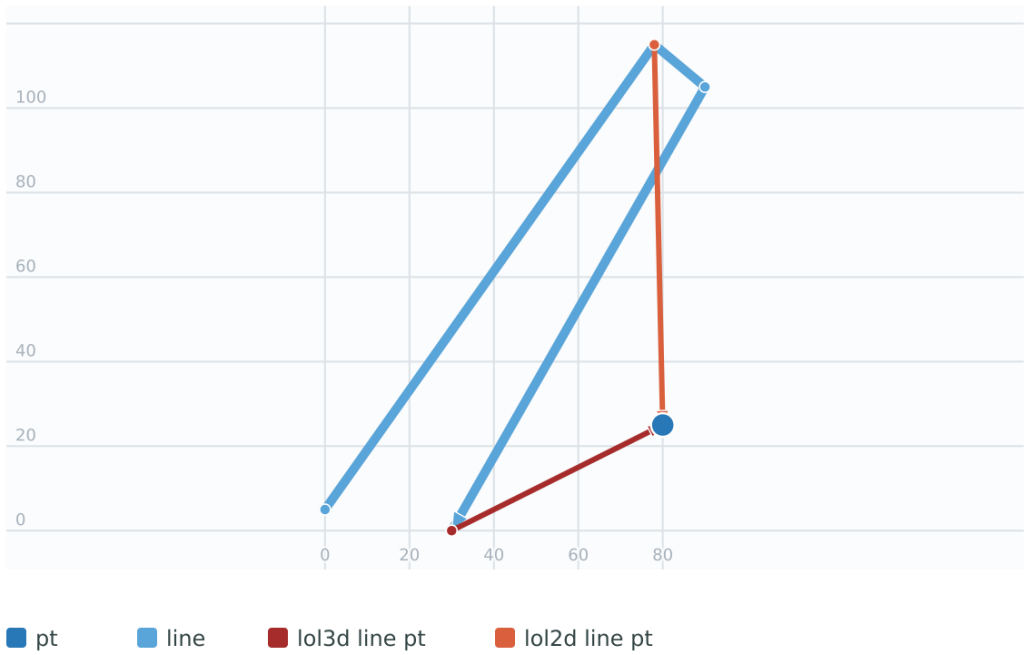
Code

```
SELECT ST_3DLongestLine(line, pt) AS lol3d_line_pt, ST_LongestLine(line, pt) AS  ↩
    lol2d_line_pt
FROM (
    SELECT 'POINT(100 100 30)'::geometry AS pt,
           'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)'::geometry AS line
) AS foo;
```

Output

```
lol3d_line_pt           |       lol2d_line_pt
--------------------------------+---------------------------
 LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)
```

Figure



Longest line between a LineString and a MultiPoint, comparing the 3D and 2D results.

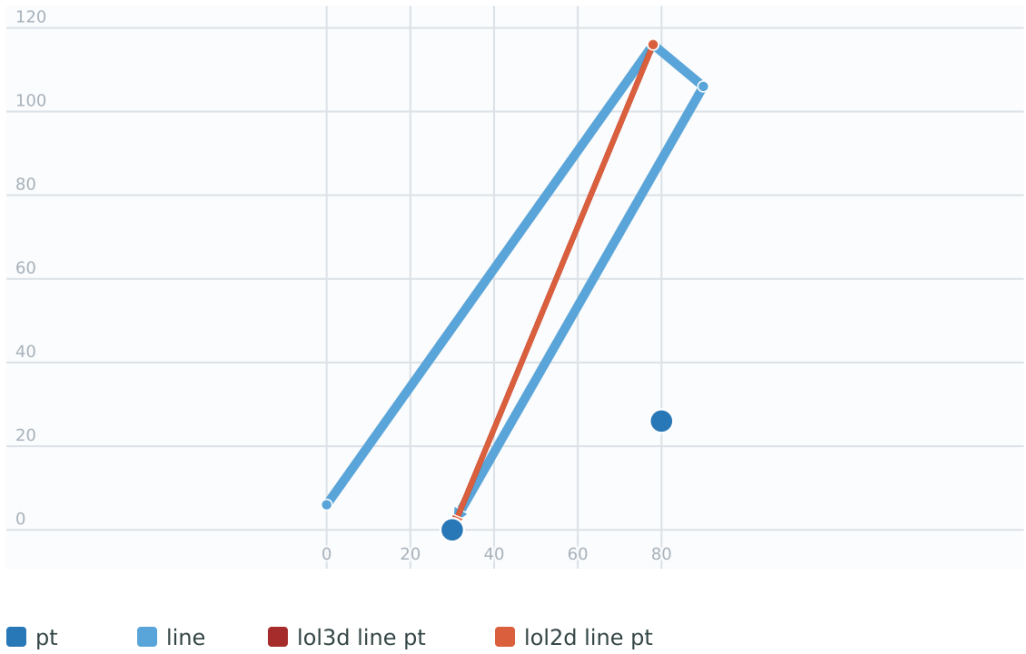
Code

```
SELECT ST_3DLongestLine(line, pt) AS lol3d_line_pt, ST_LongestLine(line, pt) AS  ↩
    lol2d_line_pt
FROM (
    SELECT 'MULTIPOINT(100 100 30,50 74 1000)'::geometry AS pt,
           'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 900)'::geometry AS line
) AS foo;
```

Output

```
lol3d_line_pt          |      lol2d_line_pt
-------------------------------+--------------------------
 LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)
```

Figure



Longest line between a Polygon and a MultiLineString, comparing the 3D and 2D results.

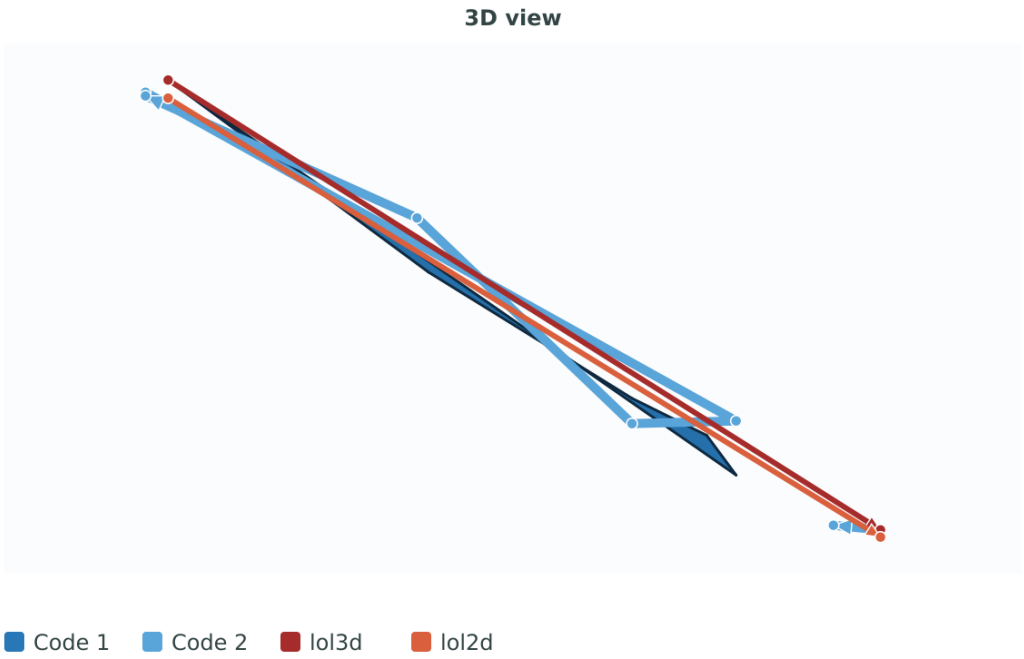
Code

```
SELECT ST_3DLongestLine(poly, mline) AS lol3d, ST_LongestLine(poly, mline) AS lol2d
FROM (
    SELECT ST_GeomFromEWKT(
            'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))'
        ) AS poly,
        ST_GeomFromEWKT(
            'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),
            (1 10 2,5 20 1))'
        ) AS mline
) AS foo;
```

Output

```
lol3d             |          lol2d
------------------------------+-------------------------
 LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)
```

Figure



Voir aussi

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_3DShortestLine, ST_3DMaxDistance

7.12.18 ST_MaxDistance

ST_MaxDistance — Renvoie la plus grande distance 2D entre deux géométries en unités projetées.

Synopsis

float **ST_MaxDistance**(geometry g);
float **ST_MaxDistance**(geometry g1, geometry g2);

Description

Renvoie la distance maximale en 2D entre deux géométries, en unités projetées. La distance maximale se situe toujours entre deux sommets. C'est la longueur de la ligne renvoyée par ST_LongestLine.

If only one geometry is provided, or g1 and g2 are the same geometry, returns the distance between the two vertices farthest apart in that geometry.

Disponibilité : 1.5.0

Enhanced: 3.7.0 - support for a single geometry input.

Exemples

Distance maximale entre un point et des lignes.

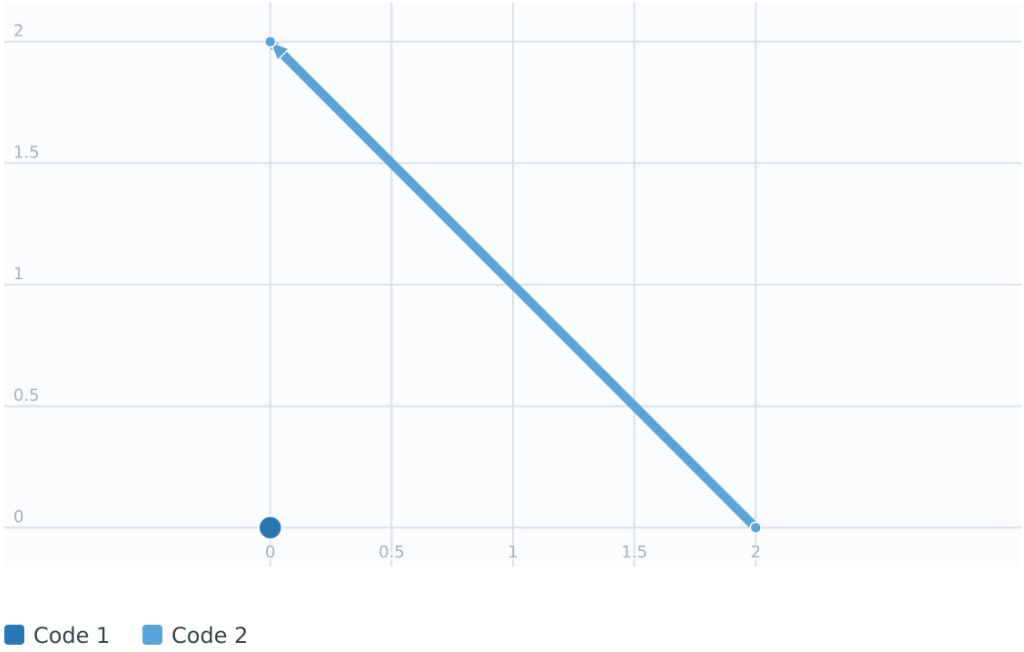
Code

```
SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
2
```

Figure



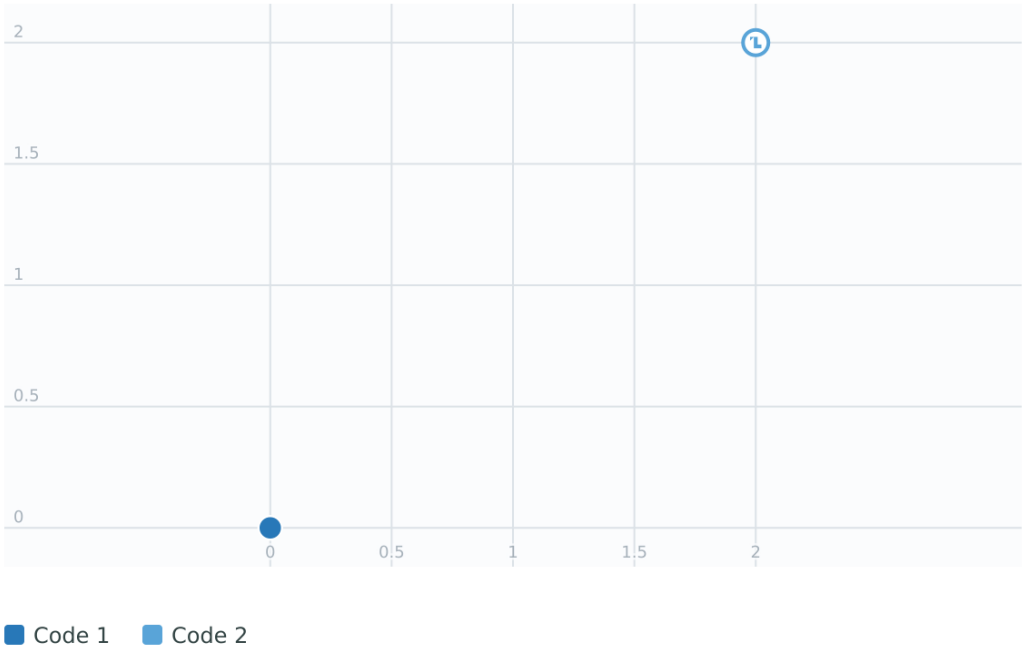
Code

```
SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 2,2 2 )'::geometry);
```

Output

```
2.82842712474619
```

Figure



Distance maximale entre les sommets d'une même géométrie.

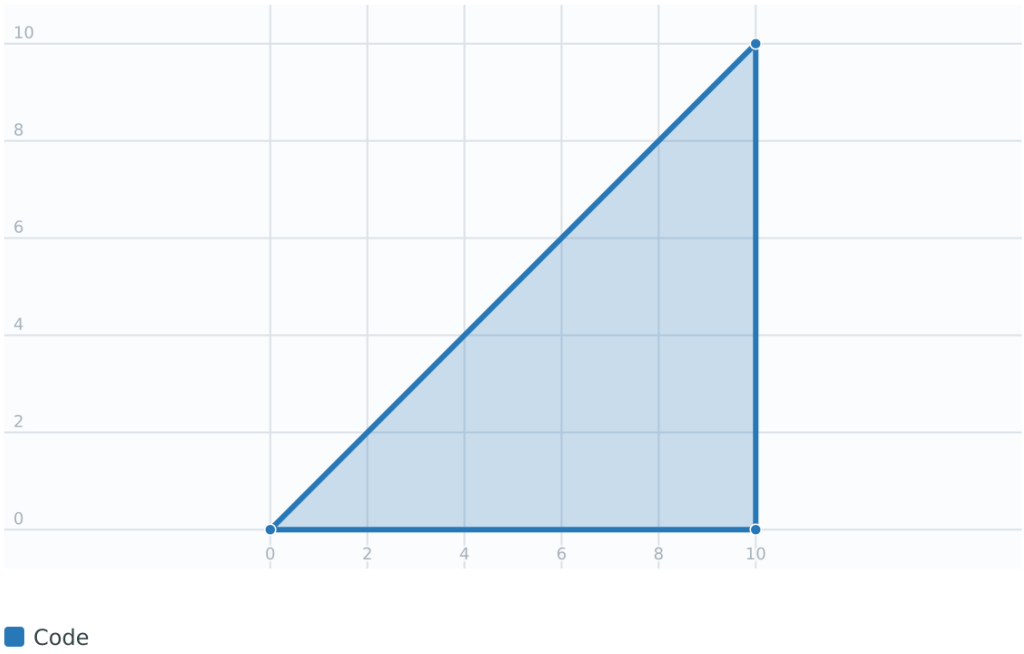
Code

```
SELECT ST_MaxDistance('POLYGON ((10 10,10 0,0 0,10 10))'::geometry);
```

Output

```
14.142135623730951
```

Figure



Voir aussi

ST_Distance, ST_LongestLine, ST_DFullyWithin

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — Renvoie la distance maximale cartésienne 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.

Synopsis

float **ST_3DMaxDistance**(geometry g1, geometry g2);

Description

Renvoie la distance cartésienne maximale tridimensionnelle entre deux géométries en unités projetées (unités de référence spatiales).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Disponibilité : 2.0.0

Modifié : 2.2.0 - Dans le cas de la 2D et de la 3D, Z n'est plus considéré comme égal à 0 en cas de Z manquant.

Exemples

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line) Note that currently no vertical datum support so Z is not transformed and assumed to be same units as final.

Code

```
SELECT ST_3DMaxDistance(
            ST_Transform('SRID=4326;POINT(-72.1235 42.3521 10000)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)':: ↩
    geometry, 2163)
            ) As dist_3d,
            ST_MaxDistance(
                ST_Transform('SRID=4326;POINT(-72.1235 42.3521 10000)'::geometry, 2163),
                ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15,-72.123 42.1546 20)':: ↩
    geometry, 2163)
            ) As dist_2d;
```

Output

```
dist_3d      |     dist_2d
-----------------+-----------------
 24383.7467488441 | 22247.8472107251
```

Voir aussi

ST_Distance, ST_3DDWithin, ST_3DMaxDistance, ST_Transform

7.12.20 ST_MinimumClearance

ST_MinimumClearance — Renvoie la clearance (le dégagement) d'une géométrie, une mesure de la robustesse d'une géométrie.

Synopsis

float **ST_MinimumClearance**(geometry g);

Description

Il est possible qu'une géométrie réponde aux critères de validité selon ST_IsValid (polygones) ou ST_IsSimple (lignes), mais qu'elle devienne invalide si l'un de ses sommets est déplacé d'une petite distance. Cela peut se produire en raison d'une perte de précision lors de la conversion vers des formats texte (tels que WKT, KML, GML, GeoJSON) ou des formats binaires qui n'utilisent pas de coordonnées en virgule flottante à double précision (par exemple, MapInfo TAB).

La clearance (le dégagement) minimum est une mesure quantitative de la résistance d'une géométrie aux changements de précision des coordonnées. Il s'agit de la plus grande distance à laquelle les sommets de la géométrie peuvent être déplacés sans créer une géométrie invalide. Des valeurs plus élevées de l'espace libre minimum indiquent une plus grande robustesse.

Si une géométrie a un dégagement minimal de `e`, alors.. :
— Aucun des deux sommets distincts de la géométrie n'est plus proche que la distance `e`.
— Aucun sommet n'est plus proche que `e` d'un segment de ligne dont il n'est pas l'extrémité.
S'il n'existe pas de dégagement minimum pour une géométrie (par exemple un point unique ou un MultiPoint dont les points sont identiques), la valeur de retour est `Infinity`.

Pour éviter les problèmes de validité causés par la perte de précision, ST_ReducePrecision peut réduire la précision des coordonnées tout en s'assurant que la géométrie polygonale reste valide.

Disponibilité : 2.3.0

Exemples

Code

```
SELECT ST_MinimumClearance('POLYGON ((0 0,1 0,1 1,0.5 3.2e-4,0 0))');
```

Output

```
0.00032
```

Figure



Voir aussi

ST_MinimumClearanceLine, ST_IsSimple, ST_IsValid, ST_ReducePrecision

7.12.21 ST_MinimumClearanceLine

ST_MinimumClearanceLine — Renvoie la chaîne de lignes à deux points couvrant le dégagement (clearance) minimum d'une géométrie.

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

Description

Renvoie les 2 points d'une LineString couvrant le dégagement (clearance) minimum d'une géométrie. Si la géométrie n'a pas de dégagement (clearance) minimum, `LINESTRING EMPTY` est retourné.

Effectué par le module GEOS.

Disponibilité : 2.3.0 - nécessite GEOS >= 3.6.0

Exemples

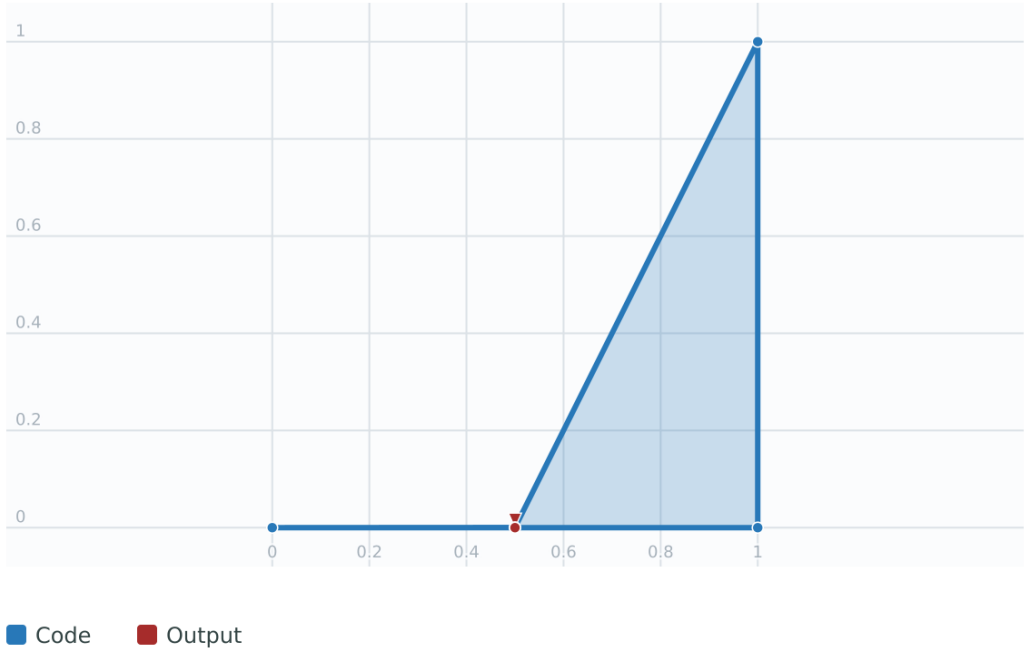
Code

```
SELECT ST_MinimumClearanceLine('POLYGON ((0 0,1 0,1 1,0.5 3.2e-4,0 0))');
```

Output

```
LINESTRING(0.5 0.00032,0.5 0)
```

Figure



Voir aussi

ST_MinimumClearance

7.12.22 ST_Perimeter

ST_Perimeter — Renvoie la longueur de la limite d'une géométrie polygonale ou d'une géographie.

Synopsis

float **ST_Perimeter**(geometry g1);
float **ST_Perimeter**(geography geog, boolean use_spheroid = true);

Description

Renvoie le périmètre 2D d'une geometry/geography s'il s'agit d'une ST_Surface, ST_MultiSurface (Polygone, MultiPolygone). 0 est retourné pour les géométries non réelles. Pour les géométries linéaires, utiliser ST_Length. Pour les types geometry, les unités de mesure du périmètre sont spécifiées par le système de référence spatiale de la géométrie.

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. The spheroid is specified by the SRID; when no SRID is provided, WGS84 is used. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

Il s'agit actuellement d'un alias de ST_Perimeter2D, mais cela pourrait changer pour prendre en charge des dimensions plus élevées.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 8.1.3, 9.5.4

Disponibilité 2.0.0 : La prise en charge du type geography a été introduite

Exemples

Geometry.

Retourne le périmètre en pieds pour les polygones et les multi-polygones. Notez que ce périmètre est en pieds car EPSG:2249 est le plan de l'État du Massachusetts en pieds

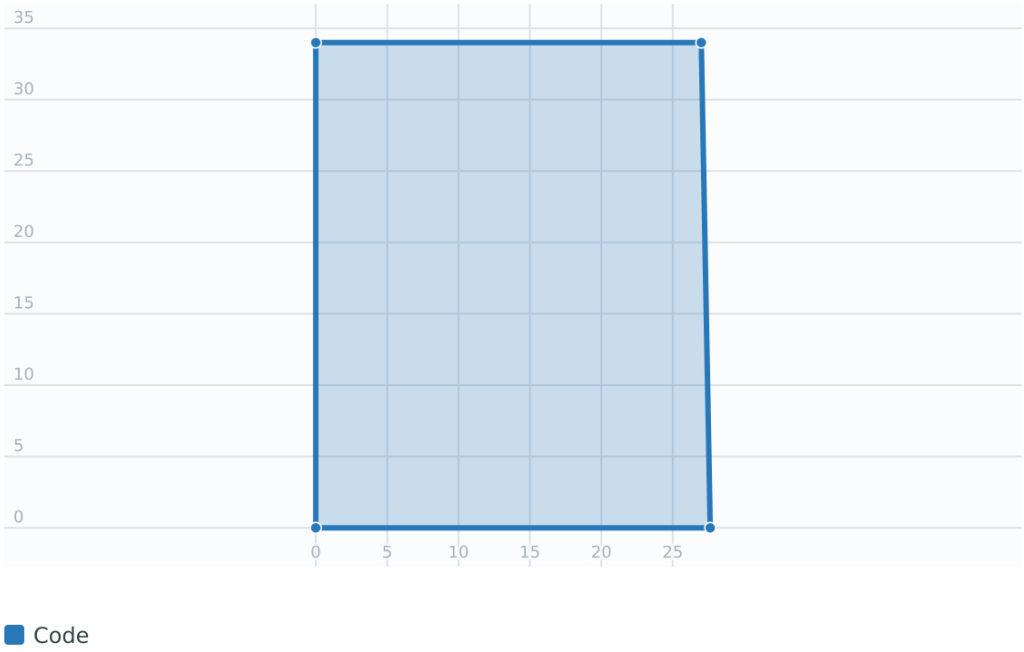
Code

```
SELECT ST_Perimeter(ST_GeomFromText(
'POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))',
2249));
```

Output

```
122.63074400009504
```

Figure



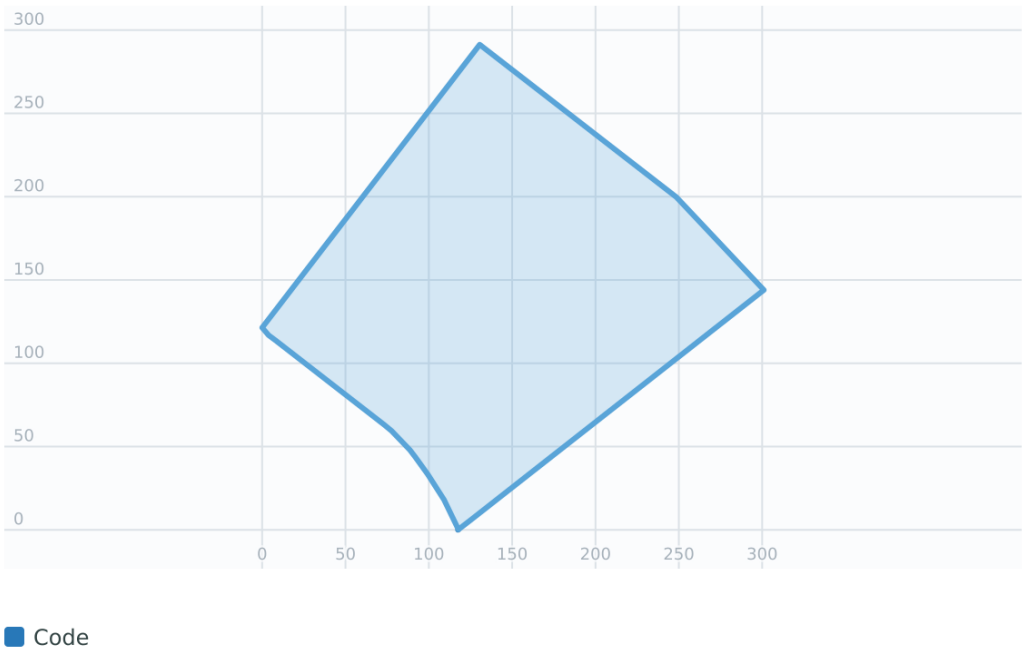
Code

```
SELECT ST_Perimeter(ST_GeomFromText(
'MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))',
2249));
```

Output

```
845.227713366825
```

Figure



Renvoie le périmètre en mètres et en pieds pour les polygones et les multi-polygones. Notez qu'il s'agit de geography (WGS 84 long lat)

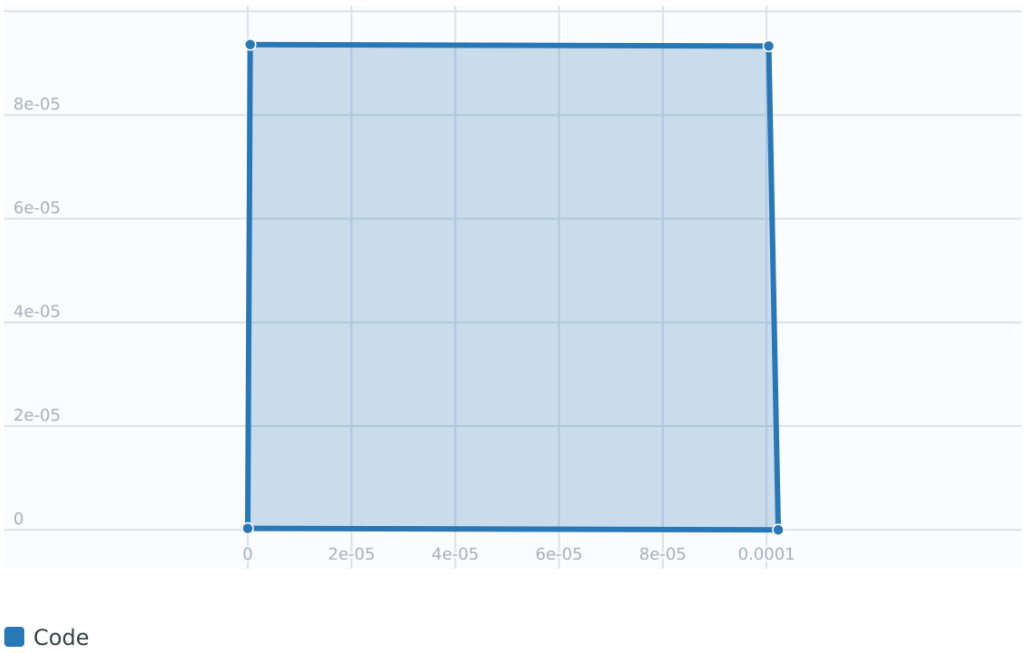
Code

```
SELECT round(ST_Perimeter(geog)::numeric, 8) As per_meters,
       round((ST_Perimeter(geog)/0.3048)::numeric, 8) As per_ft
FROM ST_GeogFromText('POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 ↩
    42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251 ↩
    42.3902896512902))') As geog;
```

Output

```
per_meters  |    per_ft
------------+--------------
 37.37904625 | 122.63466619
```

Figure



This example uses a MultiPolygon.

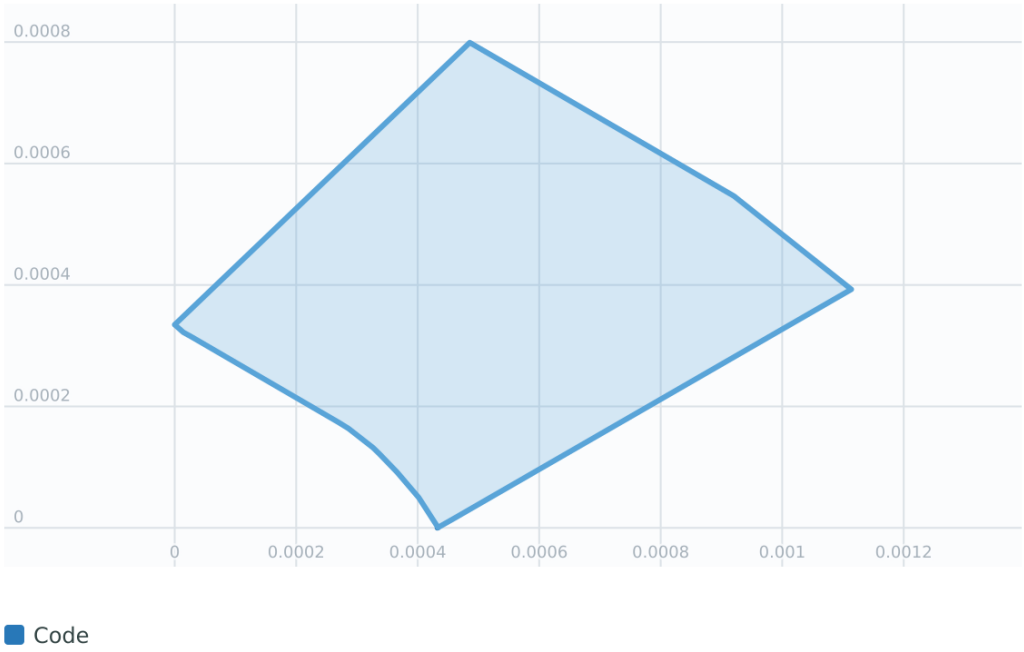
Code

```
SELECT round(ST_Perimeter(geog)::numeric, 8) As per_meters,
       round(ST_Perimeter(geog, false)::numeric, 8) As per_sphere_meters,
       round((ST_Perimeter(geog)/0.3048)::numeric, 8) As per_ft
FROM ST_GeogFromText('MULTIPOLYGON(((-71.1044543107478 42.340674480411,-71.1044542869917 ↩
    42.3406744369506,
-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411)),
((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124 ↩
    42.3407653385914,
-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427 ↩
    42.340837442371,
-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677 ↩
    42.3409959528211,
-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,
-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,
-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))') As geog;
```

Output

```
per_meters  | per_sphere_meters |    per_ft
-------------+-------------------+-------------
 257.63428368 |      257.41231145 | 845.25683623
```

Figure



Voir aussi

ST_GeogFromText, ST_GeomFromText, ST_Length

7.12.23 ST_Perimeter2D

ST_Perimeter2D — Renvoie le périmètre 2D d'une géométrie polygonale. Alias de `ST_Perimeter`.

Synopsis

float **ST_Perimeter2D**(geometry geomA);

Description

Renvoie le périmètre bidimensionnel d'une géométrie polygonale.



Note
Il s'agit actuellement d'un alias de ST_Perimeter. Dans les versions futures, ST_Perimeter pourrait renvoyer le périmètre de la dimension la plus élevée pour une géométrie. Cette possibilité est encore à l'étude

Voir aussi

ST_Perimeter

7.12.24 ST_3DPerimeter

ST_3DPerimeter — Renvoie le périmètre 3D d'une géométrie polygonale.

Synopsis

float **ST_3DPerimeter**(geometry geomA);

Description

Renvoie le périmètre tridimensionnel de la géométrie, s'il s'agit d'un polygone ou d'un multi-polygone. Si la géométrie est bidimensionnelle, le périmètre bidimensionnel est renvoyé.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode implémente la spécification SQL/MM. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Modifié : 2.0.0 Dans les versions antérieures, il s'appelait ST_Perimeter3D

Exemples

Périmètre d'un polygone légèrement surélevé en l'air dans l'état du Massachusetts, en pieds de plan

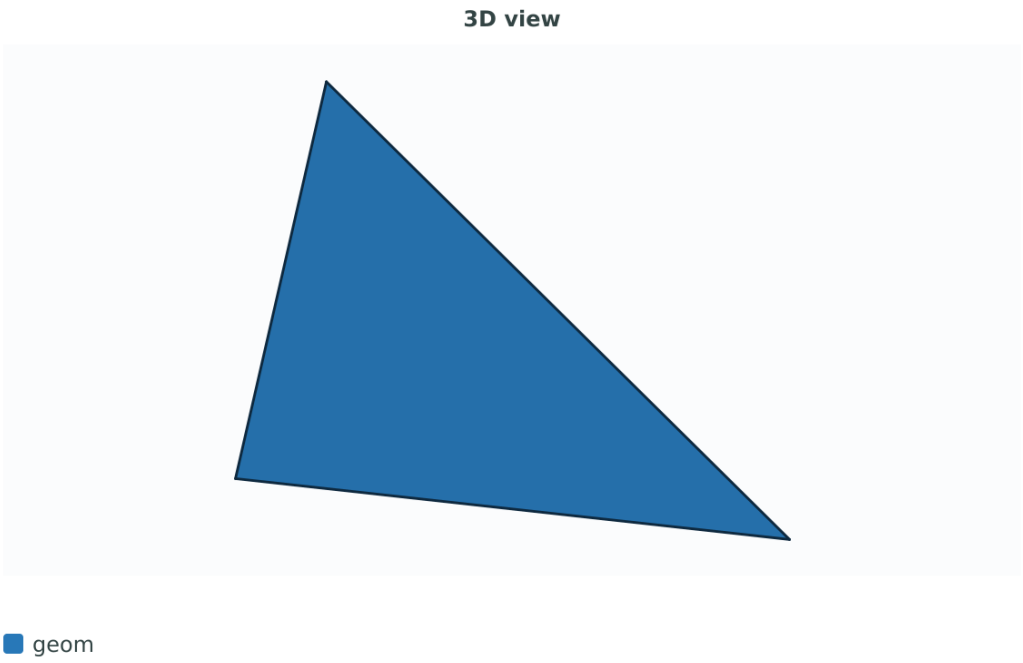
Code

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2D(geom), ST_Perimeter(geom) FROM
            (SELECT 'SRID=2249;POLYGON((743238 2967416 2,743238 2967450 1,
743265.625 2967416 1,743238 2967416 2))'::geometry As geom) As foo;
```

Output

```
ST_3DPerimeter  |  st_perimeter2d  |   st_perimeter
------------------+------------------+------------------
105.46579359767418 | 105.43299727218763 | 105.43299727218763
```

Figure



Voir aussi

ST_GeomFromEWKT, ST_Perimeter, ST_Perimeter2D

7.12.25 ST_ShortestLine

ST_ShortestLine — Renvoie la ligne 2D la plus courte entre deux géométries

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

Description

Renvoie la ligne bidimensionnelle la plus courte entre deux géométries. La ligne renvoyée commence dans `geom1` et se termine dans `geom2`. Si `geom1` et `geom2` se croisent, le résultat est une ligne dont le début et la fin se situent à un point d'intersection. La longueur de la ligne est la même que celle que ST_Distance renvoie pour g1 et g2.

Amélioré : 3.4.0 - Prise en charge de la geography.

Disponibilité : 1.5.0

Exemples

Shortest line between a Point and a LineString.

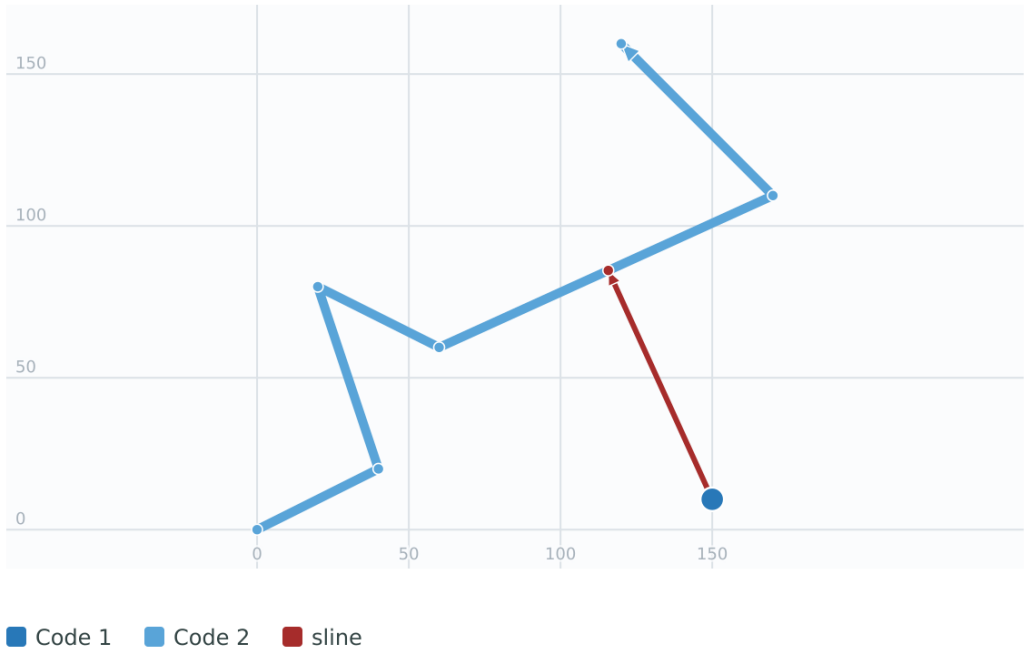
Code

```
SELECT ST_ShortestLine(
        'POINT (160 40)',
        'LINESTRING (10 30,50 50,30 110,70 90,180 140,130 190)') As sline;
```

Output

```
LINESTRING(160 40,125.75342465753425 115.34246575342466)
```

Figure



Shortest line between polygons.

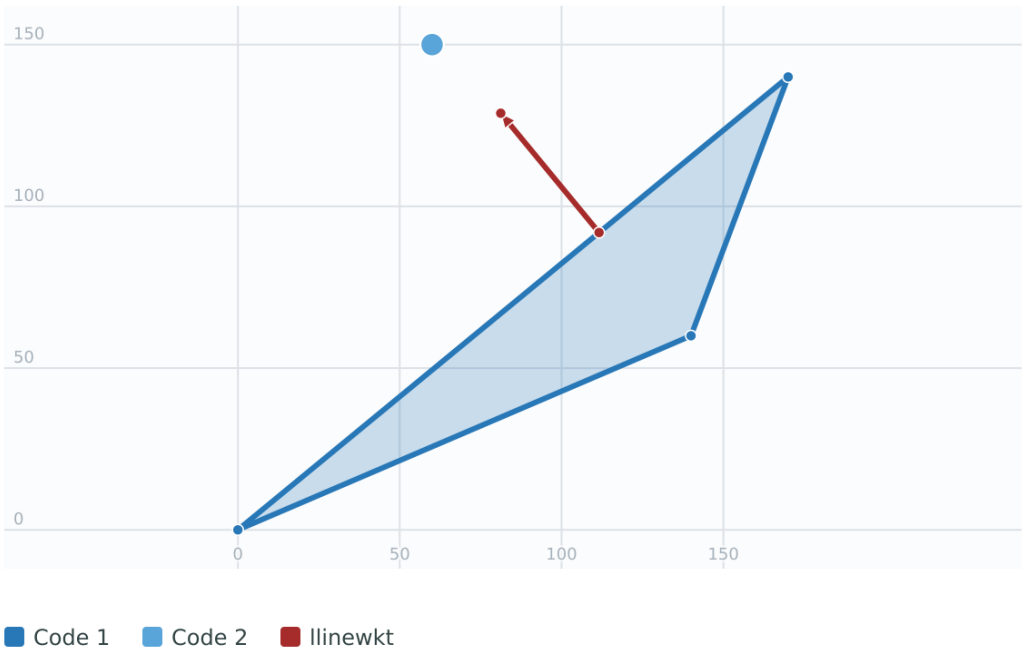
Code

```
SELECT ST_ShortestLine(
         'POLYGON ((190 150,20 10,160 70,190 150))',
         ST_Buffer('POINT(80 160)', 30)
                 ) AS llinewkt;
```

Output

```
LINESTRING(131.59149149528952 101.89887534906195,101.21320343559643 138.78679656440357)
```

Figure



Voir aussi

ST_ClosestPoint, ST_Distance, ST_LongestLine, ST_MaxDistance

7.12.26 ST_3DShortestLine

ST_3DShortestLine — Renvoie la ligne 3D la plus courte entre deux géométries

Synopsis

geometry **ST_3DShortestLine**(geometry g1, geometry g2);

Description

Renvoie la ligne tridimensionnelle la plus courte entre deux géométries. La fonction ne renvoie que la première ligne la plus courte, si la fonction en trouve plusieurs. Si g1 et g2 se croisent en un seul point, la fonction renvoie une ligne dont le début et la fin se trouvent à ce point d'intersection. Si g1 et g2 se croisent en plus d'un point, la fonction renvoie une ligne dont le début et la fin se situent au même point, mais qui peut être n'importe lequel des points d'intersection. La ligne renvoyée commencera toujours en g1 et se terminera en g2. La longueur 3D de la ligne renvoyée par cette fonction sera toujours la même que ST_3DDistance renvoie pour g1 et g2.

Disponibilité : 2.0.0

Modifié : 2.2.0 - si 2 géométries 2D sont saisies, un point 2D est renvoyé (au lieu de l'ancien comportement supposant 0 pour Z manquant). Dans le cas de 2D et 3D, Z n'est plus supposé être 0 pour Z manquant.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Compare the 3D shortest line to the 2D shortest line for a LineString and a Point.

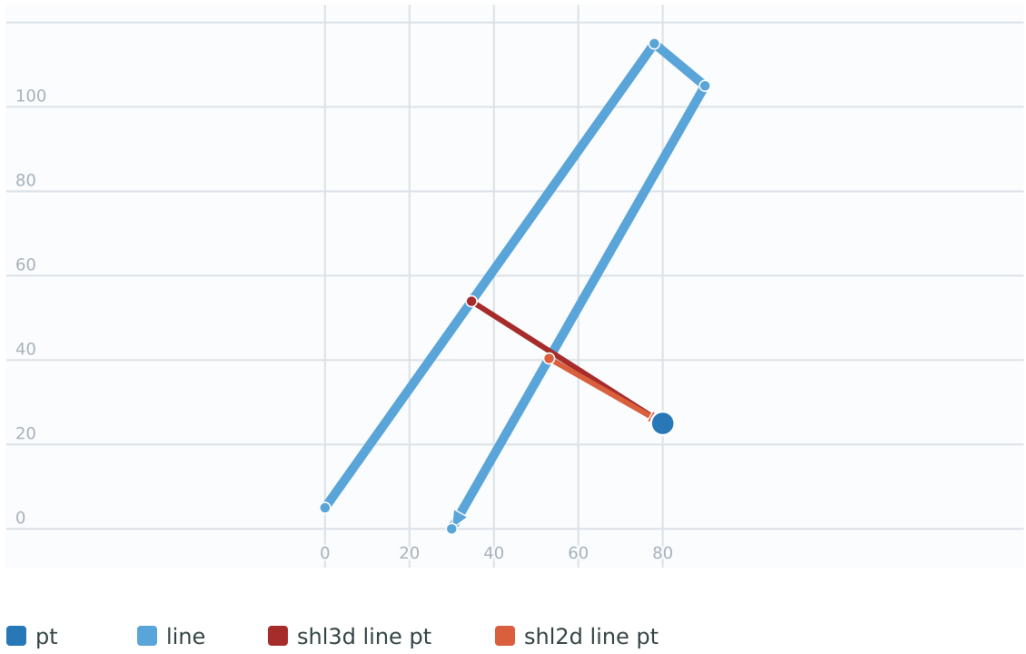
Code

```
WITH input AS (
  SELECT 'POINT(100 100 30)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 1000)'::geometry AS line
)
SELECT ST_3DShortestLine(line, pt) AS shl3d_line_pt,
       ST_ShortestLine(line, pt) AS shl2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------
shl3d_line_pt  | LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)
shl2d_line_pt  | LINESTRING(73.07692307692307 115.38461538461539,100 100)
```

Figure



Compare the 3D shortest line to the 2D shortest line for a LineString and a MultiPoint.

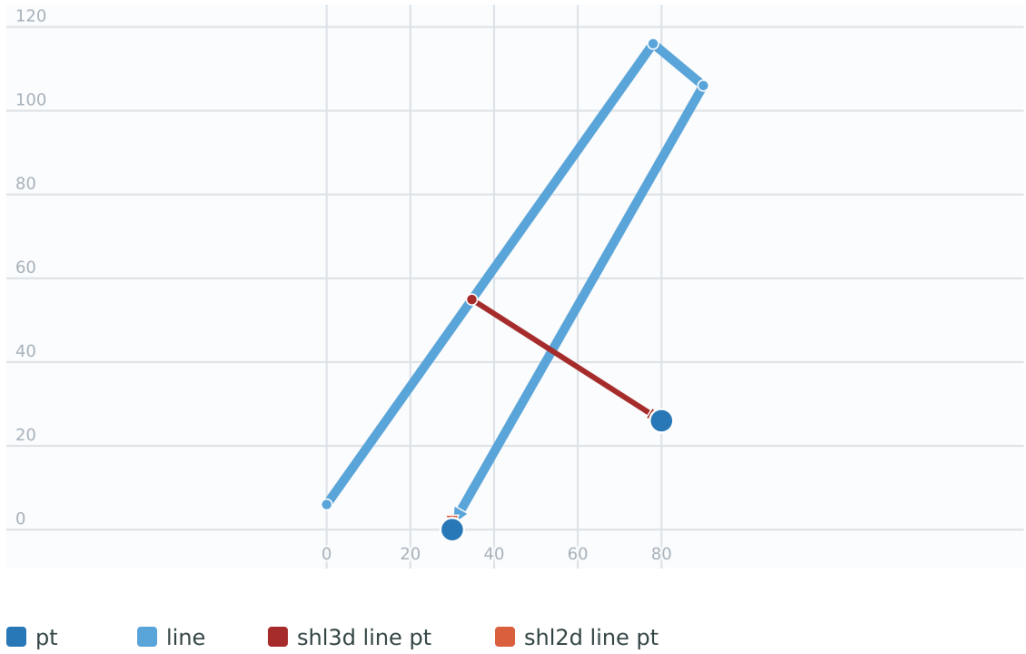
Code

```
WITH input AS (
  SELECT 'MULTIPOINT(100 100 30,50 74 1000)'::geometry AS pt,
         'LINESTRING (20 80 20,98 190 1,110 180 3,50 75 900)'::geometry AS line
)
SELECT ST_3DShortestLine(line, pt) AS shl3d_line_pt,
       ST_ShortestLine(line, pt) AS shl2d_line_pt
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------
shl3d_line_pt  | LINESTRING(54.69937988676193 128.93502291722837 11.547586950660556,100 100 30)
shl2d_line_pt  | LINESTRING(50 75,50 74)
```

Figure



Compare the 3D shortest line to the 2D shortest line for a Polygon and a MultiLineString.

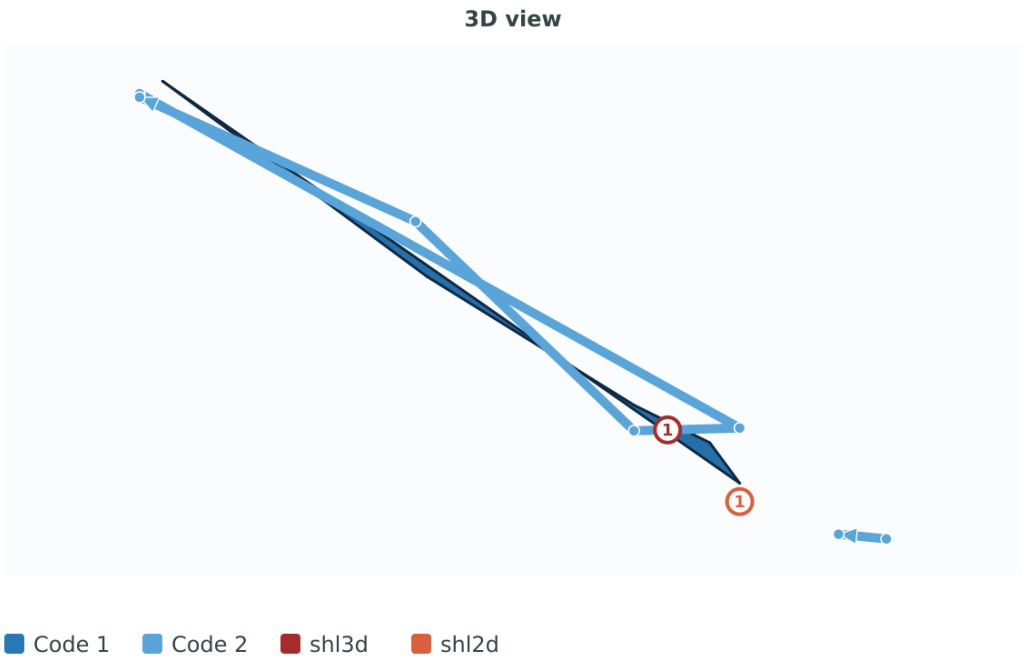
Code

```
WITH input AS (
  SELECT ST_GeomFromEWKT(
           'POLYGON((175 150 5,20 40 5,35 45 5,50 60 5,100 100 5,175 150 5))'
         ) AS poly,
         ST_GeomFromEWKT(
           'MULTILINESTRING((175 155 2,20 40 20,50 60 -2,125 100 1,175 155 1),
                            (1 10 2,5 20 1))'
         ) AS mline
)
SELECT ST_3DShortestLine(poly, mline) AS shl3d,
       ST_ShortestLine(poly, mline) AS shl2d
FROM input;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------
shl3d          | LINESTRING(40.45454545454545 53.63636363636364 5,40.45454545454545 53.63636363636364 5)
shl2d          | LINESTRING(20 40,20 40)
```

Figure



Voir aussi

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_ShortestLine, ST_3DMaxDistance

7.13 Fonctions de superposition

7.13.1 ST_ClipByBox2D

ST_ClipByBox2D — Calcule la partie d'une géométrie située à l'intérieur d'un rectangle.

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

Description

Coupe une géométrie par une boîte 2D d'une manière rapide et tolérante, mais qui peut être invalide. Les géométries d'entrée topologiquement invalides n'entraînent pas la levée d'exceptions. La géométrie de sortie n'est pas garantie valide (en particulier, des auto-intersections pour un polygone peuvent être introduites).

Effectué par le module GEOS.

Disponibilité : 2.2.0

Exemples

This example relies on the implicit cast from geometry to box2d for the second parameter.

Code

```
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0, 0, 10, 10)) FROM mytab;
```

Voir aussi

ST_Intersection, ST_MakeBox2D, ST_MakeEnvelope

7.13.2 ST_Difference

ST_Difference — Calcule une géométrie représentant la partie de la géométrie A qui n'intersecte pas la géométrie B.

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Description

Renvoie une géométrie représentant la partie de la géométrie A qui n'intersecte pas la géométrie B. Ceci est équivalent à `A - ST_Intersection(A,B)`. Si A est entièrement contenue dans B, une géométrie atomique vide du type approprié est renvoyée.



Note
Il s'agit de la seule fonction de superposition pour laquelle l'ordre d'entrée est important. ST_Difference(A, B) renvoie toujours une partie de A.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Effectué par le module GEOS

Amélioration : 3.1.0 accepte un paramètre gridSize.

Nécessite GEOS >= 3.9.0 pour utiliser le paramètre gridSize.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.20

Cette fonction prend en charge la 3D et ne supprime pas l'indice z. Cependant, le résultat est calculé en utilisant uniquement XY. Les valeurs Z résultantes sont copiées, moyennées ou interpolées.

Exemples

La différence entre les lignes 2D.

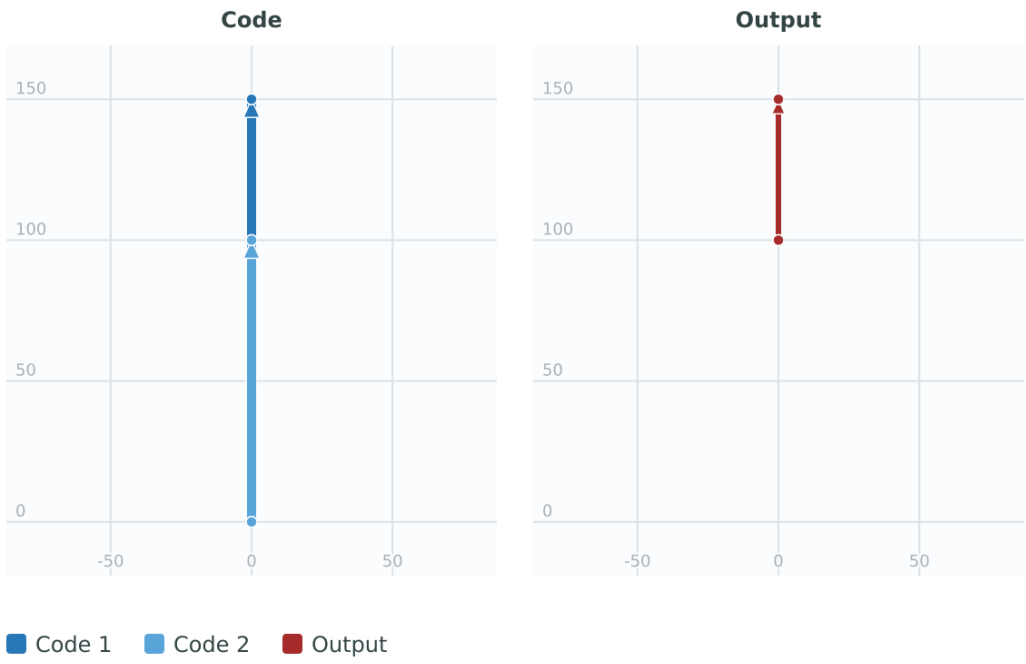
Code

```
SELECT ST_Difference('LINESTRING(50 100,50 200)'::geometry,
        'LINESTRING(50 50,50 150)'::geometry
    );
```

Output

```
LINESTRING(50 150,50 200)
```

Figure



La différence entre les points 3D.

Code

```
SELECT ST_Difference('MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' ::  ↩
    geometry,
 'POINT(-118.614 38.281 5)' :: geometry
);
```

Output

```
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

Figure



Voir aussi

ST_SymDifference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.3 ST_Intersection

ST_Intersection — Calcule une géométrie représentant la partie partagée des géométries A et B.

Synopsis

geometry **ST_Intersection**(geometry geomA , geometry geomB , float8 gridSize = -1);
geography **ST_Intersection**(geography geogA , geography geogB);

Description

Renvoie une géométrie représentant l'intersection des points de deux géométries. En d'autres termes, la partie de la géométrie A et de la géométrie B qui est partagée entre les deux géométries.

Si les géométries n'ont aucun point commun (c'est-à-dire qu'elles sont disjointes), une géométrie atomique vide du type approprié est renvoyée.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

ST_Intersection en conjonction avec ST_Intersects est utile pour découper les géométries, comme dans les requêtes sur les boîtes de délimitation, les tampons ou les régions, lorsque vous n'avez besoin que de la partie d'une géométrie qui se trouve à l'intérieur d'un pays ou d'une région d'intérêt.



Note

Spatial predicates such as ST_Intersects and overlay functions such as `ST_Intersection` are computed by different algorithms. For very close or nearly coincident coordinates, floating-point precision and geometric robustness may make a predicate return true while the computed intersection is empty, or place an intersection point slightly away from the input line. For repeatable overlay results, snap inputs to a suitable grid first, for example with ST_ReducePrecision, or use the `gridSize` parameter when all input vertices already lie on the grid.



Note



Pour le type geography, il s'agit d'un fin wrapper autour de l'implémentation de la géométrie. Il détermine d'abord le meilleur SRID qui correspond à la boîte de délimitation des 2 objets géographiques (si les objets géographiques sont dans une demi-zone UTM mais pas le même UTM choisira l'un d'eux) (en favorisant UTM ou Lambert Azimuthal Equal Area (LAEA) pôle nord/sud, et en se rabattant sur mercator dans le pire des cas) et ensuite l'intersection dans ce ref spatial planaire le mieux adapté et retransforme à nouveau à la géographie WGS84.



AVERTISSEMENT

Cette fonction abandonne les valeurs de coordonnées M si elles sont présentes.



AVERTISSEMENT

If working with 3D geometries, you may want to use SFCGAL based CG_3DIntersection which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

Effectué par le module GEOS

Amélioration : 3.1.0 accepte un paramètre gridSize

Nécessite GEOS >= 3.9.0 pour utiliser le paramètre gridSize

Modifié : 3.0.0 ne dépend pas de SFCGAL.

Disponibilité : La version 1.5 a introduit la prise en charge du type de données geography.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3



Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.18



Cette fonction prend en charge la 3D et ne supprime pas l'indice z. Cependant, le résultat est calculé en utilisant uniquement XY. Les valeurs Z résultantes sont copiées, moyennées ou interpolées.

Exemples

Code

```
SELECT ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
POINT EMPTY
```

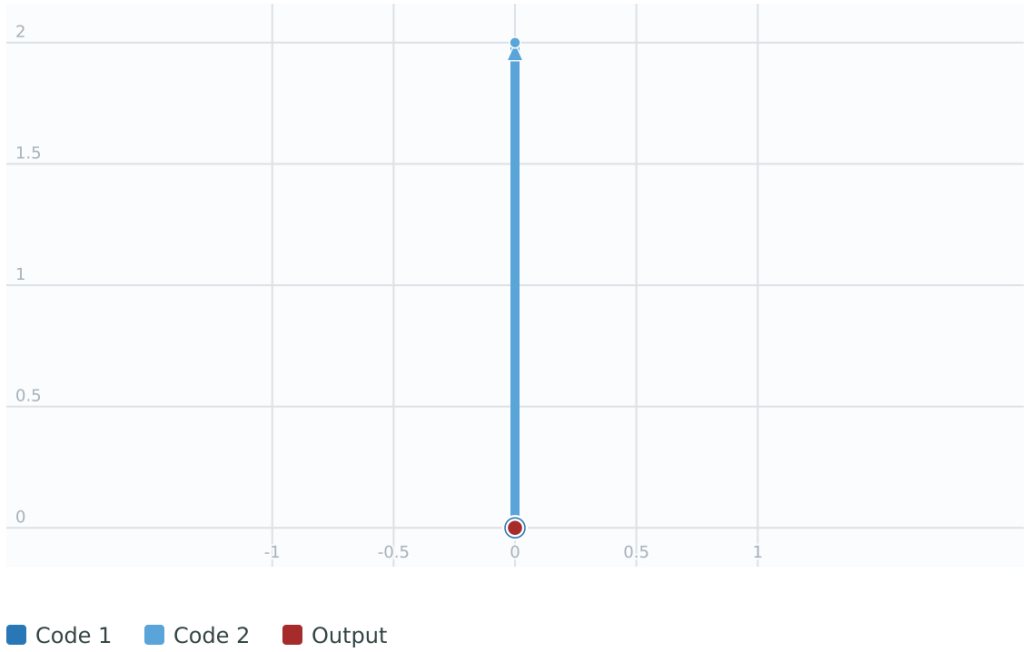
Code

```
SELECT ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Output

```
POINT(0 0)
```

Figure



Découper toutes les lignes (pistes) par pays. Nous supposons ici que les géométries des pays sont des POLYGONES ou des MULTIPOLYGONES. NOTE : nous ne gardons que les intersections qui résultent en un LINESTRING ou MULTILINESTRING car nous ne nous soucions pas des tracés qui partagent juste un point. Le dump est nécessaire pour étendre une collection géométrique en parties individuelles MULT*. La méthode ci-dessous est assez générique et fonctionnera pour les polys, etc. en changeant simplement la clause where.

Code

```
select clipped.gid, clipped.f_name, clipped_geom
from (
         select trails.gid, trails.f_name,
             (ST_Dump(ST_Intersection(country.geom, trails.geom))).geom clipped_geom
         from country
              inner join trails on ST_Intersects(country.geom, trails.geom)
     ) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

Pour les polys, par exemple les repères polygonaux, vous pouvez également utiliser la méthode parfois plus rapide qui consiste à mettre en mémoire tampon tout ce qui a une valeur de 0,0, à l'exception d'un polygone, afin d'obtenir une collection géométrique vide. (Ainsi, une collection géométrique contenant des polygones, des lignes et des points mis en mémoire tampon par 0.0 ne laisserait que les polygones et dissoudrait l'enveloppe de la collection.)

Code

```
select poly.gid,
    ST_Multi(ST_Buffer(ST_Intersection(country.geom, poly.geom),
             0.0
        )
```

```
    ) clipped_geom
from country
     inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

Note this is not a true 3D intersection. It uses 2.5D geometries (geometries with Z ordinates, but where calculations are performed in 2D). The same input is compared with CG_3DIntersection, which performs a true 3D intersection.

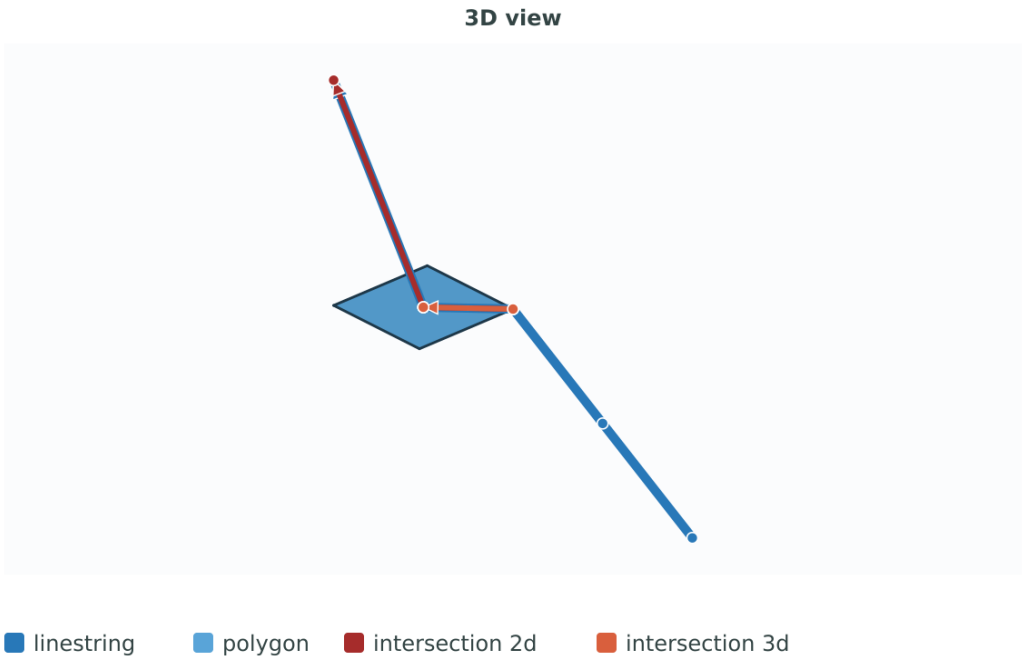
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)'::geometry AS ↩
    input_linestring,
         'POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8))'::geometry AS input_polygon
)
SELECT input_linestring AS input_linestring,
       input_polygon AS input_polygon,
       ST_Intersection(input_linestring, input_polygon) AS intersection_2d,
       CG_3DIntersection(input_linestring, input_polygon) AS intersection_3d
FROM data;
```

Output

```
LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10) | POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8)) | LINESTRING Z (1 1 ↩
     8,0.5 0.5 8,0 0 10) | LINESTRING Z (1 1 8,0.5 0.5 8)
```

Figure



Voir aussi

CG_3DIntersection, ST_Difference, ST_Union, ST_ClipByBox2D, ST_Dimension, ST_Dump, ST_Force2D, ST_SymDifference, ST_Intersects, ST_Multi, ST_ReducePrecision

7.13.4 ST_MemUnion

ST_MemUnion — Fonction d'agrégation qui fusionne les géométries d'une manière efficace sur le plan de la mémoire mais plus lente

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

Description

Une fonction d'agrégation qui fusionne les géométries en entrée, en les fusionnant pour produire une géométrie de résultat sans chevauchement. Le résultat peut être une géométrie unique, une MultiGéométrie ou une Collection de Géométries.



Note
Produit le même résultat que ST_Union, mais utilise moins de mémoire et plus de temps processeur. Cette fonction d'agrégation fonctionne par l'union des géométries de manière incrémentale, contrairement à l'agrégation ST_Union qui accumule d'abord un tableau et en unit ensuite le contenu à l'aide d'un algorithme rapide.



Cette fonction prend en charge la 3D et ne supprime pas l'indice z. Cependant, le résultat est calculé en utilisant uniquement XY. Les valeurs Z résultantes sont copiées, moyennées ou interpolées.

Exemples

Code

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Voir aussi

ST_Union

7.13.5 ST_Node

ST_Node — Nœuds d'une collection de lignes.

Synopsis

geometry **ST_Node**(geometry geom);

Description

Renvoie une (Multi)LineString représentant la version entièrement "nodée" d'une collection de lignes. Le "noding" préserve tous les nœuds d'entrée et introduit le plus petit nombre possible de nouveaux nœuds. La ligne résultante est dissoute (les lignes en double sont supprimées).

Il s'agit d'un bon moyen de créer des lignes entièrement "nodées" pouvant être utilisées dans ST_Polygonize.

ST_UnaryUnion peut également être utilisé pour créer des nœuds et dissoudre des lignes. Il est possible de spécifier une taille de grille, ce qui permet d'obtenir des résultats plus simples et plus robustes. Voir aussi ST_Union pour une variante agrégée.



Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Effectué par le module GEOS.

Disponibilité : 2.0.0

Modifié : 2.4.0 cette fonction utilise GEOSNode en interne au lieu de GEOSUnaryUnion. Cela peut entraîner un ordre et une direction différents des lignes résultantes par rapport à PostGIS < 2.4.

Exemples

Noding a 3D LineString which self-intersects.

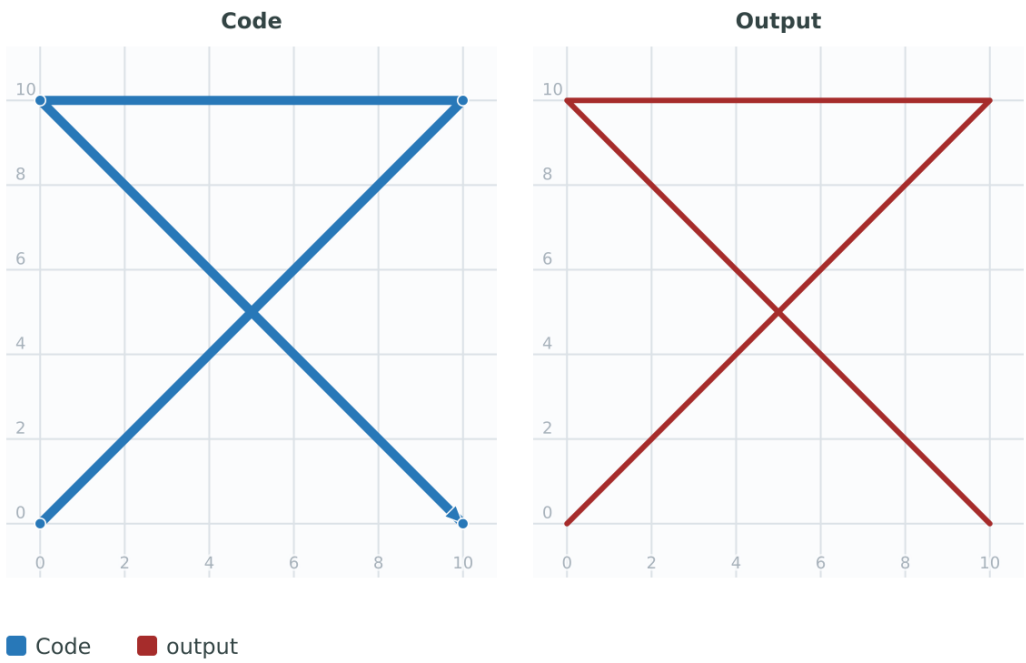
Code

```
SELECT ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)'::geometry) As  output;
```

Output

```
MULTILINESTRING Z ((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))
```

Figure



"Noding" de deux lignes qui partagent une structure linéaire commune. Notez que la ligne de résultat est dissoute.

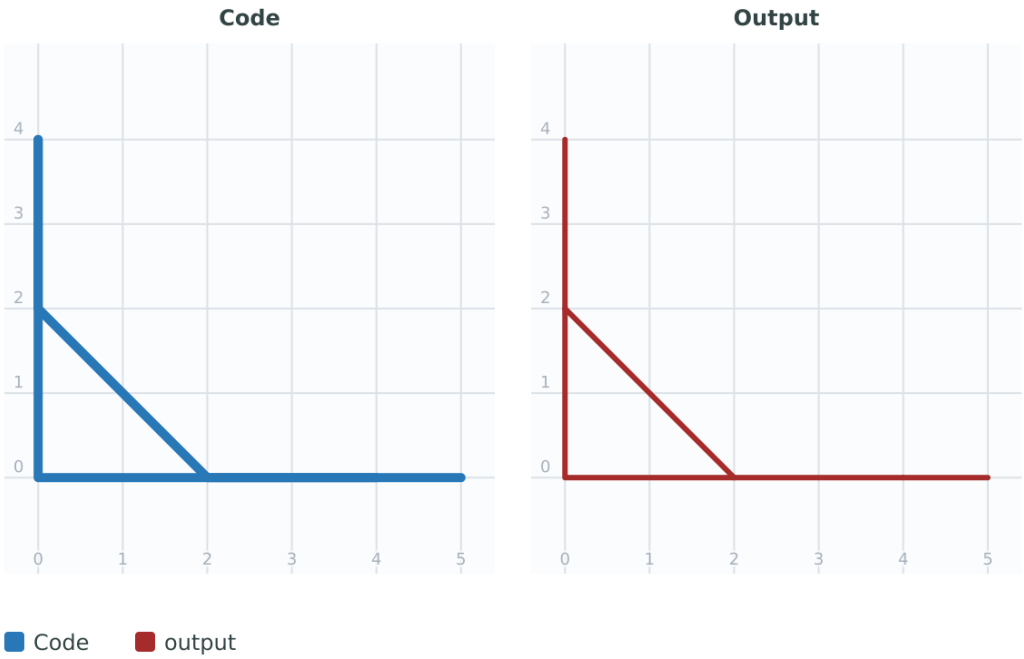
Code

```
SELECT ST_Node('MULTILINESTRING ((2 5,2 1,7 1),(6 1,4 1,2 3,2 5))'::geometry) As  output;
```

Output

```
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

Figure



Voir aussi

ST_UnaryUnion, ST_Union

7.13.6 ST_Split

ST_Split — Renvoie une collection de géométries créées en divisant une géométrie par une autre géométrie.

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

Description

Cette fonction permet de diviser une ligne par un (Multi)Point, une (Multi)Ligne ou un (Multi)Polygone, ou un (Multi)Polygone par une ligne. Lorsqu'un (Multi)Polygone est utilisé, ses composantes linéaires (la frontière) sont utilisées pour diviser l'entrée. La géométrie résultante est toujours une collection.

Cette fonction est en quelque sorte l'inverse de ST_Union. L'application de ST_Union à la collection retournée devrait théoriquement donner la géométrie originale (bien qu'en raison de l'arrondi numérique, cela puisse ne pas être exactement le cas).



Note
Si l'entrée et la "lame" ne se croisent pas en raison de problèmes de précision numérique, l'entrée peut ne pas être divisée comme prévu. Pour éviter cette situation, il peut être nécessaire d'accrocher d'abord l'entrée à la "lame", en utilisant ST_Snap avec une petite tolérance.

Disponibilité : 2.0.0 nécessite GEOS

Amélioration : la version 2.2.0 prend en charge la division d'une ligne par une limite multiligne, multipoint ou (multi)polygone.

Amélioration : la prise en charge de la division d'un polygone par une ligne multiple a été introduite dans la version 2.5.0.

Exemples

Split a circular polygon by a diagonal line.

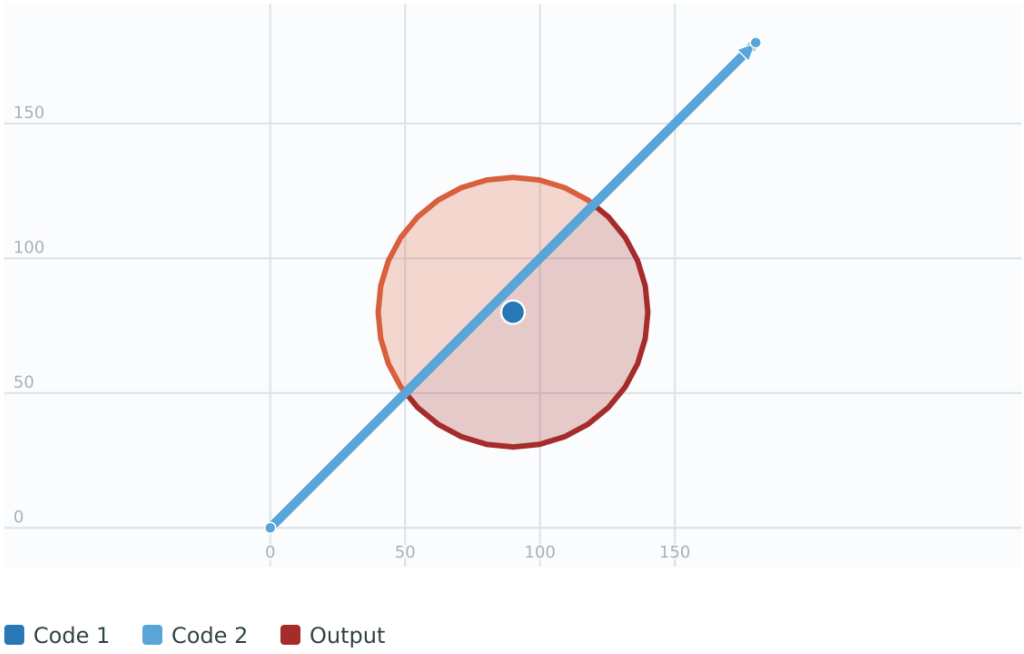
Code

```
SELECT
          ST_Split(
            ST_Buffer('POINT(100 90)'::geometry, 50),
            'LINESTRING(10 10,190 190)'::geometry
          );
```

Output

```
GEOMETRYCOLLECTION(POLYGON((150 90,149 80.2,146.2 70.9,141.6 62.2,135.4 54.6,127.8 48.4,119.1 43.8,109.8 41,100 40,90.2 ↩
     41,80.9 43.8,72.2 48.4,64.6 54.6,60.1 60.1,129.9 129.9,135.4 125.4,141.6 117.8,146.2 109.1,149 99.8,150 90)),  ↩
    POLYGON((60.1 60.1,58.4 62.2,53.8 70.9,51 80.2,50 90,51 99.8,53.8 109.1,58.4 117.8,64.6 125.4,72.2 131.6,80.9  ↩
    136.2,90.2 139,100 140,109.8 139,119.1 136.2,127.8 131.6,129.9 129.9,60.1 60.1)))
```

Figure



Divise une MultiLineString par un point, où le point se trouve exactement sur les deux éléments de la LineString.

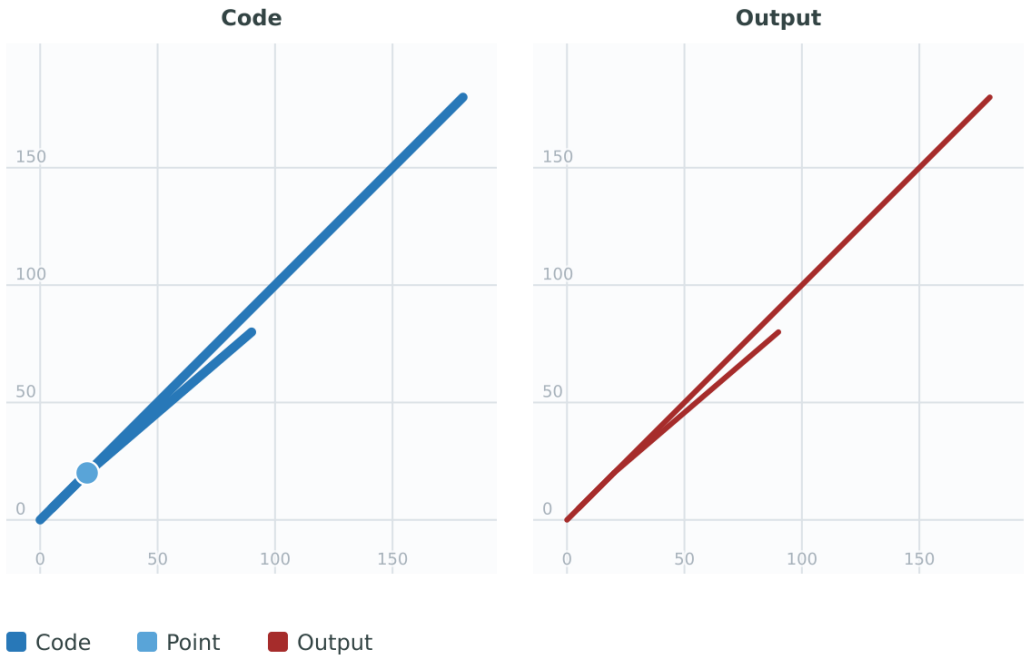
Code

```
SELECT ST_Split('MULTILINESTRING((10 10,190 190),(15 15,30 30,100 90))',
    ST_Point(30, 30));
```

Output

```
GEOMETRYCOLLECTION(
    LINESTRING(10 10,30 30),
    LINESTRING(30 30,190 190),
    LINESTRING(15 15,30 30),
    LINESTRING(30 30,100 90)
)
```

Figure



Diviser une ligne par un point, lorsque le point ne se trouve pas exactement sur la ligne. Montre l'utilisation de ST_Snap pour accrocher la ligne au point afin de permettre son découpage.

Code

```
WITH data AS (SELECT
  'LINESTRING(0 0,100 100)'::geometry AS line,
  'POINT(51 50)':: geometry AS point
)
SELECT ST_Split(ST_Snap(line, point, 1), point) AS snapped_split,
       ST_Split(line, point) AS not_snapped_not_split
       FROM data;
```

Figure



Voir aussi

ST_Snap, ST_Union

7.13.7 ST_Subdivide

ST_Subdivide — Calcule une subdivision rectiligne d'une géométrie.

Synopsis

setof geometry **ST_Subdivide**(geometry geom, integer max_vertices=256, float8 gridSize = -1);

Description

Renvoie un ensemble de géométries résultant de la division de `geom` en parties à l'aide de lignes rectilignes, chaque partie ne contenant pas plus de `max_vertices`.

`max_vertices` must be 5 or more, as 5 points are needed to represent a closed box.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Les opérations "point dans polygone" et autres opérations spatiales sont normalement plus rapides pour les ensembles de données subdivisés et indexés. Étant donné que les boîtes de délimitation des parties couvrent généralement une zone plus petite que la boîte b de la géométrie d'origine, les requêtes d'index produisent moins de cas "hit". Les cas "hit" sont plus rapides car les opérations spatiales exécutées par la revérification de l'index traitent moins de points.



Note
When casting a subdivided geometry to geography, the resulting geography may differ from the original. Subdivision adds vertices in planar (geometry) space. If vertices are inserted along the boundary, they will alter the geographical representation, where edges are interpreted as geodesic segments. To minimize distortion, first densify the geography using ST_Segmentize to add geodesic vertices, then cast to geometry before subdivision.



Note
Il s'agit d'une fonction de retour d'ensemble (SRF) qui renvoie un ensemble de lignes contenant des valeurs de géométrie uniques. Elle peut être utilisée dans une liste SELECT ou une clause FROM pour produire un ensemble de résultats avec un enregistrement pour chaque géométrie de résultat.

Effectué par le module GEOS.

Disponibilité : 2.2.0

Amélioration : 2.5.0 réutilise les points existants lors de la division d'un polygone, le nombre de vertex est réduit de 8 à 5.

Amélioration : 3.1.0 accepte un paramètre gridSize.

Nécessite GEOS >= 3.9.0 pour utiliser le paramètre gridSize

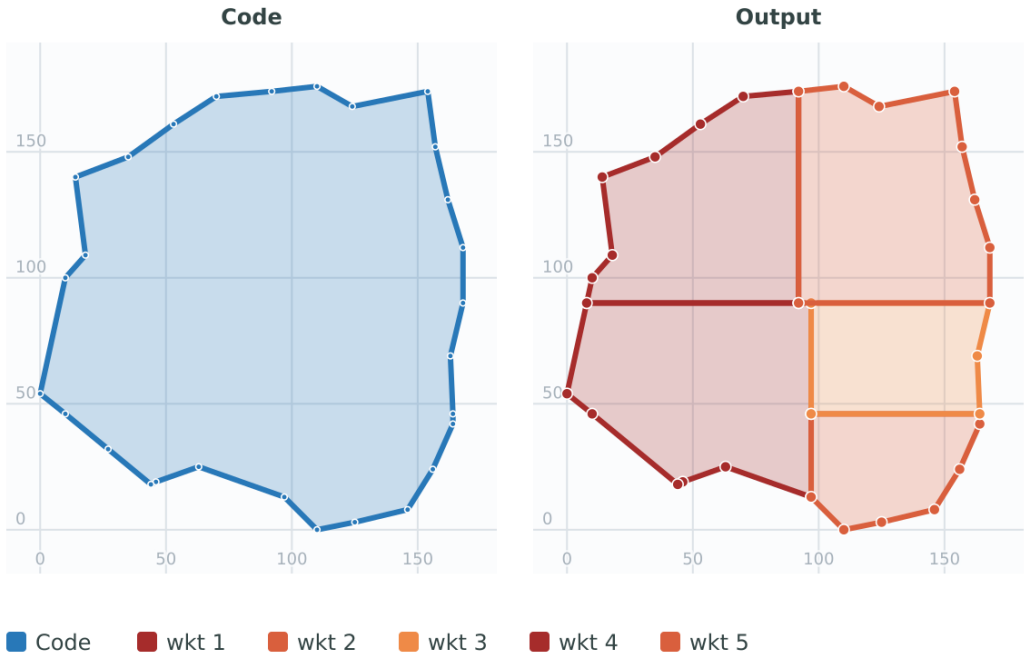
Exemples

Exemple: Subdivisez un polygone en parties ne comportant pas plus de 10 sommets et attribuez à chaque partie un identifiant unique.

Code

```
SELECT row_number() OVER() As rn,geom As wkt
    FROM (SELECT ST_Subdivide('POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 ↩
     110,40 119,36 150,
        57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
        190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry, 10))  AS f( ↩
    geom);
```

Figure



Example: Densify a long geography line using ST_Segmentize(geography, distance), and use ST_Subdivide to split the resulting line into sublines of 8 vertices. Densification minimizes the impact of changes to the geography representation of a geometry when subdividing.

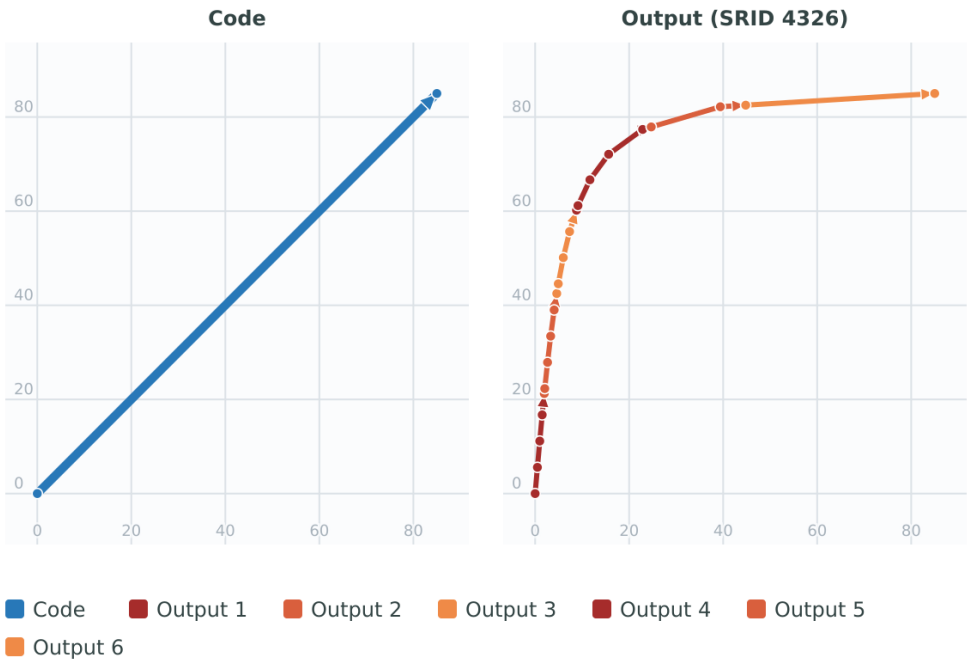
Code

```
SELECT ST_Subdivide(ST_Segmentize('LINESTRING(0 0,85 85)'::geography,
                          1200000)::geometry, 8);
```

Output

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897 11.1527721155093,1.50101059639722 ↩
    16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673 27.8740533545155,3.29994062412787 ↩
    33.443216802941,4.04836719489742 39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639 50.1195229244701,7.3290919767674 ↩
    55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779 61.1785363177625,11.6558166691368 ↩
    66.6648504160202,15.642041247655 72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 ↩
    77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744 82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

Figure



Exemple: Subdiviser les géométries complexes d'une table existante. Les enregistrements de la géométrie d'origine sont supprimés de la table source et de nouveaux enregistrements sont insérés pour chaque géométrie de résultat subdivisée.

Code

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom)
> 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
    SELECT fid, column1, column2, column3,
           ST_Subdivide(geom, 255) as geom
    FROM complex_areas_to_subdivide;
```

Exemple: Créer une nouvelle table contenant les géométries subdivisées, en conservant la clé de la géométrie d'origine afin que la nouvelle table puisse être jointe à la table source. Étant donné que ST_Subdivide est une fonction (tableau) qui renvoie un ensemble de lignes à valeur unique, cette syntaxe produit automatiquement un tableau avec une ligne pour chaque partie du résultat.

Code

```
CREATE TABLE subdivided_geoms AS
    SELECT pkey, ST_Subdivide(geom) AS geom
    FROM original_geoms;
```

Voir aussi

ST_ClipByBox2D, ST_Segmentize, ST_Split, ST_NPoints, ST_ReducePrecision

7.13.8 ST_SymDifference

ST_SymDifference — Calcule une géométrie représentant les parties des géométries A et B qui ne s'intersectent pas.

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Description

Renvoie une géométrie représentant les parties des géométries A et B qui ne s'intersectent pas. Ceci est équivalent à `ST_Union(A,B) - ST_Intersection(A,B)`. On parle de différence symétrique car `ST_SymDifférence(A,B) = ST_SymDifférence(B,`

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Effectué par le module GEOS

Amélioration : 3.1.0 accepte un paramètre gridSize.

Nécessite GEOS >= 3.9.0 pour utiliser le paramètre gridSize

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 5.1.21

Cette fonction prend en charge la 3D et ne supprime pas l'indice z. Cependant, le résultat est calculé en utilisant uniquement XY. Les valeurs Z résultantes sont copiées, moyennées ou interpolées.

Exemples

This example shows a 2D-safe symmetric difference of two LineStrings.

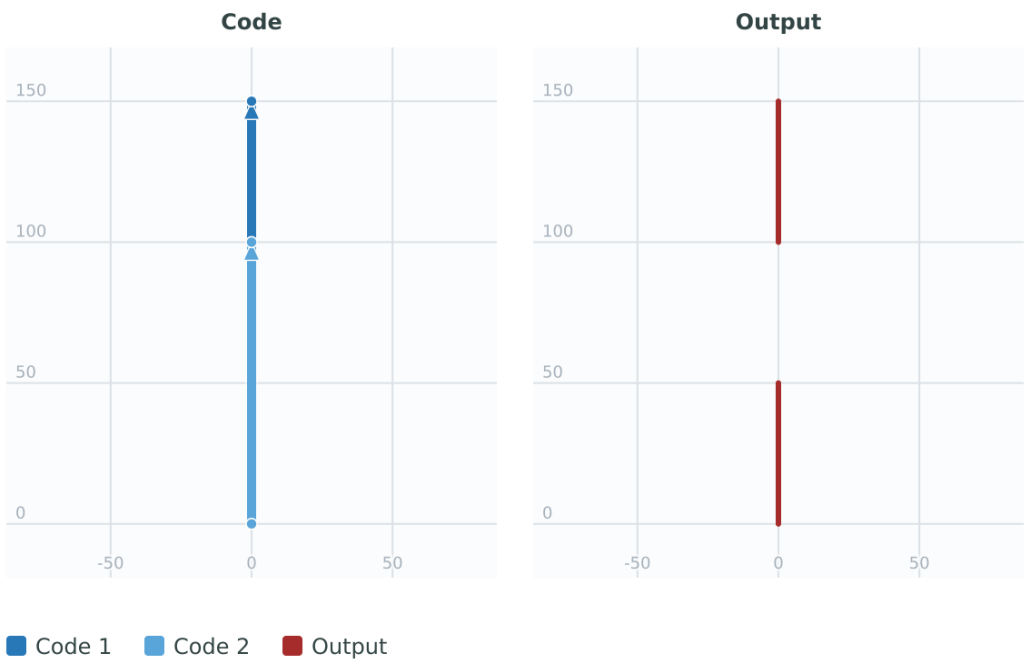
Code

```
SELECT ST_SymDifference('LINESTRING(50 100,50 200)',
        'LINESTRING(50 50,50 150)'
    );
```

Output

```
MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

Figure



In 3D, this example does not produce the expected result.

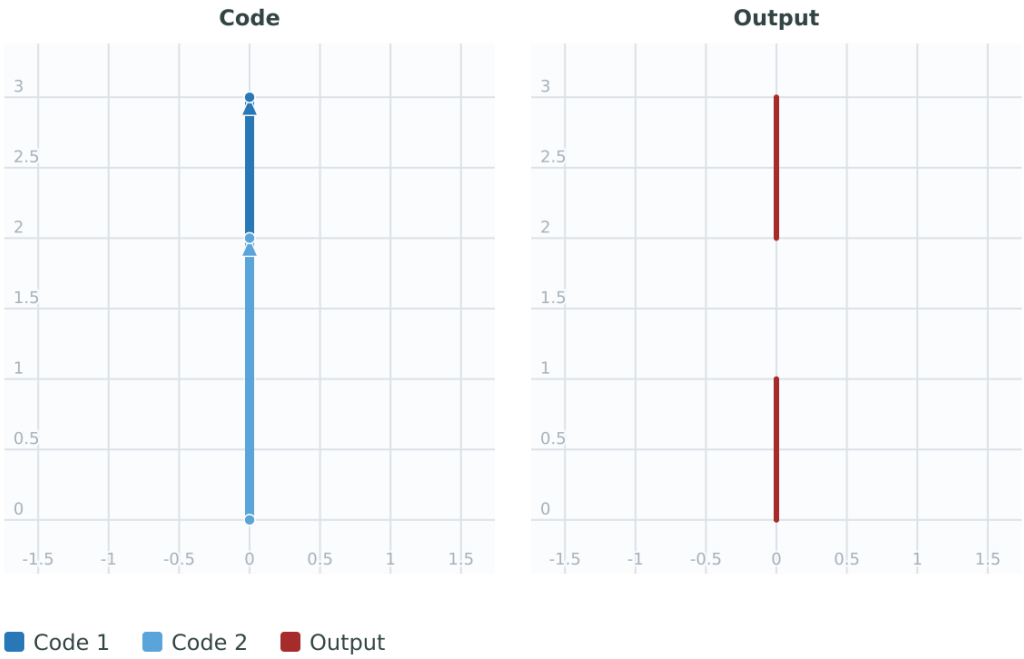
Code

```
SELECT ST_SymDifference('LINESTRING(1 2 1,1 4 2)',
    'LINESTRING(1 1 3,1 3 4)');
```

Output

```
MULTILINESTRING((1 3 4,1 4 2),(1 1 3,1 2 1))
```

Figure



Voir aussi

ST_Difference, ST_Intersection, ST_Union, ST_ReducePrecision

7.13.9 ST_UnaryUnion

ST_UnaryUnion — Calcule l'union des composantes d'une seule géométrie.

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

Description

Variante à entrée unique de ST_Union. L'entrée peut être une géométrie unique, une MultiGeometry ou une GeometryCollection. L'union est appliquée aux éléments individuels de l'entrée.

Cette fonction peut être utilisée pour corriger les MultiPolygones qui ne sont pas valides en raison du chevauchement de leurs composants. Toutefois, les composants d'entrée doivent tous être valides. Un composant d'entrée non valide, tel qu'un polygone en nœud papillon, peut provoquer une erreur. Pour cette raison, il peut être préférable d'utiliser ST_MakeValid.

Une autre utilisation de cette fonction est de nouer et de dissoudre une collection de lignes qui se croisent ou se chevauchent pour les rendre simple. (ST_Node fait également cela, mais ne fournit pas l'option `gridSize`.)

Il est possible de combiner ST_UnaryUnion avec ST_Collect pour affiner le nombre de géométries à unir en une seule fois. Cela permet de faire un compromis entre l'utilisation de la mémoire et le temps de calcul, en trouvant un équilibre entre ST_Union et ST_MemUnion.

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z. Cependant, le résultat est calculé en utilisant uniquement XY. Les valeurs Z résultantes sont copiées, moyennées ou interpolées.

Amélioration : 3.1.0 accepte un paramètre gridSize.

Nécessite GEOS >= 3.9.0 pour utiliser le paramètre gridSize

Disponibilité : 2.0.0

Voir aussi

ST_Union, ST_MemUnion, ST_MakeValid, ST_Collect, ST_Node, ST_ReducePrecision

7.13.10 ST_Union

ST_Union — Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.

Synopsis

geometry **ST_Union**(geometry g1, geometry g2);
geometry **ST_Union**(geometry g1, geometry g2, float8 gridSize);
geometry **ST_Union**(geometry[] g1_array);
geometry **ST_Union**(geometry set g1field);
geometry **ST_Union**(geometry set g1field, float8 gridSize);

Description

Unifie les géométries d'entrée, fusionne les géométries pour produire une géométrie de résultat sans chevauchement. Le résultat peut être une géométrie atomique, une MultiGeometry ou une Geometry Collection. Il existe plusieurs variantes :

Variante à deux entrées: renvoie une géométrie qui est l'union de deux géométries d'entrée. Si l'une des entrées est NULL, NULL est renvoyé.

Variante tableau: renvoie une géométrie qui est l'union d'un tableau de géométries.

Variante agrégée: renvoie une géométrie qui est l'union d'un ensemble de géométries. La fonction ST_Union() est une fonction "agrégée" dans la terminologie de PostgreSQL. Cela signifie qu'elle opère sur des lignes de données, de la même manière que les fonctions SUM() et AVG() et, comme la plupart des agrégats, elle ignore les géométries NULL.

Voir ST_UnaryUnion pour une variante non agrégée, à entrée unique.

Le tableau ST_Union et ses variantes utilisent l'algorithme rapide d'union en cascade décrit dans http://blog.cleverelephant.ca/-2009/01/must-faster-unions-in-postgis-14.html

If the optional `gridSize` parameter is given (GEOS-3.9.0 or higher required), all result vertices are guaranteed to fall on a snap-rounded grid of the specified size. Note that operations performed on a grid may contain small artifacts produced during grid alignment, see ST_ReducePrecision.



Note
ST_Collect peut parfois être utilisé à la place de ST_Union, si le résultat n'a pas besoin d'être non chevauchant. ST_Collect est généralement plus rapide que ST_Union car il n'effectue aucun traitement sur les géométries collectées.

Effectué par le module GEOS.

ST_Union crée une MultiLineString et ne fusionne pas les LineStrings en une seule LineString. Utilisez ST_LineMerge pour fusionner les LineStrings.

NOTE : cette fonction s'appelait auparavant GeomUnion(), qui a été renommée "Union" car UNION est un mot réservé de SQL.

Amélioration : 3.1.0 accepte un paramètre gridSize.

Nécessite GEOS >= 3.9.0 pour utiliser le paramètre gridSize

Modifié : 3.0.0 ne dépend pas de SFCGAL.

Disponibilité : 1.4.0 - ST_Union a été amélioré. ST_Union(geomarray) a été introduit ainsi qu'une collecte d'agrégats plus rapide dans PostgreSQL.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3



Note
La version agrégée n'est pas explicitement définie dans l'OGC SPEC.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 : 5.1.19 l'indice z (élévation) lorsque des polygones sont impliqués.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z. Cependant, le résultat est calculé en utilisant uniquement XY. Les valeurs Z résultantes sont copiées, moyennées ou interpolées.

Exemples

Exemple d'agrégat

Code

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Exemple non agrégé

Code

```
SELECT ST_Union('POINT(1 2)'::geometry, 'POINT(-2 3)'::geometry);
```

Output

```
MULTIPOINT((1 2),(-2 3))
```

Figure



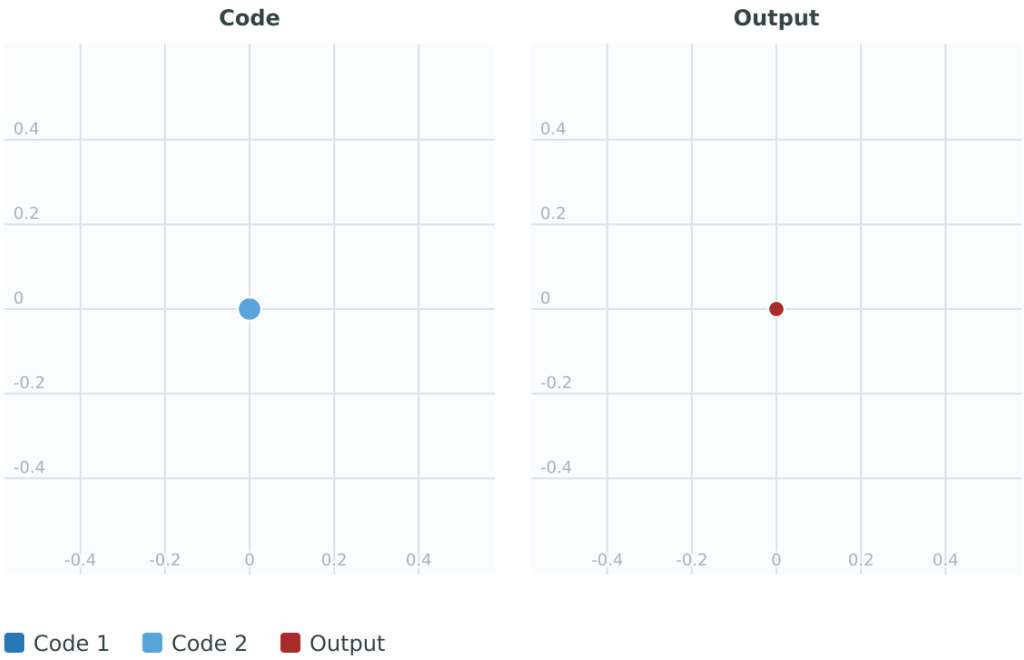
Code

```
SELECT ST_Union('POINT(1 2)'::geometry, 'POINT(1 2)'::geometry);
```

Output

```
POINT(1 2)
```

Figure



3D example - sort of supports 3D (and with mixed geometry types!)

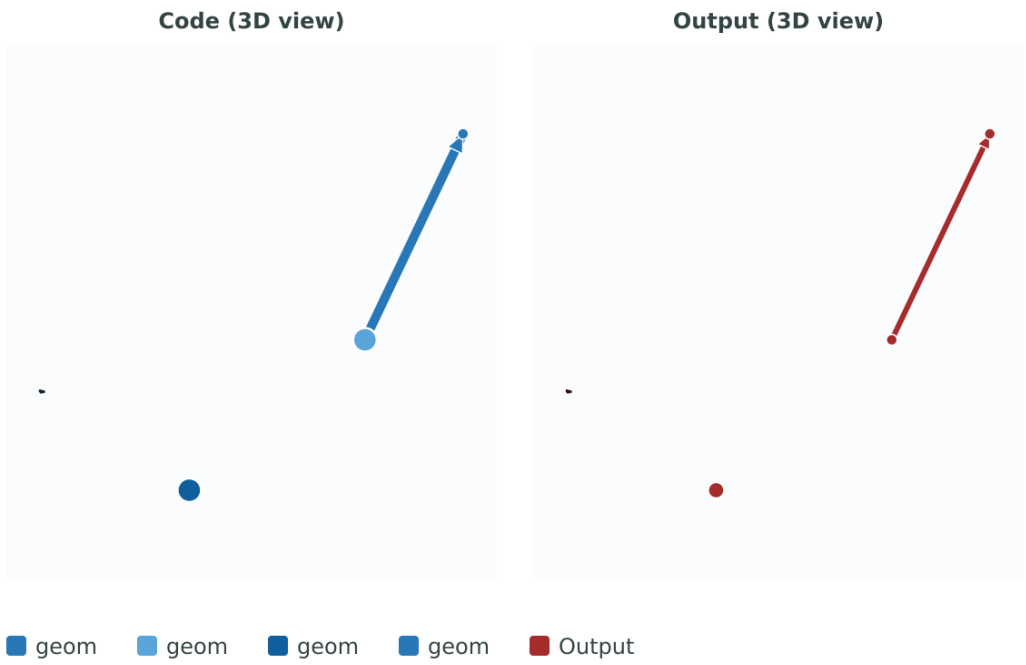
Code

```
SELECT ST_Union(geom)
FROM (
    SELECT 'POLYGON Z((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5))'::geometry geom
    UNION ALL
    SELECT 'POINT Z(5 5 5)'::geometry geom
    UNION ALL
    SELECT 'POINT Z(-2 3 1)'::geometry geom
    UNION ALL
    SELECT 'LINESTRING Z(5 5 5,10 10 10)'::geometry geom
) AS foo;
```

Output

```
GEOMETRYCOLLECTION(POINT(-2 3 1), LINESTRING(5 5 5,10 10 10), POLYGON((-7 4.2 5,-7.1 4.2 5,-7.1 4.3 5,-7 4.2 5)))
```

Figure



This 3D example does not mix dimensions.

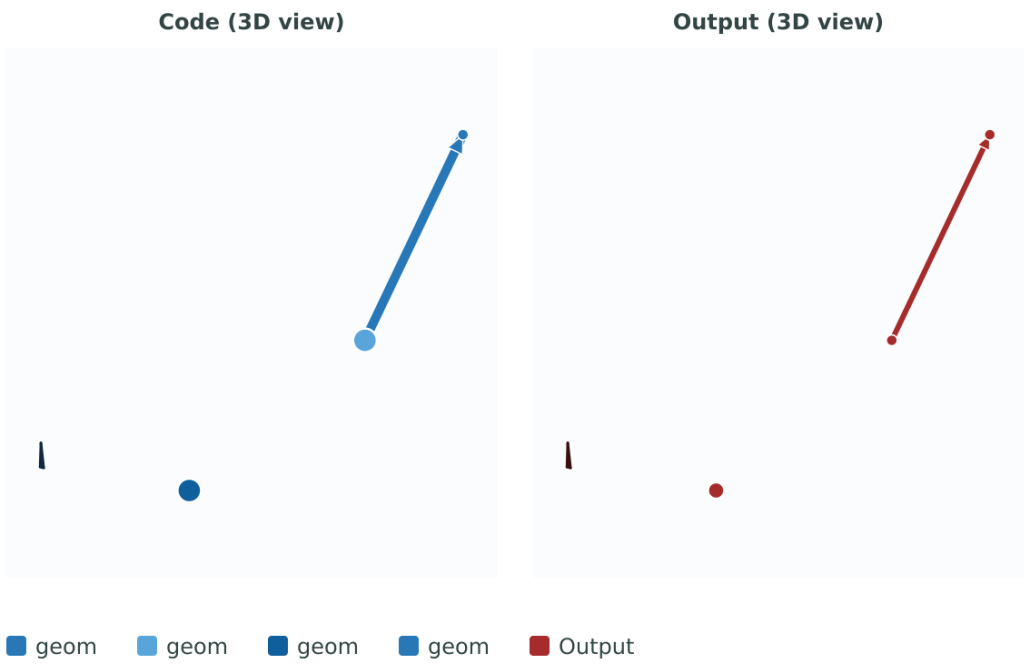
Code

```
SELECT ST_Union(geom)
FROM (
    SELECT 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2))'::geometry geom
    UNION ALL
    SELECT 'POINT(5 5 5)'::geometry geom
    UNION ALL
    SELECT 'POINT(-2 3 1)'::geometry geom
    UNION ALL
    SELECT 'LINESTRING(5 5 5,10 10 10)'::geometry geom
) AS foo;
```

Output

```
GEOMETRYCOLLECTION(POINT(-2 3 1), LINESTRING(5 5 5,10 10 10), POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,-7 4.2 2)))
```

Figure



This example unions an array built from a query.

Code

```
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));
```

This example unions an array of linestrings.

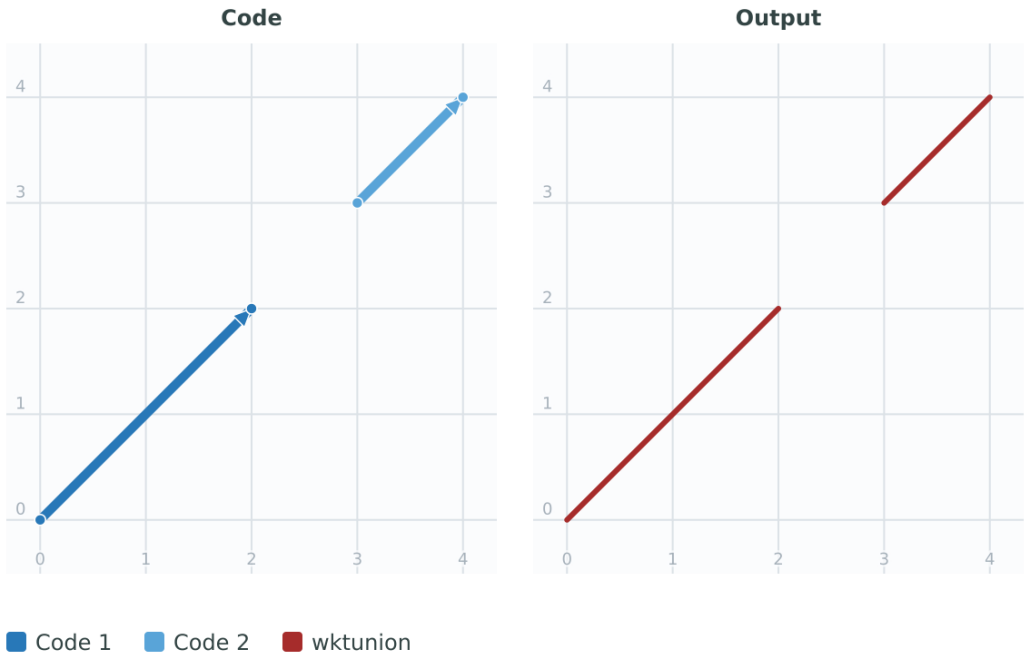
Code

```
SELECT ST_Union(ARRAY[
    'LINESTRING(1 2,3 4)'::geometry,
    'LINESTRING(4 5,5 6)'::geometry
]) As wktunion;
```

Output

```
MULTILINESTRING((1 2,3 4),(4 5,5 6))
```

Figure



Voir aussi

ST_CoverageUnion, ST_UnaryUnion, ST_MemUnion, ST_Collect, ST_Intersection, ST_Difference, ST_SymDifference, ST_LineMerg

7.14 Traitement des géométries

7.14.1 ST_Buffer

ST_Buffer — Calcule une géométrie couvrant tous les points situés à une distance donnée d'une géométrie.

Synopsis

geometry **ST_Buffer**(geometry g1, float radius_of_buffer, text buffer_style_parameters = '');
geometry **ST_Buffer**(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography **ST_Buffer**(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography **ST_Buffer**(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);

Description

Calcule un POLYGONE ou un MULTIPOLYGONE représentant tous les points dont la distance par rapport à une geometry/-geography est inférieure ou égale à une distance donnée. Une distance négative rétrécit la géométrie au lieu de l'étendre. Une distance négative peut réduire complètement un polygone, auquel cas POLYGON EMPTY est renvoyé. Pour les points et les lignes, les distances négatives renvoient toujours des résultats vides.

Pour les types geometry, la distance est spécifiée dans les unités du système de référence spatiale de la géométrie. Pour les types geography, la distance est spécifiée en mètres.

Le troisième paramètre facultatif contrôle la précision et le style du tampon. La précision des arcs de cercle dans le tampon est spécifiée en tant que nombre de segments de ligne utilisés pour approximer un quart de cercle (la valeur par défaut est de 8). Le style du tampon peut être spécifié en fournissant une liste de paires clé=valeur séparées par des blancs, comme suit :

— 'quad_segs=#' : nombre de segments de ligne utilisés pour approximer un quart de cercle (8 par défaut).
— 'endcap=round|flat|square' : style d'endcap (la valeur par défaut est "round"). 'butt' est accepté comme synonyme de 'flat'.
— 'join=round|mitre|bevel' : style de jointure (la valeur par défaut est "round"). 'miter' est accepté comme synonyme de 'mitre'.
— 'mitre_limit=#.#' : limite du rapport d'angle (n'affecte que le style d'assemblage en angle). 'miter_limit' est accepté comme synonyme de 'mitre_limit'.
— 'side=both|left|right' : defaults to 'both'. 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square when 'left' or 'right' are specified.



Note

Pour le type geography, il s'agit d'un fin wrapper autour de l'implémentation de la géométrie. Il détermine un système de référence spatiale planaire qui correspond le mieux à la boîte de délimitation de l'objet géographique (en essayant UTM, le pôle Nord/Sud de Lambert Azimuthal Equal Area (LAEA), et enfin Mercator). Le tampon est calculé dans l'espace planaire, puis retransformé en WGS84. Cela peut ne pas produire le comportement souhaité si l'objet d'entrée est beaucoup plus grand qu'une zone UTM ou s'il traverse la ligne de changement de date



Note

Buffer can handle invalid inputs and the output is always a valid polygonal geometry. Buffering by distance 0 is sometimes used as a way of repairing invalid polygons. ST_MakeValid is more suitable for this process as it can handle multi-polygons.



Note

La mise en mémoire tampon est parfois utilisée pour effectuer une recherche à l'intérieur de la distance. Dans ce cas, il est plus efficace d'utiliser ST_DWithin.



Note

Cette fonction ignore la dimension Z. Elle donne toujours un résultat en 2D, même lorsqu'elle est utilisée sur une géométrie en 3D.

Amélioration : 2.5.0 - La prise en charge de la géométrie ST_Buffer a été améliorée pour permettre la spécification de la mise en mémoire tampon latérale `side=both|left|right`.

Disponibilité : 1.5 - ST_Buffer a été amélioré pour prendre en charge différents types de terminaisons et de jointures. Ceux-ci sont utiles, par exemple, pour convertir les lignes de route en routes polygonales avec des bords plats ou carrés au lieu de bords arrondis. Un petit wrapper pour la geography a été ajouté.

Effectué par le module GEOS.



Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3



Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1.30

Exemples

The textual results are rounded to whole coordinate units for readability. Each result remains available as text, while the manual initially favors the shared input/output figure.

quad_segs=8 (par défaut)

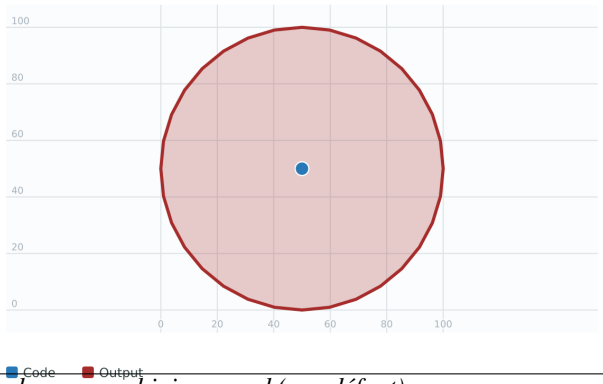
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=8');
```

Output

```
POLYGON((150 90,149 80,146 71,142 62,135 55,128 48,119
44,110 41,100 40,90 41,81 44,72 48,65 55,58 62,54 71,51
80,50 90,51 100,54 109,58 118,65 125,72 132,81 136,90
139,100 140,110 139,119 136,128 132,135 125,142 118,146
109,149 100,150 90))
```

Figure



quad_segs=2

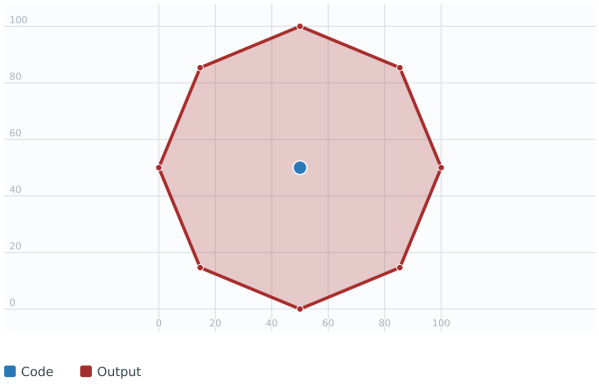
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('POINT(100 90)'),
  50, 'quad_segs=2');
```

Output

```
POLYGON((150 90,135 55,100 40,65 55,50 90,65 125,100
140,135 125,150 90))
```

Figure



endcap=round join=round (par défaut)

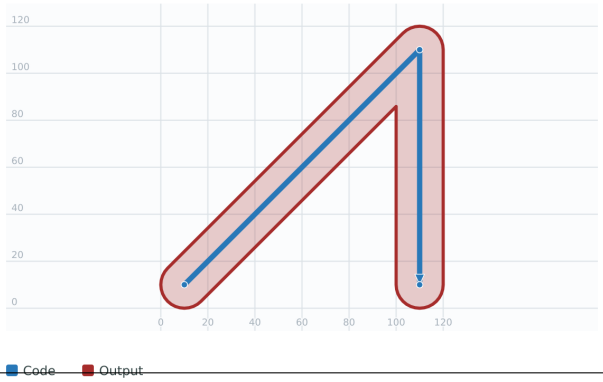
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'endcap=round join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152
160,154 159,156 158,157 157,158 156,159 154,160 152,160
150,160 50,160 48,159 46,158 44,157 43,156 42,154 41,152
40,150 40,148 40,146 41,144 42,143 43,142 44,141 46,140
48,140 50,140 126,57 43,56 42,54 41,52 40,50 40,48 40,46
41,44 42,43 43,42 44,41 46,40 48,40 50,40 52,41 54,42
56,43 57,143 157))
```

Figure



endcap=square

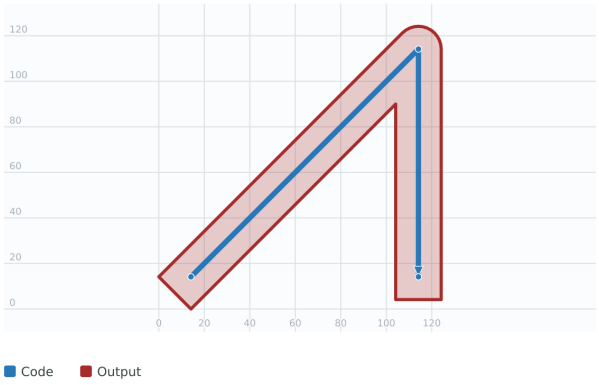
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'endcap=square join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152
160,154 159,156 158,157 157,158 156,159 154,160 152,160
150,160 50,160 40,140 40,140 126,57 43,50 36,36 50,143
157))
```

Figure



endcap=flat

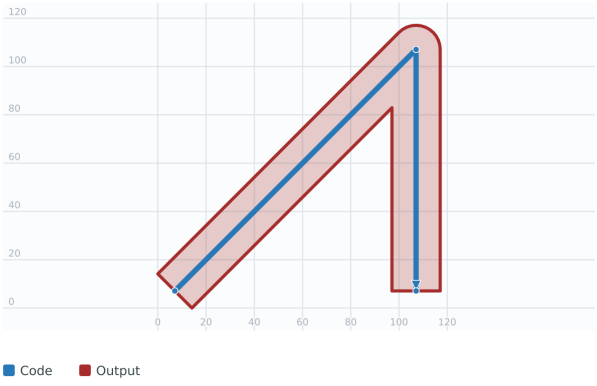
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'endcap=flat join=round');
```

Output

```
POLYGON((143 157,144 158,146 159,148 160,150 160,152
160,154 159,156 158,157 157,158 156,159 154,160 152,160
150,160 50,140 50,140 126,57 43,43 57,143 157))
```

Figure



join=bevel

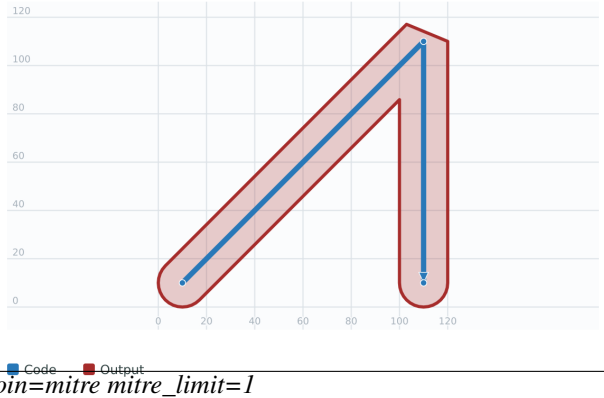
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'join=bevel');
```

Output

```
POLYGON((143 157,160 150,160 50,160 48,159 46,158 44,157
43,156 42,154 41,152 40,150 40,148 40,146 41,144 42,143
43,142 44,141 46,140 48,140 50,140 126,57 43,56 42,54
41,52 40,50 40,48 40,46 41,44 42,43 43,42 44,41 46,40
48,40 50,40 52,41 54,42 56,43 57,143 157))
```

Figure



join=mitre mitre_limit=5.0 (limite de mitre par défaut)

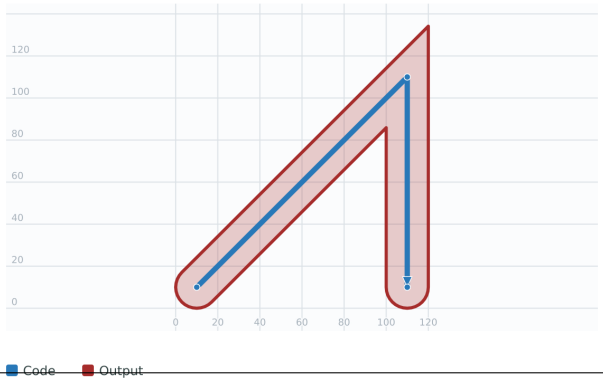
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'join=mitre mitre_limit=5.0');
```

Output

```
POLYGON((160 174,160 50,160 48,159 46,158 44,157 43,156
42,154 41,152 40,150 40,148 40,146 41,144 42,143 43,142
44,141 46,140 48,140 50,140 126,57 43,56 42,54 41,52 40,
50 40,48 40,46 41,44 42,43 43,42 44,41 46,40 48,40 50,40
52,41 54,42 56,43 57,160 174))
```

Figure



join=mitre mitre_limit=1

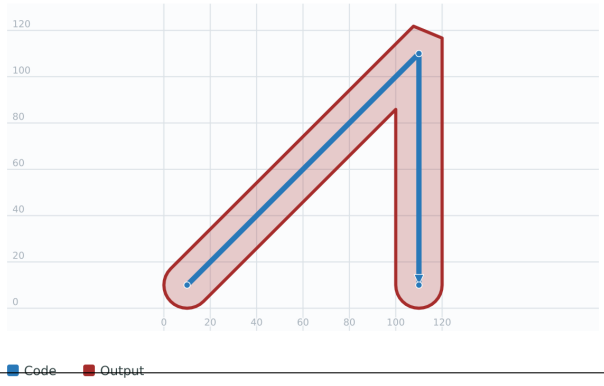
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'join=mitre mitre_limit=1.0');
```

Output

```
POLYGON((148 162,160 157,160 50,160 48,159 46,158 44,157
43,156 42,154 41,152 40,150 40,148 40,146 41,144 42,143
43,142 44,141 46,140 48,140 50,140 126,57 43,56 42,54
41,52 40,50 40,48 40,46 41,44 42,43 43,42 44,41 46,40
48,40 50,40 52,41 54,42 56,43 57,148 162))
```

Figure



side=left

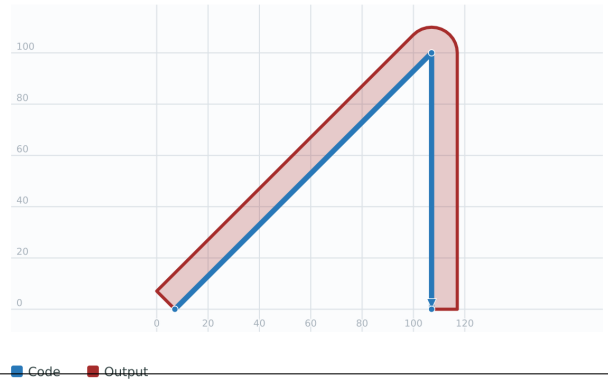
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'side=left');
```

Output

```
POLYGON((150 50,150 150,50 50,43 57,143 157,144 158,146
159,148 160,150 160,152 160,154 159,156 158,157 157,158
156,159 154,160 152,160 150,160 50,150 50))
```

Figure



side=right

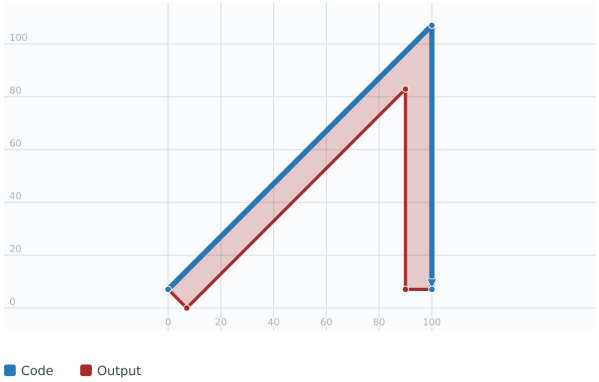
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'side=right');
```

Output

```
POLYGON((50 50,150 150,150 50,140 50,140 126,57 43,50
50))
```

Figure



side=left join=mitre

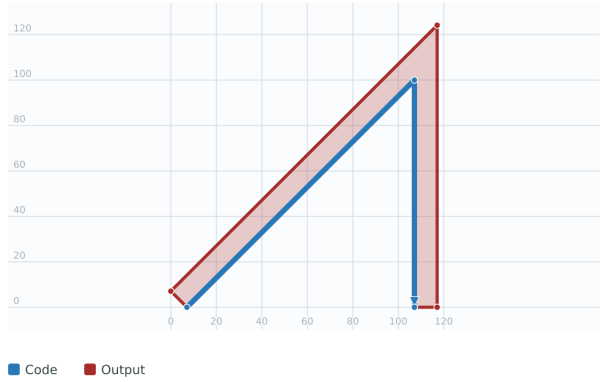
Code

```
SELECT ST_Buffer(
  ST_GeomFromText('LINESTRING(50 50,150 ↩
    150,150 50)'),
  10, 'side=left join=mitre');
```

Output

```
POLYGON((150 50,150 150,50 50,43 57,160 174,160 50,150
50))
```

Figure



right-hand winding, polygon boundary side=left

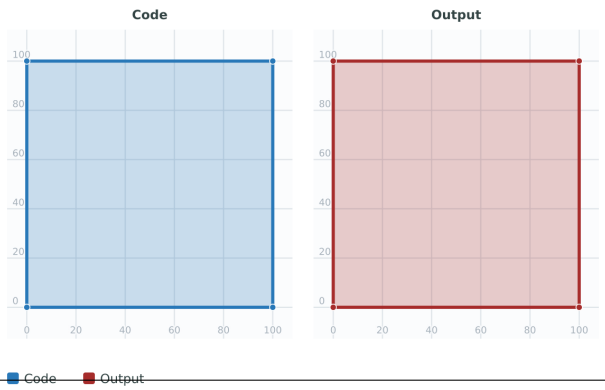
Code

```
SELECT ST_Buffer(
  ST_ForceRHR(ST_Boundary(ST_GeomFromText ↩
    (
    'POLYGON ((50 50,50 150,150 150,150 ↩
    50,50 50))'))),
  20, 'side=left');
```

Output

```
POLYGON((50 50,50 150,150 150,150 50,50 50))
```

Figure



right-hand winding, polygon boundary side=right

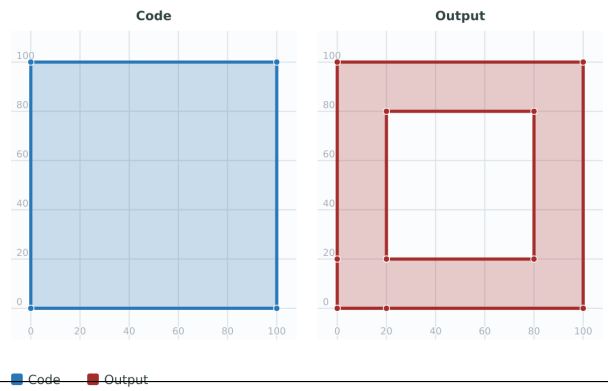
Code

```
SELECT ST_Buffer(
  ST_ForceRHR(ST_Boundary(ST_GeomFromText ↩
    (
    'POLYGON ((50 50,50 150,150 150,150  ↩
    50,50 50))')))),
  20, 'side=right');
```

Output

```
POLYGON((50 50,50 70,50 150,150 150,150 50,70 50,50 50),
(70 70,130 70,130 130,70 130,70 70))
```

Figure



A buffered point approximates a circle. A buffered point forcing approximation of (see diagram) 2 points per quarter circle is poly with 8 sides (see diagram)

Code

```
SELECT ST_NPoints(ST_Buffer('POINT(100 90)'::geometry, 50)) As promisingcircle_pcount,
ST_NPoints(ST_Buffer('POINT(100 90)'::geometry, 50, 2)) As lamecircle_pcount;
```

Output

```
promisingcircle_pcount | lamecircle_pcount
-----------------------+-------------------
                    33 |                 9
```

A lighter, less smooth circle using only 2 points per quarter circle is an octagon.

The following example creates a 100 meter octagon in NAD83 longitude/latitude by transforming to the Massachusetts state plane meter projection and then buffering. The figure shows the input and output in separate panels because the coordinates are in different SRIDs.

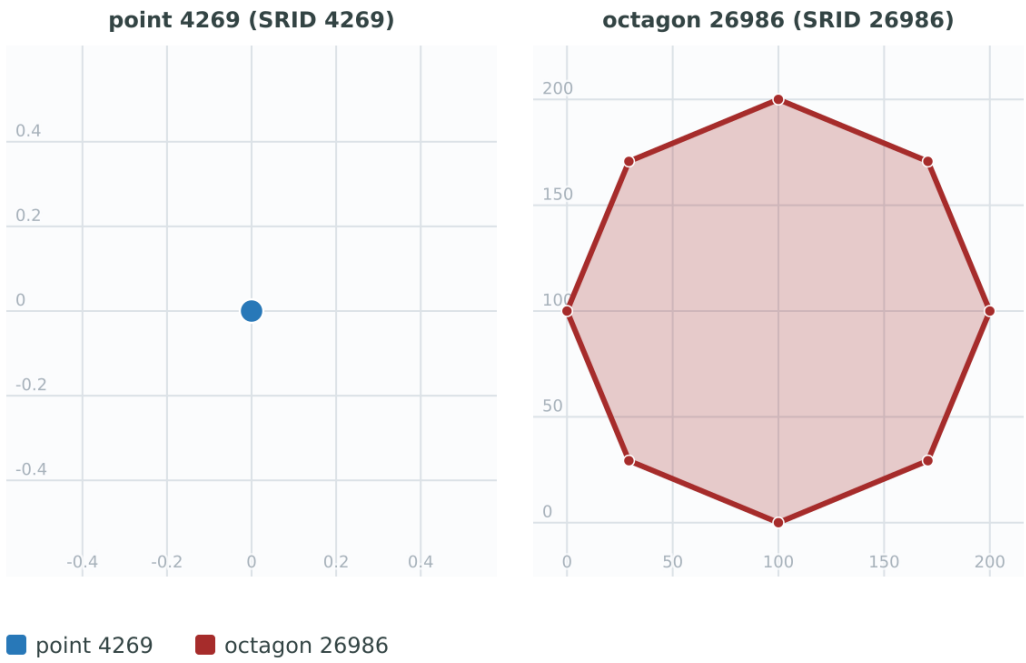
Code

```
WITH input AS (
  SELECT ST_SetSRID(ST_Point(-71.063526, 42.35785), 4269) AS point_4269
), projected AS (
  SELECT point_4269,
         ST_Transform(point_4269, 26986) AS point_26986
  FROM input
)
SELECT point_4269 AS input_point_4269,
       ST_Buffer(point_26986, 100, 2) AS octagon_26986
FROM projected;
```

Output

```
SRID=4269;POINT(-71.063526 42.35785) | SRID=26986;POLYGON((236057.5905746494 900908.7599186979,236028.30125276805 ↩
    900838.0492405792,235957.5905746494 900808.7599186979,235886.87989653074 900838.0492405792,235857.5905746494 ↩
    900908.7599186979,235886.87989653074 900979.4705968165,235957.5905746494 901008.7599186979,236028.30125276805 ↩
    900979.4705968165,236057.5905746494 900908.7599186979))
```

Figure



Buffering a polygon by a positive distance and then by the same negative distance is a morphological closing operation. It can be useful for smoothing noisy boundaries, such as simplifying a coastline by removing narrow inlets and bays. The result intentionally changes the shape and area of the input, and the distance must be chosen in the units of the input SRS. For data in a geographic SRS, transform to a projected SRS suited to the operation first; for cartographic visualization in Web Mercator, using that display SRS can be appropriate.

Smooth a coastline using a 1 km closing operation, then simplify the result. The example uses Massachusetts State Plane meter units for the operation.

Code

```
WITH projected AS (
  SELECT id, ST_Transform(geom, 26986) AS geom
  FROM coast
),
closed AS (
  SELECT id, ST_Buffer(ST_Buffer(geom, 1000), -1000) AS geom
  FROM projected
)
SELECT id, ST_Transform(ST_SimplifyPreserveTopology(geom, 50), 4326) AS geom
FROM closed;
```

The same technique is described in Paul Ramsey's Removing Complexities. The final simplification step is optional; use ST_SimplifyPres when polygonal validity matters, and check the result against the cartographic or analytical tolerance required by the application.

A small negative buffer can also be used as a heuristic to find narrow polygon spikes. The query below compares how much area and perimeter remain after erosion. A large `erosion_index` indicates that the perimeter dropped proportionally more than the area. This is only a preselection aid; the threshold and buffer distance must be chosen for the data units and the kind of spike being checked.

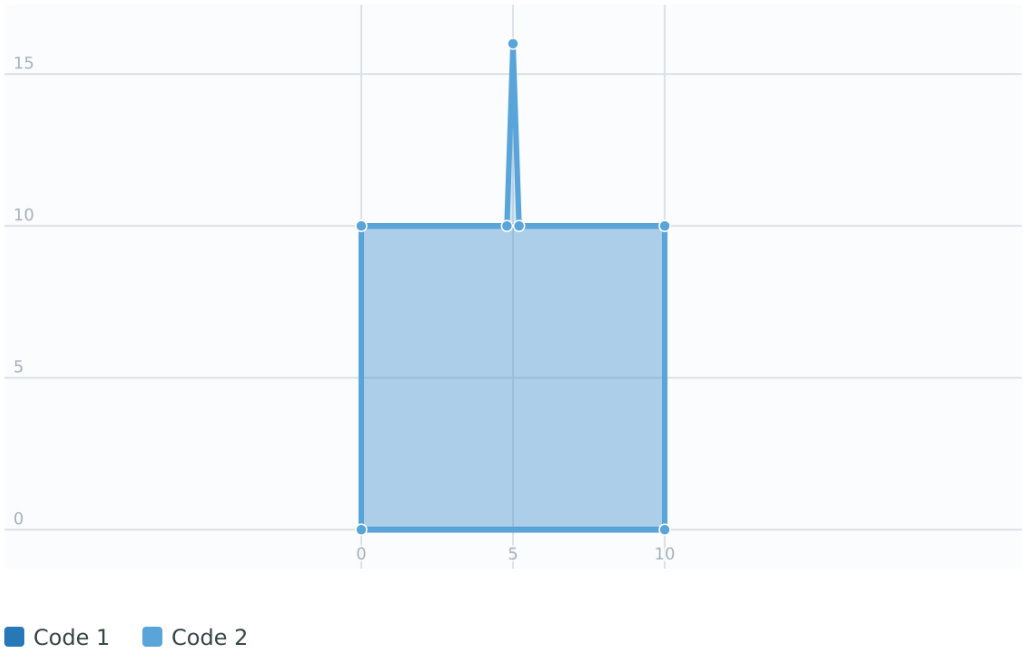
Code

```
WITH sample(id, geom) AS (
  VALUES
    ('plain', 'POLYGON((0 0,10 0,10 10,0 10,0 0))'::geometry),
    ('spike', 'POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0))'::geometry)
),
eroded AS (
  SELECT id, geom, ST_Buffer(geom, -0.5) AS eroded_geom
  FROM sample
),
scored AS (
  SELECT id,
         round((
           ST_Area(eroded_geom) / ST_Area(geom) /
           NULLIF(ST_Perimeter(eroded_geom) / ST_Perimeter(geom), 0)
         )::numeric, 3) AS erosion_index
  FROM eroded
  WHERE NOT ST_IsEmpty(eroded_geom)
)
SELECT id, erosion_index, erosion_index > 1.1 AS possible_spike
FROM scored
ORDER BY id;
```

Output

```
id    | erosion_index | possible_spike
-------+---------------+----------------
 plain |         0.900 | f
 spike |         1.147 | t
```

Figure



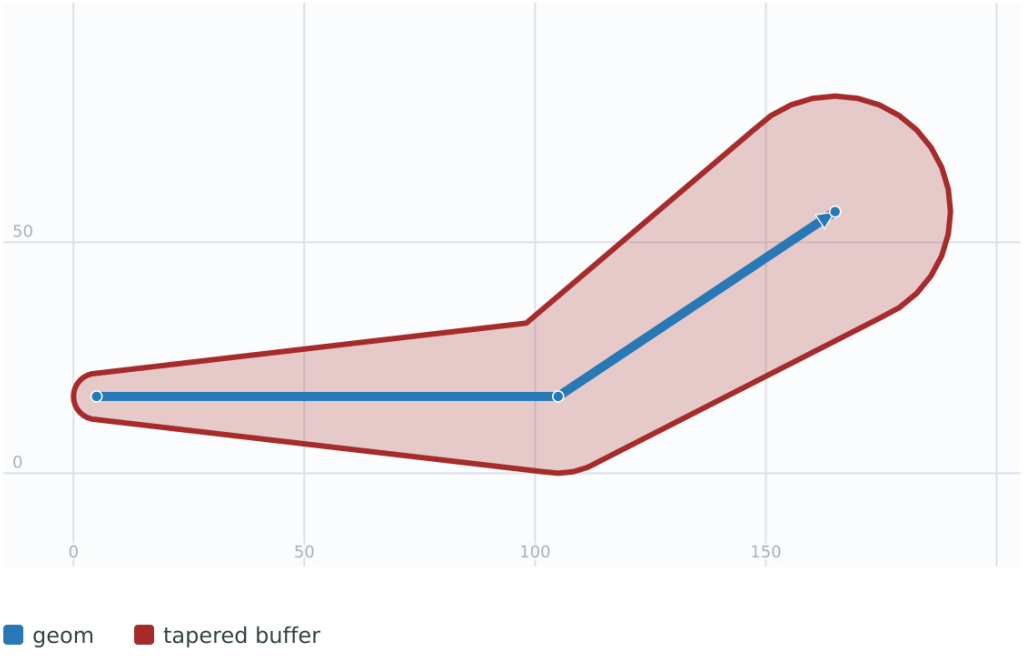
A variable-distance buffer can be approximated by buffering each vertex of a line by a distance interpolated along the line, building the convex hull for each consecutive pair of vertex buffers, and unioning the segment hulls. As with any geometry buffer, distances are in the units of the input SRS.

Taper a line buffer from 5 to 25 units along the line.

Code

```
WITH line AS (
  SELECT 'LINESTRING(0 0, 100 0, 160 40)'::geometry AS geom
),
vertices AS (
  SELECT (dump).path[1] AS n,
         (dump).geom AS geom,
         line.geom AS line_geom
  FROM line
  CROSS JOIN LATERAL ST_DumpPoints(line.geom) AS dump
),
radii AS (
  SELECT n,
         geom,
         5 + (25 - 5) * ST_LineLocatePoint(line_geom, geom) AS radius
  FROM vertices
),
segments AS (
  SELECT n,
         geom,
         radius,
         lead(geom) OVER (ORDER BY n) AS next_geom,
         lead(radius) OVER (ORDER BY n) AS next_radius
  FROM radii
)
SELECT ST_Union(
         ST_ConvexHull(
           ST_Collect(
             ST_Buffer(geom, radius),
             ST_Buffer(next_geom, next_radius)
           )
         )
       ) AS tapered_buffer
FROM segments
WHERE next_geom IS NOT NULL;
```

Figure



Voir aussi

ST_Collect, ST_ConvexHull, ST_DWithin, ST_DumpPoints, ST_LineLocatePoint, ST_SetSRID, ST_SimplifyPreserveTopology, ST_Transform, ST_Union, ST_MakeValid

7.14.2 ST_BuildArea

ST_BuildArea — Crée une géométrie polygonale formée par le tracé d'une géométrie.

Synopsis

geometry **ST_BuildArea**(geometry geom);

Description

Crée une géométrie aréolaire formée par les lignes constitutives de la géométrie d'entrée. L'entrée peut être une LineString, MultiLineString, Polygon, MultiPolygon ou une GeometryCollection. Le résultat est un Polygone ou un MultiPolygone, en fonction de l'entrée. Si la ligne d'entrée ne forme pas de polygone, NULL est renvoyé.

Contrairement à ST_MakePolygon, cette fonction accepte les anneaux formés par plusieurs lignes et peut former un nombre quelconque de polygones.

Cette fonction convertit les anneaux intérieurs en trous. Pour transformer les anneaux intérieurs en polygones, utilisez ST_Polygonize.



Note
Les lignes d'entrée doivent être correctement nouées pour que cette fonction fonctionne correctement. ST_Node peut être utilisé pour nouer des lignes.
Si le réseau de lignes d'entrée est croisé, cette fonction produira des polygones non valides. ST_MakeValid peut être utilisé pour s'assurer que la sortie est valide.

Disponibilité : 1.1.0

Exemples

Building an area from five input lines.

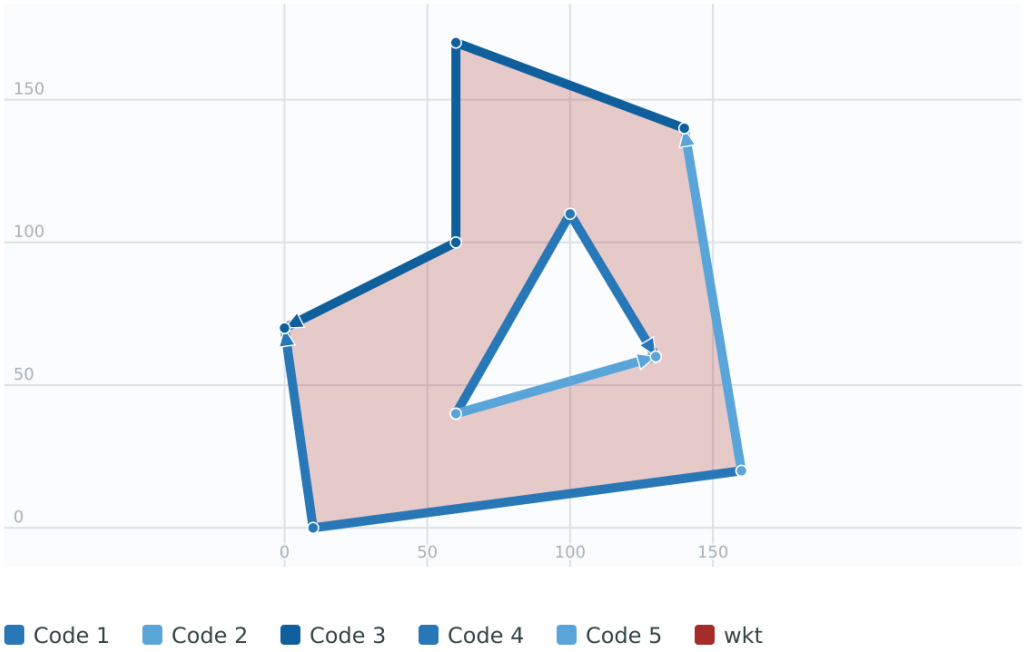
Code

```
WITH data(geom) AS (VALUES
   ('LINESTRING (180 40,30 20,20 90)'::geometry),
  ('LINESTRING (180 40,160 160)'::geometry),
  ('LINESTRING (160 160,80 190,80 120,20 90)'::geometry),
  ('LINESTRING (80 60,120 130,150 80)'::geometry),
  ('LINESTRING (80 60,150 80)'::geometry)
)
SELECT ST_BuildArea(ST_Collect(geom )) AS wkt
    FROM data;
```

Output

```
POLYGON((180 40,30 20,20 90,80 120,80 190,160 160,180 40),(150 80,120 130,80 60,150 80))
```

Figure

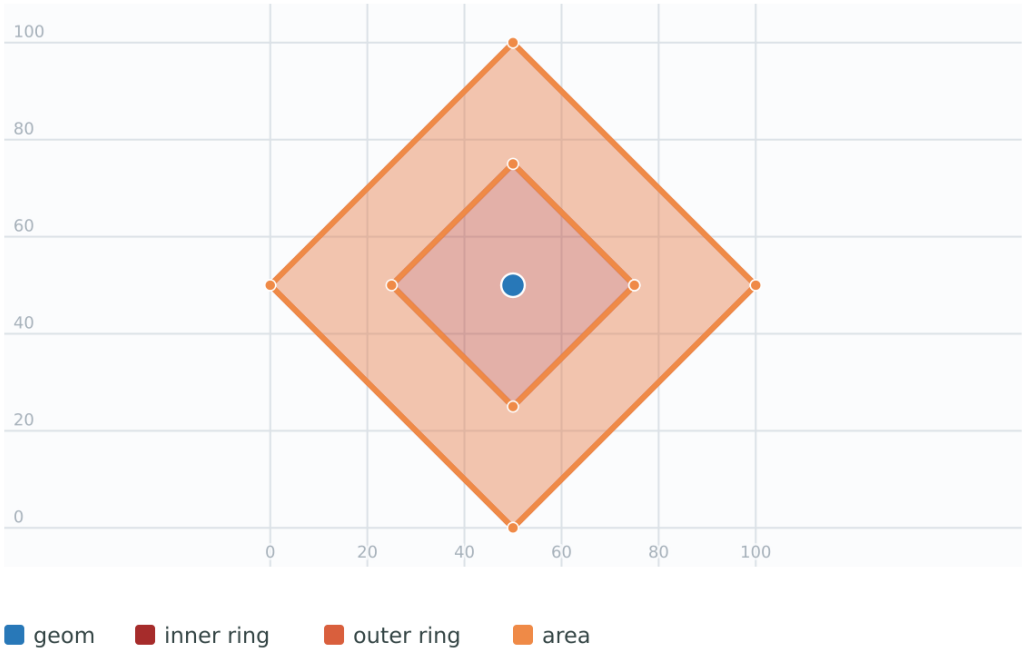


Create a donut from two circular polygons.

Code

```
WITH input AS (
  SELECT 'POINT(100 90)'::geometry AS geom
),
rings AS (
  SELECT ST_Buffer(geom, 25, 'quad_segs=1') AS inring,
         ST_Buffer(geom, 50, 'quad_segs=1') AS outring
  FROM input
)
SELECT inring AS inner_ring,
       outring AS outer_ring,
       ST_BuildArea(ST_Collect(inring, outring)) AS area
FROM rings;
```

Figure



Input linework that crosses at non-endpoint intersections must be noded before building areas:

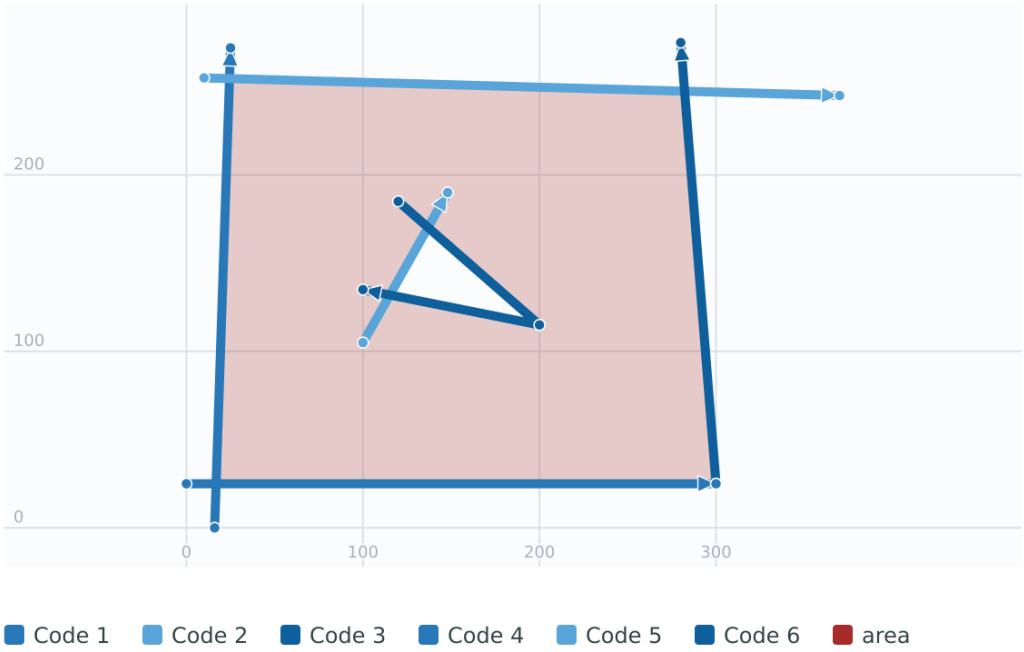
Code

```
WITH lines(geom) AS (VALUES
  ('LINESTRING(36 35,45 307)'::geometry),
  ('LINESTRING(30 290,390 280)'::geometry),
  ('LINESTRING(320 60,300 310)'::geometry),
  ('LINESTRING(20 60,320 60)'::geometry),
  ('LINESTRING(120 140,168 225)'::geometry),
  ('LINESTRING(140 220,220 150,120 170)'::geometry)
),
noded AS (
  SELECT ST_Node(ST_UnaryUnion(ST_Collect(geom))) AS geom FROM lines
)
SELECT ST_BuildArea(geom) AS area
FROM noded;
```

Output

```
POLYGON((320 60,36.8 60,44.4 289.6,302.2 282.4,320 60),(156.9 205.3,135.2 167,220 150,156.9 205.3))
```

Figure



Voir aussi

ST_Collect, ST_MakePolygon, ST_MakeValid, ST_Node, ST_Polygonize, ST_BdPolyFromText, ST_BdMPolyFromText (wrappers de cette fonction avec l'interface standard de l'OGC)

7.14.3 ST_Centroid

ST_Centroid — Renvoie le centre géométrique d'une géométrie.

Synopsis

geometry **ST_Centroid**(geometry g1);
geography **ST_Centroid**(geography g1, boolean use_spheroid = true);

Description

Calcule un point qui est le centre de masse géométrique d'une géométrie. Pour [MULTI]POINTs, le centroïde est la moyenne arithmétique des coordonnées entrées. Pour les [MULTI]LINESTRINGs, le centroïde est calculé en utilisant la longueur pondérée de chaque segment de ligne. Pour les [MULTI]POLYGONs, le centroïde est calculé en termes de surface. Si une géométrie vide est fournie, une GEOMETRYCOLLECTION vide est renvoyée. Si NULL est fourni, NULL est renvoyé. Si CIRCULARSTRING ou COMPOUNDCURVE sont fournis, ils sont convertis en linestring avec CurveToLine d'abord, puis de la même manière que pour LINESTRING

Pour les entrées de dimensions mixtes, le résultat est égal au centroïde de la composante Géométries de la dimension la plus élevée (puisque les géométries de dimension inférieure n'apportent aucun "poids" au centroïde).

Notez que pour les géométries polygonales, le centroïde n'est pas nécessairement situé à l'intérieur du polygone. Par exemple, voir le diagramme ci-dessous du centroïde d'un polygone en forme de C. Pour construire un point garanti à l'intérieur d'un polygone, utilisez ST_PointOnSurface.

Nouveau dans la version 2.3.0 : prend en charge CIRCULARSTRING et COMPOUNDCURVE (en utilisant CurveToLine)

Disponibilité : la prise en charge du type geography a été introduite dans la version 2.4.0.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1.

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 8.1.4, 9.5.5

Exemples

Centroid of a MultiPoint.

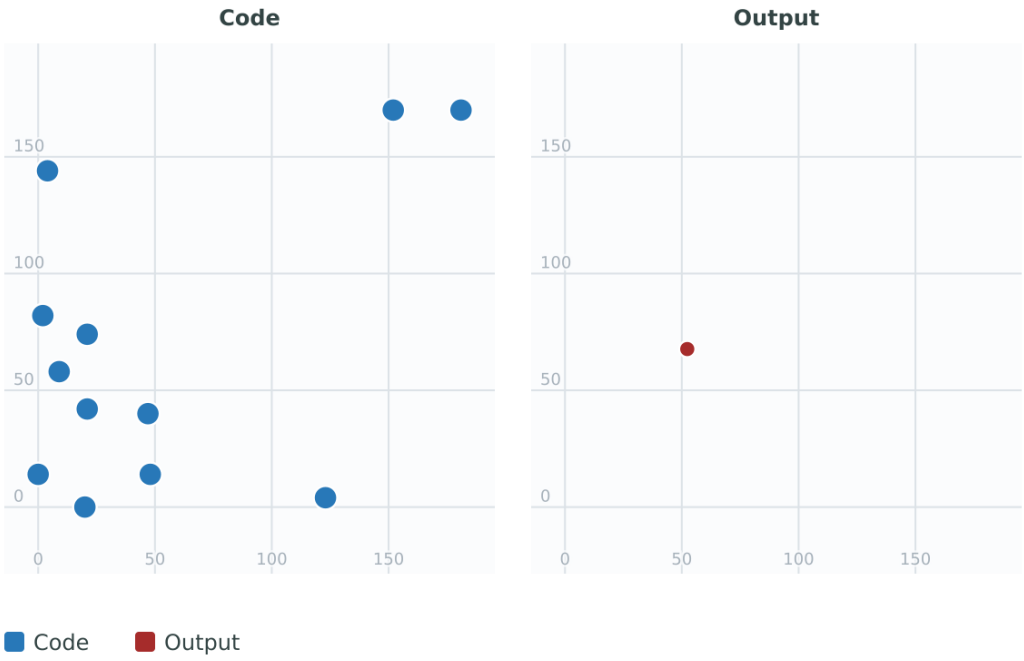
Code

```
SELECT ST_Centroid(
  'MULTIPOINT (8 24,10 92,12 154,17 68,28 10,29 52,29 84,55 50,56 24,131 14,160 180,189 180) ↩
    ');
```

Output

```
POINT(60.333 77.667)
```

Figure



Centroid of a LineString.

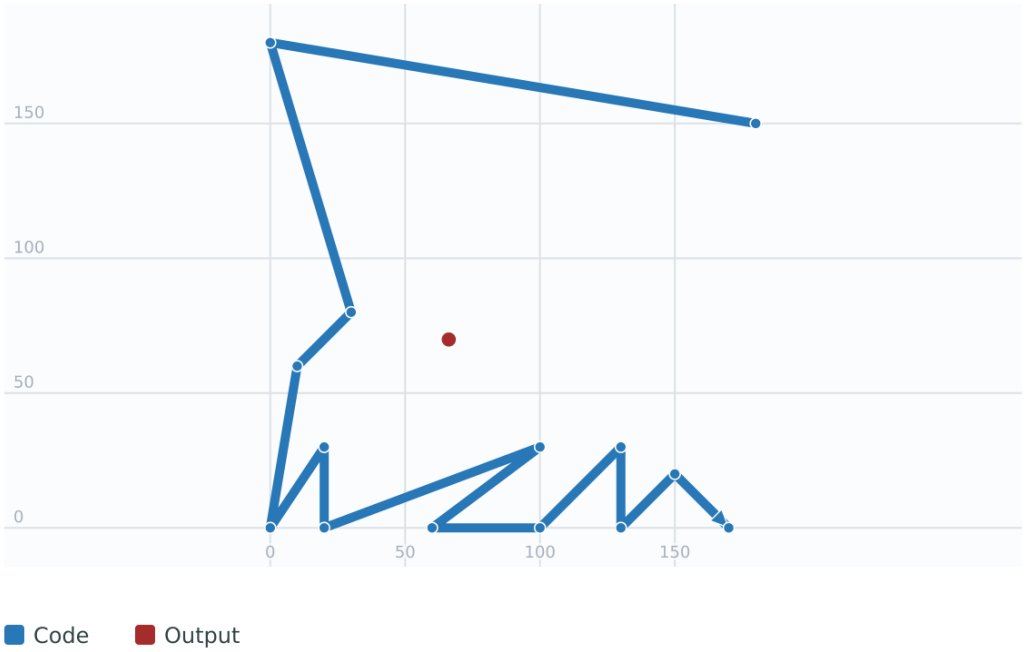
Code

```
SELECT ST_Centroid(
  'LINESTRING (190 160,10 190,40 90,20 70,10 10,30 40,30 10,110 40,70 10,110 10,140 40,140  ↩
    10,160 30,180 10)');
```

Output

```
POINT(76.191 79.876)
```

Figure



Centroid of a Polygon.

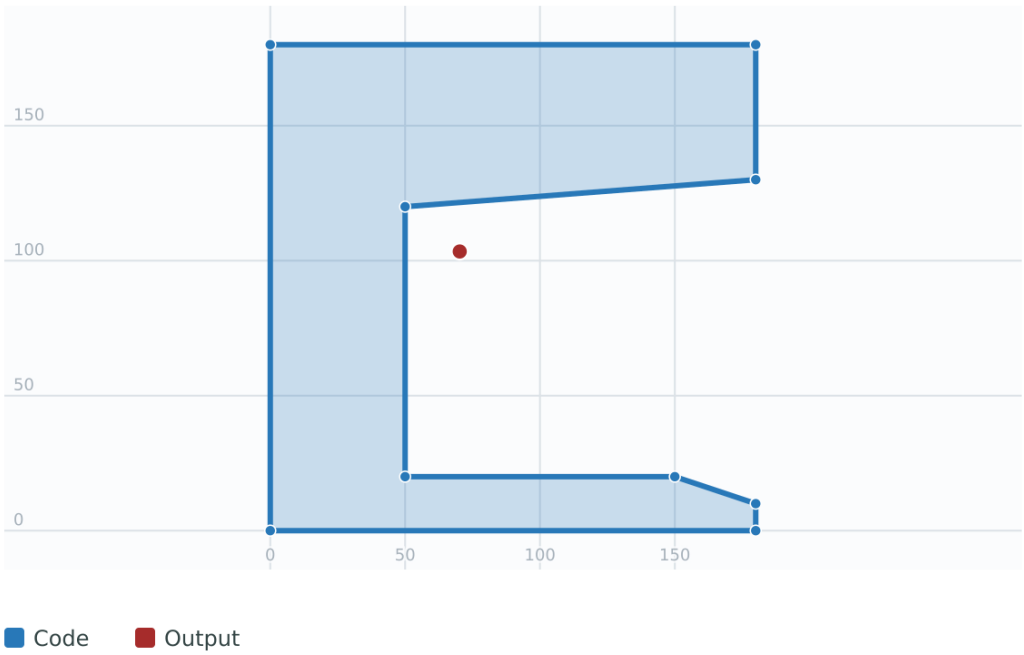
Code

```
SELECT ST_Centroid(
  'POLYGON ((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))');
```

Output

```
POINT(80.251 113.405)
```

Figure



Centroid of the highest-dimension components of a GeometryCollection.

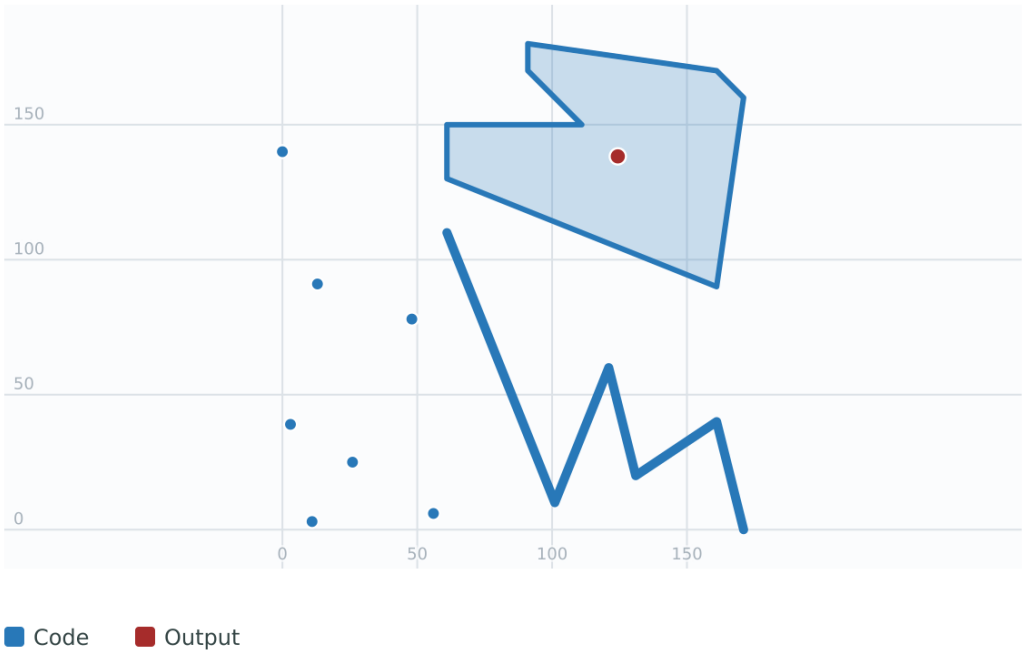
Code

```
SELECT ST_Centroid(
  'GEOMETRYCOLLECTION (POLYGON ((190 170,180 100,80 140,80 160,130 160,110 180,110 190,180 ↩
   180,190 170)),
    LINESTRING (80 120,120 20,140 70,150 30,180 50,190 10),
    MULTIPOINT (19 150,22 49,30 13,32 101,45 35,67 88,75 16))');
```

Output

```
POINT(143.361 148.263)
```

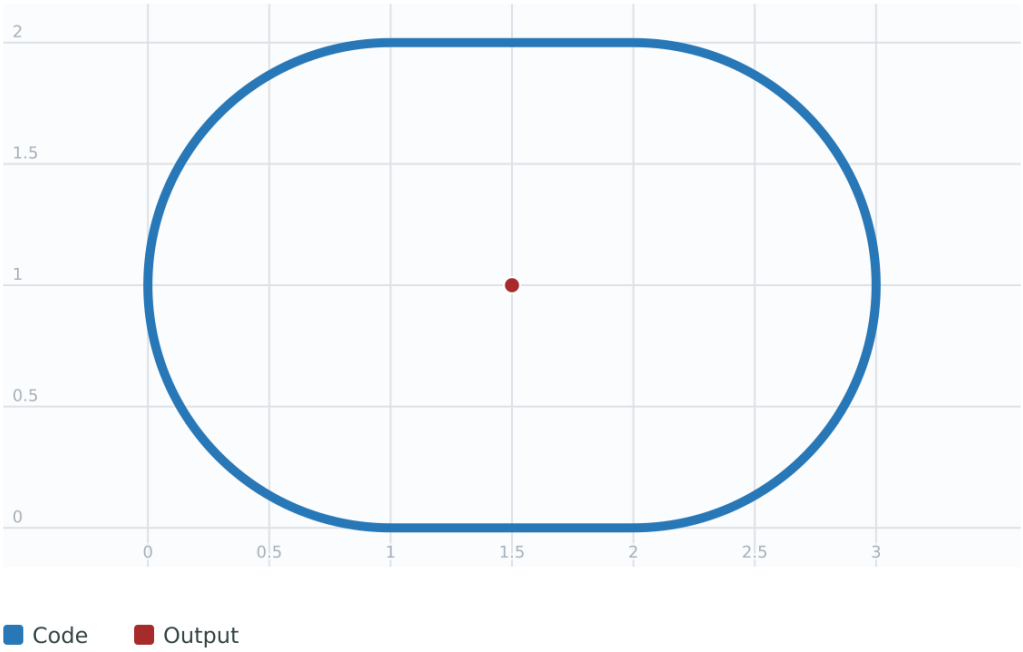
Figure



Code

```
SELECT ST_Centroid(g)
FROM  ST_GeomFromText('CIRCULARSTRING(0 2,-1 1,0 0,0.5 0,1 0,2 1,1 2,0.5 2,0 2)')  AS g ;
```

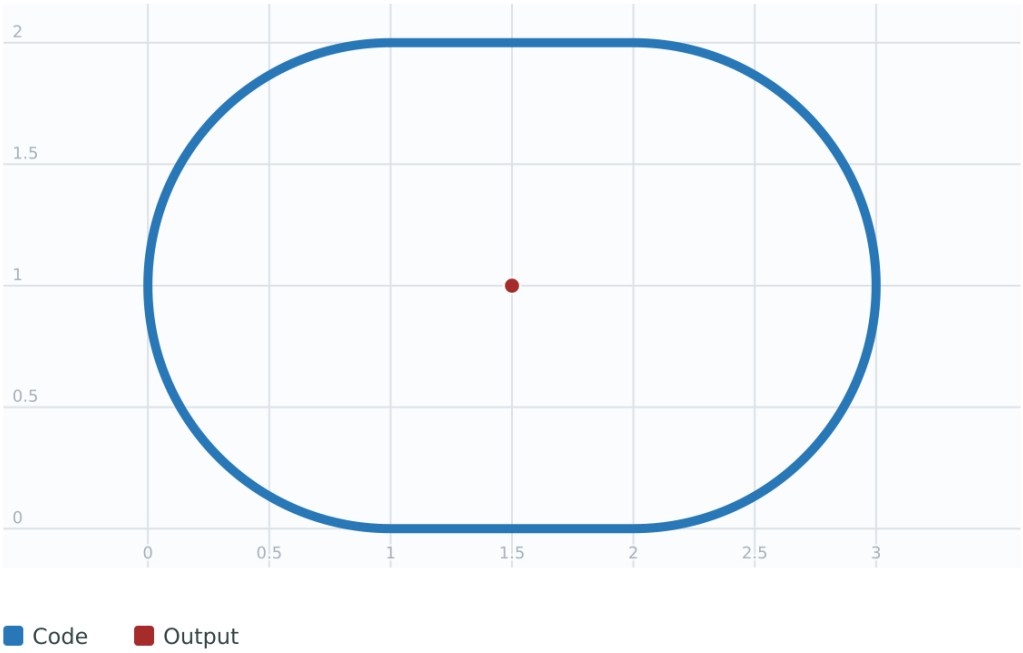
Figure



Code

```
SELECT ST_Centroid(g)
FROM  ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2,-1 1,0 0),(0 0,0.5 0,1 0),  ↩
    CIRCULARSTRING( 1 0,2 1,1 2),(1 2,0.5 2,0 2))' ) AS g;
```

Figure



Centroids of the parts of a MultiPolygon.

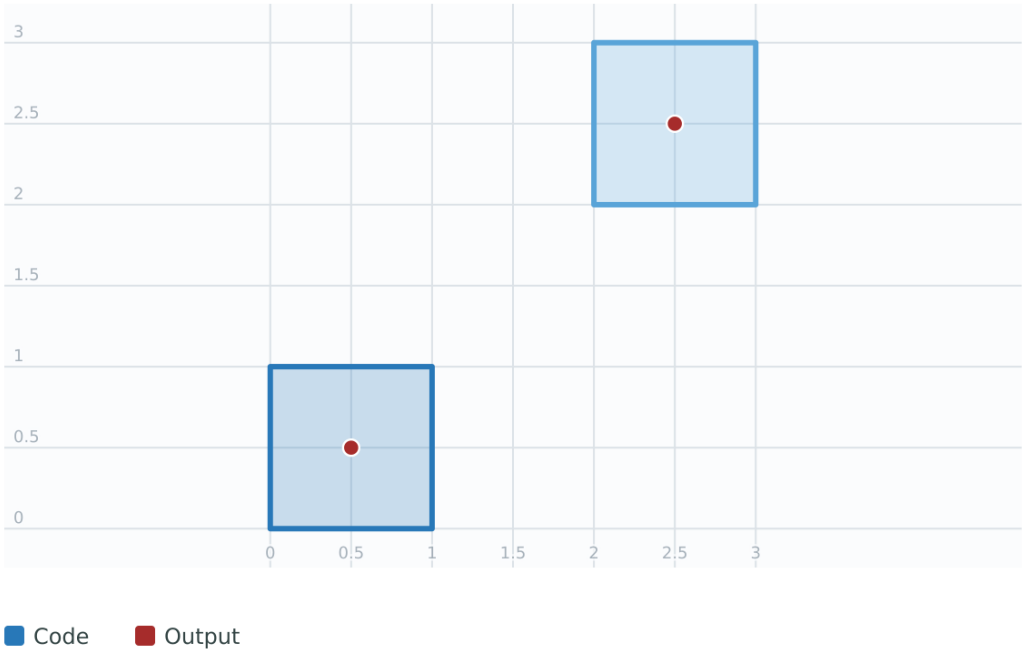
Code

```
SELECT ST_Collect(ST_Centroid(geom) ORDER BY path)
FROM ST_Dump('MULTIPOLYGON (
    ((0 0,0 1,1 1,1 0,0 0)),
    ((2 2,2 3,3 3,3 2,2 2))
  )'::geometry);
```

Output

```
MULTIPOINT((0.5 0.5),(2.5 2.5))
```

Figure



Voir aussi

ST_Collect, ST_Dump, ST_PointOnSurface, ST_GeometricMedian

7.14.4 ST_ChaikinSmoothing

ST_ChaikinSmoothing — Renvoie une version lissée d'une géométrie, en utilisant l'algorithme Chaikin

Synopsis

geometry **ST_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndPoints = false);

Description

Lisse une géométrie linéaire ou polygonale en utilisant Chaikin's algorithm. Le degré de lissage est contrôlé par le paramètre `nIterations`. À chaque itération, chaque sommet intérieur est remplacé par deux sommets situés à 1/4 de la longueur des segments de ligne précédant et suivant le sommet. Un degré raisonnable de lissage est fourni par 3 itérations ; le maximum est limité à 5.

Si `preserveEndPoints` est égal à true, les extrémités des anneaux de polygone ne sont pas lissées. Les extrémités des lignes sont toujours préservées.



Note
Le nombre de sommets double à chaque itération, de sorte que la géométrie résultante peut comporter beaucoup plus de points que la géométrie d'entrée. Pour réduire le nombre de points, utilisez une fonction de simplification sur le résultat (voir ST_Simplify, ST_SimplifyPreserveTopology et ST_SimplifyVW).

Le résultat comporte des valeurs interpolées pour les dimensions Z et M lorsqu'elles sont présentes.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Disponibilité : 2.5.0

Exemples

Smoothing a Polygon using 1, 2 and 3 iterations.

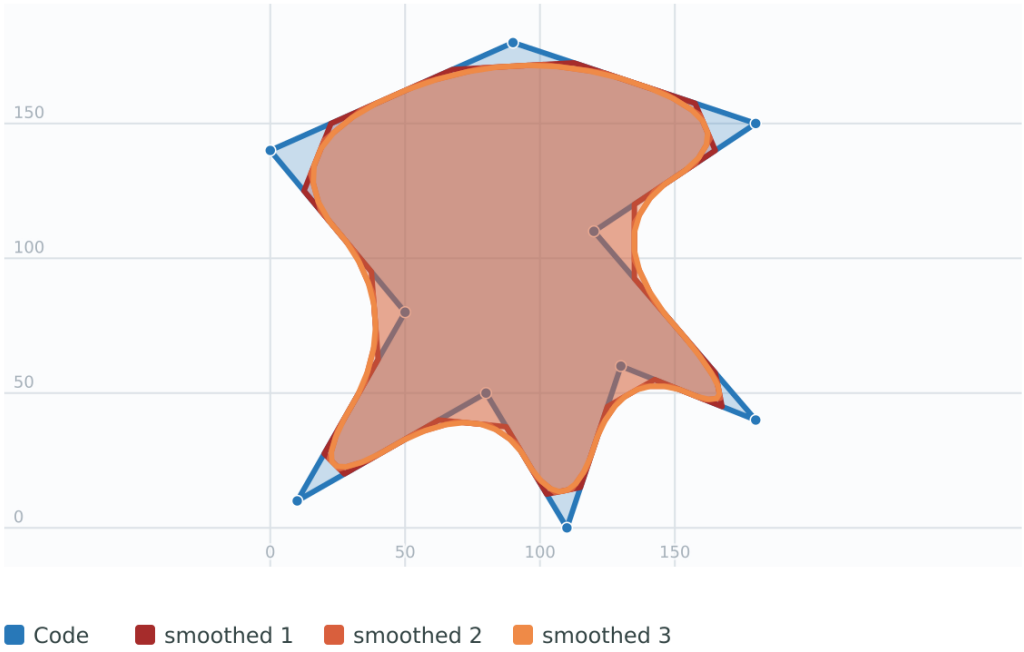
Code

```
SELECT n AS iterations, ST_ChaikinSmoothing(
    'POLYGON ((20 20,60 90,10 150,100 190,190 160,130 120,190 50,140 70,120 10,90 60,20 20))'
    , n
  ) AS smoothed
FROM generate_series(1, 3) AS n;
```

Output

```
1 | POLYGON((30 37.5,50 72.5,47.5 105,22.5 135,32.5 160,77.5 180,122.5 182.5,167.5 167.5,175 150,145 130,145 102.5,175
    67.5,177.5 55,152.5 65,135 55,125 25,112.5 22.5,97.5 47.5,72.5 50,37.5 30,30 37.5))
2 | POLYGON((35 46.25,45 63.75,49.375 80.625,48.125 96.875,41.25 112.5,28.75 127.5,25 141.25,30 153.75,43.75 165,66.25
    175,88.75 180.625,111.25 181.875,133.75 178.75,156.25 171.25,169.375 163.125,173.125 154.375,167.5 145,152.5
    135,145 123.125,145 109.375,152.5 93.75,167.5 76.25,175.625 64.375,176.875 58.125,171.25 57.5,158.75 62.5,148.125
    62.5,139.375 57.5,132.5 47.5,127.5 32.5,121.875 24.375,115.625 23.125,108.75 28.75,101.25 41.25,91.25 48.125,78.75
     49.375,63.75 45,46.25 35,35.625 31.875,31.875 35.625,35 46.25))
3 | POLYGON((37.5 50.625,42.5 59.375,46.09375 67.96875,48.28125 76.40625,49.0625 84.6875,48.4375 92.8125,46.40625
    100.78125,42.96875 108.59375,38.125 116.25,31.875 123.75,27.8125 130.9375,25.9375 137.8125,26.25 144.375,28.75
    150.625,33.4375 156.5625,40.3125 162.1875,49.375 167.5,60.625 172.5,71.875 176.40625,83.125 179.21875,94.375
    180.9375,105.625 181.5625,116.875 181.09375,128.125 179.53125,139.375 176.875,150.625 173.125,159.53125
    169.21875,166.09375 165.15625,170.3125 160.9375,172.1875 156.5625,171.71875 152.03125,168.90625 147.34375,163.75
    142.5,156.25 137.5,150.625 132.03125,146.875 126.09375,145 119.6875,145 112.8125,146.875 105.46875,150.625
    97.65625,156.25 89.375,163.75 80.625,169.53125 73.28125,173.59375 67.34375,175.9375 62.8125,176.5625
    59.6875,175.46875 57.96875,172.65625 57.65625,168.125 58.75,161.875 61.25,156.09375 62.5,150.78125 62.5,145.9375
    61.25,141.5625 58.75,137.65625 55,134.21875 50,131.25 43.75,128.75 36.25,126.09375 30.46875,123.28125
    26.40625,120.3125 24.0625,117.1875 23.4375,113.90625 24.53125,110.46875 27.34375,106.875 31.875,103.125
    38.125,98.75 42.96875,93.75 46.40625,88.125 48.4375,81.875 49.0625,75 48.28125,67.5 46.09375,59.375 42.5,50.625
    37.5,43.59375 34.21875,38.28125 32.65625,34.6875 32.8125,32.8125 34.6875,32.65625 38.28125,34.21875 43.59375,37.5
    50.625))
```

Figure



Smoothing a LineString using 1, 2 and 3 iterations.

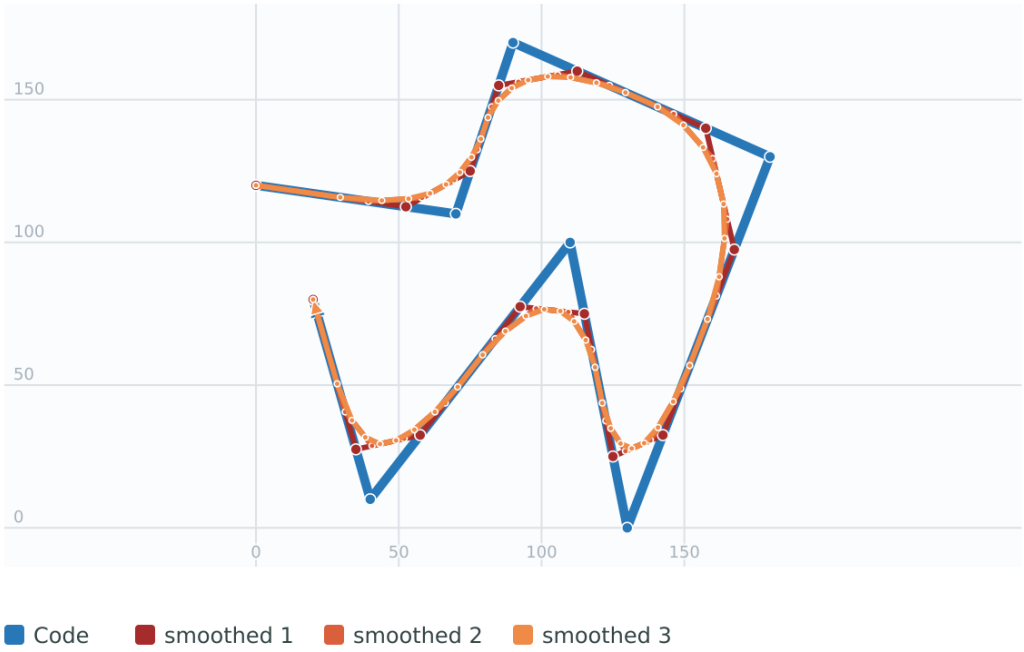
Code

```
SELECT n AS iterations, ST_ChaikinSmoothing(
    'LINESTRING (10 140,80 130,100 190,190 150,140 20,120 120,50 30,30 100)', n
  ) AS smoothed
FROM generate_series(1, 3) AS n;
```

Output

```
1 | LINESTRING(10 140,62.5 132.5,85 145,95 175,122.5 180,167.5 160,177.5 117.5,152.5 52.5,135 45,125 95,102.5 97.5,67.5 ↩
     52.5,45 47.5,30 100)
2 | LINESTRING(10 140,49.375 134.375,68.125 135.625,79.375 141.875,87.5 152.5,92.5 167.5,101.875 176.25,115.625  ↩
    178.75,133.75 175,156.25 165,170 149.375,175 128.125,171.25 101.25,158.75 68.75,148.125 50.625,139.375  ↩
    46.875,132.5 57.5,127.5 82.5,119.375 95.625,108.125 96.875,93.75 86.25,76.25 63.75,61.875 51.25,50.625 48.75,41.25 ↩
     60.625,30 100)
3 | LINESTRING(10 140,39.53125 135.78125,54.0625 134.6875,63.4375 135.3125,70.9375 137.1875,76.5625 140.3125,81.40625  ↩
    144.53125,85.46875 149.84375,88.75 156.25,91.25 163.75,94.84375 169.6875,99.53125 174.0625,105.3125  ↩
    176.875,112.1875 178.125,120.15625 177.8125,129.21875 175.9375,139.375 172.5,150.625 167.5,159.6875  ↩
    161.09375,166.5625 153.28125,171.25 144.0625,173.75 133.4375,174.0625 121.40625,172.1875 107.96875,168.125  ↩
    93.125,161.875 76.875,156.09375 64.21875,150.78125 55.15625,145.9375 49.6875,141.5625 47.8125,137.65625  ↩
    49.53125,134.21875 54.84375,131.25 63.75,128.75 76.25,125.46875 85.78125,121.40625 92.34375,116.5625  ↩
    95.9375,110.9375 96.5625,104.53125 94.21875,97.34375 88.90625,89.375 80.625,80.625 69.375,72.65625 60.625,65.46875 ↩
     54.375,59.0625 50.625,53.4375 49.375,48.28125 51.71875,43.59375 57.65625,38.4375 70.46875,30 100)
```

Figure



Smoothing a triangle shows the weakest visual change, so it is kept last.

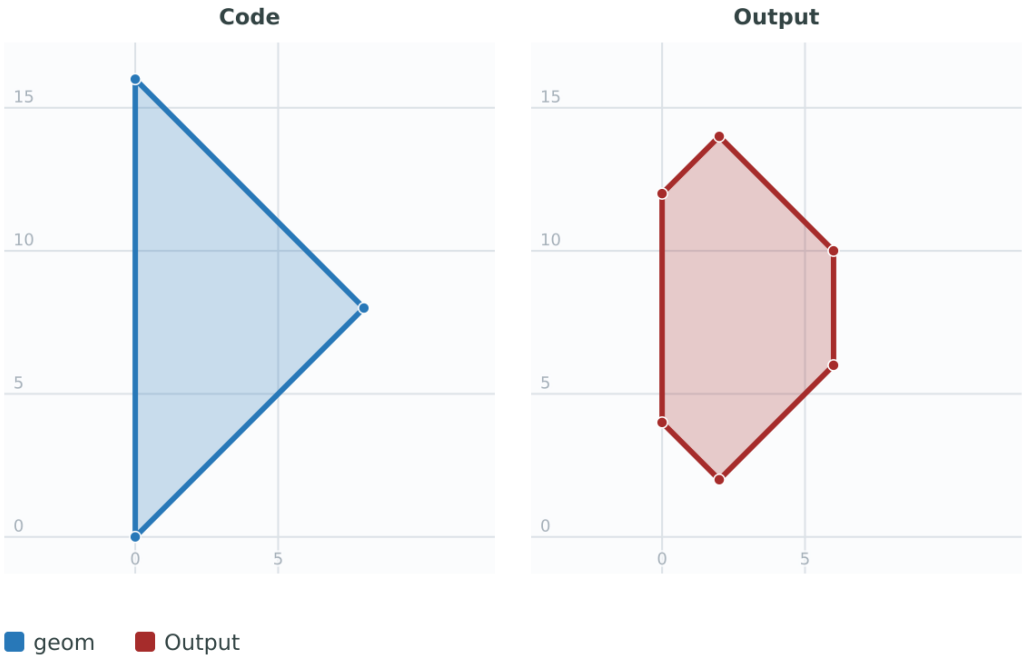
Code

```
SELECT ST_ChaikinSmoothing(geom) smoothed
FROM (SELECT  'POLYGON((0 0,8 8,0 16,0 0))'::geometry geom) AS foo;
```

Output

```
POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))
```

Figure



Voir aussi

ST_Simplify, ST_SimplifyPreserveTopology, ST_SimplifyVW

7.14.5 ST_CatmullRomSmoothing

ST_CatmullRomSmoothing — Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm

Synopsis

geometry **ST_CatmullRomSmoothing**(geometry geom, integer nSegments = 5);

Description

Smoothes a linear or polygonal geometry using the Catmull-Rom spline algorithm. Unlike ST_ChaikinSmoothing, this is an *interpolating* spline: the output curve passes through every original vertex. Between each pair of consecutive original vertices, `nSegments` - 1 new intermediate points are inserted.

The `nSegments` parameter controls the density of the output. With `nSegments` = 5 (the default), each original span is divided into 5 sub-segments (inserting 4 new points per span). The minimum value is 2.

At least 4 input vertices are required to apply smoothing; geometries with fewer vertices are returned unchanged. Points and multipoints are always returned unchanged.



Note
The output vertex count grows as `1 + (N-1) * nSegments` for open lines, and `1 + N * nSegments` for closed rings, where N is the number of input vertices. For large geometries, use a simplification function on the result to reduce the number of points (see ST_Simplify, ST_SimplifyPreserveTopology and ST_SimplifyVW).

Le résultat comporte des valeurs interpolées pour les dimensions Z et M lorsqu'elles sont présentes.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Availability: 3.7.0

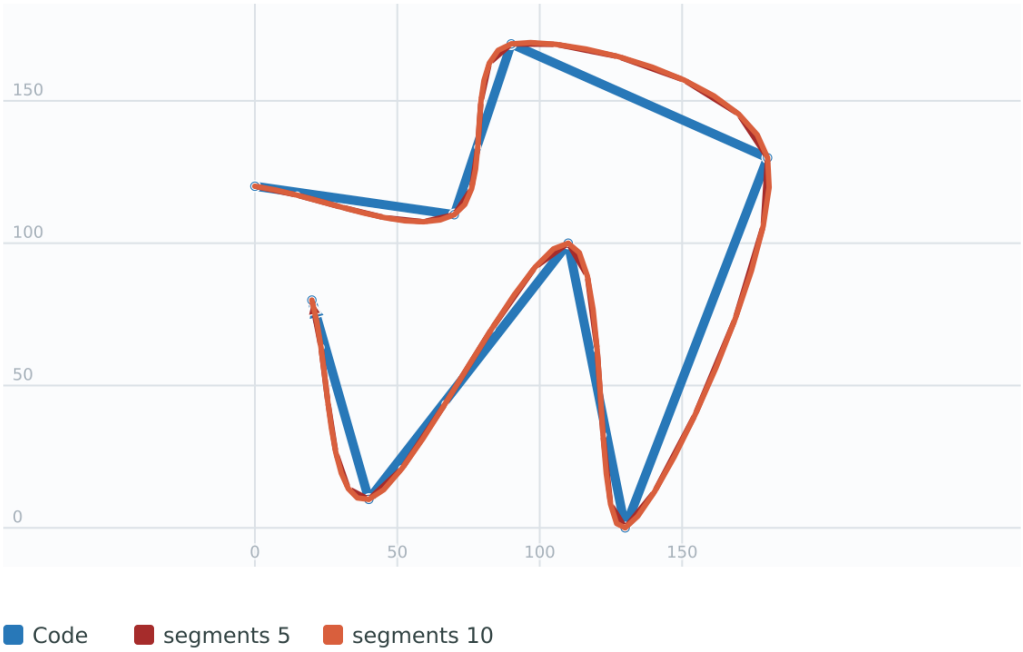
Exemples

Smoothing a LineString using nSegments = 5 and 10.

Code

```
WITH input(geom) AS (VALUES
  ('LINESTRING (10 140,80 130,100 190,190 150,140 20,120 120,50 30,30 100)'::geometry)
)
SELECT ST_CatmullRomSmoothing(geom, 5) AS segments_5,
  ST_CatmullRomSmoothing(geom, 10) AS segments_10
FROM input;
```

Figure



Smoothing a 4-point collinear line with default nSegments=5 shows the minimal change case.

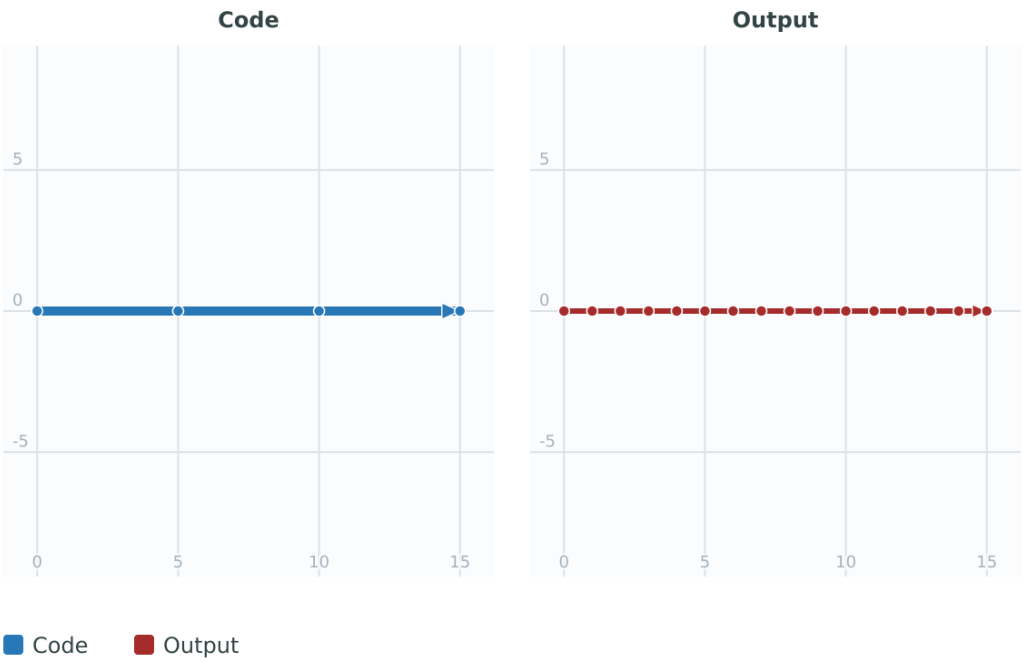
Code

```
SELECT ST_CatmullRomSmoothing('LINESTRING(0 0,5 0,10 0,15 0)');
```

Output

```
LINESTRING(0 0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 0,10 0,11 0,12 0,13 0,14 0,15 0)
```

Figure

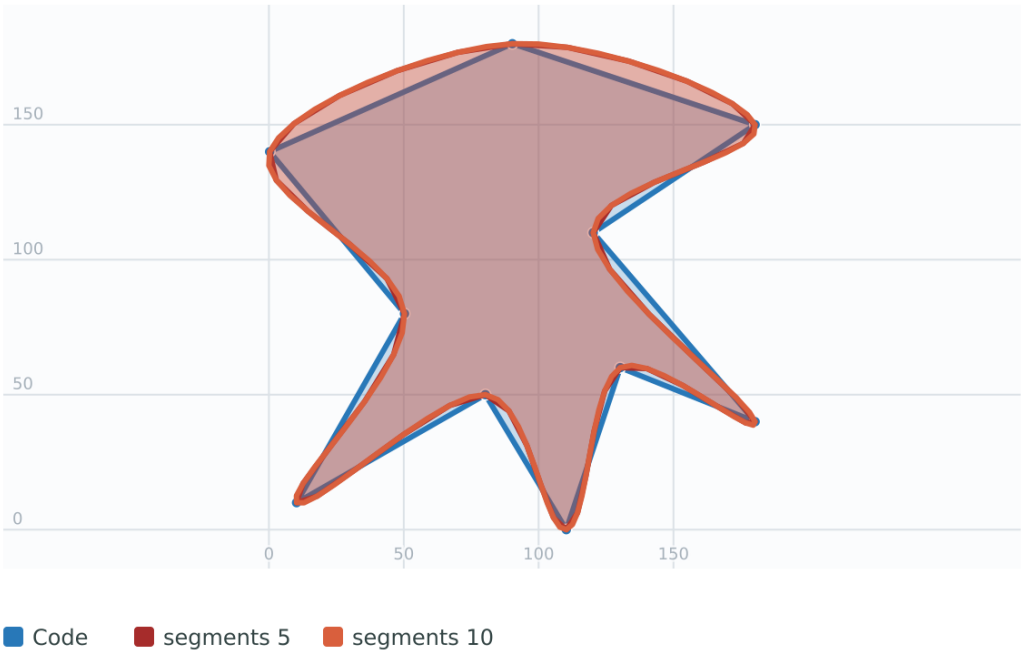


Smoothing a Polygon using nSegments = 5 and 10. This larger example is kept last.

Code

```
WITH input(geom) AS (VALUES
  ('POLYGON ((20 20,60 90,10 150,100 190,190 160,130 120,190 50,140 70,120 10,90 60,20 20))' ↩
    ::geometry)
)
SELECT ST_CatmullRomSmoothing(geom, 5) AS segments_5,
  ST_CatmullRomSmoothing(geom, 10) AS segments_10
FROM input;
```

Figure



Voir aussi

ST_ChaikinSmoothing, ST_Simplify, ST_SimplifyPreserveTopology, ST_SimplifyVW

7.14.6 ST_ConcaveHull

ST_ConcaveHull — Calcule une géométrie éventuellement concave qui contient tous les sommets de la géométrie d'entrée

Synopsis

geometry **ST_ConcaveHull**(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);

Description

Une enveloppe concave est une géométrie (généralement) concave qui contient l'entrée et dont les sommets sont un sous-ensemble des sommets de l'entrée. Dans le cas général, l'enveloppe concave est un polygone. L'enveloppe concave de deux ou plusieurs points colinéaires est une ligne à deux points. L'enveloppe concave d'un ou plusieurs points identiques est un Point. Le polygone ne contiendra pas de trous à moins que l'argument facultatif `param_allow_holes` ne soit spécifié comme vrai.

On peut considérer qu'une enveloppe concave "rétrécit" un ensemble de points. Cela diffère de la convex hull, qui revient à enrouler un élastique autour des points. Une enveloppe concave a généralement une surface plus petite et représente une limite plus naturelle pour les points d'entrée.

La `param_pctconvex` contrôle la concavité de l'enveloppe calculée. Une valeur de 1 produit une enveloppe convexe. Les valeurs comprises entre 1 et 0 produisent des enveloppes de concavité croissante. Une valeur de 0 produit une enveloppe avec une concavité maximale (mais toujours un polygone unique). Le choix d'une valeur appropriée dépend de la nature des données d'entrée, mais des valeurs comprises entre 0,3 et 0,1 produisent souvent des résultats raisonnables.



Note
Techniquement, la `param_pctconvex` détermine une longueur comme une fraction de la différence entre les arêtes les plus longues et les plus courtes dans la triangulation de Delaunay des points d'entrée. Les arêtes plus longues que cette longueur sont "érodées" de la triangulation. Les triangles restants forment l'enveloppe concave.

Pour les entrées ponctuelles et linéaires, l'enveloppe englobe tous les points de l'entrée. Pour les entrées polygonales, l'enveloppe englobera tous les points de l'entrée *et aussi* toutes les zones couvertes par l'entrée. Si vous souhaitez obtenir une enveloppe ponctuelle d'une entrée polygonale, convertissez-la d'abord en points à l'aide de ST_Points.

This is not an aggregate function. To compute the concave hull of a set of geometries use ST_Collect (e.g. `ST_ConcaveHull(ST_Col ), 0.80)`.

Disponibilité : 2.0.0

Amélioré : 3.3.0, implémentation native de GEOS activée pour GEOS 3.11+

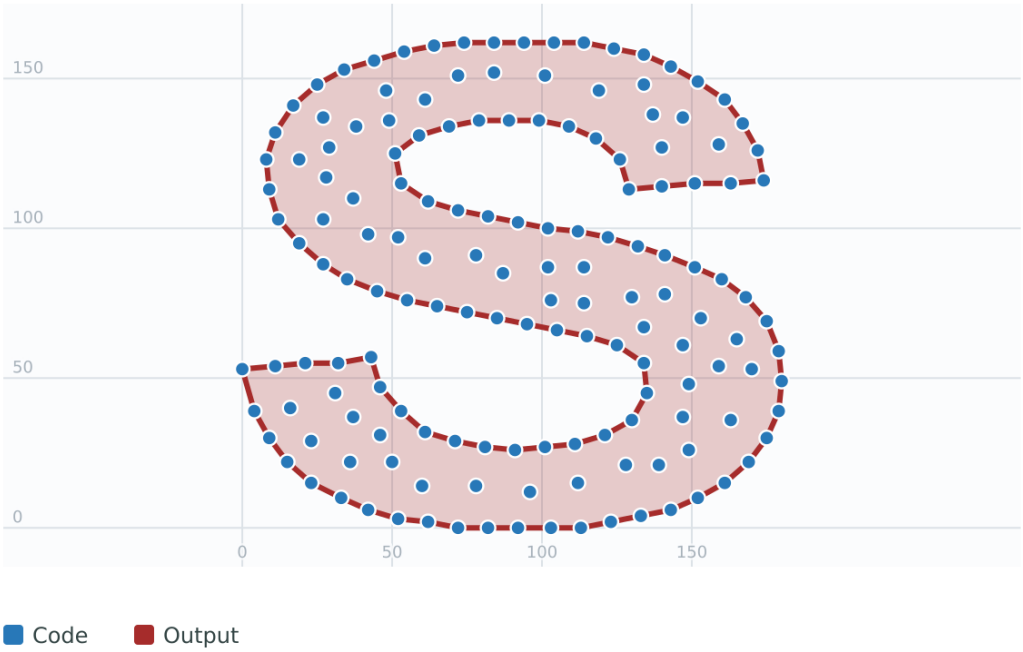
Exemples

Concave hull of a MultiPoint.

Code

```
SELECT ST_ConcaveHull('MULTIPOINT ((10 72),(53 76),(56 66),(63 58),(71 51),(81 48),(91 46) ↩
    ,(101 45),(111 46),(121 47),(131 50),(140 55),(145 64),(144 74),(135 80),(125 83),(115  ↩
    85),(105 87),(95 89),(85 91),(75 93),(65 95),(55 98),(45 102),(37 107),(29 114),(22 122) ↩
    ,(19 132),(18 142),(21 151),(27 160),(35 167),(44 172),(54 175),(64 178),(74 180),(84  ↩
    181),(94 181),(104 181),(114 181),(124 181),(134 179),(144 177),(153 173),(162 168),(171 ↩
     162),(177 154),(182 145),(184 135),(139 132),(136 142),(128 149),(119 153),(109 155) ↩
    ,(99 155),(89 155),(79 153),(69 150),(61 144),(63 134),(72 128),(82 125),(92 123),(102  ↩
    121),(112 119),(122 118),(132 116),(142 113),(151 110),(161 106),(170 102),(178 96),(185 ↩
     88),(189 78),(190 68),(189 58),(185 49),(179 41),(171 34),(162 29),(153 25),(143 23) ↩
    ,(133 21),(123 19),(113 19),(102 19),(92 19),(82 19),(72 21),(62 22),(52 25),(43 29),(33 ↩
     34),(25 41),(19 49),(14 58),(21 73),(31 74),(42 74),(173 134),(161 134),(150 133),(97  ↩
    104),(52 117),(157 156),(94 171),(112 106),(169 73),(58 165),(149 40),(70 33),(147 157) ↩
    ,(48 153),(140 96),(47 129),(173 55),(144 86),(159 67),(150 146),(38 136),(111 170),(124 ↩
     94),(26 59),(60 41),(71 162),(41 64),(88 110),(122 34),(151 97),(157 56),(39 146),(88  ↩
    33),(159 45),(47 56),(138 40),(129 165),(33 48),(106 31),(169 147),(37 122),(71 109) ↩
    ,(163 89),(37 156),(82 170),(180 72),(29 142),(46 41),(59 155),(124 106),(157 80),(175  ↩
    82),(56 50),(62 116),(113 95),(144 167))',
         0.1 );
```

Figure

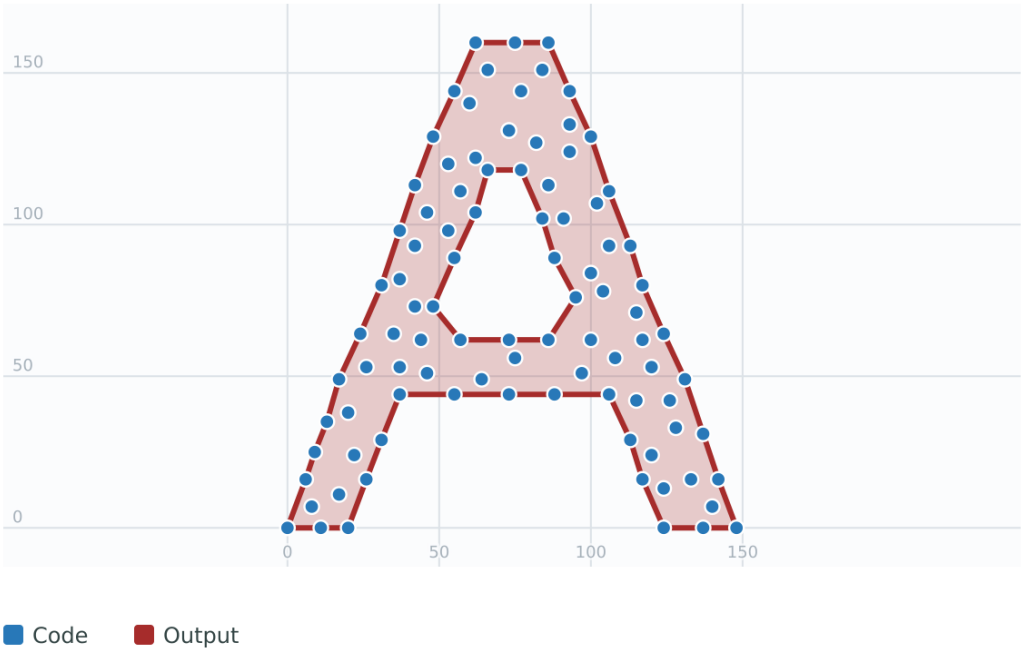


Concave hull of a MultiPoint, allowing holes.

Code

```
SELECT ST_ConcaveHull('MULTIPOINT ((132 64),(114 64),(99 64),(81 64),(63 64),(57 49),(52 36) ↩
    ,(46 20),(37 20),(26 20),(32 36),(39 55),(43 69),(50 84),(57 100),(63 118),(68 133),(74  ↩
    149),(81 164),(88 180),(101 180),(112 180),(119 164),(126 149),(132 131),(139 113),(143  ↩
    100),(150 84),(157 69),(163 51),(168 36),(174 20),(163 20),(150 20),(143 36),(139 49) ↩
    ,(132 64),(99 151),(92 138),(88 124),(81 109),(74 93),(70 82),(83 82),(99 82),(112 82) ↩
    ,(126 82),(121 96),(114 109),(110 122),(103 138),(99 151),(34 27),(43 31),(48 44),(46  ↩
    58),(52 73),(63 73),(61 84),(72 71),(90 69),(101 76),(123 71),(141 62),(166 27),(150 33) ↩
    ,(159 36),(146 44),(154 53),(152 62),(146 73),(134 76),(143 82),(141 91),(130 98),(126  ↩
    104),(132 113),(128 127),(117 122),(112 133),(119 144),(108 147),(119 153),(110 171) ↩
    ,(103 164),(92 171),(86 160),(88 142),(79 140),(72 124),(83 131),(79 118),(68 113),(63  ↩
    102),(68 93),(35 45))',
         0.15, true );
```

Figure



Comparaison entre l'enveloppe concave d'un polygone et l'enveloppe concave des points qui le composent. L'enveloppe respecte les limites du polygone, ce qui n'est pas le cas de l'enveloppe basée sur les points.

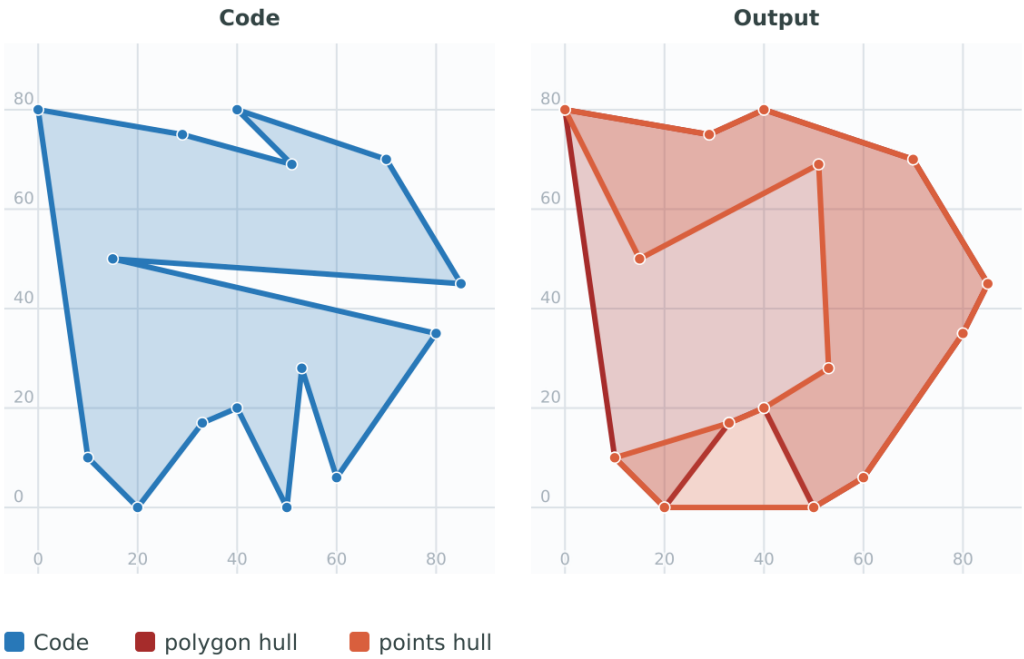
Code

```
WITH data(geom) AS (VALUES
    ('POLYGON ((10 90,39 85,61 79,50 90,80 80,95 55,25 60,90 45,70 16,63 38,60 10,50 30,43 ↩
     27,30 10,20 20,10 90))'::geometry)
)
SELECT ST_ConcaveHull(geom, 0.1) AS polygon_hull,
       ST_ConcaveHull(ST_Points(geom), 0.1) AS points_hull
     FROM data;
```

Output

```
POLYGON((60 10,50 30,43 27,30 10,20 20,10 90,39 85,50 90,80 80,95 55,90 45,70 16,60 10)) | POLYGON((25 60,10 90,39 ↩
     85,50 90,80 80,95 55,90 45,70 16,60 10,30 10,20 20,43 27,50 30,63 38,61 79,25 60))
```

Figure



Utilisation avec ST_Collect pour calculer l'enveloppe concave d'un ensemble géométrique.

This example estimates an infected area from point observations.

Code

```
SELECT disease_type,
    ST_ConcaveHull(ST_Collect(obs_pnt), 0.3 ) AS geom
  FROM disease_obs
  GROUP BY disease_type;
```

Voir aussi

ST_ConvexHull, ST_Collect, CG_AlphaShape, CG_OptimalAlphaShape

7.14.7 ST_ConvexHull

ST_ConvexHull — Calcule l'enveloppe convexe d'une géométrie.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Description

Calcule l'enveloppe convexe d'une géométrie. L'enveloppe convexe est la plus petite géométrie convexe qui englobe toutes les géométries de l'entrée.

On peut considérer l'enveloppe convexe comme la géométrie obtenue en enroulant un élastique autour d'un ensemble de géométries. Ceci est différent d'une concave hull qui est analogue à un "rétrécissement" des géométries. Une enveloppe convexe est souvent utilisée pour déterminer une zone affectée sur la base d'un ensemble d'observations ponctuelles.

Dans le cas général, l'enveloppe convexe est un polygone. L'enveloppe convexe de deux ou plusieurs points colinéaires est une LineString à deux points. L'enveloppe convexe d'un ou plusieurs points identiques est un Point.

Il ne s'agit pas d'une fonction d'agrégation. Pour calculer l'enveloppe convexe d'un ensemble de géométries, utilisez ST_Collect pour les agréger dans une collection de géométries (par exemple `ST_ConvexHull(ST_Collect(geom))`.

Effectué par le module GEOS

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1.16

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

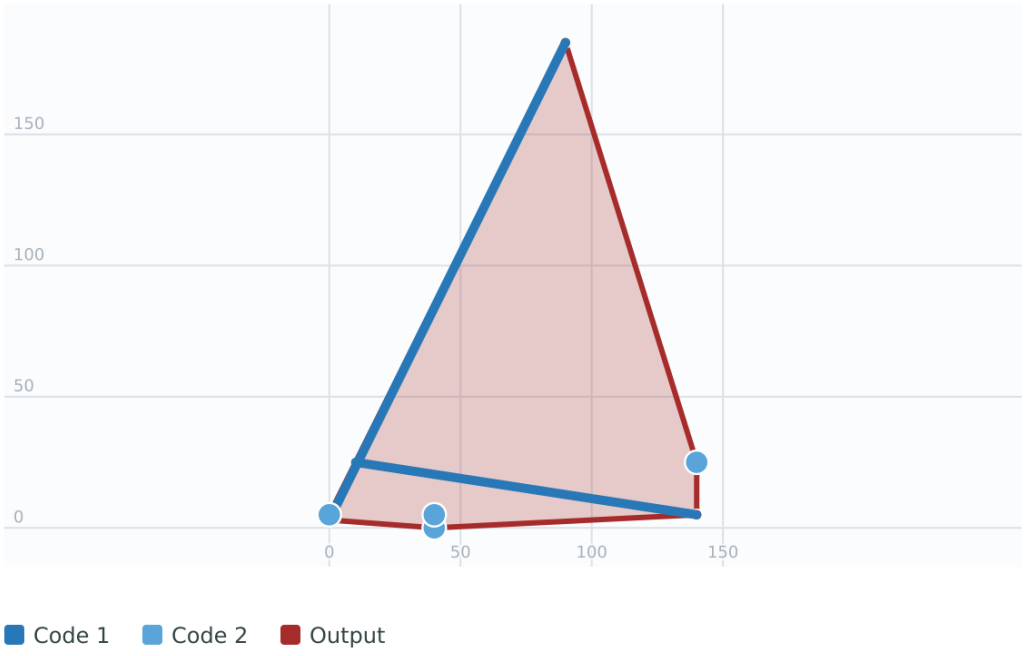
Exemples

Compute the convex hull of a MultiLineString and a MultiPoint.

Code

```
SELECT ST_ConvexHull(ST_Collect(ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10,20 ↩
    30))'),
            ST_GeomFromText('MULTIPOINT(50 5,150 30,50 10,10 10)')
            ));
```

Figure



Utilisation avec ST_Collect pour calculer les enveloppes convexes des ensembles géométriques.

This example returns an estimated infected area from point observations.

Code

```
SELECT d.disease_type,
    ST_ConvexHull(ST_Collect(d.geom)) As geom
    FROM disease_obs As d
    GROUP BY d.disease_type;
```

Voir aussi

ST_Collect, ST_ConcaveHull, ST_MinimumBoundingCircle

7.14.8 ST_DelaunayTriangles

ST_DelaunayTriangles — Renvoie la triangulation de Delaunay des sommets d'une géométrie.

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

Description

Calcule la Delaunay triangulation des sommets de la géométrie d'entrée. La `tolerance` optionnelle peut être utilisée pour regrouper les sommets proches, ce qui améliore la robustesse dans certaines situations. La géométrie résultante est délimitée par l'enveloppe convexe des sommets de la géométrie d'entrée. La représentation de la géométrie résultante est déterminée par le code `flags` :

— `0` - une collection de géométries de POLYGONES triangulaires (par défaut)
— `1` - une MULTILINESTRING des arêtes de la triangulation
— `2` - Un TIN de la triangulation

Effectué par le module GEOS.

Disponibilité : 2.1.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

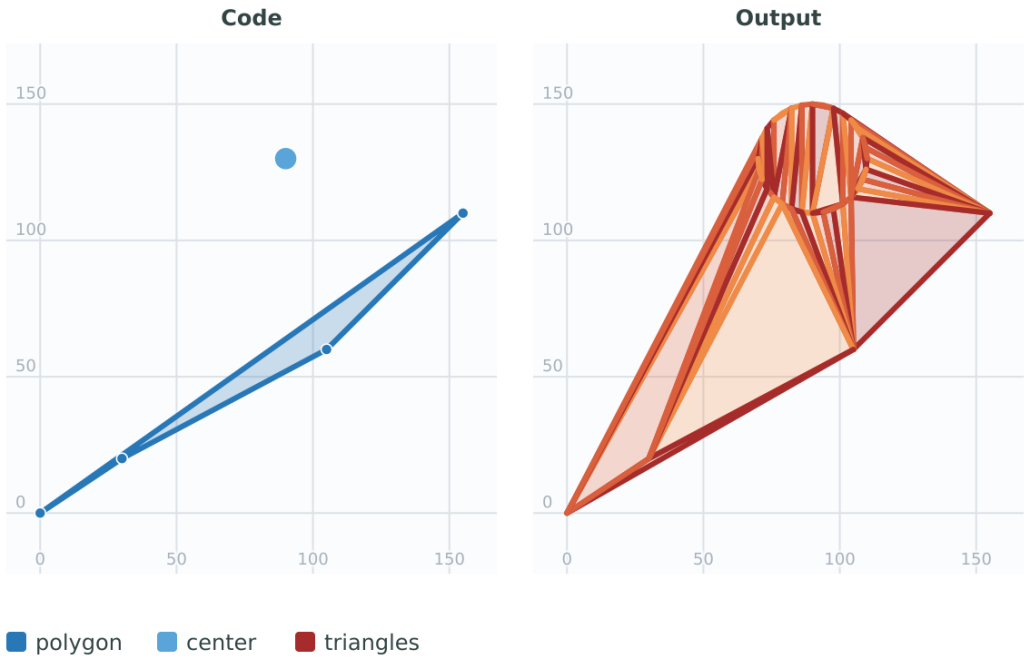
Exemples

Triangulation of the vertices of a polygon and a buffered point.

Code

```
WITH data AS (
  SELECT polygon, ST_Buffer(center, 20) AS point_buffer
  FROM (SELECT
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry AS polygon,
    'POINT(110 170)'::geometry AS center
  ) AS input
)
SELECT ST_DelaunayTriangles(ST_Union(polygon, point_buffer)) AS triangles
FROM data;
```

Figure

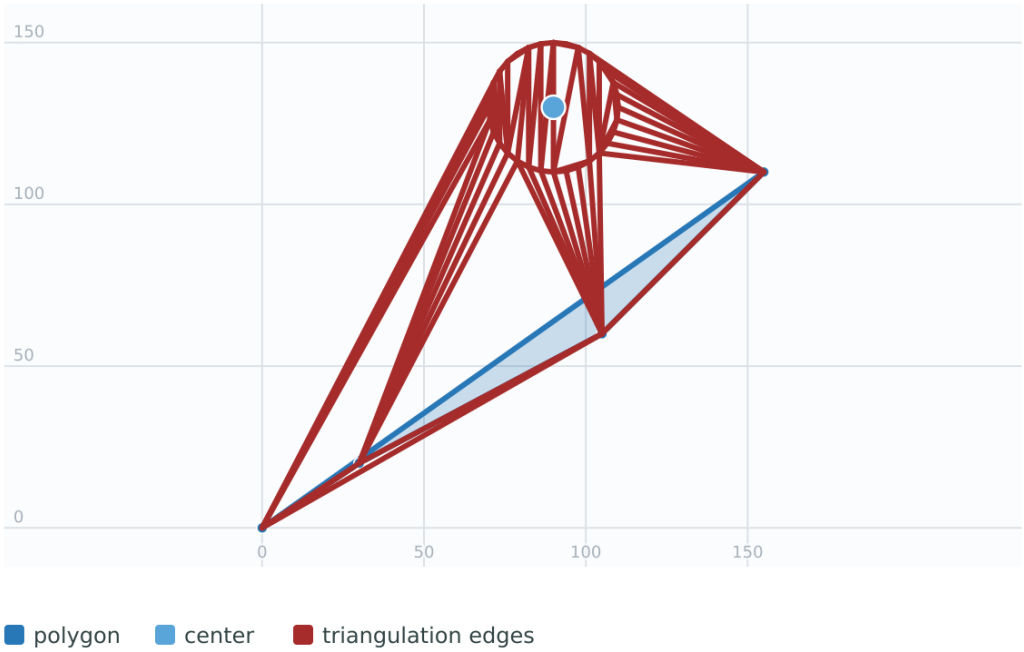


Triangulation edges returned as a `MULTILINESTRING`.

Code

```
WITH data AS (
  SELECT polygon, ST_Buffer(center, 20) AS point_buffer
  FROM (SELECT
    'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry AS polygon,
    'POINT(110 170)'::geometry AS center
  ) AS input
)
SELECT ST_DelaunayTriangles(ST_Union(polygon, point_buffer), 0.001, 1) AS ↩
    triangulation_edges
FROM data;
```

Figure

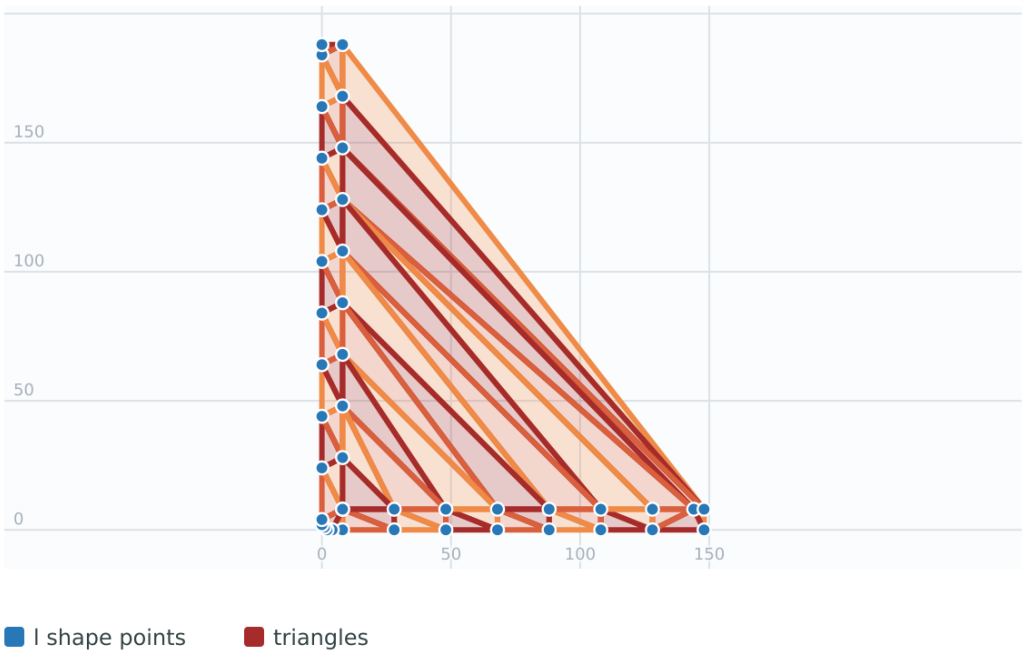


Triangulation of 42 points forming an L shape.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,
    150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,
    14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,
    6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,
    14 94,14 74,14 54,14 34,14 14)'::geometry AS l_shape_points
)
SELECT ST_DelaunayTriangles(l_shape_points) AS triangles
FROM data;
```

Figure



Exemple utilisant des sommets avec des valeurs Z.

Code

```
SELECT ST_DelaunayTriangles(
  'MULTIPOINT Z(14 14 10,150 14 100,34 6 25,20 10 150)'::geometry);
```

Voir aussi

ST_VoronoiPolygons, ST_TriangulatePolygon, CG_ConstrainedDelaunayTriangles, ST_VoronoiLines, ST_ConvexHull

7.14.9 ST_FilterByM

ST_FilterByM — Supprime les vertices en fonction de leur valeur M

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

Description

Filtre les points de sommet en fonction de leur valeur M. Renvoie une géométrie contenant uniquement les points de sommet dont la valeur M est supérieure ou égale à la valeur min et inférieure ou égale à la valeur max. Si l'argument max-value est omis, seule la valeur min est prise en compte. Si le quatrième argument est omis, la valeur m ne figurera pas dans la géométrie résultante. Si la géométrie résultante a trop peu de points de vertex pour son type de géométrie, une géométrie vide sera renvoyée. Dans une collection de géométries, les géométries qui n'ont pas assez de points seront simplement éliminées silencieusement.

Cette fonction est principalement destinée à être utilisée en conjonction avec ST_SetEffectiveArea. ST_EffectiveArea définit la surface effective d'un sommet dans sa valeur m. Avec ST_FilterByM, il est possible d'obtenir une version simplifiée de la géométrie sans aucun calcul, simplement en filtrant



Note
Il existe une différence entre ST_SimplifyVW et ST_FilterByM en ce qui concerne la géométrie renvoyée lorsque le nombre de points répondant aux critères n'est pas suffisant. ST_SimplifyVW renvoie la géométrie avec suffisamment de points alors que ST_FilterByM renvoie une géométrie vide



Note
Notez que la géométrie renvoyée peut être invalide



Note
Cette fonction renvoie toutes les dimensions, y compris les valeurs Z et M

Disponibilité : 2.5.0

Exemples

Une ligne est filtrée

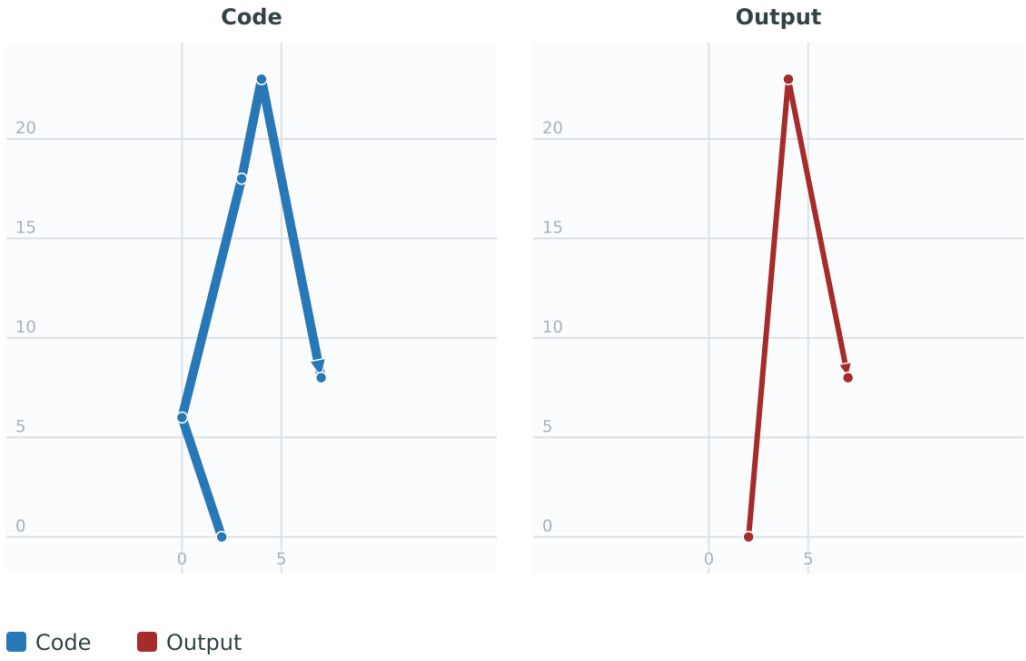
Code

```
SELECT ST_FilterByM(geom, 30) simplified
FROM (SELECT  ST_SetEffectiveArea('LINESTRING(5 2,3 8,6 20,7 25,10 10)'::geometry) geom) As ↩
    foo;
```

Output

```
LINESTRING(5 2,7 25,10 10)
```

Figure



Voir aussi

ST_SetEffectiveArea, ST_SimplifyVW

7.14.10 ST_GeneratePoints

ST_GeneratePoints — Génère un multipoint de points aléatoires contenus dans un polygone ou un multipolygone.

Synopsis

geometry **ST_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

Description

ST_GeneratePoints génère un multipoint composé d'un nombre donné de points pseudo-aléatoires situés dans la zone d'entrée. Le paramètre facultatif `seed` est utilisé pour régénérer une séquence déterministe de points et doit être supérieur à zéro.

Disponibilité : 2.3.0

Amélioration : 3.0.0, ajout du paramètre seed

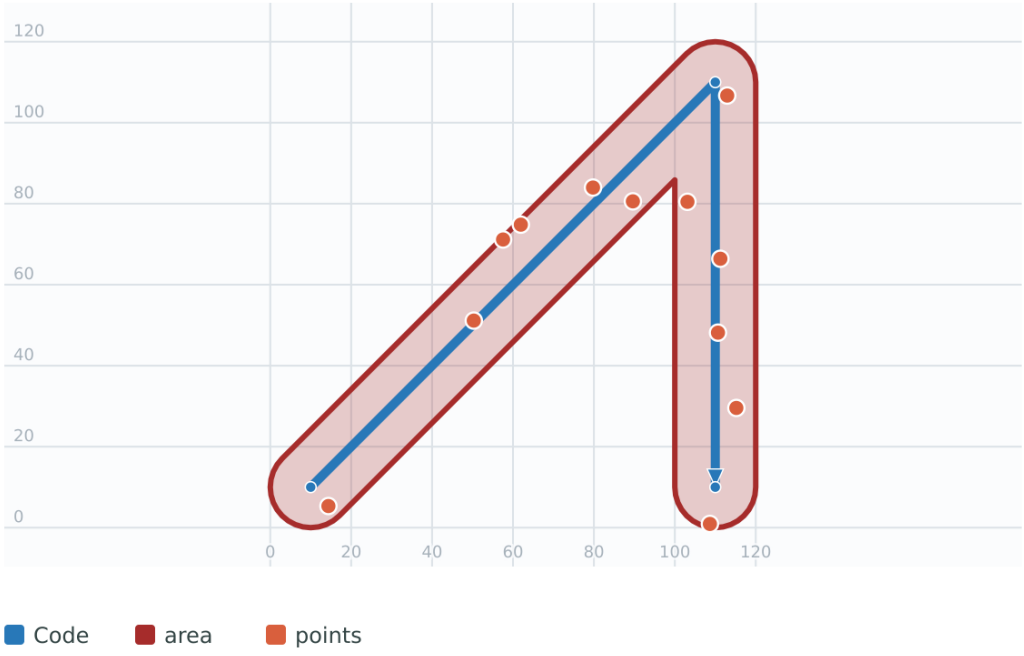
Exemples

Generate 12 points inside a buffered LineString using the deterministic seed 1996.

Code

```
WITH source AS (
  SELECT ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),
           10, 'endcap=round join=round') AS geom
)
SELECT geom AS area,
       ST_GeneratePoints(geom, 12, 1996) AS points
FROM source;
```

Figure



Etant donné un tableau de polygones s, retourner 12 points individuels par polygone. Les résultats seront différents à chaque exécution.

Code

```
SELECT s.id, dp.path[1] AS pt_id, dp.geom
FROM s, ST_DumpPoints(ST_GeneratePoints(s.geom, 12)) AS dp;
```

Generate dot-density points from a polygon table, using the absolute value of an attribute as the number of points and keeping the sign of the attribute for styling.

Code

```
SELECT z.id,
       CASE WHEN z.count_delta < 0 THEN -1 ELSE 1 END AS sign,
       dp.path[1] AS pt_id,
       dp.geom
FROM zones AS z
CROSS JOIN LATERAL ST_DumpPoints(
    ST_GeneratePoints(z.geom, abs(z.count_delta)::integer, z.id + 1)
) AS dp
WHERE z.count_delta <> 0;
```

Voir aussi

ST_DumpPoints

7.14.11 ST_GeometricMedian

ST_GeometricMedian — Renvoie la médiane géométrique d'un MultiPoint.

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

Description

Calcule la médiane géométrique approximative d'une géométrie MultiPoint à l'aide de l'algorithme de Weiszfeld. La médiane géométrique est le point qui minimise la somme des distances aux points d'entrée. Elle fournit une mesure de centralité moins sensible aux points aberrants que le centroïde (centre de masse).

L'algorithme itère jusqu'à ce que la variation de la distance entre les itérations successives soit inférieure au paramètre `tolerance` fourni. Si cette condition n'est pas remplie après `max_iterations` itérations, la fonction produit une erreur et se termine, à moins que `fail_fif_not_converged` ne soit fixé à `false` (par défaut).

Si l'argument `tolerance` n'est pas fourni, la valeur de la tolérance est calculée sur la base de l'étendue de la géométrie d'entrée.

Si elle est présente, les valeurs M des points d'entrée sont interprétées comme leurs poids relatifs.

Disponibilité : 2.3.0

Amélioré : 2.5.0 Ajout de la prise en charge de M comme poids des points.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

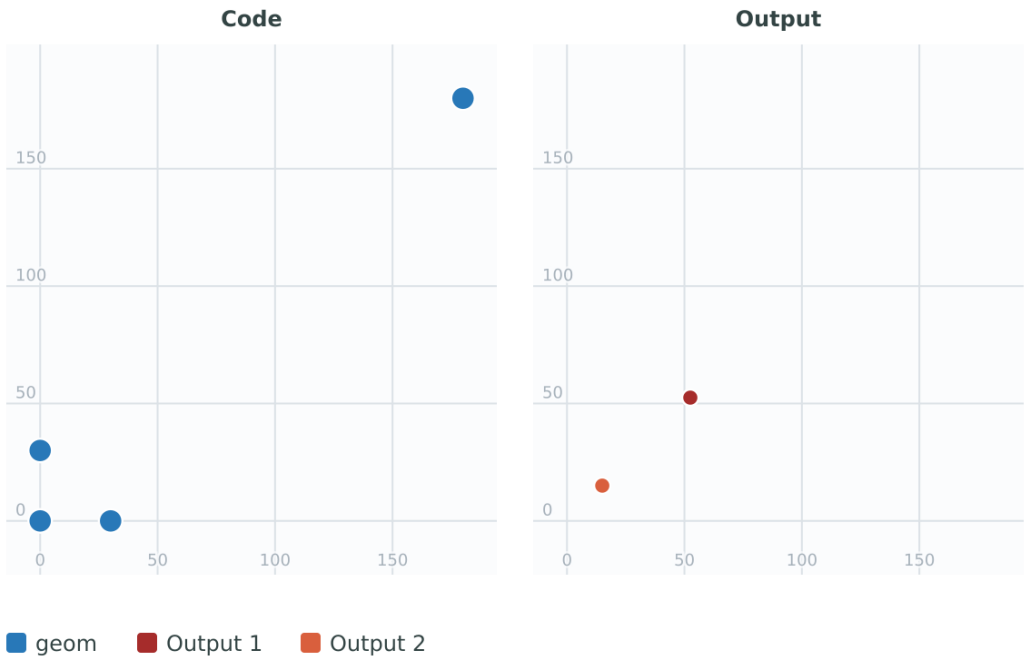
Exemples

Compare the geometric median and centroid of a MultiPoint.

Code

```
WITH test AS (
SELECT 'MULTIPOINT((10 10),(10 40),(40 10),(190 190))'::geometry geom)
SELECT
  ST_Centroid(geom) centroid,
  ST_GeometricMedian(geom) median
FROM test;
```

Figure



Voir aussi

ST_Centroid

7.14.12 ST_LineMerge

ST_LineMerge — Renvoie les lignes formées par la couture d'une MultiLineString.

Synopsis

geometry **ST_LineMerge**(geometry amultilinestring);
geometry **ST_LineMerge**(geometry amultilinestring, boolean directed);

Description

Renvoie une LineString ou une MultiLineString formée par l'assemblage des éléments de ligne d'une MultiLineString. Les lignes sont jointes à leurs extrémités aux intersections à 2 voies. Les lignes ne sont pas jointes aux intersections à 3 voies ou plus.

Si **directed** est TRUE, ST_LineMerge ne modifiera pas l'ordre des points dans les LineStrings, de sorte que les lignes ayant des directions opposées ne seront pas fusionnées



Note
A n'utiliser qu'avec les MultiLineString/LineStrings. Les autres types de géométrie renvoient une GeometryCollection vide

Effectué par le module GEOS.

Amélioration : 3.3.0 accepte un paramètre direct.

GEOS >= 3.11.0 est nécessaire pour utiliser le paramètre direct.

Disponibilité : 1.1.0



AVERTISSEMENT
Cette fonction supprime la dimension M.

Exemples

Cardinality 2

Nodes with cardinality 2 are merged away.

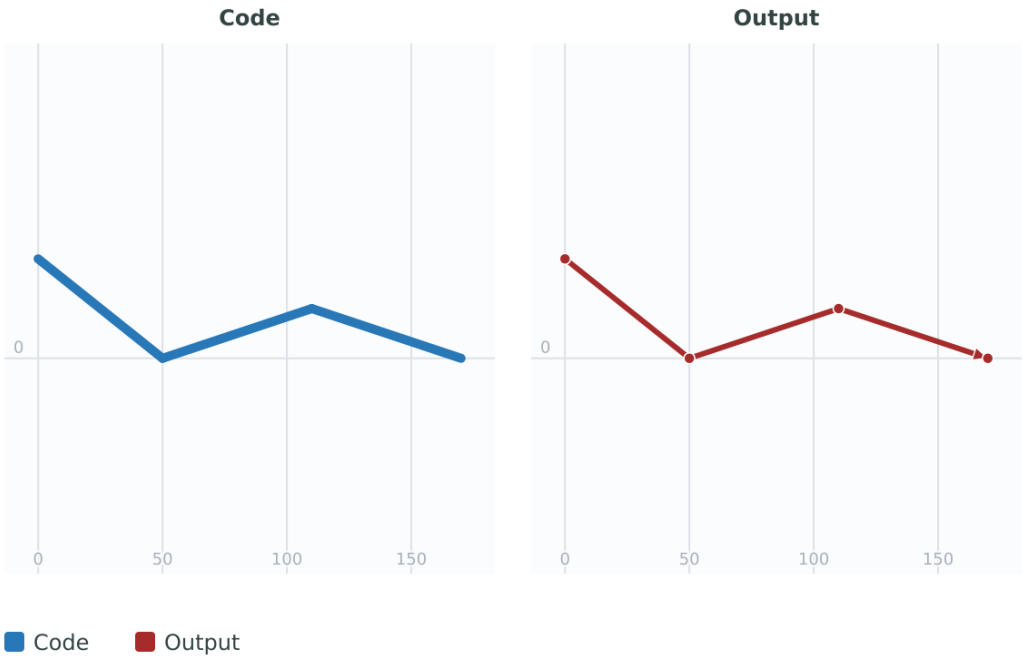
Code

```
SELECT ST_LineMerge('MULTILINESTRING((10 160,60 120),(120 140,60 120),(120 140,180 120))'
);
```

Output

```
LINESTRING(10 160,60 120,120 140,180 120)
```

Figure



Cardinality 3 or More

Les lignes ne sont pas fusionnées aux intersections de degré > 2.

Code

```
SELECT ST_LineMerge('MULTILINESTRING((10 160,60 120),(120 140,60 120),(120 140,180 120),(100 ↩
    180,120 140))'
);
```

Output

```
MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

Figure



Non-Touching Lines

Si la fusion n'est pas possible en raison de lignes qui ne se touchent pas, la MultiLineString d'origine est renvoyée.

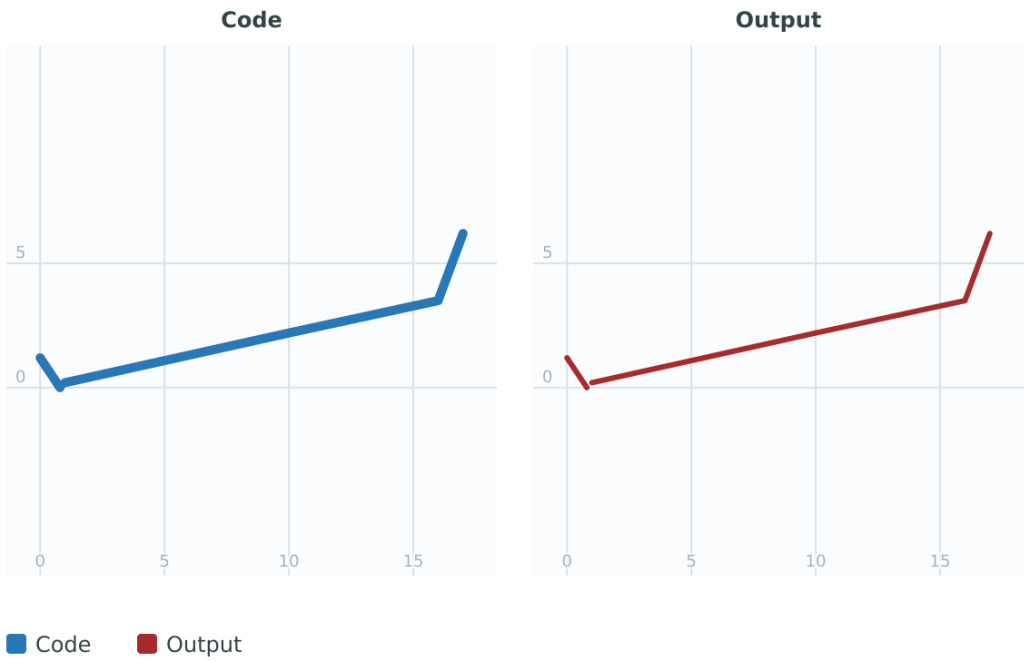
Code

```
SELECT ST_LineMerge('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 ↩
    -32))'
);
```

Output

```
MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))
```

Figure



Directed Parameter

Lines with opposite directions are not merged if *directed* is TRUE.

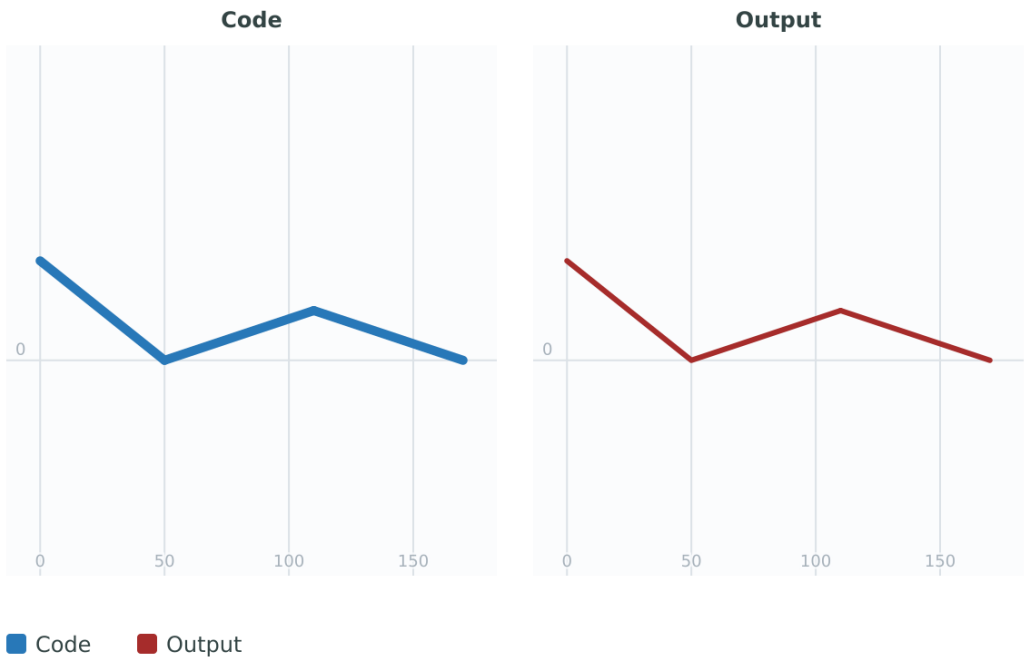
Code

```
SELECT ST_LineMerge('MULTILINESTRING((60 30,10 70),(120 50,60 30),(120 50,180 30))',
TRUE);
```

Output

```
MULTILINESTRING((120 50,60 30,10 70),(120 50,180 30))
```

Figure



Z-dimension Handling

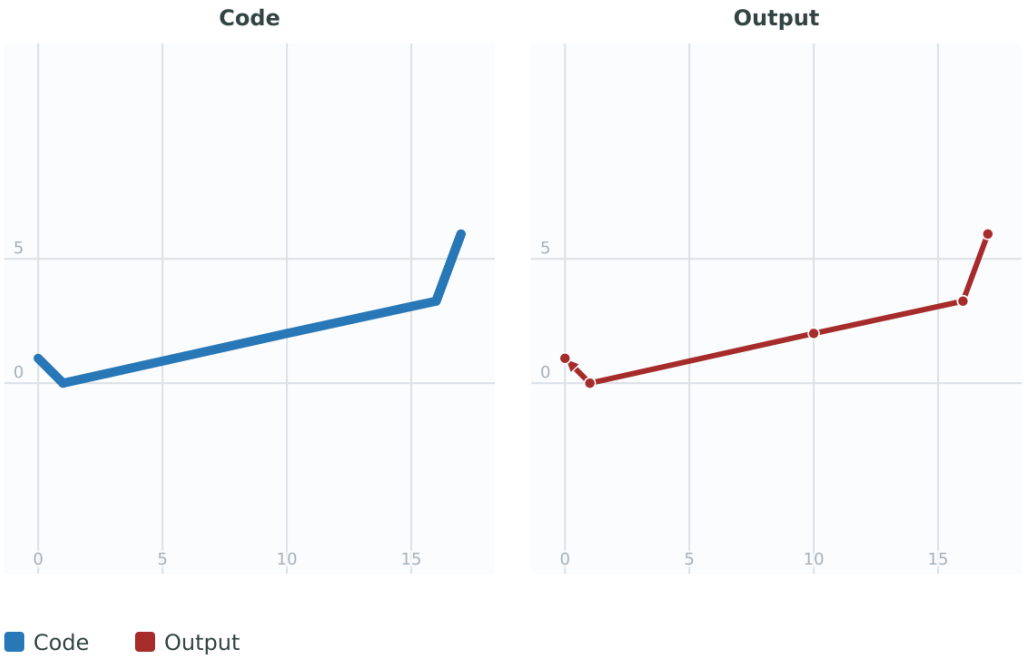
Code

```
SELECT ST_LineMerge('MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6),(-29 -27 ↩
    12,-30 -29.7 5),(-45 -33 1,-46 -32 11))'
        );
```

Output

```
LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

Figure



Voir aussi

ST_Segmentize, ST_LineSubstring

7.14.13 ST_MaximumInscribedCircle

ST_MaximumInscribedCircle — Calcule le plus grand cercle contenu dans une géométrie.

Synopsis

(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

Description

Trouve le plus grand cercle contenu dans un (multi)polygone, ou qui ne chevauche aucune ligne ni aucun point. Renvoie un enregistrement avec des champs :

— `center` - point central du cercle
— `nearest` - un point de la géométrie le plus proche du centre
— `radius` - rayon du cercle

Pour les entrées polygonales, le cercle est inscrit dans les anneaux de délimitation, en utilisant les anneaux internes comme limites. Pour les entrées linéaires et ponctuelles, le cercle est inscrit dans l'enveloppe convexe de l'entrée, en utilisant les lignes et les points de l'entrée comme autres limites.

Disponibilité : 3.1.0.

Nécessite GEOS >= 3.9.0.

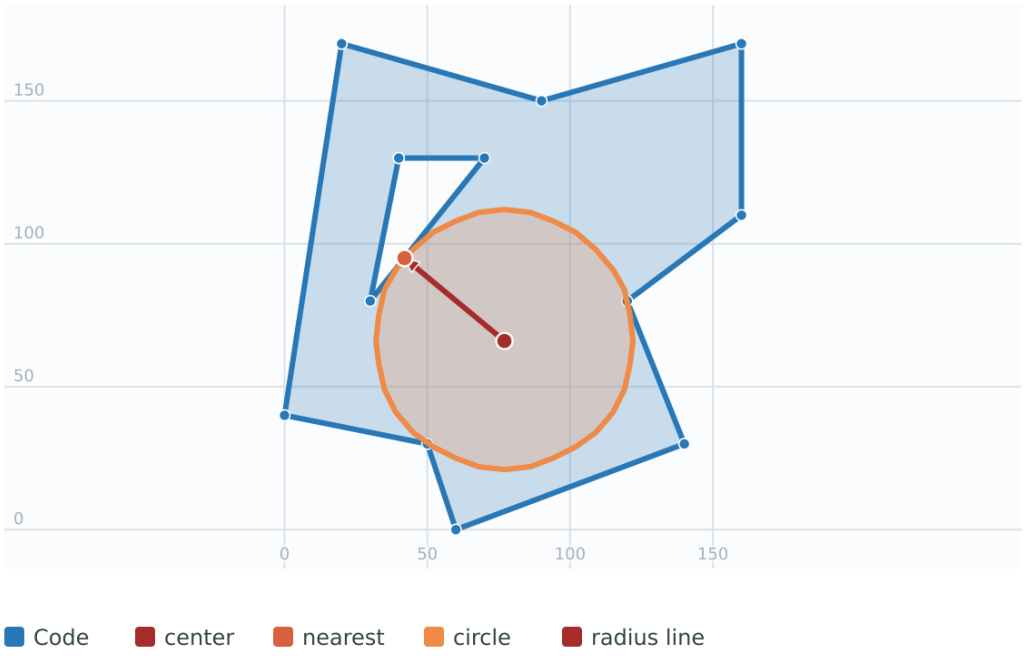
Exemples

Maximum inscribed circle of a polygon. Center, nearest point, radius, circle, and radius line are returned.

Code

```
WITH mic AS (
  SELECT * FROM ST_MaximumInscribedCircle(
    'POLYGON ((40 180,110 160,180 180,180 120,140 90,160 40,80 10,70 40,20 50,40 180),
              (60 140,50 90,90 140,60 140))')
)
SELECT round(radius::numeric, 3) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line
FROM mic;
```

Figure

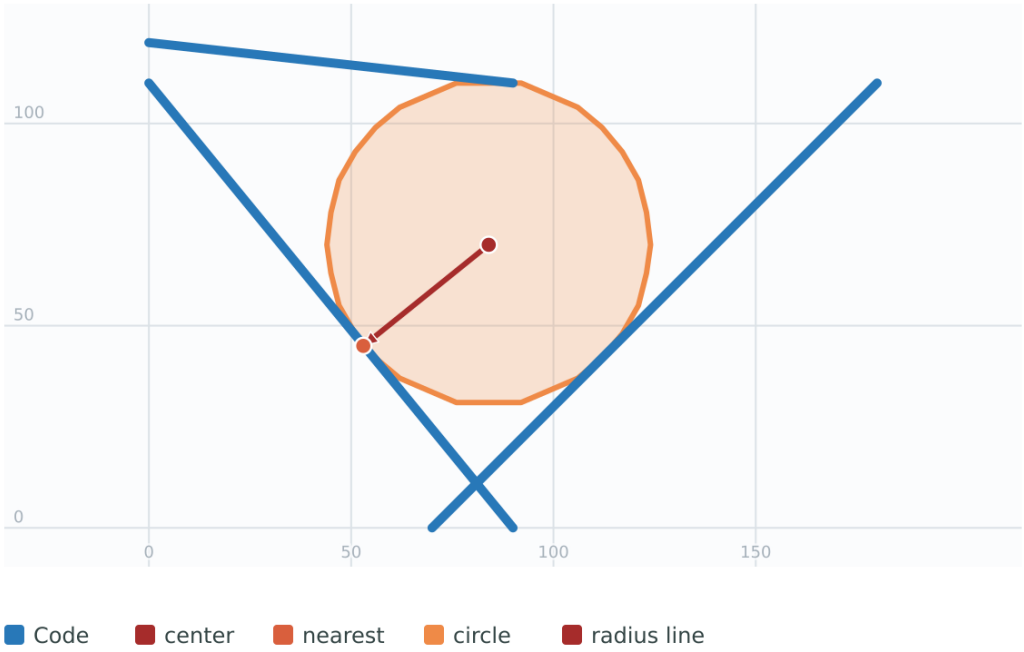


Maximum inscribed circle of a MultiLineString.

Code

```
WITH mic AS (
  SELECT * FROM ST_MaximumInscribedCircle(
    'MULTILINESTRING((100 40,10 150),(10 160,100 150),(190 150,80 40))')
)
SELECT round(radius::numeric, 3) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line
FROM mic;
```

Figure



Voir aussi

ST_MinimumBoundingRadius, ST_LargestEmptyCircle

7.14.14 ST_LargestEmptyCircle

ST_LargestEmptyCircle — Calcule le plus grand cercle ne recouvrant pas une géométrie.

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

Description

Trouve le plus grand cercle qui ne chevauche pas un ensemble d'obstacles de points et de lignes. (Les géométries polygonales peuvent être incluses en tant qu'obstacles, mais seules leurs lignes de démarcation sont utilisées). Le centre du cercle est contraint de se situer à l'intérieur d'une limite polygonale, qui par défaut est l'enveloppe convexe de la géométrie d'entrée. Le centre du cercle est le point à l'intérieur de la frontière qui est le plus éloigné des obstacles. Le cercle lui-même est fourni par le point central et un point le plus proche situé sur un obstacle déterminant le rayon du cercle.

If a boundary is supplied it must be a valid polygonal geometry with positive area.

Le centre du cercle est déterminé avec une précision donnée, spécifiée par une tolérance de distance, à l'aide d'un algorithme itératif. Si la distance de précision n'est pas spécifiée, une valeur par défaut raisonnable est utilisée.

Renvoie un enregistrement avec des champs :

— `center` - point central du cercle
— `nearest` - un point de la géométrie le plus proche du centre
— `radius` - rayon du cercle

Pour trouver le plus grand cercle vide à l'intérieur d'un polygone, voir ST_MaximumInscribedCircle.

Disponibilité : 3.4.0.

Nécessite GEOS >= 3.9.0.

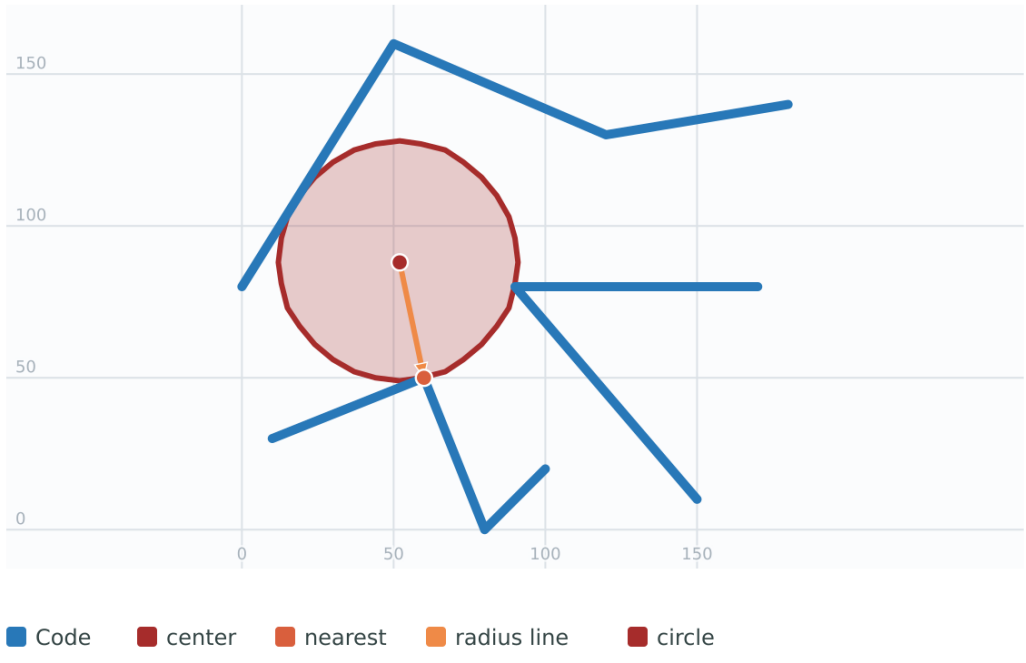
Exemples

Largest empty circle within a set of lines.

Code

```
WITH obstacles(geom) AS (VALUES ('MULTILINESTRING (
        (10 100,60 180,130 150,190 160),
        (20 50,70 70,90 20,110 40),
        (160 30,100 100,180 100))'::geometry)),
circle AS (
  SELECT result.*
  FROM obstacles
  CROSS JOIN LATERAL ST_LargestEmptyCircle(obstacles.geom) AS result
)
SELECT round(radius::numeric, 6) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle
FROM circle;
```

Figure



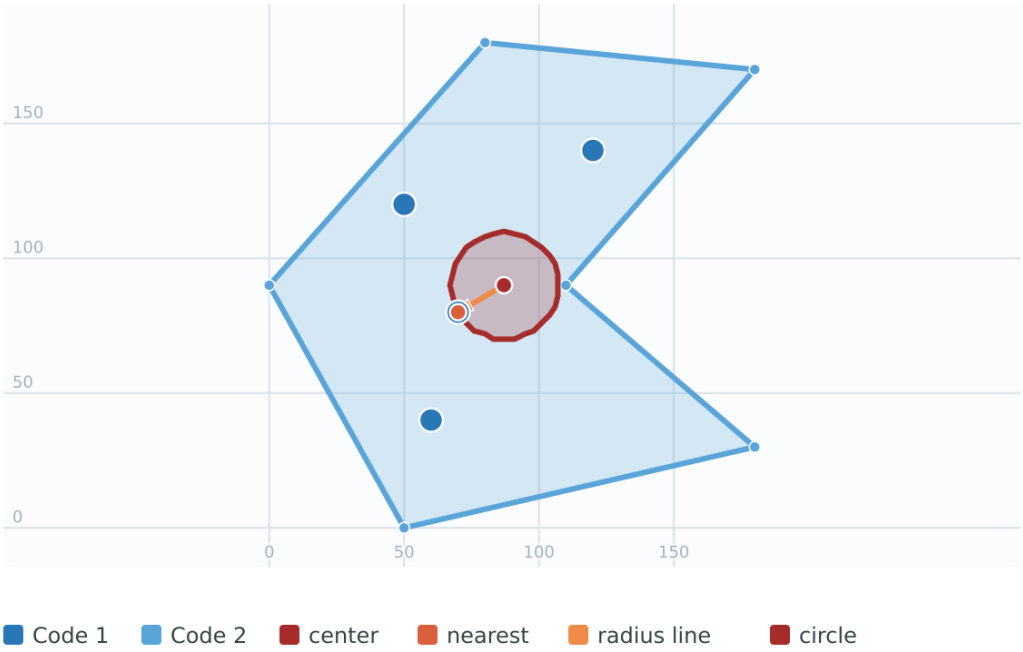
Largest empty circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary is included both as an obstacle and as the constraint for the circle center.

Code

```
WITH inputs(points, boundary) AS (VALUES (
  'MULTIPOINT ((70 50),(60 130),(130 150),(80 90))'::geometry,
  'POLYGON ((90 190,10 100,60 10,190 40,120 100,190 180,90 190))'::geometry
```

```
)),
circle AS (
  SELECT result.*
  FROM inputs
  CROSS JOIN LATERAL ST_LargestEmptyCircle(
    ST_Collect(points, boundary), 0, boundary
  ) AS result
)
SELECT round(radius::numeric, 6) AS radius,
       ST_SnapToGrid(center, 1) AS center,
       ST_SnapToGrid(nearest, 1) AS nearest,
       ST_SnapToGrid(ST_MakeLine(center, nearest), 1) AS radius_line,
       ST_SnapToGrid(ST_Buffer(center, radius), 1) AS circle
FROM circle;
```

Figure



Voir aussi

ST_MinimumBoundingRadius

7.14.15 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Retourne le plus petit cercle polygonal qui contient une géométrie.

Synopsis

geometry **ST_MinimumBoundingCircle**(geometry geomA, integer num_segs_per_qt_circ=48);

Description

Retourne le plus petit cercle polygonal qui contient une géométrie.



Note
Le cercle de délimitation est approximé par un polygone avec une valeur par défaut de 48 segments par quart de cercle. Le polygone étant une approximation du cercle de délimitation minimal, certains points de la géométrie d'entrée peuvent ne pas être contenus dans le polygone. L'approximation peut être améliorée en augmentant le nombre de segments. Pour les applications où une approximation ne convient pas, on peut utiliser ST_MinimumBoundingRadius.

A utiliser avec ST_Collect pour obtenir le cercle minimal de délimitation d'un ensemble de géométries.

Pour calculer deux points situés sur le cercle minimum (le "diamètre maximum"), utilisez ST_LongestLine.

Le rapport de la surface d'un polygone divisée par la surface de son cercle circonscrit minimal est appelé *Score de compacité de Reock*.

Effectué par le module GEOS.

Disponibilité : 1.4.0

Exemples

Code

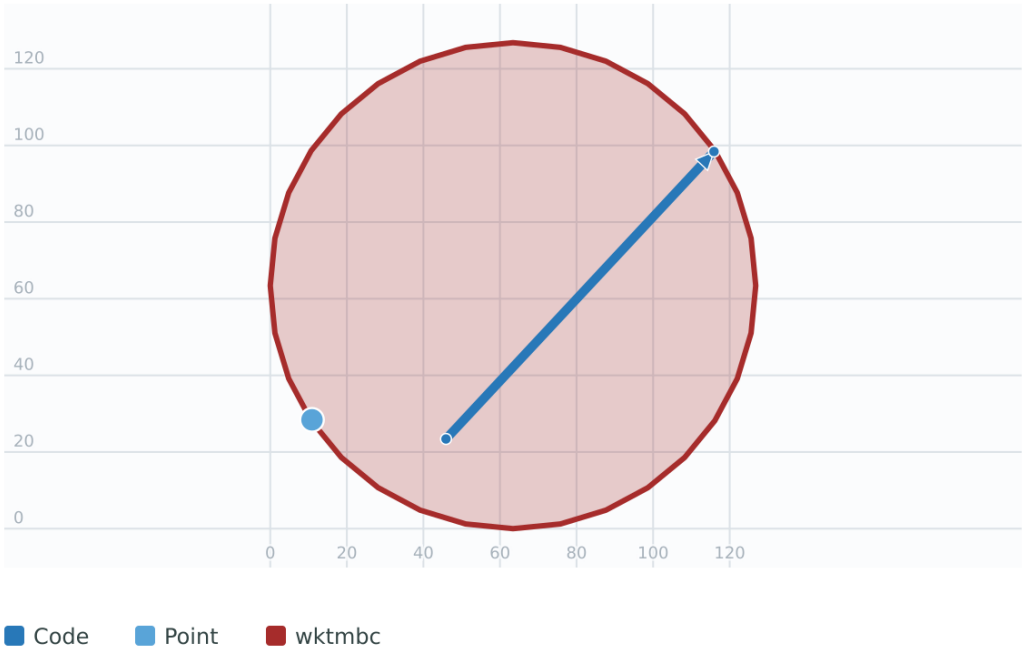
```
SELECT d.disease_type,
ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

Minimum bounding circle of a point and linestring, using 8 segments to approximate a quarter circle.

Code

```
SELECT ST_MinimumBoundingCircle(ST_Collect('LINESTRING(55 75,125 150)',
ST_Point(20, 80)), 8
) As wktmbc;
```

Figure



Voir aussi

ST_Collect, ST_MinimumBoundingRadius, ST_LargestEmptyCircle, ST_LongestLine

7.14.16 ST_MinimumBoundingRadius

ST_MinimumBoundingRadius — Renvoie le point central et le rayon du plus petit cercle contenant une géométrie.

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

Description

Calcule le point central et le rayon du plus petit cercle contenant une géométrie. Retourne un enregistrement avec des champs :
— `center` - point central du cercle
— `radius` - rayon du cercle
A utiliser avec ST_Collect pour obtenir le cercle minimal de délimitation d'un ensemble de géométries.

Pour calculer deux points situés sur le cercle minimum (le "diamètre maximum"), utilisez ST_LongestLine.

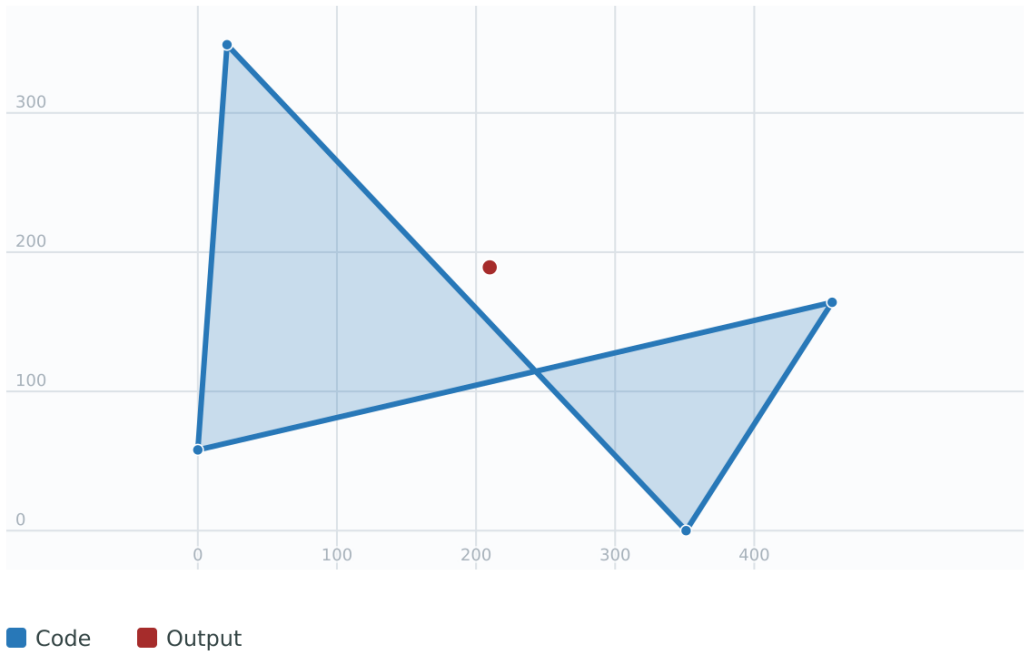
Disponibilité - 2.3.0

Exemples

Code

```
SELECT center, radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 ↩
    65136,26096 65427,26426 65078))');
```

Figure



Voir aussi

ST_Collect, ST_MinimumBoundingCircle, ST_LongestLine

7.14.17 ST_OrientedEnvelope

ST_OrientedEnvelope — Renvoie un rectangle de surface minimale contenant une géométrie.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Description

Renvoie le rectangle rotatif de surface minimale entourant une géométrie. Notez qu'il peut exister plus d'un rectangle de ce type. Peut renvoyer un point ou une ligne dans le cas d'entrées dégénérées.

Disponibilité : 2.5.0.

Nécessite GEOS >= 3.6.0.

Exemples

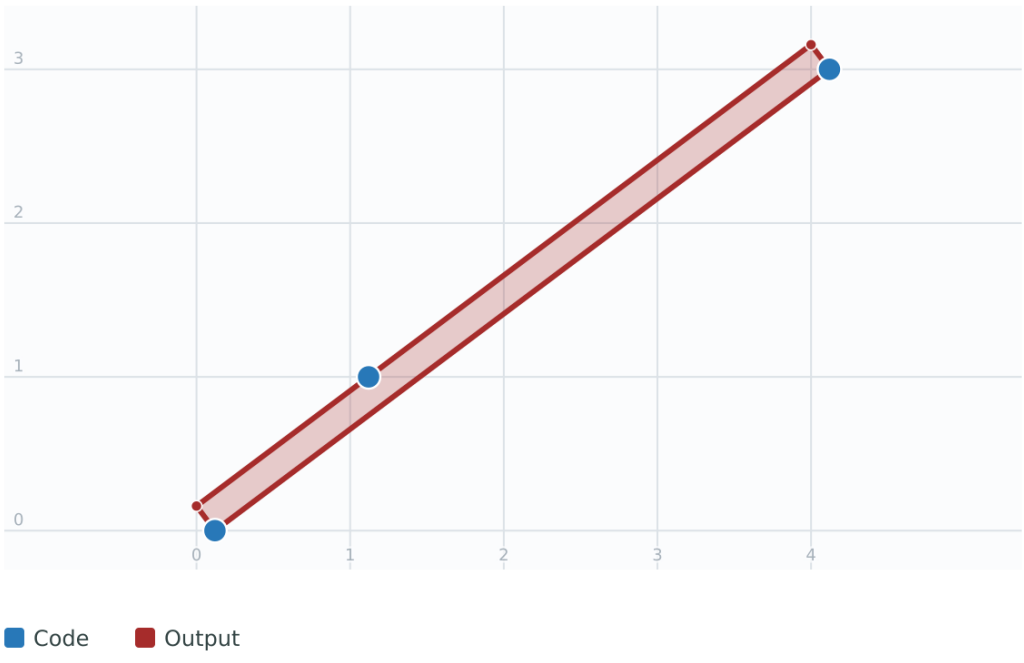
Code

```
SELECT ST_OrientedEnvelope('MULTIPOINT ((0 0),(-1 -1),(3 2))');
```

Output

```
POLYGON((3 2,-1 -1,-1.12 -0.84,2.88 2.16,3 2))
```

Figure



Oriented envelope of a Point and LineString.

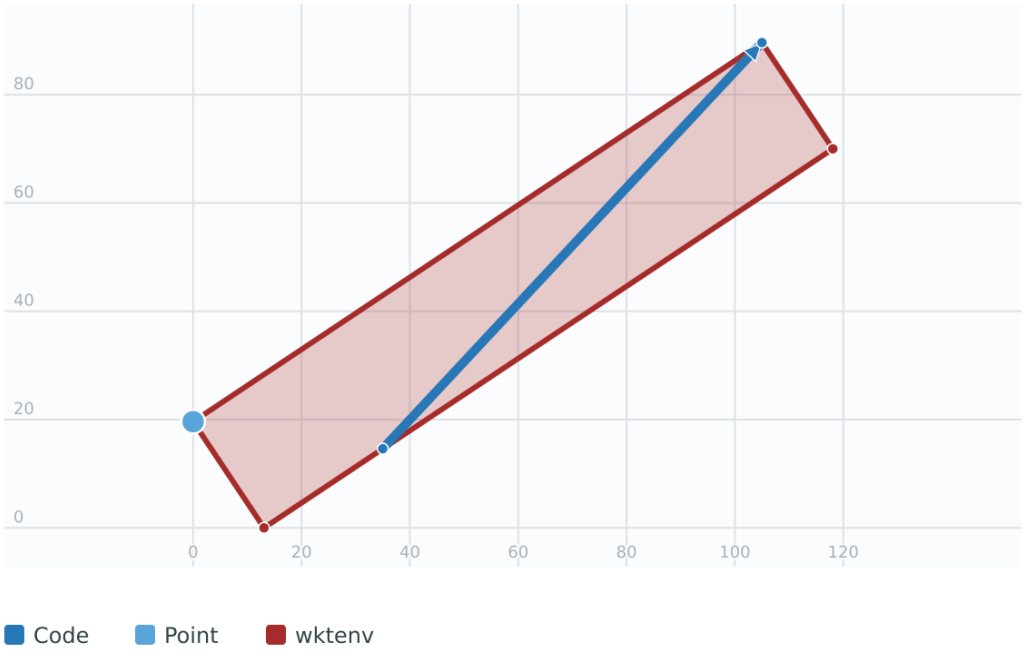
Code

```
SELECT ST_OrientedEnvelope(ST_Collect('LINESTRING(55 75,125 150)',
                ST_Point(20, 80))
                ) As wktenv;
```

Output

```
POLYGON((20 80,125 150,138.0769230769241 130.38461538461573,33.07692307692291 60.384615384615195,20 80))
```

Figure



Voir aussi

ST_Envelope ST_MinimumBoundingCircle

7.14.18 ST_OffsetCurve

ST_OffsetCurve — Renvoie une ligne décalée par rapport à une distance et un côté donnés à partir d'une ligne en entrée.

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters='');

Description

Renvoie une ligne décalée par rapport à une distance et un côté donnés à partir d'une ligne en entrée. Tous les points des géométries renvoyées ne sont pas plus éloignés que la distance donnée de la géométrie d'entrée. Utile pour calculer des lignes parallèles autour d'une ligne centrale.

Pour une distance positive, le décalage se situe à gauche de la ligne d'entrée et conserve la même direction. Pour une distance négative, il se trouve sur le côté droit et dans la direction opposée.

Les unités de distance sont mesurées en unités du système de référence spatiale.

Notez que la sortie peut être un MULTILINESTRING ou un EMPTY pour certaines géométries d'entrée en forme de puzzle.

Le troisième paramètre facultatif permet de spécifier une liste de paires clé/valeur séparées par des blancs afin de modifier les opérations comme suit :

— 'quad_segs=#' : nombre de segments utilisés pour approximer un quart de cercle (8 par défaut).
— 'join=round|mitre|bevel' : style de jointure (la valeur par défaut est "round"). 'miter' est également accepté comme synonyme de 'mitre'.
— 'mitre_limit=#.#' : limite du rapport de "mitre" (n'affecte que le style d'assemblage "mitred"). 'miter_limit' est également accepté comme synonyme de 'mitre_limit'.

Effectué par le module GEOS.

Comportement modifié dans GEOS 3.11 de sorte que les courbes de décalage ont désormais la même direction que la ligne d'entrée, tant pour les décalages positifs que négatifs.

Disponibilité : 2.0

Amélioration : 2.5 - ajout de la prise en charge de GEOMETRYCOLLECTION et MULTILINESTRING



Note
Cette fonction ignore la dimension Z. Elle donne toujours un résultat en 2D, même lorsqu'elle est utilisée sur une géométrie en 3D.

Exemples

Compute an open buffer around roads.

Code

```
SELECT ST_Union(
 ST_OffsetCurve(f.geom, f.width/2, 'quad_segs=4 join=round'),
 ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```

Offset 15 units with `quad_segs=4 join=round`.

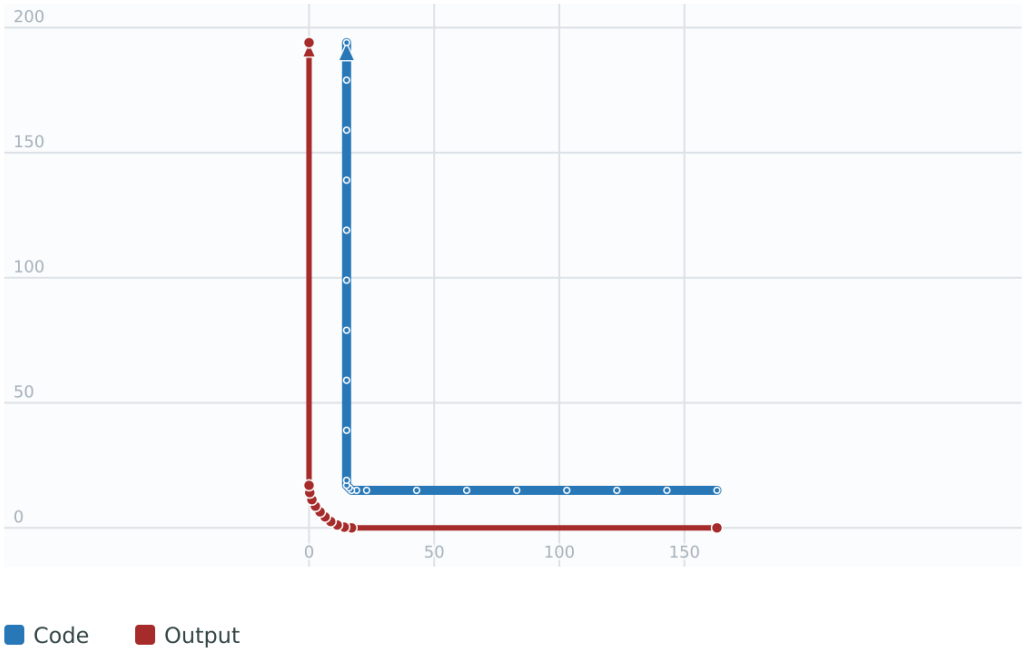
Code

```
SELECT ST_OffsetCurve(
    ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)'),
    15, 'quad_segs=4 join=round');
```

Output

```
LINESTRING(164 1,18 1,15.073645169758077 1.288220793951544,
    12.259748514523654 2.141807012330698,9.66644650470597 3.527955815461818,
    7.393398282201788 5.393398282201788,5.393398282201788 7.393398282201788,
    3.527955815461818 9.66644650470597,2.141807012330696 12.259748514523654,
    1.288220793951544 15.073645169758075,1 18,1 195)
```

Figure



Offset -15 units with `quad_segs=4 join=round`.

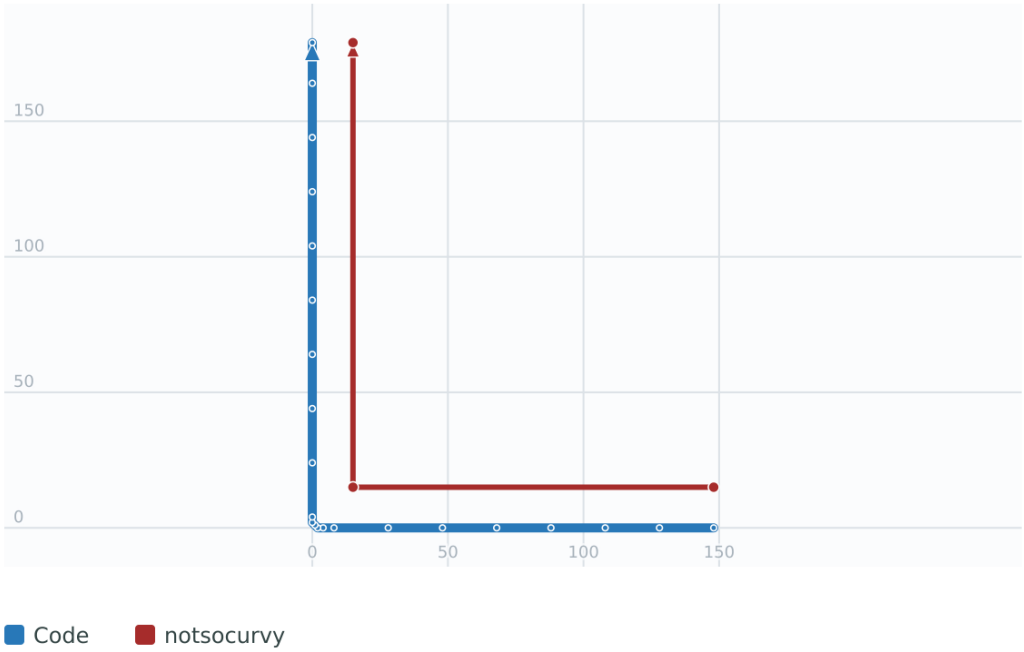
Code

```
SELECT ST_OffsetCurve(
    geom,
    -15,
    'quad_segs=4 join=round') As notsocurvy
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
LINESTRING(164 31,31 31,31 195)
```

Figure



Apply two negative offsets to obtain a smoother curve; the distances combine to -45 units.

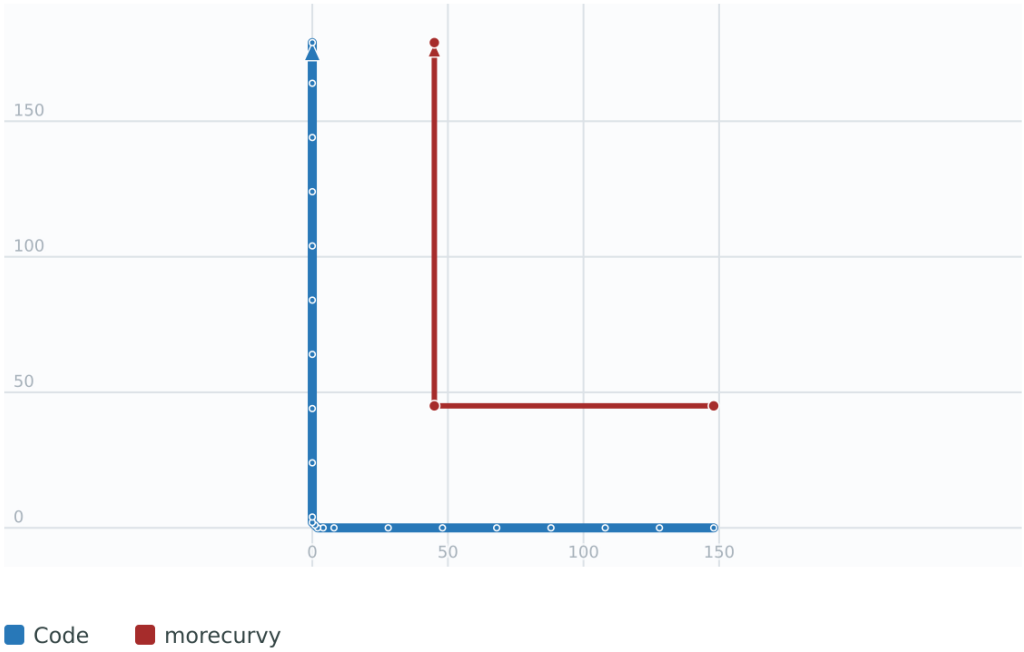
Code

```
SELECT ST_OffsetCurve(
    ST_OffsetCurve(
        geom,
        -30,
        'quad_segs=4 join=round'),
    -15,
    'quad_segs=4 join=round') As morecurvy
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
LINESTRING(164 61,61 61,61 195)
```

Figure



Collect the positive offset and the double-offset to produce parallel curves.

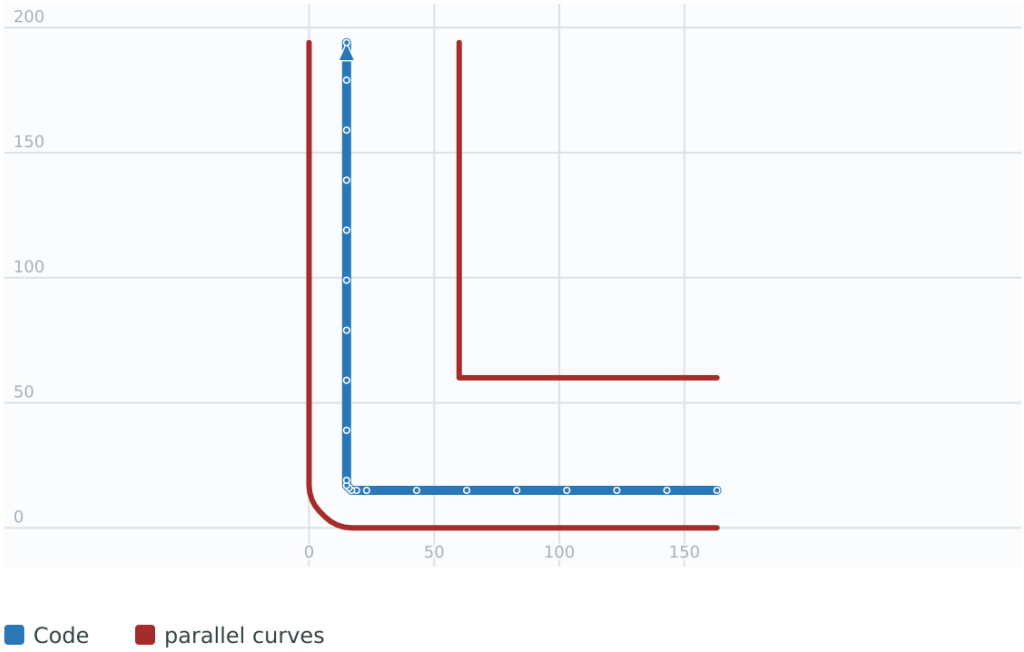
Code

```
SELECT ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 join=round'),
    ST_OffsetCurve(
        ST_OffsetCurve(
            geom,
            -30,
            'quad_segs=4 join=round'),
        -15,
        'quad_segs=4 join=round')
) As parallel_curves
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
MULTILINESTRING((164 1,18 1,15.073645169758077 1.288220793951544,
12.259748514523654 2.141807012330698,9.66644650470597 3.527955815461818,
7.393398282201788 5.393398282201788,5.393398282201788 7.393398282201788,
3.527955815461818 9.66644650470597,2.141807012330696 12.259748514523654,
1.288220793951544 15.073645169758075,1 18,1 195),(164 61,61 61,61 195))
```

Figure



Offset 15 units with a bevel join.

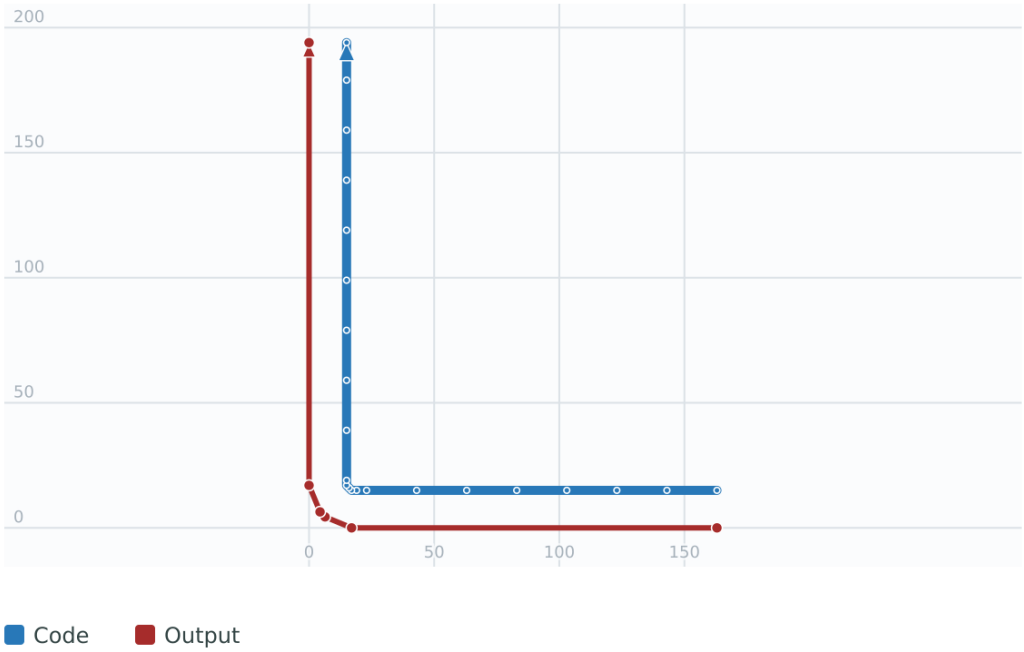
Code

```
SELECT ST_OffsetCurve(
    ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)'),
        15, 'quad_segs=4 join=bevel');
```

Output

```
LINESTRING(164 1,18 1,7.393398282201788 5.393398282201788,
    5.393398282201788 7.393398282201788,1 18,1 195)
```

Figure



Collect offsets on both sides using a mitre join and a mitre limit of 2.2.

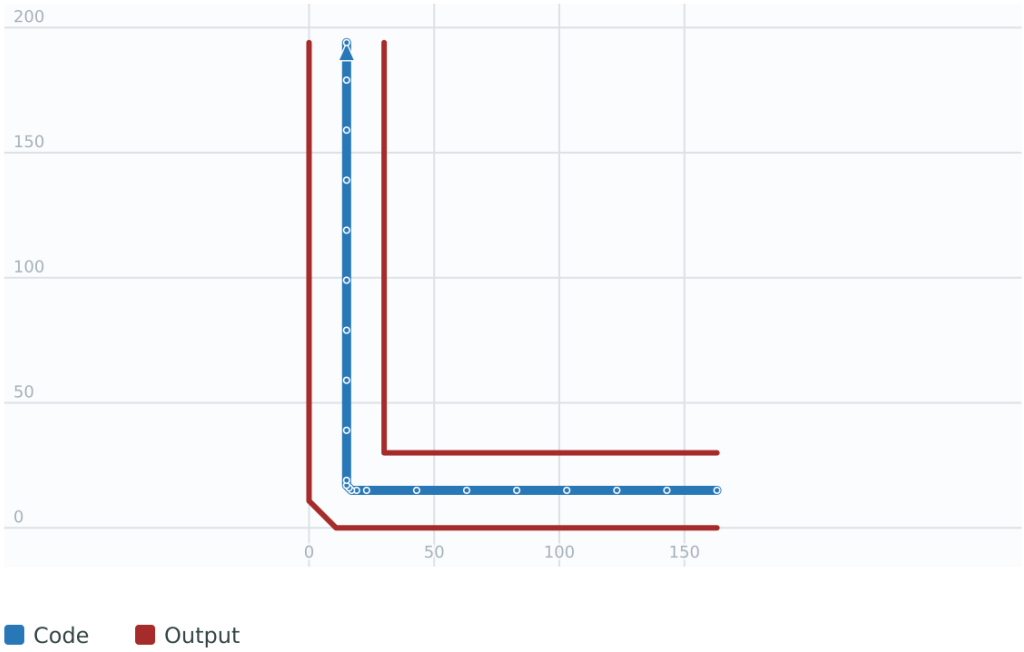
Code

```
SELECT ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 join=mitre mitre_limit=2.2'),
    ST_OffsetCurve(geom, -15, 'quad_segs=4 join=mitre mitre_limit=2.2')
    )
    FROM ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') As geom;
```

Output

```
MULTILINESTRING((164 1,11.786796564403577 1,1 11.786796564403577,1 195),
    (164 31,31 31,31 195))
```

Figure



Voir aussi

ST_Buffer

7.14.19 ST_PointOnSurface

ST_PointOnSurface — Calcule un point dont on garantit qu'il se trouve dans un polygone ou sur une géométrie.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Description

Renvoie un `POINT` qui est garanti de se trouver à l'intérieur d'une surface (`POLYGON`, `MULTIPOLYGON`, et `CURVEPOLYGON`). Dans PostGIS, cette fonction fonctionne également pour les géométries de lignes et de points.

Cette méthode implémente la spécification OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2

Cette méthode implémente la spécification SQL/MM. SQL-MM 3: 8.1.5, 9.5.6. Les spécifications définissent ST_PointOnSurface pour les géométries de surface uniquement. PostGIS étend la fonction pour prendre en charge tous les types de géométrie courants. D'autres bases de données (Oracle, DB2, ArcSDE) semblent ne prendre en charge cette fonction que pour les surfaces. SQL Server 2008 prend en charge tous les types de géométrie courants.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Point on surface of a MultiPoint.

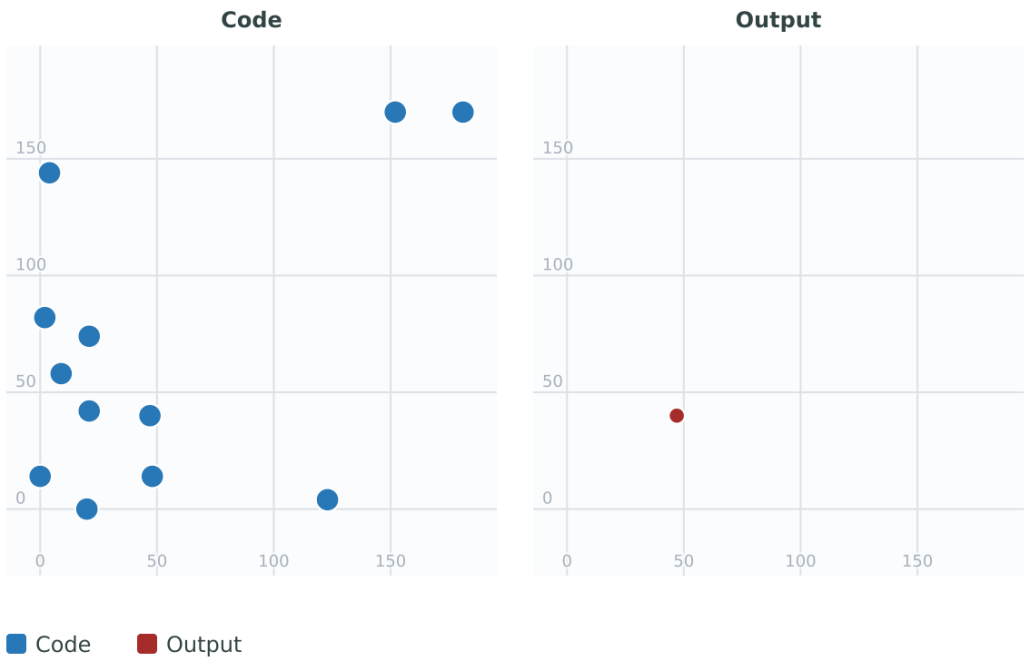
Code

```
SELECT ST_PointOnSurface(
  'MULTIPOINT (8 24,10 92,12 154,17 68,28 10,29 52,29 84,55 50,56 24,131 14,160 180,189 180) ↩
    ');
```

Output

```
POINT(55 50)
```

Figure



Point on surface of a LineString.

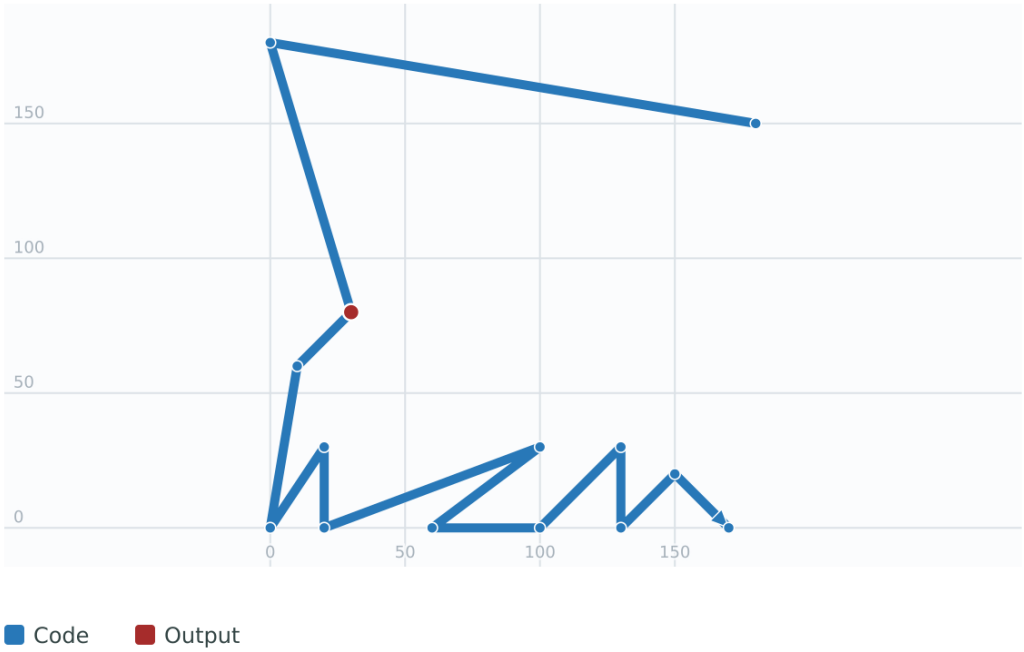
Code

```
SELECT ST_PointOnSurface(
  'LINESTRING (190 160,10 190,40 90,20 70,10 10,30 40,30 10,110 40,70 10,110 10,140 40,140  ↩
    10,160 30,180 10)');
```

Output

```
POINT(40 90)
```

Figure



Point on surface of a Polygon.

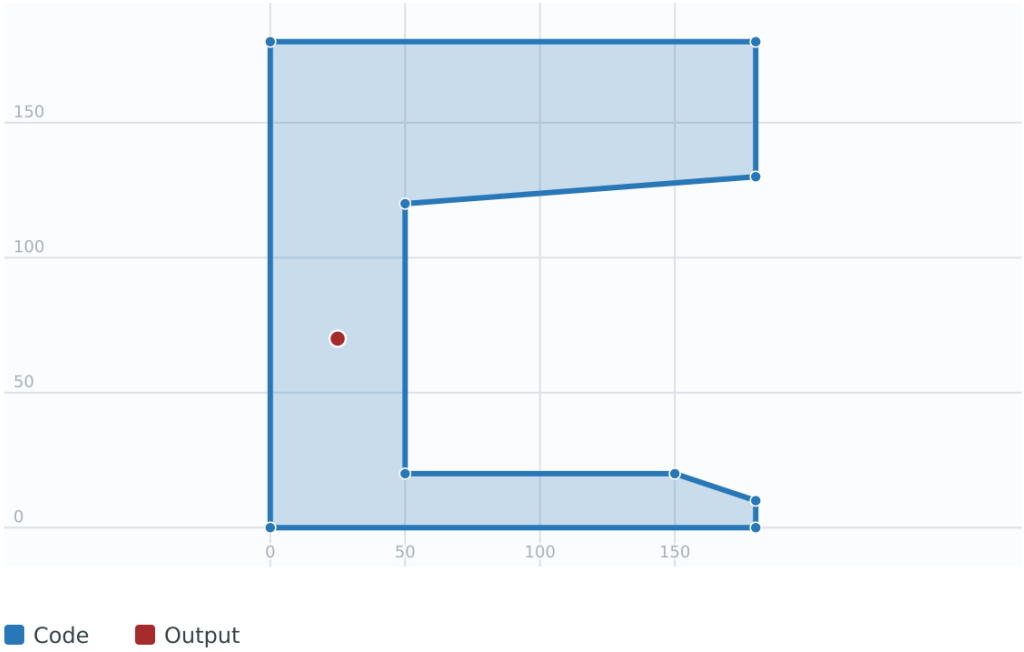
Code

```
SELECT ST_PointOnSurface(
  'POLYGON ((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))');
```

Output

```
POINT(35 80)
```

Figure



Point on surface of the highest-dimension components of a GeometryCollection.

Code

```
SELECT ST_PointOnSurface(
  'GEOMETRYCOLLECTION (POLYGON ((190 170,180 100,80 140,80 160,130 160,110 180,110 190,180 ↩
   180,190 170)),
    LINESTRING (80 120,120 20,140 70,150 30,180 50,190 10),
    MULTIPOINT (19 150,22 49,30 13,32 101,45 35,67 88,75 16))');
```

Output

```
POINT(133.571 150)
```

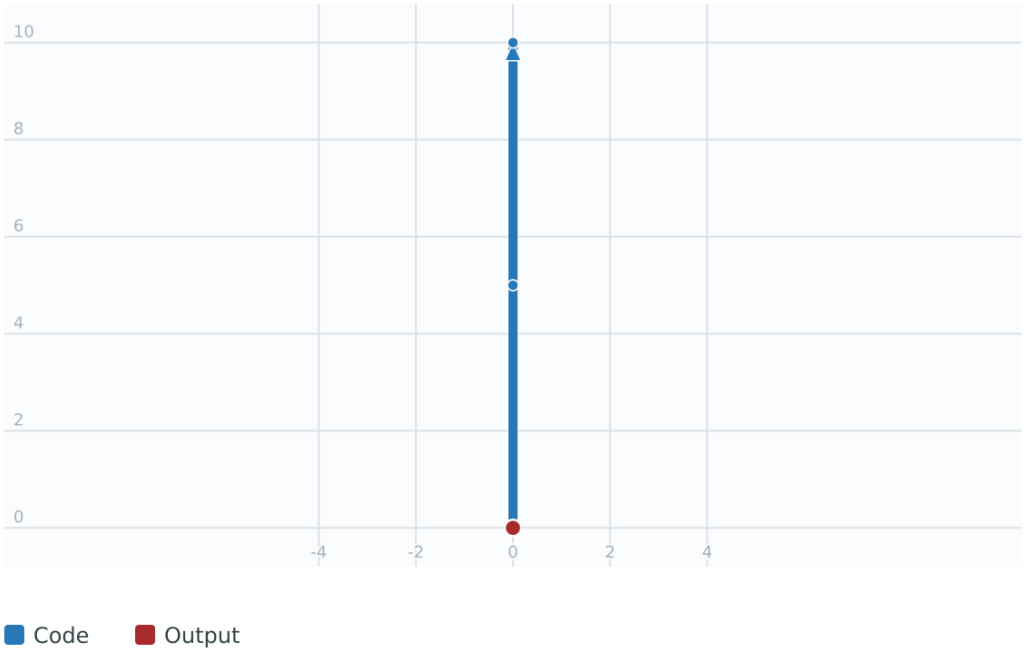
Figure



Code

```
SELECT ST_PointOnSurface('LINESTRING(0 5 1,0 0 1,0 10 2)');
```

Figure

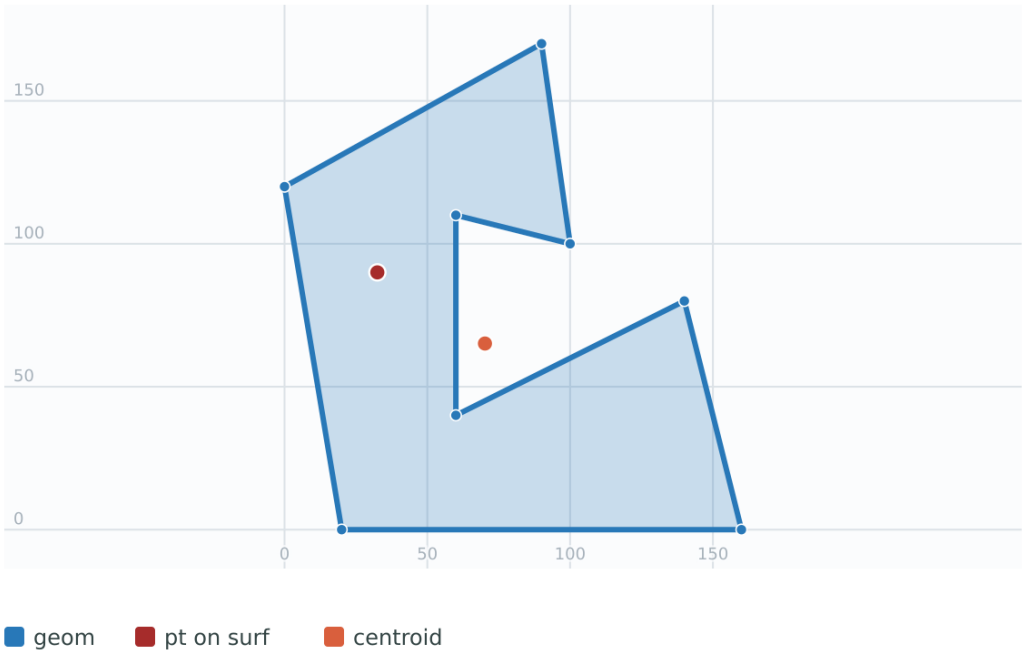


Exemple: Le résultat de ST_PointOnSurface est garanti à l'intérieur des polygones, alors que le point calculé par ST_Centroid peut se trouver à l'extérieur.

Code

```
SELECT ST_PointOnSurface(geom) AS pt_on_surf, ST_Centroid(geom) AS centroid
    FROM (SELECT 'POLYGON ((130 120,120 190,30 140,50 20,190 20,
                        170 100,90 60,90 130,130 120))'::geometry AS geom) AS t;
```

Figure



Voir aussi

ST_Centroid, ST_MaximumInscribedCircle

7.14.20 ST_Polygonize

ST_Polygonize — Calcule une collection de polygones formés à partir du tracé d'un ensemble de géométries.

Synopsis

geometry **ST_Polygonize**(geometry set geomfield);
geometry **ST_Polygonize**(geometry[] geom_array);

Description

Crée une GeometryCollection contenant les polygones formés par le tracé d'un ensemble de géométries. Si le tracé en entrée ne forme aucun polygone, une GeometryCollection vide est renvoyée.

Cette fonction crée des polygones couvrant toutes les zones délimitées. Si le résultat est destiné à former une géométrie polygonale valide, utilisez ST_BuildArea pour éviter que des trous ne soient remplis.



Note
La géométrie d'entrée doit être correctement nouée pour que cette fonction fonctionne correctement. Pour s'assurer que l'entrée est nouée, utilisez ST_Node sur la géométrie d'entrée avant de la polygoniser.



Note
Les collections de géométries peuvent être difficiles à manipuler avec des outils externes. Utilisez ST_Dump pour convertir le résultat polygonisé en polygones séparés.

Effectué par le module GEOS.

Disponibilité : 1.0.0RC1

Exemples

Polygonize a set of input lines.

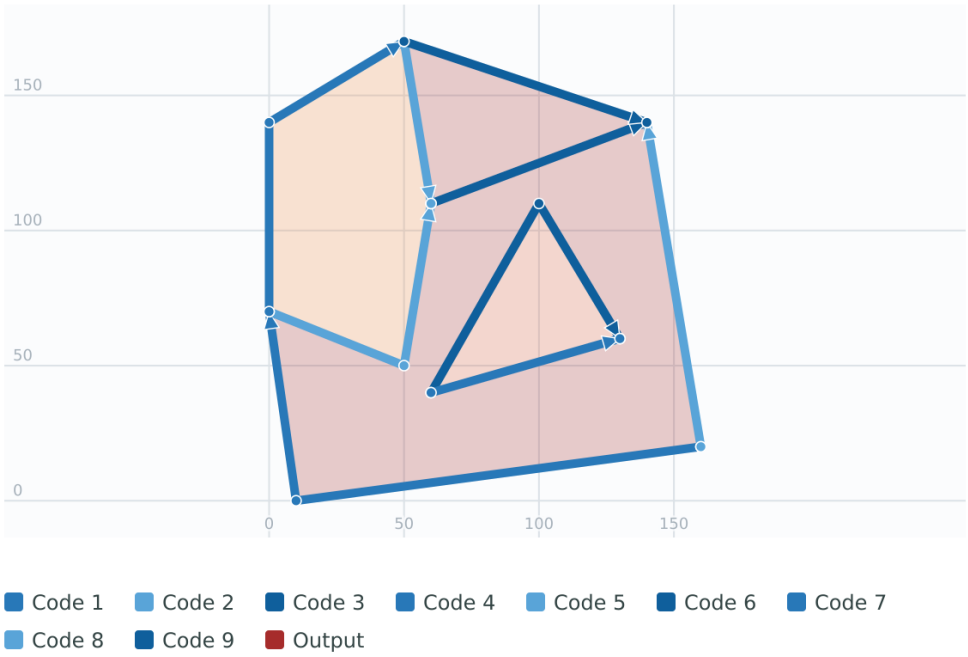
Code

```
WITH data(geom) AS (
  VALUES
    ('LINESTRING (180 40,30 20,20 90)'::geometry),
    ('LINESTRING (180 40,160 160)'::geometry),
    ('LINESTRING (80 60,120 130,150 80)'::geometry),
    ('LINESTRING (80 60,150 80)'::geometry),
    ('LINESTRING (20 90,70 70,80 130)'::geometry),
    ('LINESTRING (80 130,160 160)'::geometry),
    ('LINESTRING (20 90,20 160,70 190)'::geometry),
    ('LINESTRING (70 190,80 130)'::geometry),
    ('LINESTRING (70 190,160 160)'::geometry)
)
SELECT ST_Polygonize(geom)
FROM data;
```

Output

```
GEOMETRYCOLLECTION(POLYGON((180 40,30 20,20 90,70 70,80 130,160 160,180 40),(150 80,120 130,80 60,150 80)), POLYGON((80 ↩
      60,120 130,150 80,80 60)), POLYGON((80 130,70 70,20 90,20 160,70 190,80 130)), POLYGON((160 160,80 130,70 190,160 ↩
      160)))
```

Figure

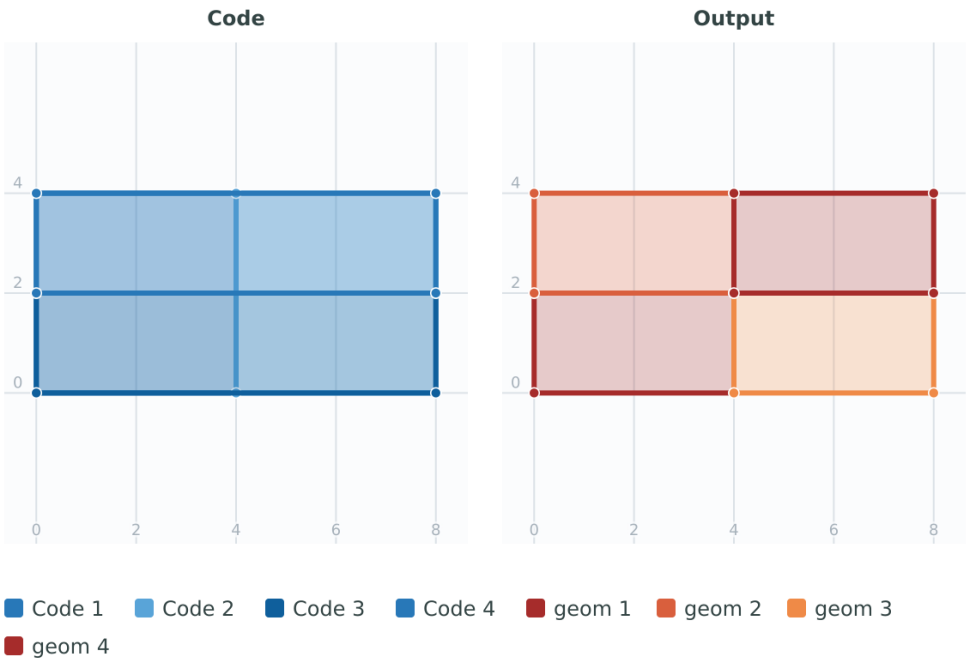


Polygonizing two polygon coverages and transferring attributes to the resultants:

Code

```
WITH poly_a(id, geom) AS (VALUES
  ('a1', 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry),
  ('a2', 'POLYGON ((4 0, 8 0, 8 4, 4 4, 4 0))'::geometry)
),
poly_b(id, geom) AS (VALUES
  ('b1', 'POLYGON ((0 0, 8 0, 8 2, 0 2, 0 0))'::geometry),
  ('b2', 'POLYGON ((0 2, 8 2, 8 4, 0 4, 0 2))'::geometry)
),
lines AS (
  SELECT ST_Boundary(geom) AS geom FROM poly_a
  UNION ALL
  SELECT ST_Boundary(geom) FROM poly_b
),
noded AS (
  SELECT ST_UnaryUnion(ST_Collect(geom)) AS geom FROM lines
),
cells AS (
  SELECT (ST_Dump(ST_Polygonize(geom))).geom AS geom FROM noded
)
SELECT a.id AS id_a,
       b.id AS id_b,
       c.geom AS geom
FROM cells AS c
LEFT JOIN poly_a AS a ON ST_Within(ST_PointOnSurface(c.geom), a.geom)
LEFT JOIN poly_b AS b ON ST_Within(ST_PointOnSurface(c.geom), b.geom)
WHERE a.id IS NOT NULL OR b.id IS NOT NULL
ORDER BY id_a, id_b;
```

Figure



Polygonisation d'une table de lignes :

Code

```
SELECT ST_Polygonize(geom_4269) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;
```

Output

```
geomtextrep
-------------
 SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096  ↩
    42.285752,-71.040878 42.285678)),
 POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794  ↩
    42.354971,-71.170511 42.354855,
 -71.17112 42.354238,-71.17166 42.353675)))
(1 row)
```

Use ST_Dump to dump out the polygonize geoms into individual polygons.

Code

```
SELECT (ST_Dump(t.polycoll)).geom AS geomtextrep
FROM (SELECT ST_Polygonize(geom_4269) AS polycoll
    FROM (SELECT geom_4269 FROM ma.suffolk_edges)
        As foo) AS t;
```

Output

```
geomtextrep
-------------
 SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,
-71.040878 42.285678))
 SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,
-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))
(2 rows)
```

Voir aussi

ST_BuildArea, ST_Boundary, ST_Collect, ST_Dump, ST_Node, ST_PointOnSurface, ST_UnaryUnion, ST_Within

7.14.21 ST_ReducePrecision

ST_ReducePrecision — Renvoie une géométrie valide dont les points sont arrondis en fonction de la tolérance de la grille.

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsize);

Description

Renvoie une géométrie valide dont tous les points sont arrondis à la tolérance de grille fournie, et dont les caractéristiques inférieures à la tolérance sont supprimées.

Contrairement à ST_SnapToGrid, la géométrie renvoyée sera valide, sans auto-intersections d'anneaux ni composants réduits.

La réduction de précision peut être utilisée pour :

— faire correspondre la précision des coordonnées à la précision des données
— réduire le nombre de coordonnées nécessaires pour représenter une géométrie
— garantir une sortie géométrique valide vers des formats qui utilisent une précision moindre (par exemple, des formats de texte tels que WKT, GeoJSON ou KML lorsque le nombre de décimales de sortie est limité).
— exporter une géométrie valide vers des systèmes qui utilisent une précision plus faible ou limitée (par exemple, SDE, valeur de tolérance Oracle)

Disponibilité : 3.1.0.

Nécessite GEOS >= 3.9.0.

Exemples

Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 0.1);
```

Output

```
POINT(1.4 19.3)
```

Figure



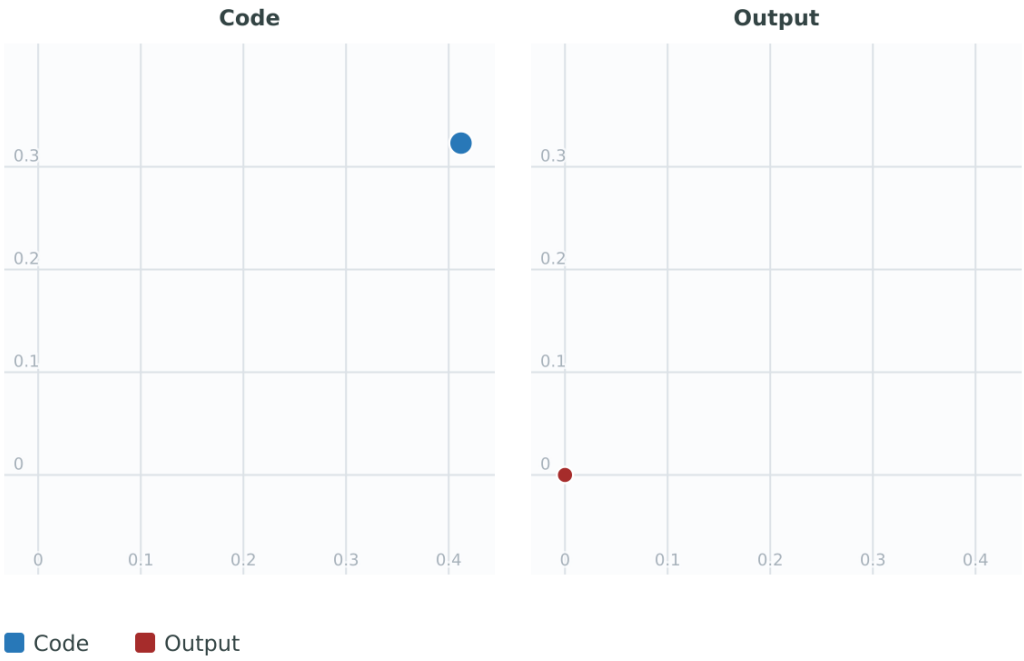
Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 1.0);
```

Output

```
POINT(1 19)
```

Figure



Code

```
SELECT ST_ReducePrecision('POINT(1.412 19.323)', 10);
```

Output

```
POINT(0 20)
```

Figure



La réduction de la précision permet de réduire le nombre de sommets

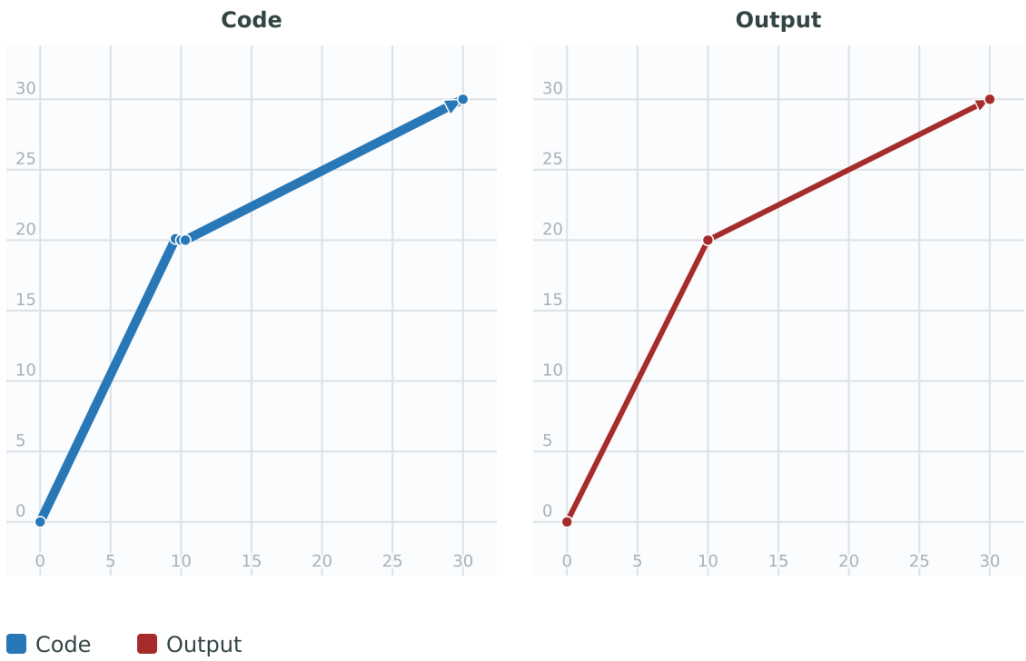
Code

```
SELECT ST_ReducePrecision('LINESTRING (10 10,19.6 30.1,20 30,20.3 30,40 40)', 1);
```

Output

```
LINESTRING (10 10, 20 30, 40 40)
```

Figure



La réduction de la précision divise les polygones si nécessaire pour garantir la validité

Code

```
WITH data(geom) AS (
  VALUES ('POLYGON ((10 10,60 60.1,70 30,40 40,50 10,10 10))'::geometry)
)
SELECT geom AS input,
       ST_ReducePrecision(geom, 10) AS reduced
FROM data;
```

Output

```
-[ RECORD 1 ]-------------------------------------------------
input   | POLYGON((10 10,60 60.1,70 30,40 40,50 10,10 10))
reduced | MULTIPOLYGON(((10 10,40 40,50 10,10 10)),((40 40,60 60,70 30,40 40)))
```

Figure



Voir aussi

ST_SnapToGrid, ST_Simplify, ST_SimplifyVW

7.14.22 ST_SharedPaths

ST_SharedPaths — Renvoie une collection contenant les chemins partagés par les deux lignes/multilignes en entrée.

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

Description

Renvoie une collection contenant les chemins partagés par les deux géométries d'entrée. Ceux qui vont dans la même direction sont dans le premier élément de la collection, ceux qui vont dans la direction opposée sont dans le deuxième élément. Les chemins eux-mêmes sont donnés dans la direction de la première géométrie.

Effectué par le module GEOS.

Disponibilité : 2.0.0

Exemples : Trouver des chemins communs

Adjacent shared path segments may be returned as separate or merged LineStrings depending on the GEOS version.

Find the shared paths of a MultiLineString and a LineString.

Code

```
SELECT ST_SharedPaths(ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 ↩
    125),
       (51 150,101 150,76 175,51 150))'),
   ST_GeomFromText('LINESTRING(151 100,126 156.25,126 125,101 150,76 175)')
   )
```

Output

```
GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),
(101 150,76 175)),MULTILINESTRING EMPTY)
```

Figure



Flip the LineString orientation to return the same paths in the opposite-direction member.

Code

```
SELECT ST_SharedPaths(ST_GeomFromText('LINESTRING(76 175,101 150,126 125,126 156.25,151 100) ↩
    '),
   ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
       (51 150,101 150,76 175,51 150))')
    )
```

Output

```
GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,101 150),(126 125,126 156.25)))
```

Figure



Voir aussi

ST_Dump, ST_GeometryN, ST_NumGeometries

7.14.23 ST_Simplify

ST_Simplify — Renvoie une représentation simplifiée d'une géométrie, en utilisant l'algorithme de Douglas-Peucker.

Synopsis

geometry **ST_Simplify**(geometry geom, float tolerance);
geometry **ST_Simplify**(geometry geom, float tolerance, boolean preserveCollapsed);

Description

Calcule une représentation simplifiée d'une géométrie en utilisant l'algorithme Douglas-Peucker. La variable `tolerance` de simplification est une valeur de distance, dans les unités du SRS d'entrée. La simplification supprime les sommets dont la distance au réseau simplifié est inférieure à la distance de tolérance. Le résultat peut ne pas être valide même si l'entrée l'est.

La fonction peut être appelée avec n'importe quel type de géométrie (y compris les GeometryCollections), mais seuls les lignes et polygones sont simplifiés. Les extrémités des lignes sont préservées.

L'indicateur `preserveCollapsed` permet de conserver les petites géométries qui seraient autrement supprimées à la tolérance donnée. Par exemple, si une ligne de 1 m de long est simplifiée avec une tolérance de 10 m, lorsque `preserveCollapsed` est vrai, la ligne ne disparaîtra pas. Cette option est utile pour le rendu, afin d'éviter que de très petits éléments ne disparaissent de la carte.



Note
La géométrie renvoyée peut perdre sa simplicité (voir ST_IsSimple), la topologie peut ne pas être préservée et les résultats polygonaux peuvent être invalides (voir ST_IsValid). Utilisez ST_SimplifyPreserveTopology pour préserver la topologie et garantir la validité.



Note
Cette fonction ne préserve pas les limites partagées entre les polygones. Utilisez ST_CoverageSimplify si cela est nécessaire.

Disponibilité : 1.2.2

Exemples

A circle simplified with a medium tolerance becomes an octagon, and with a large tolerance disappears.

Code

```
SELECT ST_NPoints(geom) AS np_before,
     ST_NPoints(ST_Simplify(geom, 0.1)) AS np01_notbadcircle,
     ST_NPoints(ST_Simplify(geom, 0.5)) AS np05_notquitecircle,
     ST_NPoints(ST_Simplify(geom, 1)) AS np1_octagon,
     ST_NPoints(ST_Simplify(geom, 10)) AS np10_gone,
     (ST_Simplify(geom, 100) is null) AS  np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10, 12) As geom) AS t;
```

Output

```
-[ RECORD 1 ]------------
np_before                | 49
np01_notbadcircle        | 33
np05_notquitecircle      | 17
np1_octagon              | 9
np10_gone                | null
np100_geometrygoesaway   | t
```

Simplification d'un ensemble de lignes. Les lignes peuvent s'intersecter après simplification.

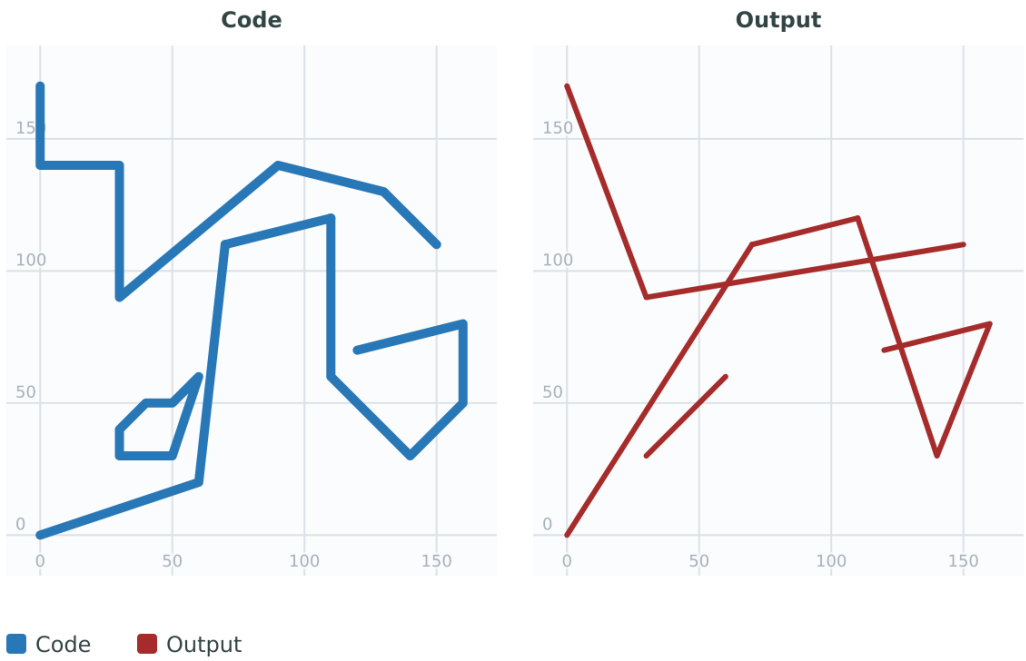
Code

```
SELECT ST_Simplify('MULTILINESTRING ((20 180,20 150,50 150,50 100,110 150,150 140,170 120) ↩
    ,(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180 60,180 90,140 80),(50 ↩
    40,70 40,80 70,70 60,60 60,50 50,50 40))'::geometry,
40);
```

Output

```
MULTILINESTRING((20 180,50 100,170 120),(20 10,90 120),(90 120,130 130),(130 130,160 40,180 90,140 80),(50 40,80 70,50 ↩
    40))
```

Figure



Simplification d'un polygone multiple. Les résultats polygonaux peuvent être invalides.

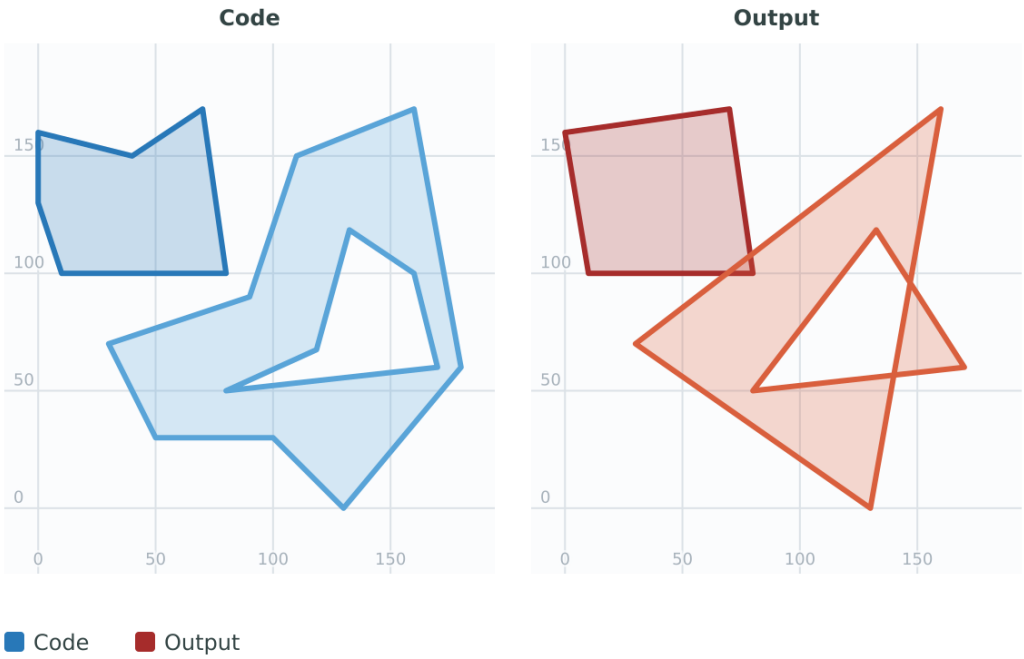
Code

```
SELECT ST_Simplify('MULTIPOLYGON (((90 110,80 180,50 160,10 170,10 140,20 110,90 110)),((40 ↩
    80,100 100,120 160,170 180,190 70,140 10,110 40,60 40,40 80),(180 70,170 110,142.5 ↩
    128.5,128.5 77.5,90 60,180 70)))'::geometry,
40);
```

Output

```
MULTIPOLYGON(((90 110,80 180,10 170,20 110,90 110)),((40 80,170 180,140 10,40 80),(180 70,142.5 128.5,90 60,180 70)))
```

Figure



Voir aussi

ST_IsSimple, ST_SimplifyPreserveTopology, ST_SimplifyVW, ST_CoverageSimplify, Topologie ST_Simplify

7.14.24 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — Renvoie une représentation simplifiée et valide d'une géométrie, en utilisant l'algorithme de Douglas-Peucker.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geom, float tolerance);

Description

Calcule une représentation simplifiée d'une géométrie en utilisant une variante de l'algorithme Douglas-Peucker qui limite la simplification pour garantir que le résultat a la même topologie que l'entrée. Le paramètre de `tolerance` de simplification est une valeur de distance, dans les unités du SRS d'entrée. La simplification supprime les sommets dont la distance au réseau simplifiée est inférieure à la distance de tolérance, tant que la topologie est préservée. Le résultat sera valide et simple si l'entrée est.

La fonction peut être appelée avec n'importe quel type de géométrie (y compris les GeometryCollections), mais seuls les éléments linéaires et polygonaux sont simplifiés. Pour les éléments polygonaux, le résultat aura le même nombre d'anneaux (coquilles et trous), et les anneaux ne se croiseront pas. Les extrémités des anneaux peuvent être simplifiées. Pour les entrées linéaires, le résultat aura le même nombre de lignes et les lignes ne se croiseront pas si elles ne le faisaient pas dans la géométrie d'origine. Les extrémités de la géométrie linéaire sont préservées.



Note
Cette fonction ne préserve pas les limites partagées entre les polygones. Utilisez ST_CoverageSimplify si cela est nécessaire.

Effectué par le module GEOS.

Disponibilité : 1.3.3

Exemples

For the same example as ST_Simplify, ST_SimplifyPreserveTopology prevents the geometry from disappearing.

Code

```
SELECT  ST_NPoints(geom) AS np_before,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.1)) AS np01_notbadcircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 0.5)) AS np05_notquitecircle,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 1))   AS np1_octagon,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 10))  AS np10_minimal,
        ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_minimal
FROM (SELECT ST_Buffer('POINT(1 3)', 10, 12) AS geom) AS t;
```

Output

```
np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_minimal |  ↩
    np100_minimal
-----------+-------------------+---------------------+-------------+--------------+--------------- ←

        49 |                33 |                  17 |           9 |            4 |          ↩
        4
```

Simplification d'un ensemble de lignes, en préservant la topologie des lignes qui ne se croisent pas.

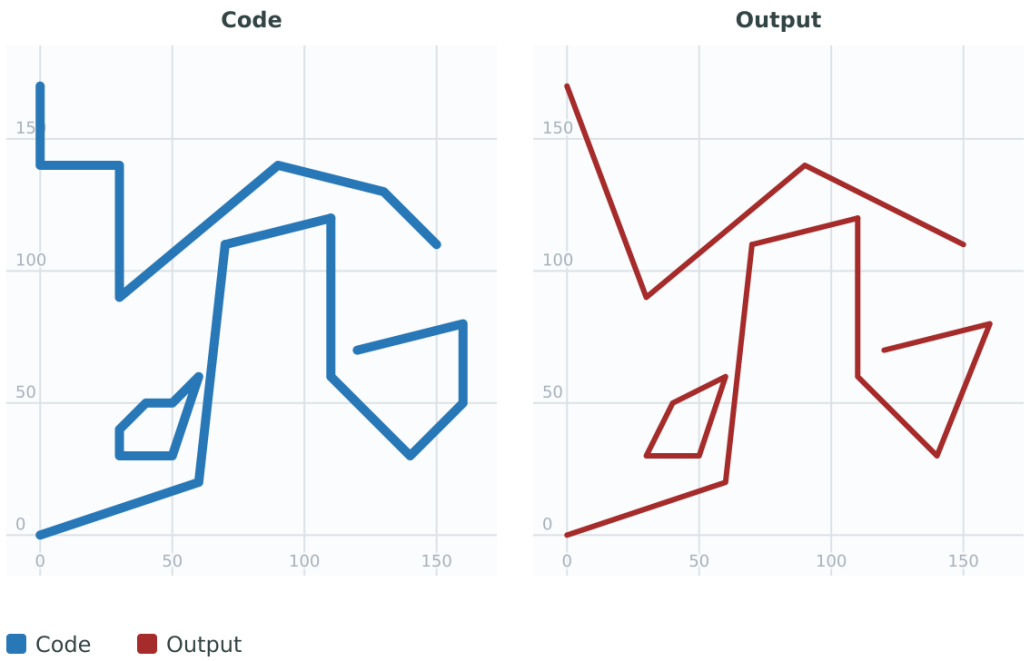
Code

```
SELECT ST_SimplifyPreserveTopology('MULTILINESTRING ((20 180,20 150,50 150,50 100,110  ↩
    150,150 140,170 120),(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180  ↩
    60,180 90,140 80),(50 40,70 40,80 70,70 60,60 60,50 50,50 40))'::geometry,
40);
```

Output

```
MULTILINESTRING((20 180,50 100,110 150,170 120),(20 10,80 30,90 120),(90 120,130 130),(130 130,130 70,160 40,180 90,140 ↩
     80),(50 40,70 40,80 70,60 60,50 40))
```

Figure



Simplification d'un multipolygone, en préservant la topologie des coquilles et des trous.

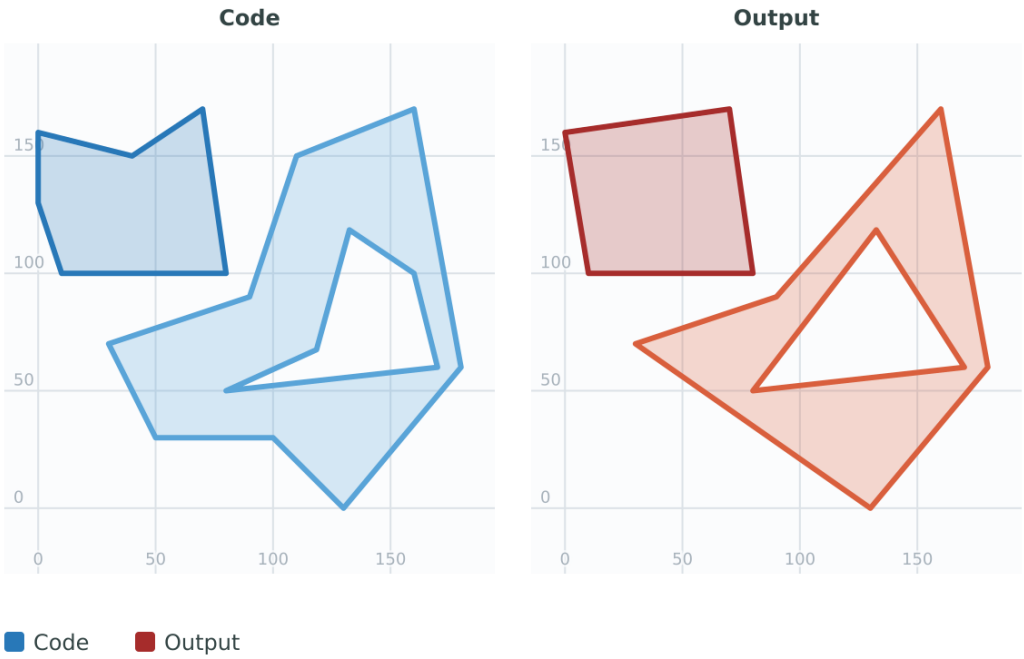
Code

```
SELECT ST_SimplifyPreserveTopology('MULTIPOLYGON (((90 110,80 180,50 160,10 170,10 140,20 ↩
    110,90 110)),((40 80,100 100,120 160,170 180,190 70,140 10,110 40,60 40,40 80),(180 ↩
    70,170 110,142.5 128.5,128.5 77.5,90 60,180 70)))'::geometry,
40);
```

Output

```
MULTIPOLYGON(((90 110,80 180,10 170,20 110,90 110)),((40 80,100 100,170 180,190 70,140 10,40 80),(180 70,142.5 128.5,90 ↩
      60,180 70)))
```

Figure



Voir aussi

ST_Simplify, ST_SimplifyVW, ST_CoverageSimplify

7.14.25 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — Calcule une enveloppe extérieure ou intérieure simplifiée préservant la topologie d'une géométrie polygonale.

Synopsis

geometry **ST_SimplifyPolygonHull**(geometry param_geom, float vertex_fraction, boolean is_outer = true);

Description

Calcule une enveloppe extérieure ou intérieure simplifiée préservant la topologie d'une géométrie polygonale. Une enveloppe extérieure couvre complètement la géométrie d'entrée. Une enveloppe intérieure est entièrement couverte par la géométrie d'entrée. Le résultat est une géométrie polygonale formée par un sous-ensemble de sommets d'entrée. Les multipolygones et les trous sont gérés et produisent un résultat ayant la même structure que la géométrie d'entrée.

La réduction du nombre de sommets est contrôlée par le paramètre `vertex_fraction`, qui est un nombre compris entre 0 et 1. Les valeurs inférieures produisent des résultats plus simples, avec un nombre de vertex plus faible et une concavité moindre. Pour les enveloppes extérieures et intérieures, une fraction de sommet de 1,0 produit la géométrie originale. Pour les enveloppes extérieures, une valeur de 0,0 produit l'enveloppe convexe (pour un polygone unique) ; pour les enveloppes intérieures, elle produit un triangle.

Le processus de simplification consiste à supprimer progressivement les angles concaves qui contiennent le moins de surface, jusqu'à ce que l'objectif de nombre de vertex soit atteint. Il empêche les arêtes de se croiser, de sorte que le résultat est toujours une géométrie polygonale valide.

Pour obtenir de meilleurs résultats avec des géométries contenant des segments de ligne relativement longs, il peut être nécessaire de "segmenter" l'entrée, comme indiqué ci-dessous.

Effectué par le module GEOS.

Disponibilité : 3.3.0.

Nécessite GEOS >= 3.11.0.

Exemples

Outer hull of a Polygon.

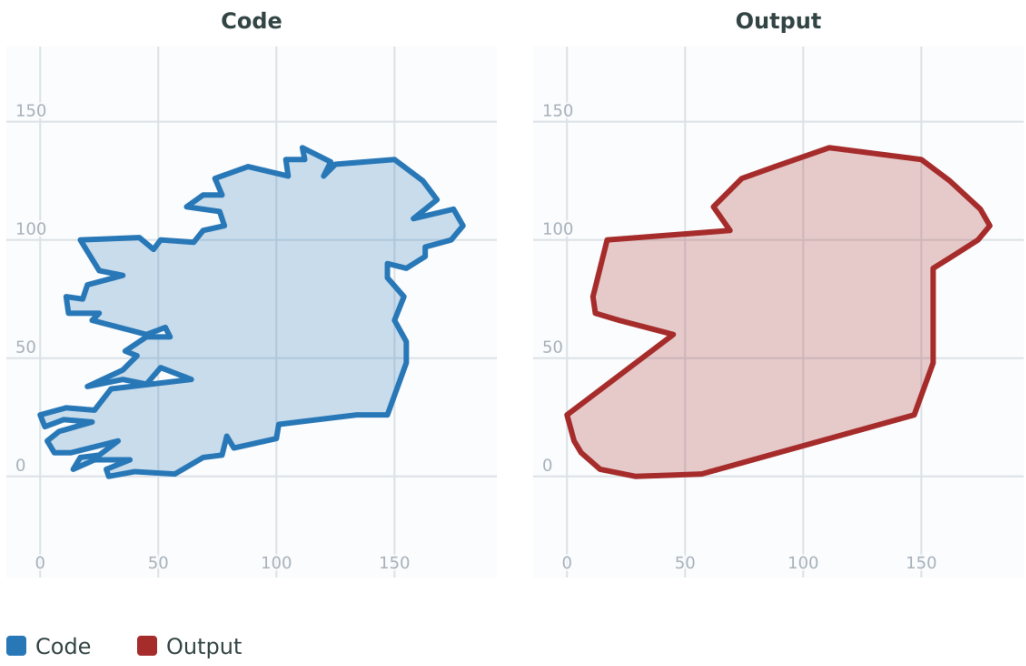
Code

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158,136 163,161 165,173 156,179 148,169 140,186 144,190 137,185 131,174 ↩
    128,174 124,166 119,158 121,158 115,165 107,161 97,166 88,166 79,158 57,145 57,112 ↩
    53,111 47,93 43,90 48,88 40,80 39,68 32,51 33,40 31,39 34,49 38,34 38,25 34,28 39,36 ↩
    40,44 46,24 41,17 41,14 46,19 50,33 54,21 55,13 52,11 57,22 60,34 59,41 68,75 72,62 ↩
    77,56 70,46 72,31 69,46 76,52 82,47 84,56 90,66 90,64 94,56 91,33 97,36 100,23 100,22 ↩
    107,29 106,31 112,46 116,36 118,28 131,53 132,59 127,62 131,76 130,80 135,89 137,87 ↩
    143,73 145,80 150,88 150,85 157,99 162,116 158,115 165,123 165,122 170,134 164,131 158)) ↩
    '::geometry,
  0.3
);
```

Output

```
POLYGON((161 165,173 156,186 144,190 137,185 131,174 124,166 119,166 79,158 57,68 32,40 31,25 34,17 41,14 46,11 57,56 ↩
    91,33 97,23 100,22 107,28 131,80 135,73 145,85 157,99 162,122 170,161 165))
```

Figure



Inner hull of a Polygon.

Code

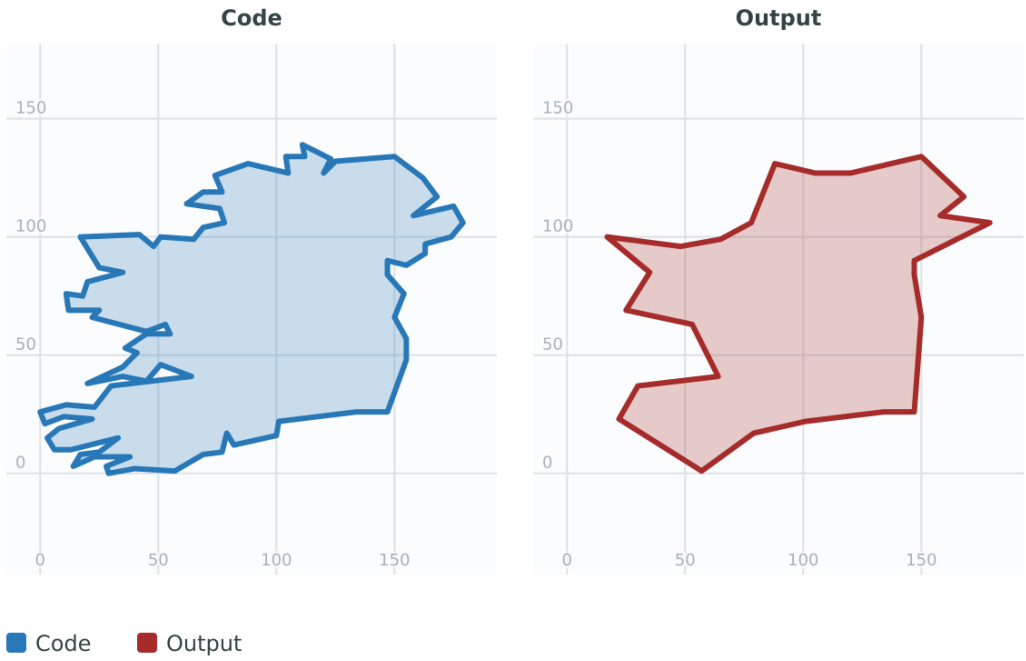
```
SELECT ST_SimplifyPolygonHull(
```

```
  'POLYGON ((131 158,136 163,161 165,173 156,179 148,169 140,186 144,190 137,185 131,174 ↩
    128,174 124,166 119,158 121,158 115,165 107,161 97,166 88,166 79,158 57,145 57,112 ↩
    53,111 47,93 43,90 48,88 40,80 39,68 32,51 33,40 31,39 34,49 38,34 38,25 34,28 39,36 ↩
    40,44 46,24 41,17 41,14 46,19 50,33 54,21 55,13 52,11 57,22 60,34 59,41 68,75 72,62 ↩
    77,56 70,46 72,31 69,46 76,52 82,47 84,56 90,66 90,64 94,56 91,33 97,36 100,23 100,22 ↩
    107,29 106,31 112,46 116,36 118,28 131,53 132,59 127,62 131,76 130,80 135,89 137,87 ↩
    143,73 145,80 150,88 150,85 157,99 162,116 158,115 165,123 165,122 170,134 164,131 158)) ↩
    '::geometry,
  0.3, false
);
```

Output

```
POLYGON((131 158,116 158,99 162,89 137,76 130,59 127,28 131,46 116,36 100,64 94,75 72,41 68,33 54,68 32,90 48,112 ↩
    53,145 57,158 57,161 97,158 115,158 121,190 137,169 140,179 148,161 165,131 158))
```

Figure

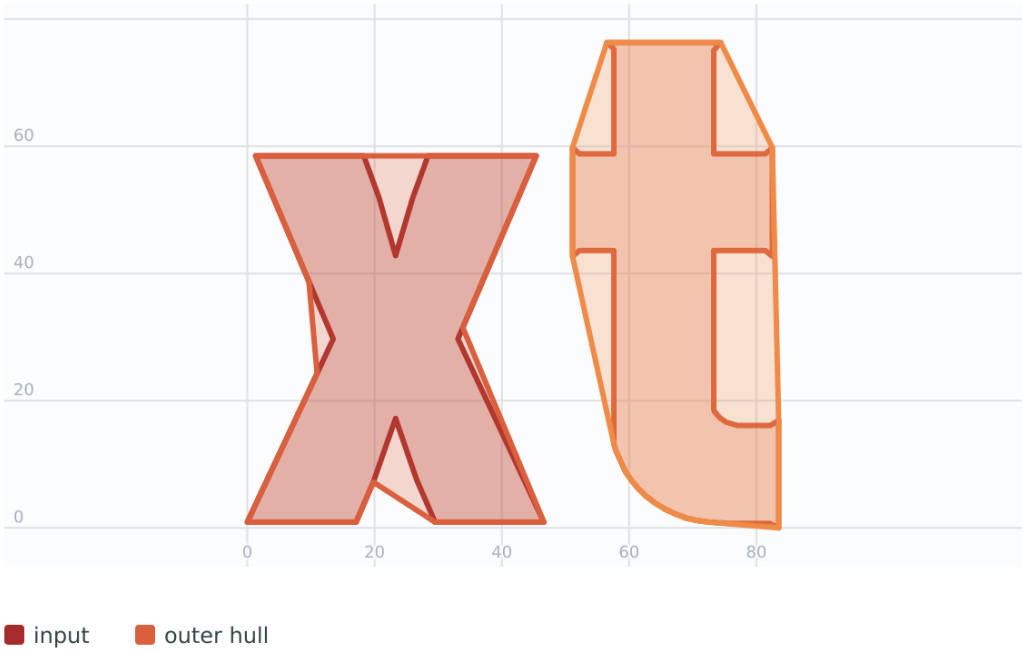


Outer hull simplification of a MultiPolygon, with segmentization.

Code

```
WITH letters AS (
  SELECT ST_Letters('xt') AS geom
)
SELECT geom AS input,
  ST_SimplifyPolygonHull(ST_Segmentize(geom, 2.0), 0.1) AS outer_hull
FROM letters;
```

Figure



Voir aussi

ST_ConvexHull, ST_SimplifyVW, ST_ConcaveHull, ST_Segmentize

7.14.26 ST_SimplifyVW

ST_SimplifyVW — Renvoie une représentation simplifiée d'une géométrie, en utilisant l'algorithme de Visvalingam-Whyatt

Synopsis

geometry **ST_SimplifyVW**(geometry geom, float tolerance);

Description

Returns a simplified representation of a geometry using the Visvalingam-Whyatt algorithm. The simplification `tolerance` is an area value, in squared units of the input SRS. Simplification removes vertices which form "corners" with area less than the tolerance. The result may not be valid even if the input is.

La fonction peut être appelée avec n'importe quel type de géométrie (y compris les GeometryCollections), mais seuls les lignes et polygones sont simplifiés. Les extrémités des lignes sont préservées.



Note
La géométrie renvoyée peut perdre sa simplicité (voir ST_IsSimple), la topologie peut ne pas être préservée et les résultats polygonaux peuvent être invalides (voir ST_IsValid). Utilisez ST_SimplifyPreserveTopology pour préserver la topologie et garantir la validité. ST_CoverageSimplify préserve également la topologie et la validité.

This is a general-purpose simplification function. It may remove narrow polygon spikes when their effective area is below the tolerance, but it is not a dedicated spike-removal or polygon-repair operation. Use ST_IsValid to check the result and ST_MakeValid to repair invalid polygonal output if needed.

Cette fonction ne préserve pas les limites partagées entre les polygones. Utilisez ST_CoverageSimplify si cela est nécessaire.

Cette fonction gère la 3D et la troisième dimension affectera le résultat.

Disponibilité : 2.2.0

Exemples

This polygon has a narrow spike whose effective area is small enough to be removed by a tolerance of 20 square units.

Code

```
SELECT geom AS original, ST_SimplifyVW(geom, 20) AS simplified
FROM (SELECT 'POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0))'::geometry AS geom) AS t ↩
    ;
```

Output

```
POLYGON((0 0,10 0,10 10,5.2 10,5 16,4.8 10,0 10,0 0)) | POLYGON((0 0,10 0,10 10,0 10,0 0))
```

Figure



A LineString is simplified with a minimum-area tolerance of 30. For a linear simplification using ST_Simplify, a comparable tolerance would be a length in input-SRS units, so the value here is roughly that linear tolerance squared.

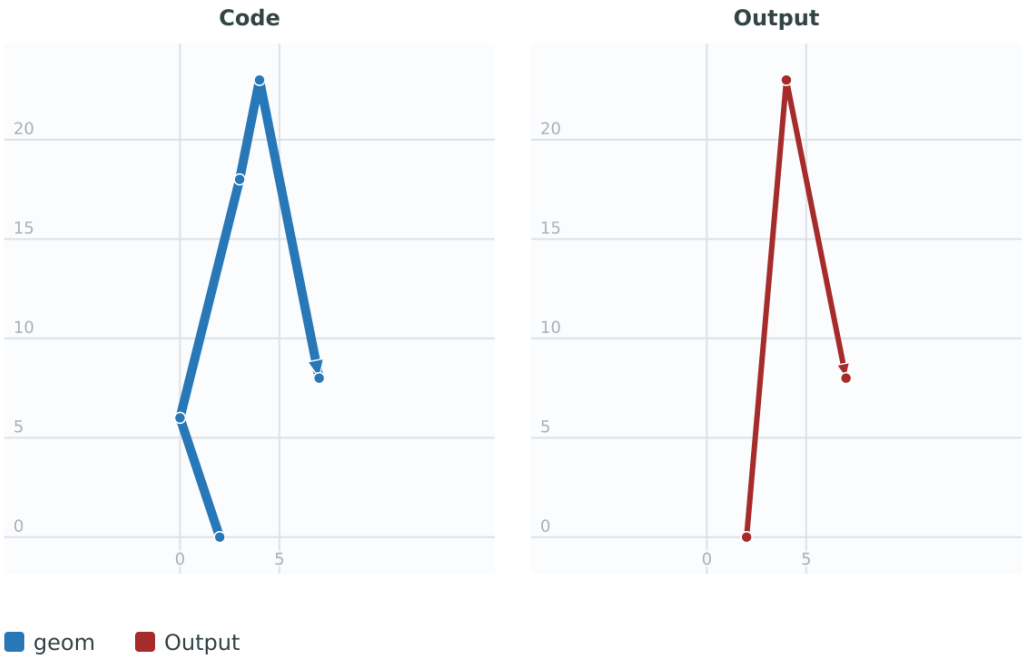
Code

```
SELECT ST_SimplifyVW(geom, 30) simplified
  FROM (SELECT 'LINESTRING(5 2,3 8,6 20,7 25,10 10)'::geometry AS geom) AS t;
```

Output

```
LINESTRING(5 2,7 25,10 10)
```

Figure



Simplification d'une ligne.

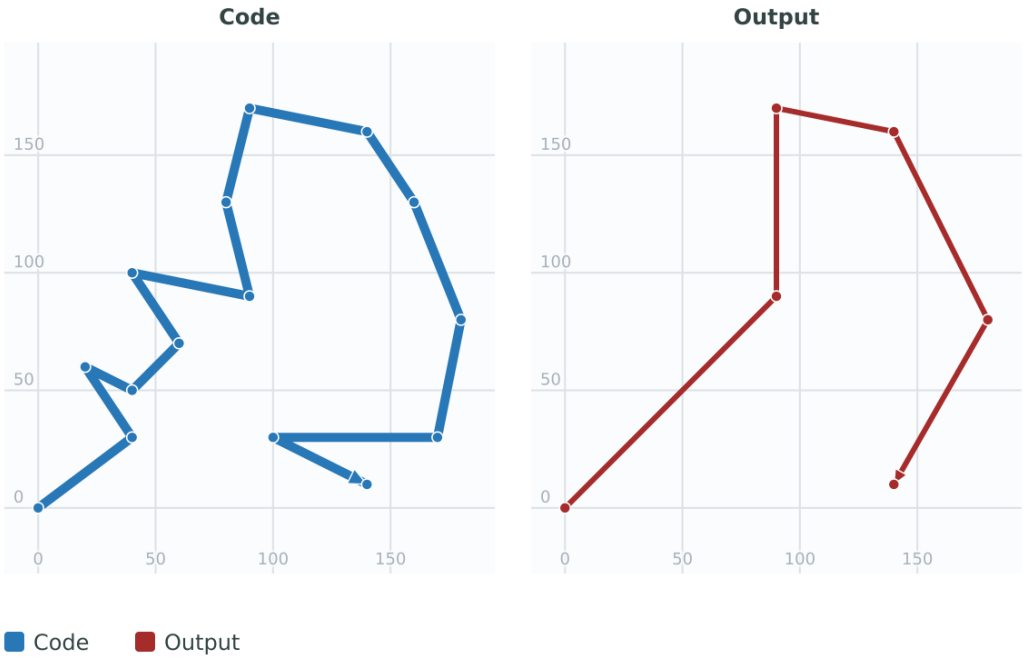
Code

```
SELECT ST_SimplifyVW('LINESTRING (10 10,50 40,30 70,50 60,70 80,50 110,100 100,90 140,100 ↩
    180,150 170,170 140,190 90,180 40,110 40,150 20)'::geometry,
1600);
```

Output

```
LINESTRING(10 10,100 100,100 180,150 170,190 90,150 20)
```

Figure



Simplification d'un polygone.

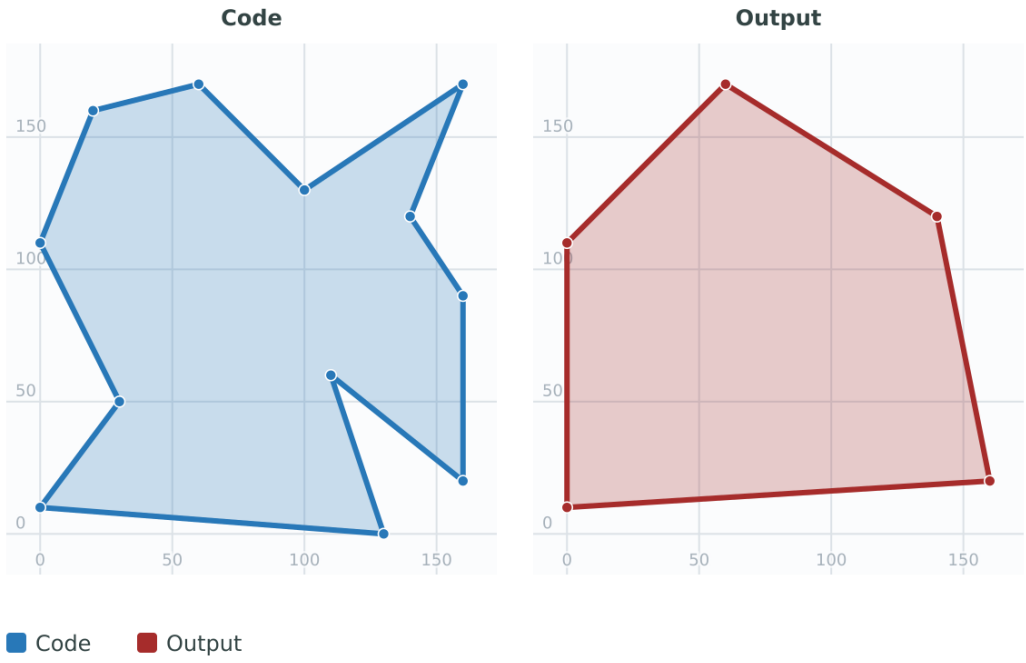
Code

```
SELECT ST_SimplifyVW('POLYGON ((20 20,50 60,20 120,40 170,80 180,120 140,180 180,160 130,180 ↩
    100,180 30,130 70,150 10,20 20))'::geometry,
1600);
```

Output

```
POLYGON((20 20,20 120,80 180,160 130,180 30,20 20))
```

Figure



Voir aussi

ST_SetEffectiveArea, ST_Simplify, ST_SimplifyPreserveTopology, ST_CoverageSimplify, Topologie ST_Simplify

7.14.27 ST_SetEffectiveArea

ST_SetEffectiveArea — Définit la surface effective de chaque sommet, en utilisant l'algorithme Visvalingam-Whyatt.

Synopsis

geometry **ST_SetEffectiveArea**(geometry geom, float threshold = 0, integer set_area = 1);

Description

Définit la surface effective de chaque sommet, en utilisant l'algorithme Visvalingam-Whyatt. La surface effective est stockée comme la valeur M du sommet. Si le paramètre optionnel "threshold" est utilisé, une géométrie simplifiée sera renvoyée, contenant uniquement les sommets dont la surface effective est supérieure ou égale à la valeur seuil.

Cette fonction peut être utilisée pour la simplification côté serveur lorsqu'un seuil est spécifié. Une autre option consiste à utiliser une valeur seuil de zéro. Dans ce cas, la géométrie complète sera renvoyée avec les surfaces effectives sous forme de valeurs M, qui peuvent être utilisées par le client pour simplifier très rapidement.

Cette fonction n'agit qu'avec des (multi)lignes et des (multi)polygones, mais vous pouvez l'appeler en toute sécurité avec n'importe quel type de géométrie. Comme la simplification se produit objet par objet, vous pouvez également utiliser une collection de géométries dans cette fonction.



Note
Notez que la géométrie retournée peut perdre sa simplicité (voir ST_IsSimple)



Note
Notez que la topologie peut ne pas être préservée et donner lieu à des géométries non valides. Utilisez (voir ST_SimplifyPreserveTopology) pour préserver la topologie.



Note
La géométrie de sortie perdra toutes les informations précédentes concernant les valeurs M



Note
Cette fonction gère la 3D et la troisième dimension affecte la surface effective

Disponibilité : 2.2.0

Exemples

Calcul de la surface effective d'une LineString. Comme nous utilisons une valeur seuil de zéro, tous les sommets de la géométrie d'entrée sont renvoyés.

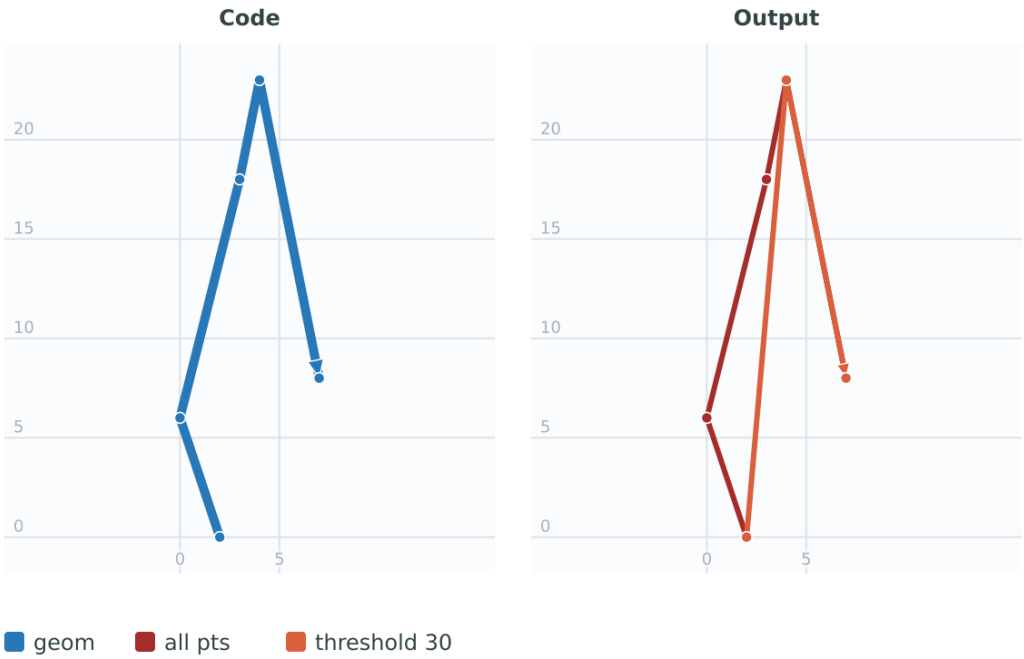
Code

```
SELECT ST_SetEffectiveArea(geom) AS all_pts,
       ST_SetEffectiveArea(geom, 30) AS threshold_30
FROM (SELECT  'LINESTRING(5 2,3 8,6 20,7 25,10 10)'::geometry geom) As foo;
```

Output

```
-[ RECORD 1 ]------------------
all_pts      | LINESTRING M (5 2 3.4e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.4e+38)
threshold_30 | LINESTRING M (5 2 3.4e+38,7 25 49.5,10 10 3.4e+38)
```

Figure



Voir aussi

ST_SimplifyVW

7.14.28 ST_TriangulatePolygon

ST_TriangulatePolygon — Calcule la triangulation de Delaunay contrainte des polygones

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

Description

Calcule la triangulation de Delaunay contrainte des polygones. Les trous et les multipolygones sont pris en charge.

La "constrained Delaunay triangulation" d'un polygone est un ensemble de triangles formés à partir des sommets du polygone et le couvrant exactement, avec l'angle intérieur total maximal parmi toutes les triangulations possibles. Elle constitue la triangulation de "meilleure qualité" du polygone.

Disponibilité : 3.3.0.

Nécessite GEOS >= 3.11.0.

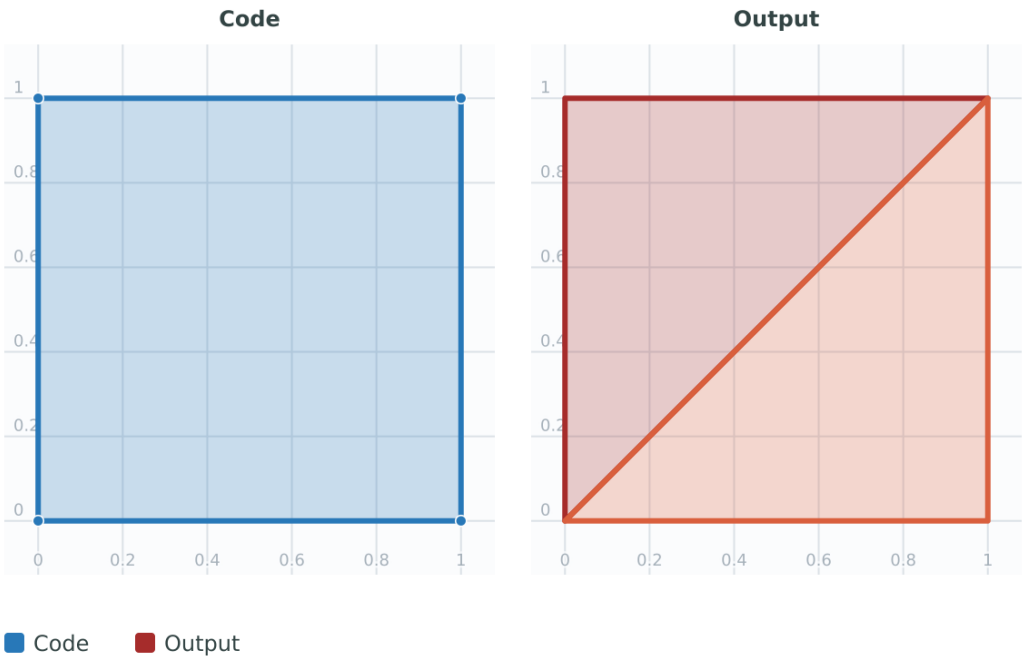
Exemples

Triangulation d'un carré.

Code

```
SELECT ST_TriangulatePolygon('POLYGON((0 0,0 1,1 1,1 0,0 0))');
```

Figure

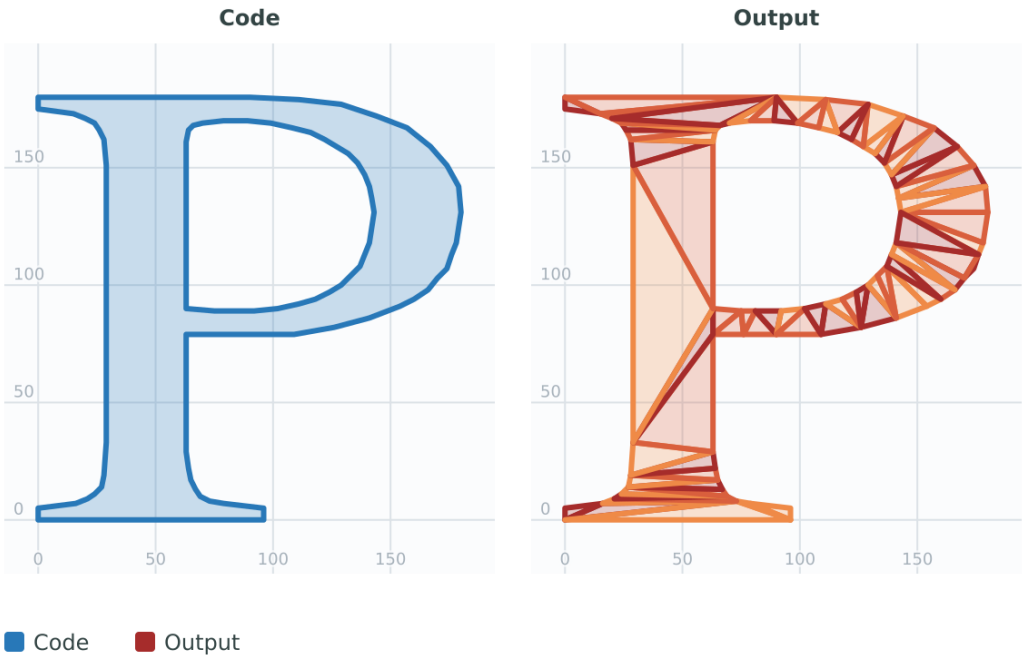


Triangulation de la lettre P.

Code

```
SELECT ST_TriangulatePolygon('POLYGON ((26 17,31 19,34 21,37 24,38 29,39 43,39 161,38 172,36 ↩
     176,34 179,30 181,25 183,10 185,10 190,100 190,121 189,139 187,154 182,167 177,177  ↩
    169,184 161,189 152,190 141,188 128,186 123,184 117,180 113,176 108,170 104,164 101,151  ↩
    96,136 92,119 89,100 89,86 89,73 89,73 39,74 32,75 27,77 23,79 20,83 18,89 17,106 15,106 ↩
     10,10 10,10 15,26 17),(152 147,151 152,149 157,146 162,142 166,137 169,132 172,126  ↩
    175,118 177,109 179,99 180,89 180,80 179,76 178,74 176,73 171,73 100,85 99,91 99,102  ↩
    99,112 100,121 102,128 104,134 107,139 110,143 114,147 118,149 123,151 128,153 141,152  ↩
    147))'
);
```

Figure

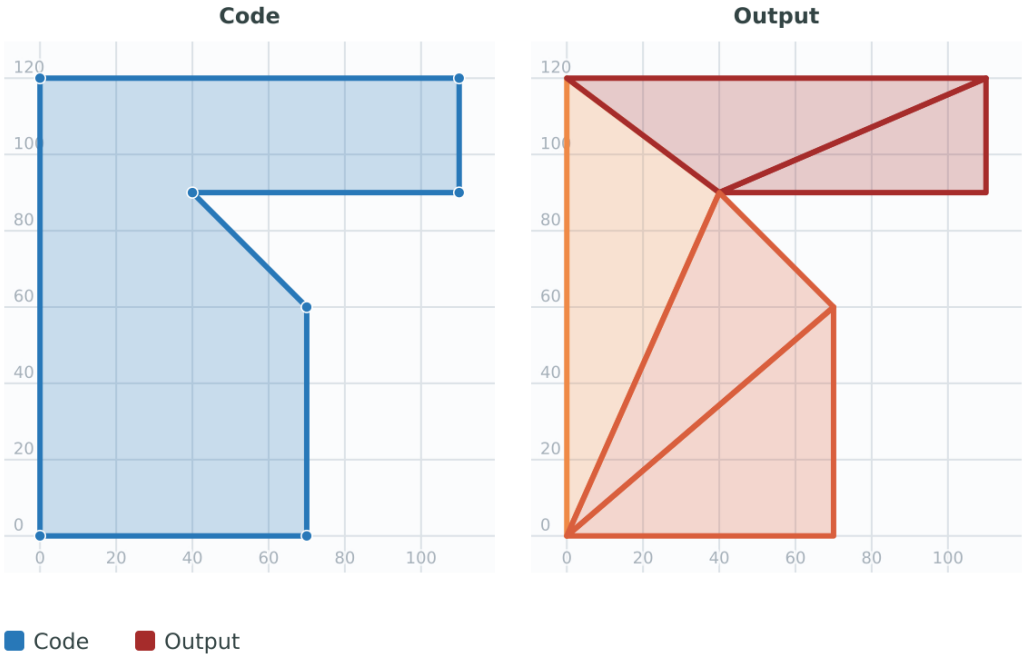


Triangulation of the polygon used in the CG_Tesselate example.

Code

```
SELECT ST_TriangulatePolygon('POLYGON (( 10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 ↩
    190 ))'::geometry
);
```

Figure



Voir aussi

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, CG_Tesselate

7.14.29 ST_VoronoiLines

ST_VoronoiLines — Renvoie les limites des polygones de Voronoï des sommets d'une géométrie.

Synopsis

geometry **ST_VoronoiLines**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Description

Calcule un Voronoi diagram bidimensionnel à partir des sommets de la géométrie fournie et renvoie les limites entre les cellules de la représentation sous la forme d'une MultiLineString. Renvoie null si la géométrie d'entrée est null. Renvoie une collection de géométries vide si la géométrie d'entrée ne contient qu'un seul sommet. Renvoie une collection de géométries vide si l'enveloppe `extend_to` a une surface nulle.

Paramètres optionnels :

— `tolerance` : distance à laquelle les sommets seront considérés comme équivalents. La robustesse de l'algorithme peut être améliorée en fournissant une distance de tolérance non nulle. (par défaut = 0.0)

— `extend_to` : S'il est présent, la représentation est étendue pour couvrir l'enveloppe de la géométrie fournie, à moins qu'elle ne soit plus petite que l'enveloppe par défaut (default = NULL, l'enveloppe par défaut est la boîte de délimitation de l'entrée élargie d'environ 50 %).

Effectué par le module GEOS.

Disponibilité : 2.3.0

Exemples

Voronoi diagram lines with a tolerance of 30 units.

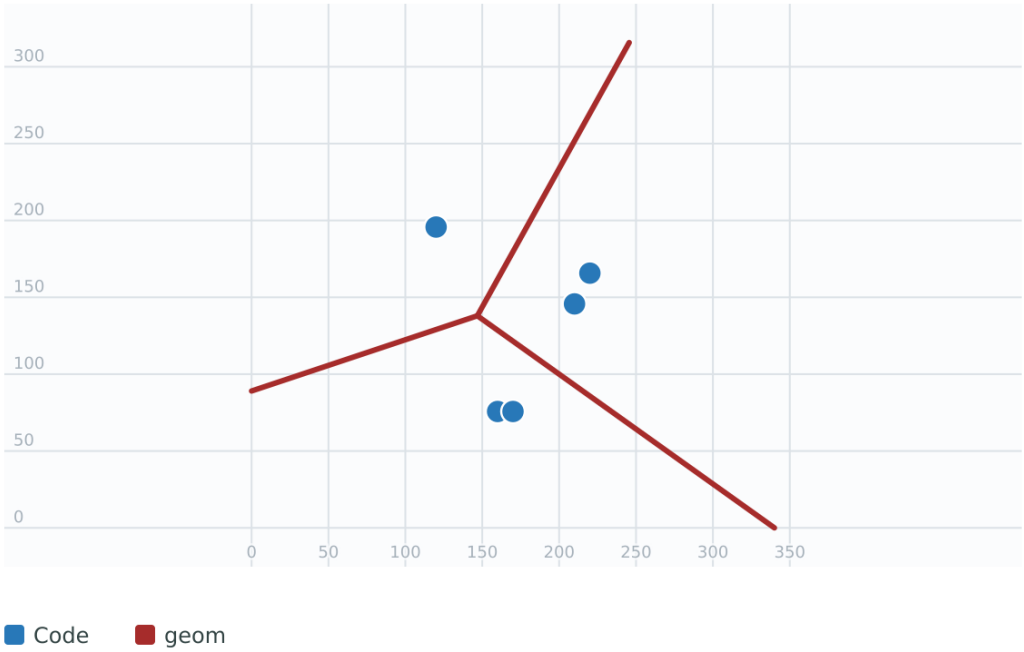
Code

```
SELECT ST_VoronoiLines('MULTIPOINT (50 30,60 30,100 100,10 150,110 120)'::geometry,
            30) AS geom;
```

Output

```
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 ↩
    43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

Figure



Voir aussi

ST_DelaunayTriangles, ST_VoronoiPolygons

7.14.30 ST_VoronoiPolygons

ST_VoronoiPolygons — Renvoie les cellules de la représentation de Voronoï des sommets d'une géométrie.

Synopsis

geometry **ST_VoronoiPolygons**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Description

Calcule un diagramme de Voronoï bidimensionnel à partir des sommets de la géométrie fournie. Le résultat est une GEOMETRYCOLLECTION de POLYGONES qui couvre une enveloppe plus grande que l'étendue des sommets d'entrée. Renvoie null si la géométrie d'entrée est null. Renvoie une collection de géométries vide si la géométrie d'entrée ne contient qu'un seul sommet. Renvoie une collection de géométries vide si l'enveloppe `extend_to` a une surface nulle.

Paramètres optionnels :
— `tolerance` : distance à laquelle les sommets seront considérés comme équivalents. La robustesse de l'algorithme peut être améliorée en fournissant une distance de tolérance non nulle. (par défaut = 0.0)
— `extend_to` : S'il est présent, la représentation est étendue pour couvrir l'enveloppe de la géométrie fournie, à moins qu'elle ne soit plus petite que l'enveloppe par défaut (default = NULL, l'enveloppe par défaut est la boîte de délimitation de l'entrée élargie d'environ 50 %).
Effectué par le module GEOS.

Disponibilité : 2.3.0

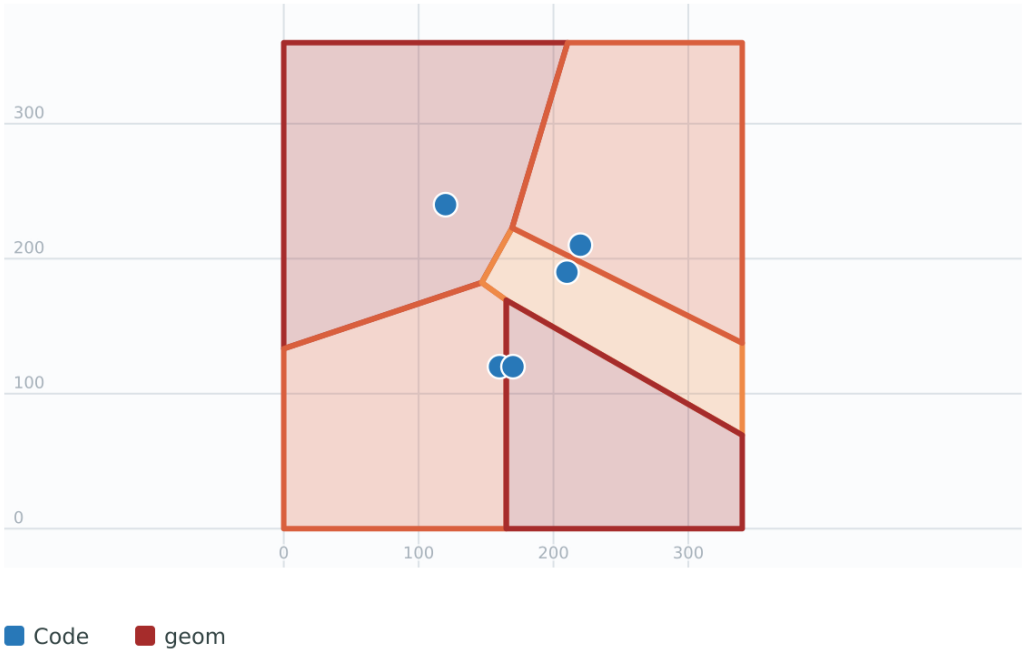
Exemples

Points overlaid on top of the Voronoi diagram.

Code

```
SELECT ST_VoronoiPolygons('MULTIPOINT (50 30,60 30,100 100,10 150,110 120)'::geometry
            ) AS geom;
```

Figure

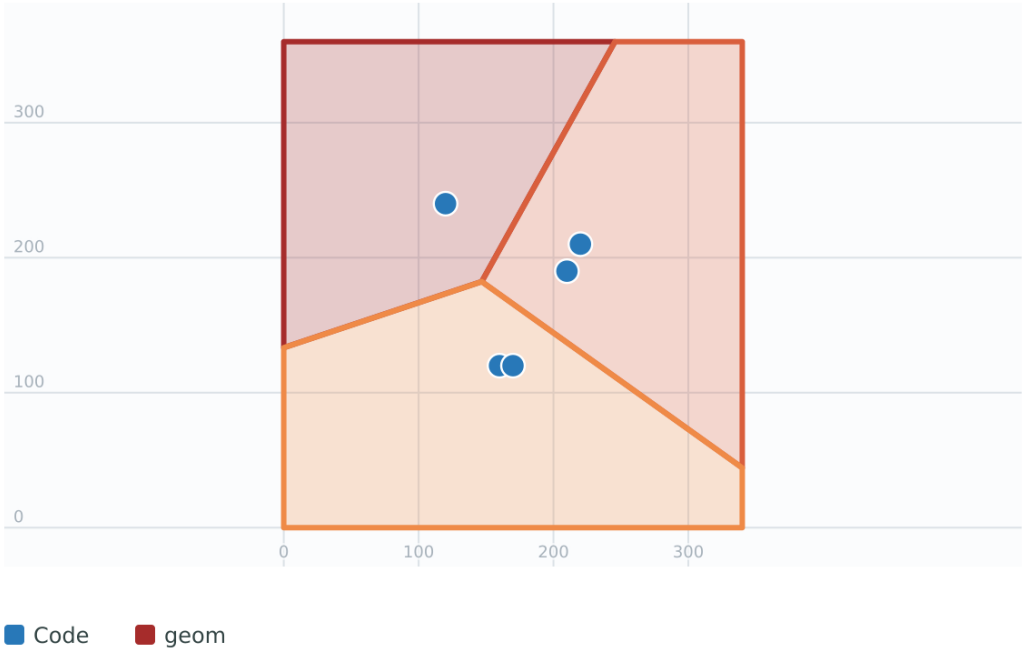


Voronoi diagram with a tolerance of 30 units.

Code

```
SELECT ST_VoronoiPolygons('MULTIPOINT (50 30,60 30,100 100,10 150,110 120)'::geometry,
            30) AS geom;
```

Figure



Voir aussi

ST_DelaunayTriangles, ST_VoronoiLines

7.15 Couvertures

7.15.1 ST_CoverageInvalidEdges

ST_CoverageInvalidEdges — Fonction window qui trouve les endroits où les polygones ne forment pas une couverture valide.

Synopsis

geometry **ST_CoverageInvalidEdges**(geometry winset geom, float8 tolerance = 0);

Description

Une fonction window qui vérifie si les polygones de la partition window forment une couverture polygonale valide. Elle renvoie des indicateurs linéaires indiquant l'emplacement des arêtes non valides (le cas échéant) dans chaque polygone.

Un ensemble de polygones valides est une couverture valide si les conditions suivantes sont remplies :

— **Non-overlapping** - les polygones ne se chevauchent pas (leurs intérieurs ne s'intersectent pas)
— **Edge-Matched** - les sommets situés le long d'arêtes communes sont identiques

En tant que fonction window, une valeur est renvoyée pour chaque polygone d'entrée. Pour les polygones qui ne respectent pas une ou plusieurs des conditions de validité, la valeur renvoyée est un MULTILINESTRING contenant les arêtes problématiques. Les polygones valides du point de vue de la couverture renvoient la valeur NULL. Les géométries non polygonales ou vides produisent également des valeurs NULL.

Les conditions permettent à une couverture valide de contenir des trous (espaces entre les polygones), pour autant que les polygones environnants soient adaptés aux bords. Toutefois, des espaces très étroits ne sont souvent pas souhaitables. Si le paramètre *tolérance* est spécifié avec une distance non nulle, les arêtes formant des espaces plus étroits seront également renvoyées comme non valides.

Les polygones dont la validité de la couverture est vérifiée doivent également être des géométries valides. Ceci peut être vérifié avec ST_IsValid.

Disponibilité : 3.4.0

Nécessite GEOS >= 3.12.0

Exemples

Invalid edges caused by overlap and non-matching vertices.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190,30 160,40 110,100 70,120 10,10 10,10 190))'::geometry),
  (2, 'POLYGON ((100 190,10 190,30 160,40 110,50 80,74 110.5,100 130,140 120,140 160,100 ↩
    190))'::geometry),
  (3, 'POLYGON ((140 190,190 190,190 80,140 80,140 190))'::geometry),
  (4, 'POLYGON ((180 40,120 10,100 70,140 80,190 80,180 40))'::geometry)
)
SELECT id, ST_CoverageInvalidEdges(geom) OVER ()
  FROM coverage;
```

Figure



This example tests the entire table for coverage validity.

Code

```
SELECT true = ALL (
    SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
    FROM coverage
    );
```

Voir aussi

ST_IsValid, ST_CoverageUnion, ST_CoverageClean, ST_CoverageSimplify

7.15.2 ST_CoverageClean

ST_CoverageClean — Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.

Synopsis

geometry **ST_CoverageClean**(geometry winset geom, float8 gapMaximumWidth = 0, float8 snappingDistance = -1, text overlapMergeStrategy = 'MERGE_LONGEST_BORDER');

Description

A window function which adjusts the edges of a set of valid polygonal geometries to produce a clean coverage. Cleaning involves:

— snapping vertices and edges to remove small discrepancies and ensure common edges are identically noded
— merging overlaps into a parent polygon
— merging narrow gaps into adjacent polygons

gapMaximumWidth controls which gaps between polygons are merged. Gaps with width <= this distance are merged into an adjacent polygon.

snappingDistance controls snapping of vertices and edges. The default (-1) automatically determines a snapping distance based on the input extent. Set to 0.0 to turn off snapping.

overlapMergeStrategy specifies how overlaps are merged into a parent polygon:

— MERGE_LONGEST_BORDER - merges into polygon with longest common border
— MERGE_MAX_AREA - merges into polygon with maximum area
— MERGE_MIN_AREA - merges into polygon with minimum area
— MERGE_MIN_INDEX - merges into polygon with smallest input index (defined by order of input polygons)

The result is a clean polygonal coverage that will pass validation by ST_CoverageInvalidEdges and can be input to coverage processing functions.



Note
To aid in determining a maximum gap width, gaps can be computed by cleaning with `gapMaximumWidth = > 0` and using ST_CoverageUnion to union the result coverage. Holes in the union correspond to gaps in the original dataset. Gap widths can be measured by extracting the holes as polygons and running ST_MaximumInscribedCircle on them; the gap width is twice the computed radius.

Availability: 3.6.0 - requires GEOS >= 3.14.0

Exemples

Clean a coverage containing overlaps and narrow gaps, and compare the cleaned polygons with the original invalid edges.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190,30 160,27 134.5,40 110,122 47,120 10,10 10,10 190))'::geometry),
  (2, 'POLYGON ((150 190,10 190,30 160,50 140,40 110,50 80,130 70,135 111,140 130,140 ↩
    160,150 190))'::geometry),
  (3, 'POLYGON ((140 190,190 190,190 80,140 80,140 190))'::geometry),
  (4, 'POLYGON ((190 10,120 10,97 77,160 90,170 70,190 80,190 10))'::geometry)
), processed AS (
  SELECT id,
         ST_CoverageClean(geom, 1) OVER () AS cleaned,
         ST_CoverageInvalidEdges(geom) OVER () AS invalid_edges
  FROM coverage
)
SELECT ST_Collect(cleaned ORDER BY id) AS cleaned,
       ST_Collect(invalid_edges ORDER BY id) AS invalid_edges
FROM processed;
```

Figure



Voir aussi

ST_CoverageInvalidEdges, ST_CoverageUnion, ST_CoverageSimplify, ST_MaximumInscribedCircle

7.15.3 ST_CoverageSimplify

ST_CoverageSimplify — Fonction window qui simplifie les bords d'une couverture polygonale.

Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

Description

Une fonction window qui simplifie les bords des polygones dans une couverture polygonale. La simplification préserve la topologie de la couverture. Cela signifie que les polygones de sortie simplifiés sont cohérents le long des arêtes communes et forment toujours une couverture valide.

La simplification utilise une variante de l'algorithme Visvalingam-Whyatt. Le paramètre `tolérance` a des unités de distance et est approximativement égal à la racine carrée des zones triangulaires à simplifier.

Pour simplifier uniquement les bords "internes" de la couverture (ceux qui sont partagés par deux polygones), définissez le paramètre `simplifyBoundary` à false.

Use this function to simplify adjacent polygon layers when shared boundaries must remain edge-matched. For a single geometry where only self-intersection and ring validity matter, see ST_SimplifyPreserveTopology.



Note
Si l'entrée n'est pas une couverture valide, il peut y avoir des artefacts inattendus dans la sortie (tels que des intersections de frontières, ou des frontières séparées qui semblaient être partagées). Utilisez ST_CoverageInvalidEdges pour déterminer si une couverture est valide.

Disponibilité : 3.4.0

Nécessite GEOS >= 3.12.0

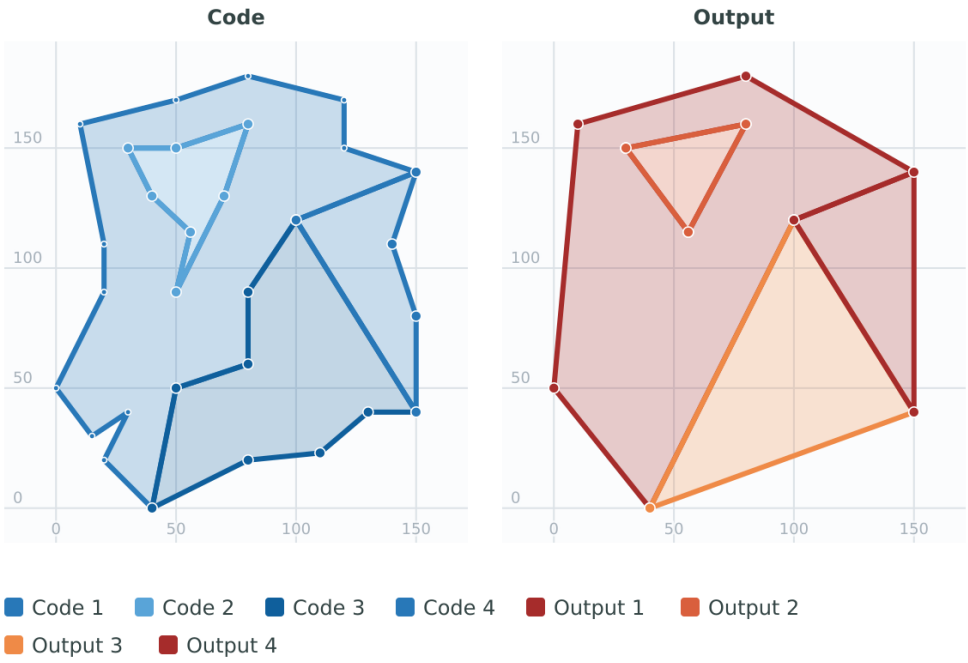
Exemples

Simplify a polygonal coverage while preserving shared boundaries.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150,110 130,90 100,90 70,60 60,50 10,30 30,40 50,25 40,10 60,30 100,30 ↩
    120,20 170,60 180,90 190,130 180,130 160,160 150),(40 160,50 140,66 125,60 100,80 140,90 ↩
     170,60 160,40 160))'::geometry),
  (2, 'POLYGON ((40 160,60 160,90 170,80 140,60 100,66 125,50 140,40 160))'::geometry),
  (3, 'POLYGON ((110 130,160 50,140 50,120 33,90 30,50 10,60 60,90 70,90 100,110 130))':: ↩
    geometry),
  (4, 'POLYGON ((160 150,150 120,160 90,160 50,110 130,160 150))'::geometry)
)
SELECT id, ST_CoverageSimplify(geom, 30) OVER ()
  FROM coverage;
```

Figure



Voir aussi

ST_CoverageInvalidEdges, ST_CoverageUnion, ST_CoverageClean

7.15.4 ST_CoverageEdges

ST_CoverageEdges — Computes the unique edges of a polygonal coverage.

Synopsis

geometry **ST_CoverageEdges**(geometry geom, integer edgetype = 0);
geometry **ST_CoverageEdges**(geometry[] geoms, integer edgetype = 0);

Description

Returns a MultiLineString representing the unique edges of a polygonal coverage. A polygonal coverage is a set of non-overlapping polygons where adjacent polygons have matching vertices along shared edges.

The *edgetype* parameter can be used to select which edges are returned:

— **0 (ALL)** - all unique edges (default)
— **1 (EXTERIOR)** - only exterior edges (non-shared)
— **2 (INTERIOR)** - only interior edges (shared)

Availability: 3.7.0

Requires GEOS >= 3.15.0

Exemples

This example returns all unique edges of a coverage.

Code

```
WITH coverage(geom) AS (
  VALUES (
    'GEOMETRYCOLLECTION(
      POLYGON((0 0,10 0,10 10,0 10,0 0)),
      POLYGON((10 0,20 0,20 10,10 10,10 0))
    )'::geometry
  )
)
SELECT geom AS input_coverage,
       ST_CoverageEdges(geom) AS all_edges
FROM coverage;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------------------
input_coverage | GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10 10,0 10,0 0)),POLYGON((10 0,20 0,20 10,10 10,10 0)))
all_edges      | MULTILINESTRING((10 0,10 10),(10 10,0 10,0 0,10 0),(10 0,20 0,20 10,10 10))
```

This example returns only the interior shared edges of a coverage.

Code

```
WITH coverage(geom) AS (
  VALUES (
    'GEOMETRYCOLLECTION(
      POLYGON((0 0,10 0,10 10,0 10,0 0)),
      POLYGON((10 0,20 0,20 10,10 10,10 0))
    )'::geometry
  )
)
SELECT geom AS input_coverage,
       ST_CoverageEdges(geom, 2) AS shared_edges
FROM coverage;
```

Output

```
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------------------
input_coverage | GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10 10,0 10,0 0)),POLYGON((10 0,20 0,20 10,10 10,10 0)))
shared_edges   | MULTILINESTRING((10 0,10 10))
```

Voir aussi

ST_CoverageInvalidEdges, ST_CoverageSimplify, ST_CoverageUnion

7.15.5 ST_CoverageUnion

ST_CoverageUnion — Calcule l'union d'un ensemble de polygones formant une couverture en supprimant les arêtes communes.

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);

Description

Une fonction agrégée qui réunit un ensemble de polygones formant une couverture polygonale. Le résultat est une géométrie polygonale couvrant la même zone que la couverture. Cette fonction produit le même résultat que ST_Union, mais utilise la structure de la couverture pour calculer l'union beaucoup plus rapidement.



Note
Si l'entrée n'est pas une couverture valide, il peut y avoir des artefacts inattendus dans la sortie (tels que des polygones non fusionnés ou se chevauchant). Utilisez ST_CoverageInvalidEdges pour déterminer si une couverture est valide.

Disponibilité : 3.4.0 - nécessite GEOS >= 3.8.0

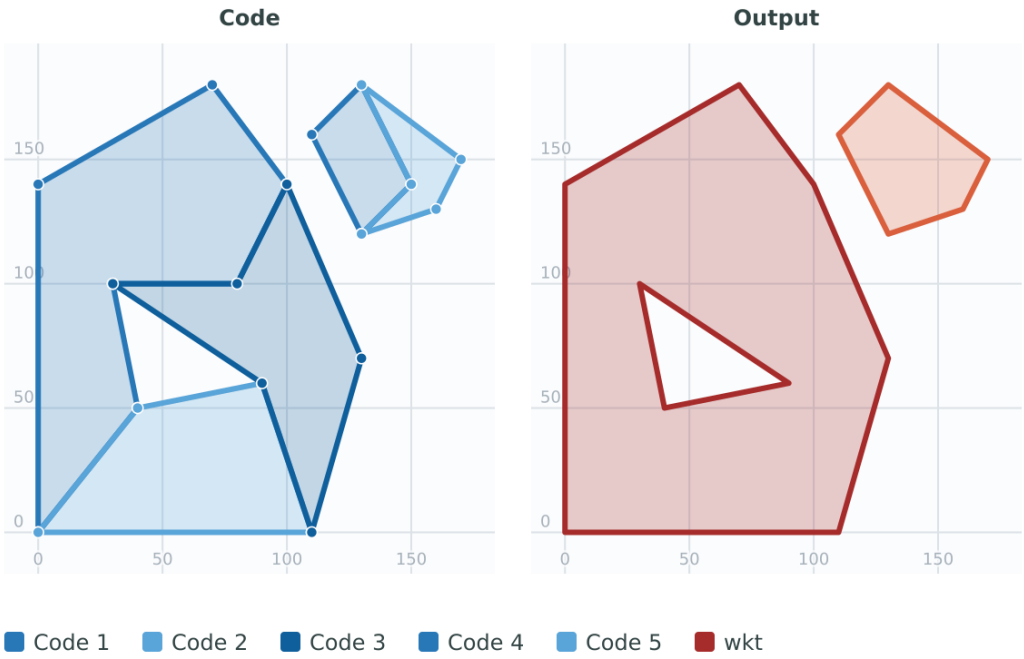
Exemples

Union a polygonal coverage.

Code

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10,10 150,80 190,110 150,90 110,40 110,50 60,10 10))'::geometry),
  (2, 'POLYGON ((120 10,10 10,50 60,100 70,120 10))'::geometry),
  (3, 'POLYGON ((140 80,120 10,100 70,40 110,90 110,110 150,140 80))'::geometry),
  (4, 'POLYGON ((140 190,120 170,140 130,160 150,140 190))'::geometry),
  (5, 'POLYGON ((180 160,170 140,140 130,160 150,140 190,180 160))'::geometry)
)
SELECT ST_CoverageUnion(geom) AS wkt
  FROM coverage;
```

Figure



Voir aussi

ST_CoverageInvalidEdges, ST_CoverageSimplify, ST_CoverageClean, ST_Union

7.16 Transformations affines

7.16.1 ST_Affine

ST_Affine — Appliquer une transformation affine 3D à une géométrie.

Synopsis

geometry **ST_Affine**(geometry geomA, float a, float b, float c, float d, float e, float f, float g, float h, float i, float xoff, float yoff, float zoff);
geometry **ST_Affine**(geometry geomA, float a, float b, float d, float e, float xoff, float yoff);

Description

Applique une transformation affine 3D à la géométrie pour effectuer des opérations telles que la translation, la rotation et la mise à l'échelle en une seule étape.

Version 1: L'appel

Code

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

représente la matrice de transformation

Code

```
/ a  b  c  xoff \
| d  e  f  yoff |
| g  h  i  zoff |
\ 0  0  0     1 /
```

et les sommets sont transformés comme suit :

Code

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

Toutes les fonctions de translation / mise à l'échelle ci-dessous sont exprimées par une telle transformation affine.

Version 2: Applique une transformation affine 2d à la géométrie. L'appel

Code

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

représente la matrice de transformation

Code

```
/  a  b  0  xoff  \       /  a  b  xoff  \
|  d  e  0  yoff  | rsp.  |  d  e  yoff  |
|  0  0  1     0  |       \  0  0     1  /
\  0  0  0     1  /
```

et les sommets sont transformés comme suit :

Code

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

Cette méthode est un sous-cas de la méthode 3D ci-dessus.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Disponibilité : 1.1.2. Le nom a été changé de Affine à ST_Affine dans la version 1.2.2



Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Rotate a 3D line 180 degrees about the z axis. Note this is long-hand for doing ST_Rotate();

Code

```
SELECT ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0, 1, 0, 0, 0)  ↩
    As using_affine,
     ST_Rotate(geom, pi()) As using_rotate
   FROM (SELECT 'LINESTRING(1 2 3,1 4 3)'::geometry As geom) As foo;
```

Figure

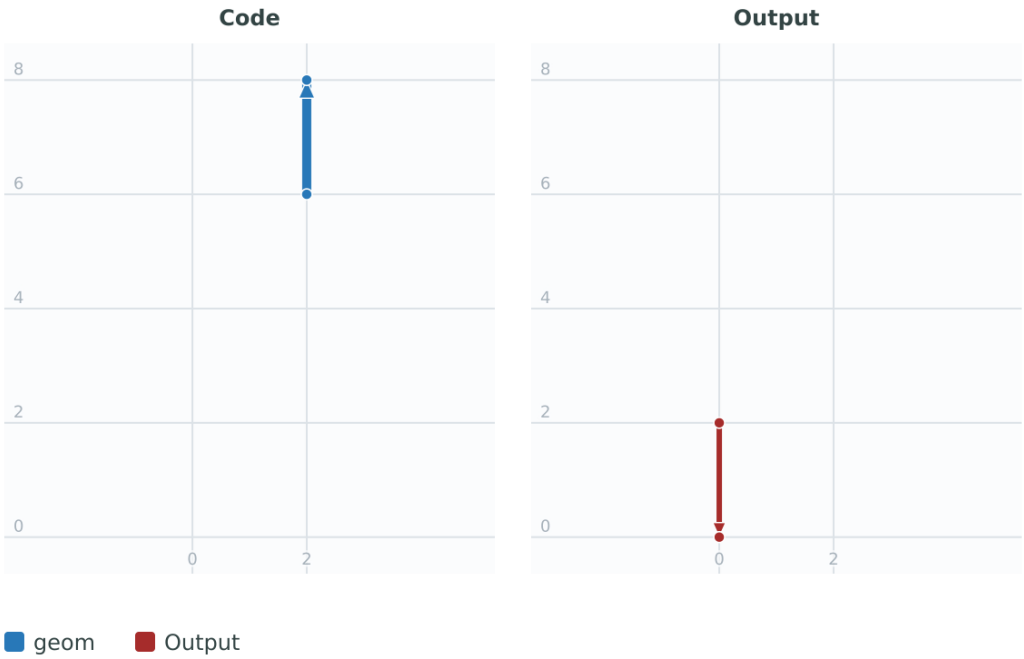


Rotate a 3D line 180 degrees in both the x and z axis.

Code

```
SELECT ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()), 0, sin(pi ↩
    ()), cos(pi()), 0, 0, 0)
    FROM (SELECT 'LINESTRING(1 2 3,1 4 3)'::geometry As geom) As foo;
```

Figure



Voir aussi

ST_Rotate, ST_Scale, ST_Translate, ST_TransScale

7.16.2 ST_Rotate

ST_Rotate — Fait pivoter une géométrie autour d'un point d'origine.

Synopsis

geometry **ST_Rotate**(geometry geomA, float rotRadians);
geometry **ST_Rotate**(geometry geomA, float rotRadians, float x0, float y0);
geometry **ST_Rotate**(geometry geomA, float rotRadians, geometry pointOrigin);

Description

Fait pivoter la géométrie rotRadians dans le sens inverse des aiguilles d'une montre autour du point d'origine. Le point d'origine de la rotation peut être spécifié sous la forme d'une géométrie POINT, ou sous la forme de coordonnées x et y. Si l'origine n'est pas spécifiée, la géométrie est tournée autour de POINT(0 0).

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Amélioration : 2.0.0 des paramètres supplémentaires ont été ajoutés pour spécifier l'origine de la rotation.

Disponibilité : 1.1.2. Nom modifié de Rotate en ST_Rotate dans la version 1.2.2

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

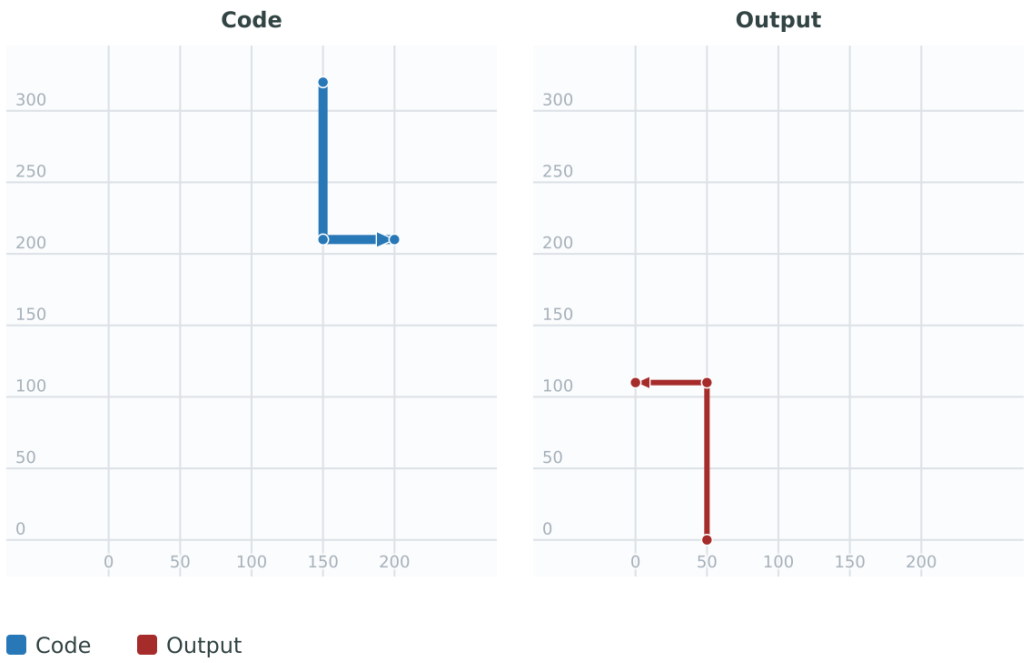
Exemples

This example rotates a geometry by 180 degrees.

Code

```
SELECT ST_Rotate('LINESTRING (50 160,50 50,100 50)'::geometry, pi());
```

Figure

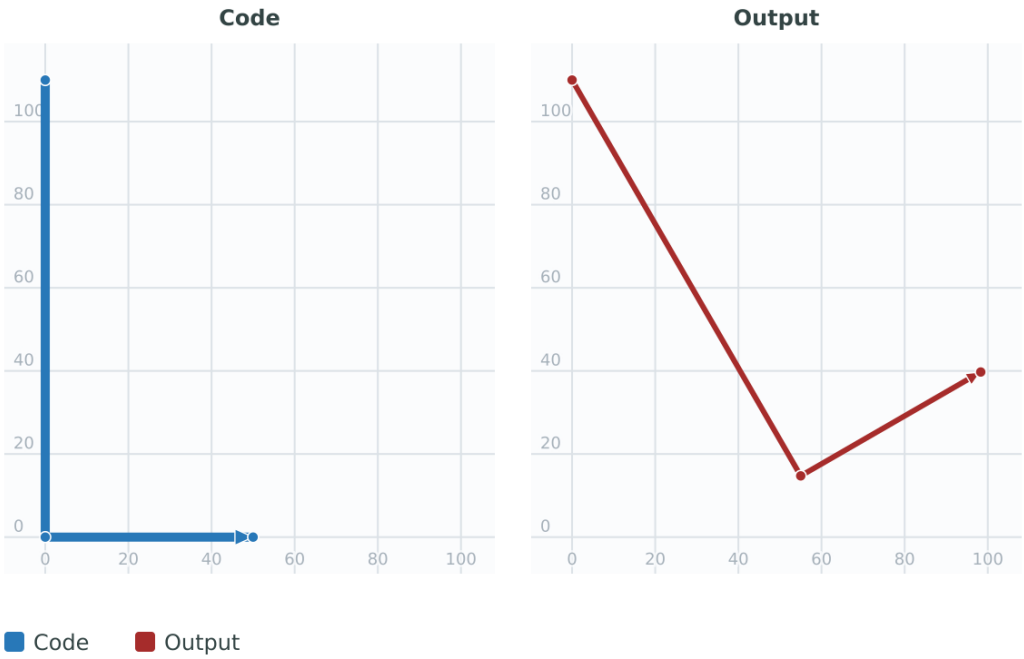


This example rotates a geometry 30 degrees counter-clockwise around x=50, y=160.

Code

```
SELECT ST_Rotate('LINESTRING (50 160,50 50,100 50)'::geometry, pi()/6, 50, 160);
```

Figure

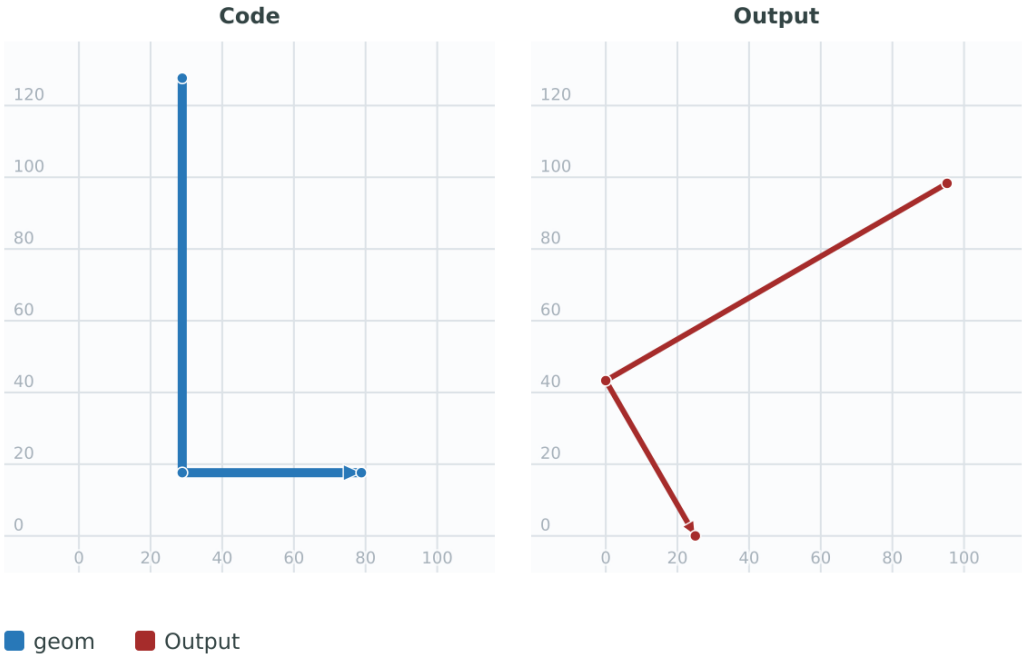


This example rotates a geometry 60 degrees clockwise around its centroid.

Code

```
SELECT ST_Rotate(geom, -pi()/3, ST_Centroid(geom))
FROM (SELECT 'LINESTRING (50 160,50 50,100 50)'::geometry AS geom) AS foo;
```

Figure



Voir aussi

ST_Affine, ST_RotateX, ST_RotateY, ST_RotateZ

7.16.3 ST_RotateX

ST_RotateX — Fait pivoter une géométrie autour de l'axe X.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

Description

Fait pivoter une géométrie geomA - rotRadians autour de l'axe X.



Note
`ST_RotateX(geomA, rotRadians)` est un raccourci pour `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).`

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Disponibilité : 1.1.2. Nom modifié de RotateX en ST_RotateX dans la version 1.2.2


Cette fonction prend en charge les surfaces Polyhedral.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

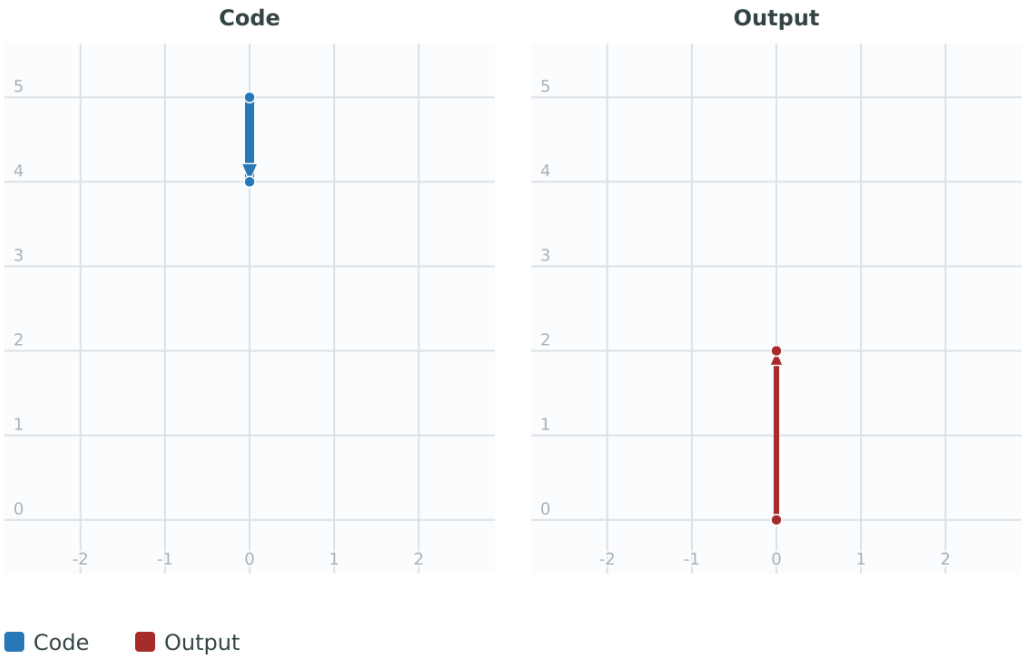
Exemples

This example rotates a line 90 degrees around the X axis.

Code

```
SELECT ST_RotateX('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure



Voir aussi

ST_Affine, ST_RotateY, ST_RotateZ

7.16.4 ST_RotateY

ST_RotateY — Fait pivoter une géométrie autour de l'axe Y.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

Description

Fait pivoter une géométrie geomA - rotRadians autour de l'axe y.



Note

`ST_RotateY(geomA, rotRadians)` est un raccourci pour `ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0)`.

Disponibilité : 1.1.2. Nom modifié de RotateY en ST_RotateY dans la version 1.2.2

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.


Cette fonction prend en charge les surfaces Polyhedral.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.


Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

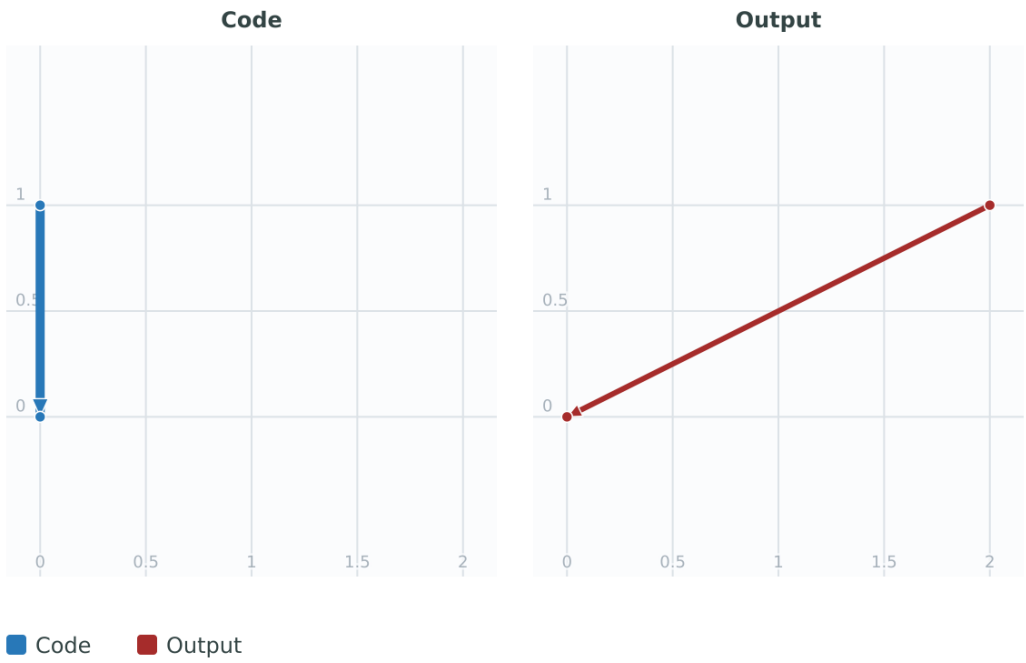
Exemples

This example rotates a line 90 degrees around the Y axis.

Code

```
SELECT ST_RotateY('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure



Voir aussi

ST_Affine, ST_RotateX, ST_RotateZ

7.16.5 ST_RotateZ

ST_RotateZ — Fait pivoter une géométrie autour de l'axe Z.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

Description

Fait pivoter une géométrie geomA - rotRadians autour de l'axe Z.



Note
C'est un synonyme de ST_Rotate

Note
`ST_RotateZ(geomA, rotRadians)` est un raccourci `SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).`

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Disponibilité : 1.1.2. Nom modifié de RotateZ en ST_RotateZ dans la version 1.2.2

Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

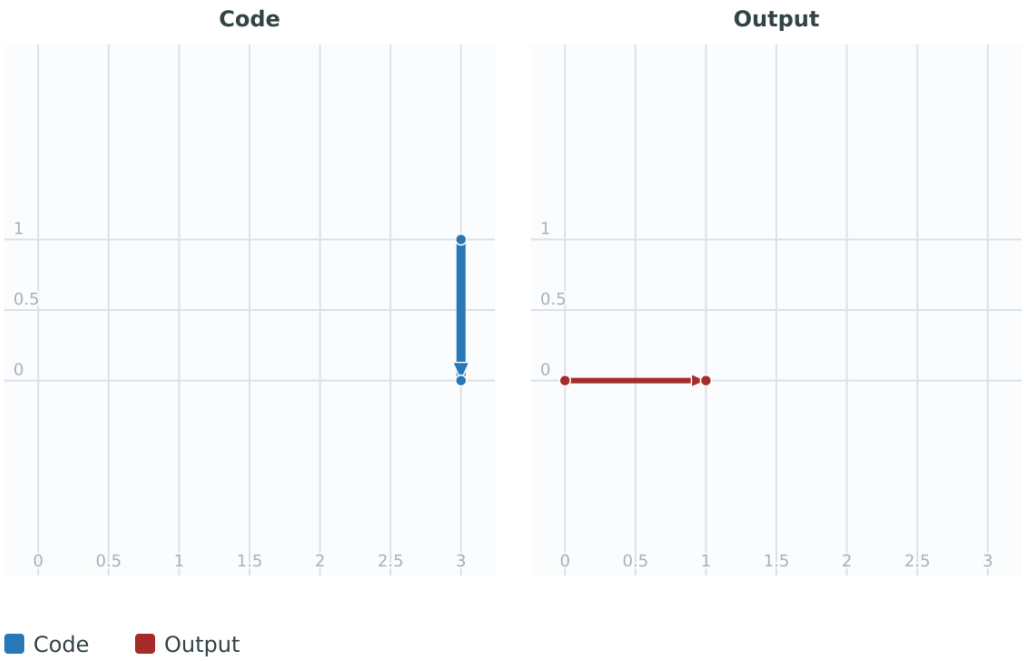
Exemples

This example rotates a line 90 degrees around the Z axis.

Code

```
SELECT ST_RotateZ('LINESTRING(1 2 3,1 1 1)', pi()/2);
```

Figure

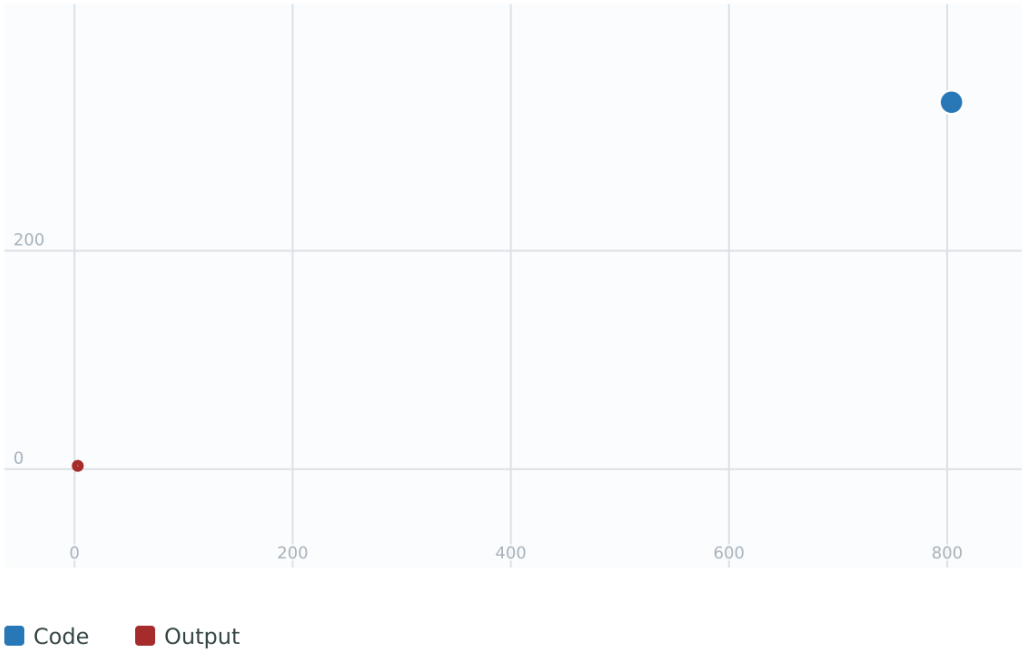


This example rotates a curved circle around the Z axis.

Code

```
SELECT ST_RotateZ(geom, pi()/2)
FROM (SELECT ST_LineToCurve(ST_Buffer('POINT(234 567)'::geometry, 3)) As geom) As foo;
```

Figure



Voir aussi

ST_Affine, ST_RotateX, ST_RotateY

7.16.6 ST_Scale

ST_Scale — Met à l'échelle une géométrie en fonction de facteurs donnés.

Synopsis

geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor);
geometry **ST_Scale**(geometry geom, geometry factor);
geometry **ST_Scale**(geometry geom, geometry factor, geometry origin);

Description

Met la géométrie à l'échelle en multipliant les ordonnées par les paramètres factor correspondants.

La version prenant une géométrie comme paramètre `factor` permet de passer un point 2d, 3dm, 3dz ou 4d pour définir le facteur d'échelle pour toutes les dimensions prises en charge. Les dimensions manquantes dans le point `factor` sont équivalentes à l'absence de mise à l'échelle de la dimension correspondante.

La variante à trois géométries permet de transmettre une "fausse origine" pour la mise à l'échelle. Cela permet une "mise à l'échelle sur place", par exemple en utilisant le centroïde de la géométrie comme fausse origine. Sans fausse origine, la mise à l'échelle s'effectue par rapport à l'origine réelle, de sorte que toutes les coordonnées sont simplement multipliées par le facteur d'échelle.



Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Disponibilité : 1.1.0.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Amélioration : La prise en charge de la mise à l'échelle de toutes les dimensions (paramètre `factor`) a été introduite dans la version 2.2.0.

Amélioration : la prise en charge de la mise à l'échelle par rapport à une origine locale (paramètre `origin`) a été introduite dans la version 2.5.0.

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

 Cette fonction prend en charge les coordonnées M.

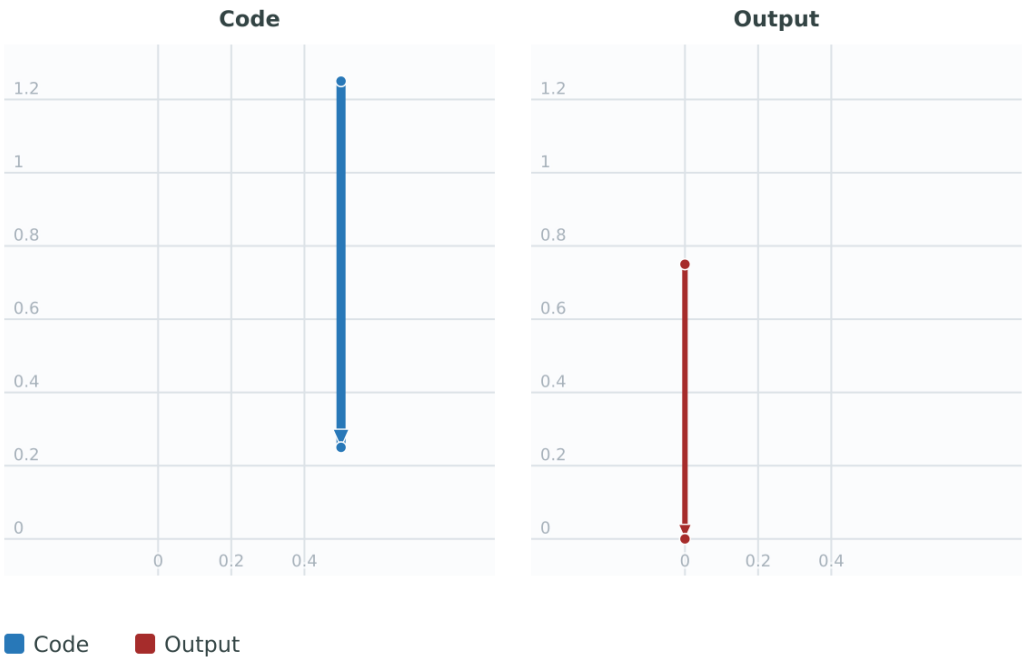
Exemples

This example scales X, Y, and Z values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3,1 1 1)', 0.5, 0.75, 0.8);
```

Figure

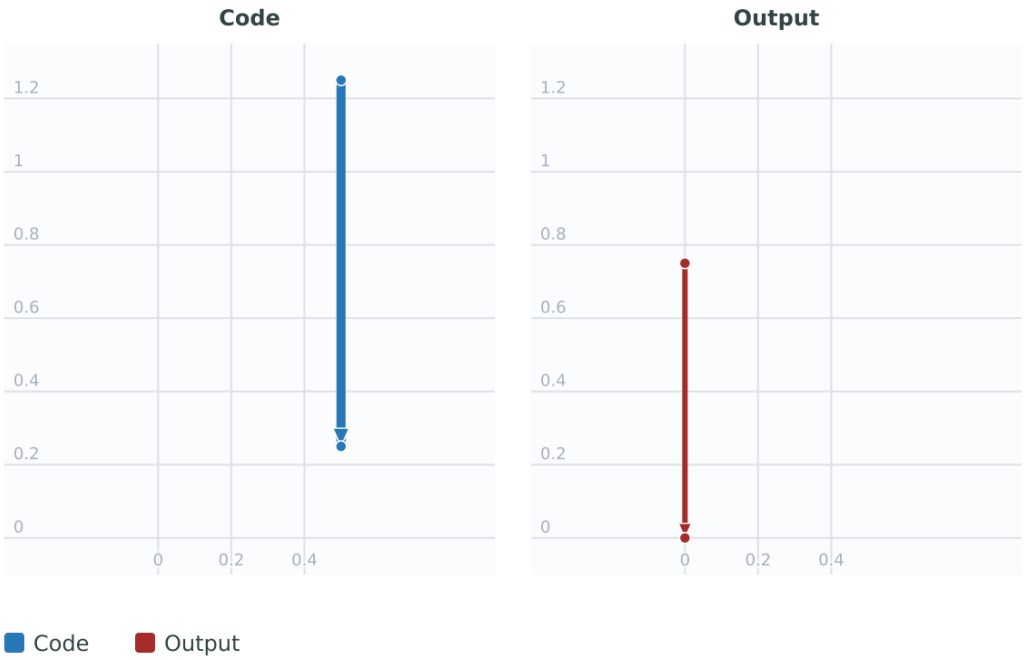


This example scales X and Y values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3,1 1 1)', 0.5, 0.75);
```

Figure

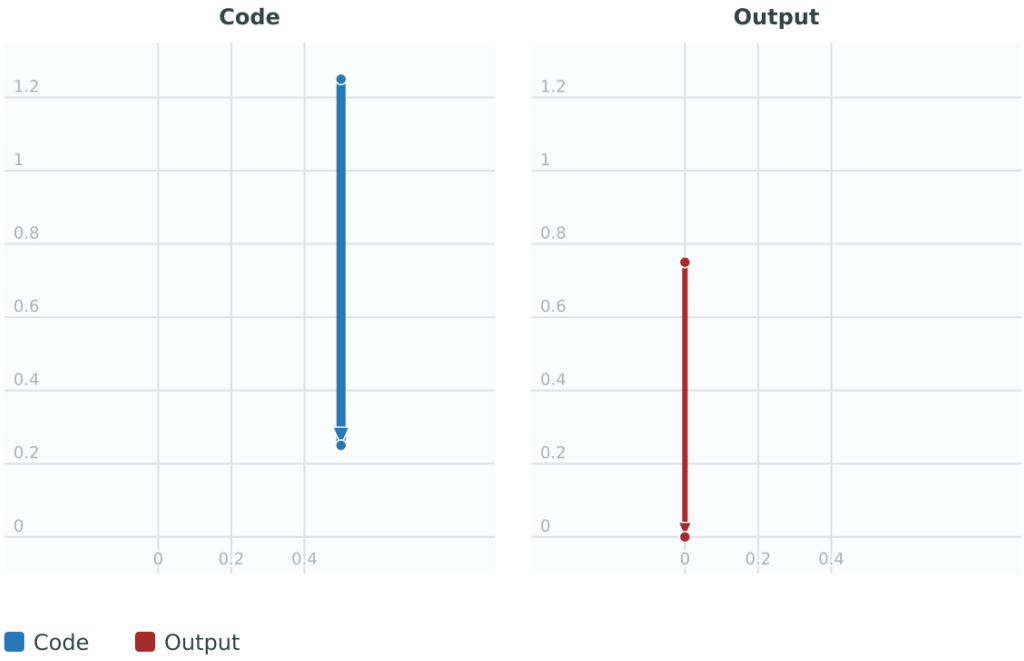


This example scales X, Y, Z, and M values.

Code

```
SELECT ST_Scale('LINESTRING(1 2 3 4,1 1 1 1)',
    ST_MakePoint(0.5, 0.75, 2, -1));
```

Figure

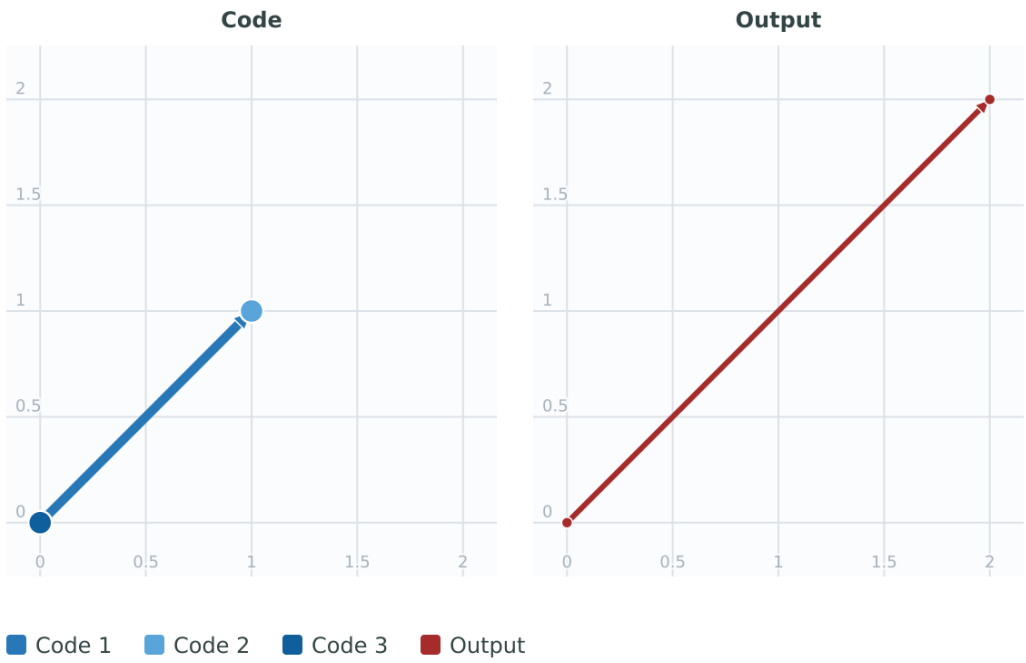


This example scales X and Y values using a false origin.

Code

```
SELECT ST_Scale('LINESTRING(1 1,2 2)', 'POINT(2 2)', 'POINT(1 1)'::geometry);
```

Figure



This example constructs an ellipse centered at (10, 20), with X radius 8, Y radius 4, and rotated 30 degrees counter-clockwise. Scaling and rotation use the center as their origin, so the ellipse keeps both its position and SRID.

Code

```
WITH params AS (
  SELECT
    ST_SetSRID(ST_Point(10, 20), 3857) AS center,
    8.0 AS x_radius,
    4.0 AS y_radius,
    radians(30) AS angle
)
SELECT ST_Rotate(
         ST_Scale(
           ST_Buffer(center, 1, 'quad_segs=32'),
           ST_MakePoint(x_radius, y_radius),
           center
         ),
         angle,
         center
       ) AS ellipse
FROM params;
```

Voir aussi

ST_Affine, ST_Buffer, ST_MakePoint, ST_Rotate, ST_TransScale

7.16.7 ST_Translate

ST_Translate — Traduit une géométrie en fonction de décalages donnés.

Synopsis

geometry **ST_Translate**(geometry g1, float deltax, float deltay);
geometry **ST_Translate**(geometry g1, float deltax, float deltay, float deltaz);

Description

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry. M coordinates are preserved.



Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Disponibilité : 1.2.2

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les coordonnées M.

Exemples

Déplacer un point de 1 degré de longitude

Code

```
SELECT ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)', 4326), 1, 0) AS wgs_transgeomtxt;
```

Output

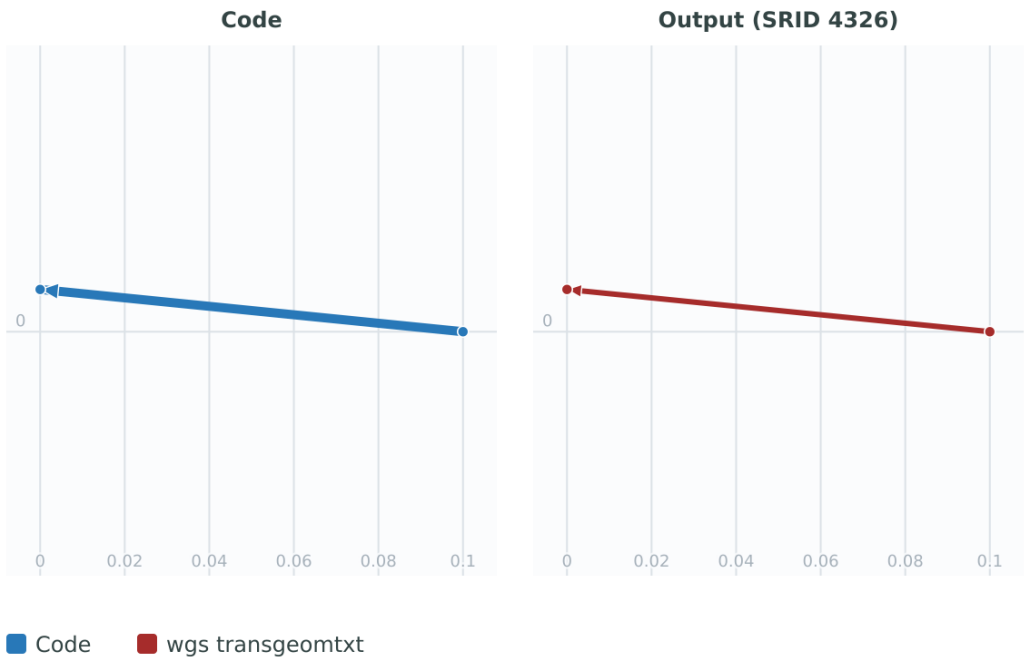
```
POINT(-70.01 42.37)
```

Déplacer une ligne d'un degré de longitude et d'un demi-degré de latitude

Code

```
SELECT ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)', 4326), 1, 0.5)  ↩
    AS wgs_transgeomtxt;
```

Figure



Déplacer un point 3D

Code

```
SELECT ST_Translate(CAST('POINT(0 0 0)' AS geometry), 5, 12, 3);
```

Output

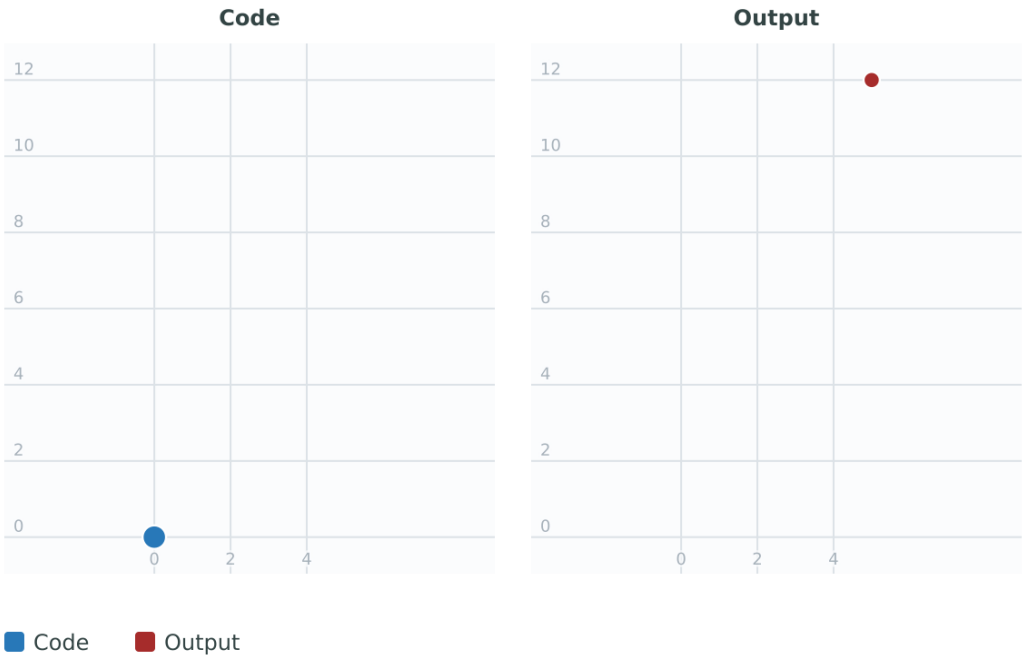
```
POINT(5 12 3)
```

Move points with a measure coordinate

Code

```
SELECT ST_Translate('POINTM(0 0 11)', 5, 12);
```

Figure



Code

```
SELECT ST_Translate('POINT(0 0 7 11)', 5, 12, 3);
```

Output

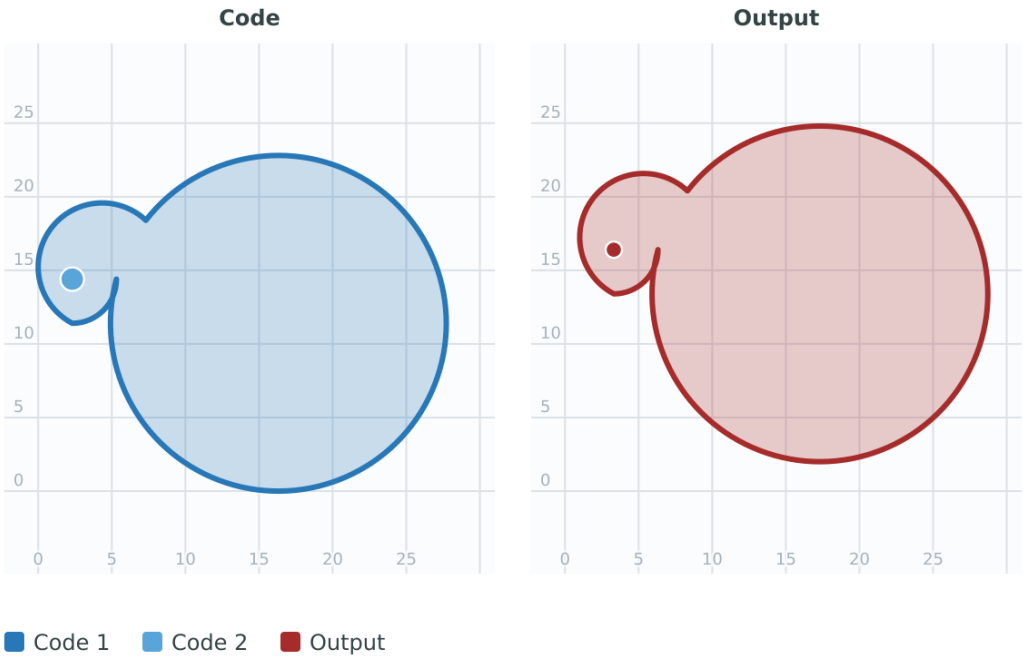
```
POINT(5 12 10 11)
```

Déplacer une courbe et un point

Code

```
SELECT ST_Translate(ST_Collect('CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1 0,-1.121 ↩
    5.1213,6 7,8 9,4 3))', 'POINT(1 3)'), 1, 2);
```

Figure



Voir aussi

ST_Affine, ST_AsText, ST_GeomFromText

7.16.8 ST_TransScale

ST_TransScale — Traduit et met à l'échelle une géométrie en fonction des paramètres offset et factor spécifiés.

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

Description

Traduit la géométrie à l'aide des paramètres deltaX et deltaY, puis la met à l'échelle à l'aide des paramètres XFactor et YFactor, uniquement en 2D.



Note
`ST_TransScale(geomA, deltaX, deltaY, XFactor, YFactor)` est un raccourci pour `ST_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX*XFactor, deltaY*YFactor, 0)`.



Note
Avant la version 1.3.4, cette fonction se bloquait si elle était utilisée avec des géométries contenant des CURVES. Ce problème est corrigé dans la version 1.3.4+

Disponibilité : 1.1.0.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_TransScale('LINESTRING(1 2 3,1 1 1)', 0.5, 1, 1, 2);
```

Figure

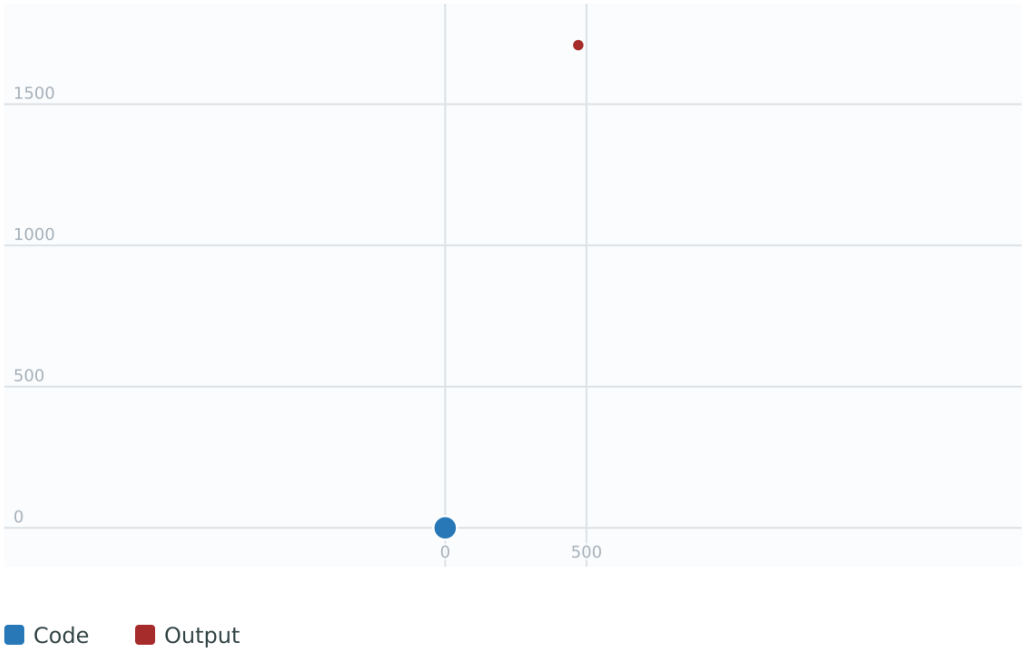


This example buffers a point to approximate a circle, converts it to a curve, translates it by 1,2, and scales it by 3,4.

Code

```
SELECT ST_TransScale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)), 1, 2, 3, 4);
```

Figure



Voir aussi

ST_Affine, ST_Translate

7.17 Fonctions de clustering

7.17.1 ST_ClusterDBSCAN

ST_ClusterDBSCAN — Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie d'entrée en utilisant l'algorithme DBSCAN.

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

Description

Une fonction window qui renvoie un numéro de cluster pour chaque géométrie d'entrée, en utilisant l'algorithme 2D Density-based spatial clustering of applications with noise (DBSCAN). Contrairement à ST_ClusterKMeans, elle ne nécessite pas la spécification du nombre de clusters, mais utilise les paramètres de distance (`eps`) et de densité (`minpoints`) pour déterminer chaque cluster.

Une géométrie d'entrée est ajoutée à un cluster si elle est l'une ou l'autre :

— Une géométrie "core", qui se trouve à `eps` distance d'au moins `minpoints` géométries d'entrée (y compris la sienne) ; ou

— Une géométrie "en bordure", qui se trouve dans `eps` distance d'une géométrie centrale.

Notez que les géométries situés en bordure peuvent se trouver à `eps` distance des géométries centrales dans plus d'un cluster. L'une ou l'autre assignation serait correcte, de sorte que la géométrie de bordure sera arbitrairement assignée à l'un des clusters disponibles. Dans cette situation, il est possible qu'un cluster correct soit généré avec moins de géométries `minpoints`.

Pour garantir une affectation déterministe des géométries de bordure (de sorte que des appels répétés à ST_ClusterDBSCAN produisent des résultats identiques), utilisez une clause `ORDER BY` dans la définition de la fonction window. Les affectations de clusters ambigus peuvent différer d'autres implémentations DBSCAN.



Note
Les géométries qui ne répondent pas aux critères d'appartenance à un cluster se voient attribuer un numéro de cluster NULL.

Disponibilité : 2.3.0

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Cluster polygons within 50 units of each other, requiring at least two polygons per cluster. The isolated polygon is reported as noise.

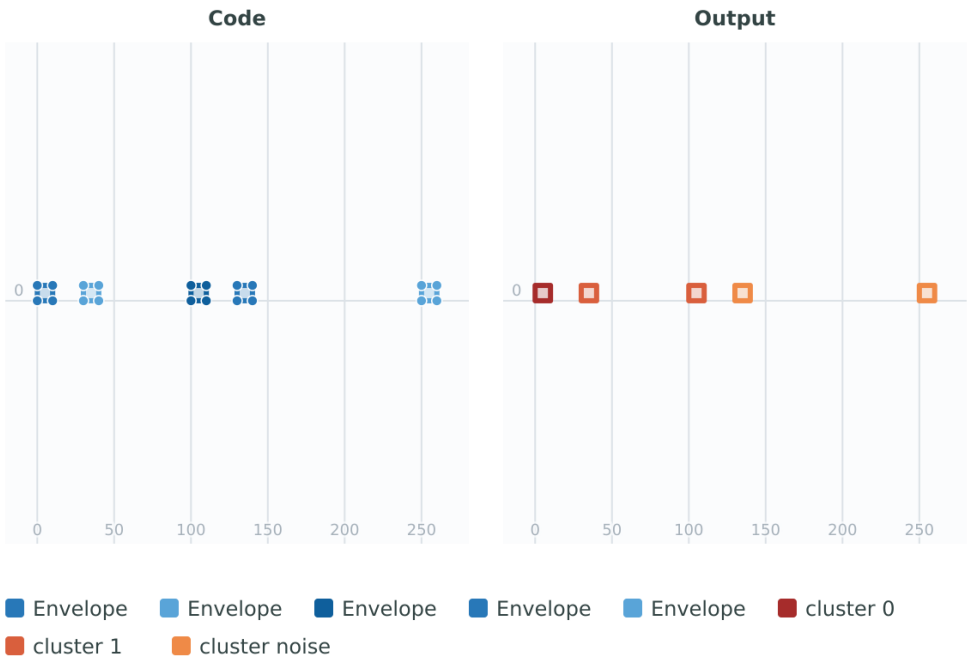
Code

```
WITH input(name, geom) AS (
  VALUES
    ('A1', ST_MakeEnvelope(0, 0, 10, 10)),
    ('A2', ST_MakeEnvelope(30, 0, 40, 10)),
    ('B1', ST_MakeEnvelope(100, 0, 110, 10)),
    ('B2', ST_MakeEnvelope(130, 0, 140, 10)),
    ('noise', ST_MakeEnvelope(250, 0, 260, 10))
), clustered AS (
  SELECT name,
         geom,
         ST_ClusterDBSCAN(geom, eps => 50, minpoints => 2)
           OVER (ORDER BY name) AS cid
  FROM input
)
SELECT COALESCE(cid::text, 'noise') AS cluster,
       string_agg(name, ',' ORDER BY name) AS members,
       ST_Collect(geom ORDER BY name) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid NULLS LAST;
```

Output

```
0 | A1,A2 | MULTIPOLYGON(((0 0,0 10,10 10,10 0,0 0)),((30 0,30 10,40 10,40 0,30 0)))
1 | B1,B2 | MULTIPOLYGON(((100 0,100 10,110 10,110 0,100 0)),((130 0,130 10,140 10,140 0,130 0)))
noise | noise | MULTIPOLYGON(((250 0,250 10,260 10,260 0,250 0)))
```

Figure



Un exemple montrant la combinaison de parcelles ayant le même numéro de cluster dans des collections géométriques.

Code

```
SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
    SELECT parcel_id, ST_ClusterDBSCAN(geom, eps => 0.5, minpoints => 5) over () AS cid,  ↩
    geom
    FROM parcels) sq
GROUP BY cid;
```

Voir aussi

ST_DWithin, ST_ClusterKMeans, ST_ClusterIntersecting, ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterWithinWin

7.17.2 ST_ClusterIntersecting

ST_ClusterIntersecting — Fonction d'agrégation qui regroupe les géométries en entrée en ensembles connectés.

Synopsis

geometry[] **ST_ClusterIntersecting**(geometry set g);

Description

Une fonction agrégée qui renvoie un tableau de GeometryCollections partitionnant les géométries d'entrée en clusters connectés et disjoints. Chaque géométrie d'un cluster intersecte au moins une autre géométrie du cluster et n'intersecte aucune géométrie d'autre cluster.

Disponibilité : 2.2.0

Exemples

Code

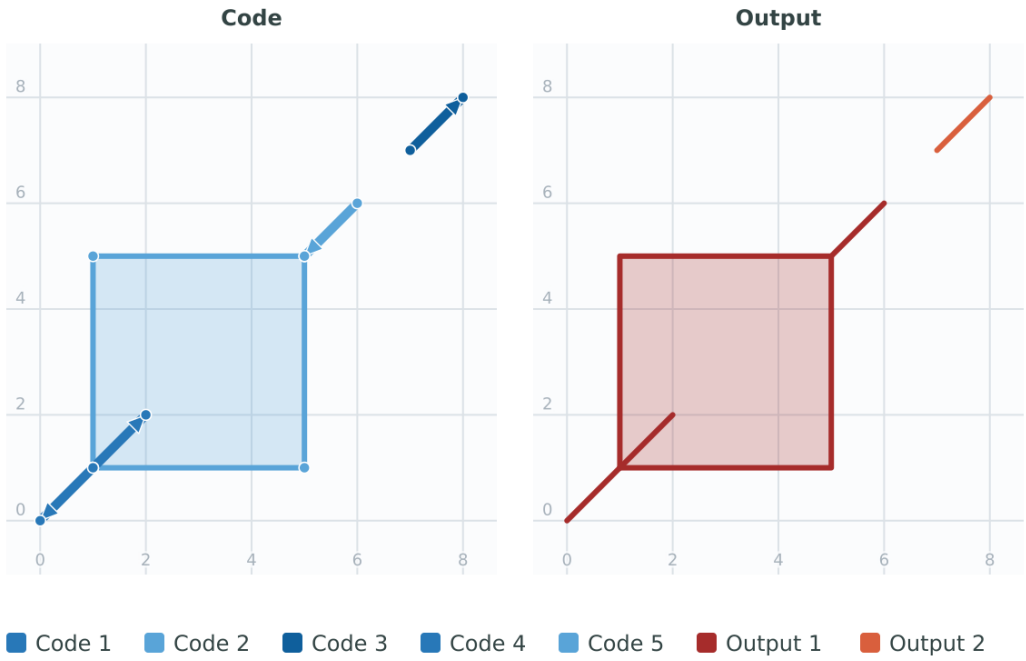
```
WITH testdata AS
  (SELECT unnest(
            ARRAY['LINESTRING (0 0,1 1)'::geometry,
            'LINESTRING (5 5,4 4)'::geometry,
            'LINESTRING (6 6,7 7)'::geometry,
            'LINESTRING (0 0,-1 -1)'::geometry,
            'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry]) AS geom)

SELECT unnest(ST_ClusterIntersecting(geom)) FROM testdata;
```

Output

```
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Figure



Voir aussi

ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterWithinWin

7.17.3 ST_ClusterIntersectingWin

ST_ClusterIntersectingWin — Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, en regroupant les géométries en entrée en ensembles connectés.

Synopsis

integer **ST_ClusterIntersectingWin**(geometry winset geom);

Description

Une fonction window qui construit des clusters connectés de géométries qui se croisent. Il est possible de parcourir toutes les géométries d'un cluster sans quitter le cluster. La valeur de retour est le numéro du cluster à laquelle participe l'argument géométrie, ou null pour les entrées nulles.

Disponibilité : 3.4.0

Exemples

Return an explicit cluster id for each input geometry. Using the conventional `cid` alias keeps the geometry column ready for color-by-cluster rendering.

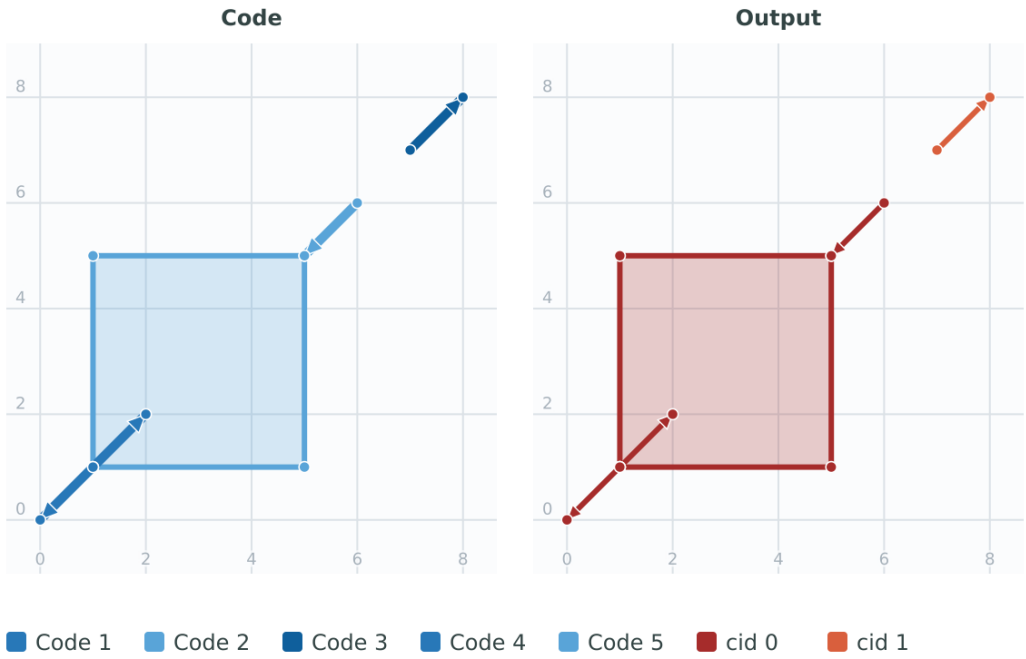
Code

```
WITH testdata(id, geom) AS (
  VALUES (1, 'LINESTRING (0 0,1 1)'::geometry),
         (2, 'LINESTRING (5 5,4 4)'::geometry),
         (3, 'LINESTRING (6 6,7 7)'::geometry),
         (4, 'LINESTRING (0 0,-1 -1)'::geometry),
         (5, 'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry)
), clustered AS (
  SELECT id,
         geom,
         ST_ClusterIntersectingWin(geom) OVER () AS cid
  FROM testdata
)
SELECT id,
       geom AS geom_wkt,
       cid
FROM clustered
ORDER BY id;
```

Output

```
id |            geom_wkt             | cid
----+---------------------------------+-----
  1 | LINESTRING(0 0,1 1)             |   0
  2 | LINESTRING(5 5,4 4)             |   0
  3 | LINESTRING(6 6,7 7)             |   1
  4 | LINESTRING(0 0,-1 -1)           |   0
  5 | POLYGON((0 0,4 0,4 4,0 4,0 0)) |   0
```

Figure



Voir aussi

ST_ClusterIntersecting, ST_ClusterWithin, ST_ClusterWithinWin

7.17.4 ST_ClusterRelateWin

ST_ClusterRelateWin — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.

Synopsis

integer **ST_ClusterRelateWin**(geometry winset geom, text relate_matrix);

Description

A window function that builds connected clusters of geometries that intersect. Geometries are added to a cluster if they share a pairwise DE9IM relationship with another member of the cluster. With this function it is possible to build a cluster of all objects that touch at boundaries, but exclude those that merely overlap.

Availability: 3.7.0

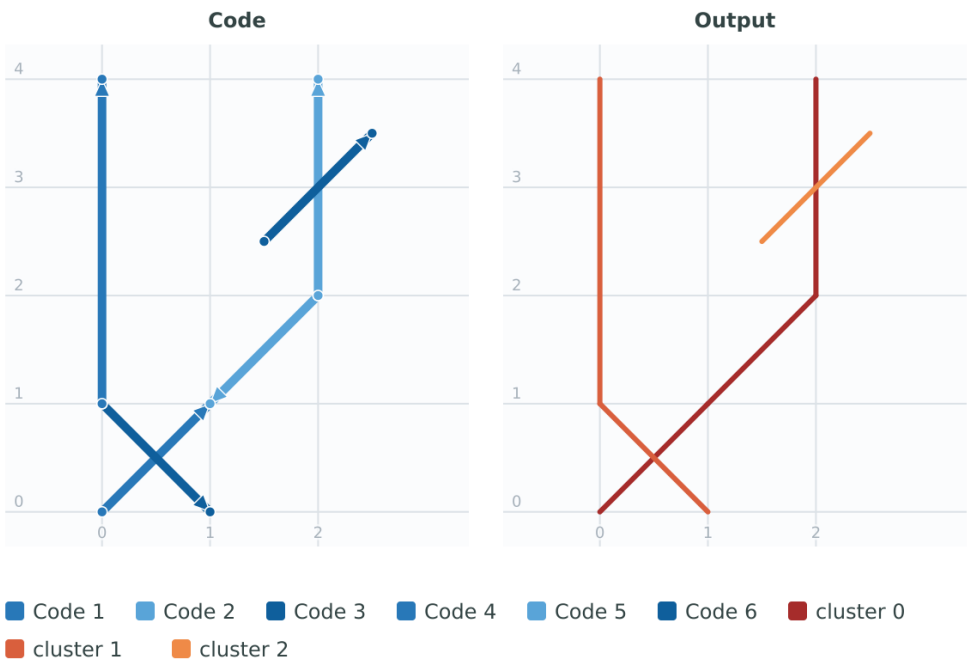
Exemples

This collection of line strings would form a single cluster using ST_ClusterIntersectingWin, but using ST_ClusterRelateWin can be clustered into three groups that connect only via their end points.

Code

```
WITH data(id, geom) AS (
  VALUES (1, 'LINESTRING(0 0,1 1)'::geometry),
         (2, 'LINESTRING(2 2,1 1)'::geometry),
         (3, 'LINESTRING(0 1,1 0)'::geometry),
         (4, 'LINESTRING(0 1,0 4)'::geometry),
         (5, 'LINESTRING(2 2,2 4)'::geometry),
         (6, 'LINESTRING(1.5 2.5,2.5 3.5)'::geometry)
), clustered AS (
  SELECT id, geom,
         ST_ClusterRelateWin(geom, '****0****') OVER () AS cluster
  FROM data
)
SELECT cluster,
       string_agg(id::text, ',' ORDER BY id) AS members,
       ST_Collect(geom ORDER BY id) AS cluster_geom
FROM clustered
GROUP BY cluster
ORDER BY cluster;
```

Figure



Voir aussi

ST_Relate, ST_ClusterIntersectingWin, ST_ClusterWithinWin

7.17.5 ST_MinimumSpanningTree

ST_MinimumSpanningTree — Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

Synopsis

integer **ST_MinimumSpanningTree**(geometry winset geom);

Description

A window function that builds connected graphs of line strings based on the Minimum Spanning Tree (MST) of the input geometries. The return value is the cluster number that the geometry argument participates in, or zero if it is not part of the minimum tree.

The Minimum Spanning Tree connects all geometries in the window partition such that the total length of the connecting lines is minimized. If the graph is not fully connected (e.g. infinite distance between some geometries), it produces a Minimum Spanning Forest, and each connected component is assigned a unique tree ID.

Availability: 3.7.0

Requires GEOS >= 3.15.0

Exemples

These six edges describe a square with both diagonals. The minimum spanning tree keeps three edges and excludes the rest.

Code

```
WITH edges(id, geom) AS (
  VALUES
    (1, 'LINESTRING(0 0,1 0)'::geometry),
    (2, 'LINESTRING(0 0,0 1)'::geometry),
    (3, 'LINESTRING(1 1,0 1)'::geometry),
    (4, 'LINESTRING(1 1,1 0)'::geometry),
    (5, 'LINESTRING(0 0,1 1)'::geometry),
    (6, 'LINESTRING(1 0,0 1)'::geometry)
), marked AS (
  SELECT id,
         geom,
         ST_MinimumSpanningTree(geom) OVER (ORDER BY id) AS tree_id
  FROM edges
)
SELECT string_agg(id::text, ',' ORDER BY id)
         FILTER (WHERE tree_id > 0) AS tree_edge_ids,
       ST_Collect(geom ORDER BY id)
         FILTER (WHERE tree_id > 0) AS tree_edges,
       string_agg(id::text, ',' ORDER BY id)
         FILTER (WHERE tree_id = 0) AS excluded_edge_ids,
       ST_Collect(geom ORDER BY id)
         FILTER (WHERE tree_id = 0) AS excluded_edges
FROM marked;
```

Output

```
-[ RECORD 1 ]-----
tree_edge_ids     | 1,2,3
tree_edges        | MULTILINESTRING((0 0,1 0),(0 0,0 1),(1 1,0 1))
excluded_edge_ids | 4,5,6
excluded_edges    | MULTILINESTRING((1 1,1 0),(0 0,1 1),(1 0,0 1))
```

Voir aussi

ST_ClusterDBSCAN, ST_ClusterKMeans, ST_ClusterIntersectingWin

7.17.6 ST_ClusterKMeans

ST_ClusterKMeans — Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée en utilisant l'algorithme K-means.

Synopsis

integer **ST_ClusterKMeans**(geometry winset geom , integer k , float8 max_radius);

Description

Renvoie le numéro de cluster K-means pour chaque géométrie en entrée. La distance utilisée pour le clustering est la distance entre les centroïdes pour les géométries 2D, et la distance entre les centres des boîtes de délimitation pour les géométries 3D. Pour les entrées POINT, la coordonnée M sera traitée comme le poids de l'entrée et doit être supérieure à 0.

`max_radius`, si elle est définie, fera en sorte que ST_ClusterKMeans génère plus de clusters que `k` en s'assurant qu'aucun cluster en sortie n'a un rayon plus grand que `max_radius`. Ceci est utile dans l'analyse d'accessibilité.

Amélioré : 3.2.0 Support pour `max_radius`

Amélioration : 3.1.0 Prise en charge des géométries et des poids en 3D

Disponibilité : 2.3.0

Exemples

Define the parcel geometries once in a CTE and reuse them across the executable examples.

Parcels color-coded by cluster number (`cid`).

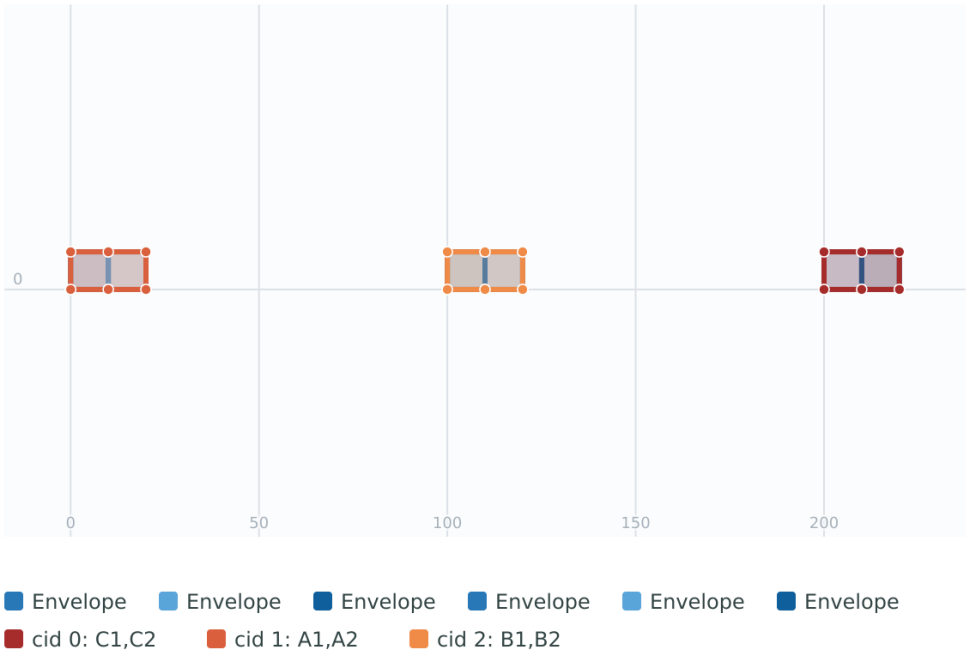
Code

```
WITH parcels(parcel_id, geom) AS (
  VALUES
    ('A1', ST_MakeEnvelope(0, 0, 10, 10)),
    ('A2', ST_MakeEnvelope(10, 0, 20, 10)),
    ('B1', ST_MakeEnvelope(100, 0, 110, 10)),
    ('B2', ST_MakeEnvelope(110, 0, 120, 10)),
    ('C1', ST_MakeEnvelope(200, 0, 210, 10)),
    ('C2', ST_MakeEnvelope(210, 0, 220, 10))
), clustered AS (
  SELECT parcel_id,
         geom,
         ST_ClusterKMeans(geom, 3) OVER (ORDER BY parcel_id) AS cid
  FROM parcels
)
SELECT cid,
       string_agg(parcel_id, ',' ORDER BY parcel_id) AS parcels,
       ST_Union(geom) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid;
```

Output

```
0 | C1,C2 | POLYGON((200 10,210 10,220 10,220 0,210 0,200 0,200 10))
1 | A1,A2 | POLYGON((0 10,10 10,20 10,20 0,10 0,0 0,0 10))
2 | B1,B2 | POLYGON((100 10,110 10,120 10,120 0,110 0,100 0,100 10))
```

Figure



Partitionnement des clusters de parcelles par type :

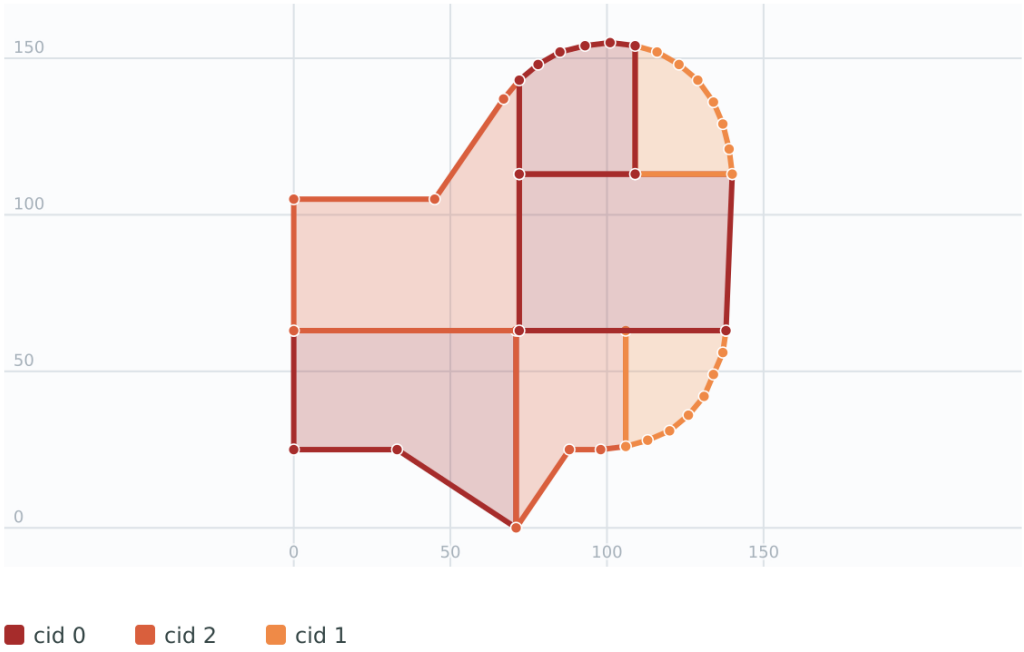
Code

```
WITH parcel_geoms AS (
  SELECT geom
  FROM ST_Subdivide(
    ST_Buffer('SRID=3857;LINESTRING(40 100,98 100,100 150,60 90)'::geometry,
              40, 'endcap=square'),
    12) AS geom
), parcels AS (
  SELECT lpad(row_number() OVER (ORDER BY ST_YMin(geom), ST_XMin(geom))::text, 3, '0') AS
    parcel_id,
         geom,
         ('{residential,commercial}'::text[])[1 + mod(row_number() OVER (ORDER BY ST_YMin(
    geom), ST_XMin(geom)), 2)] AS type
  FROM parcel_geoms
)
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid,
       parcel_id,
       type,
       ST_SnapToGrid(geom, 1) AS parcel_geom
FROM parcels
ORDER BY type, parcel_id;
```

Output

```
cid | parcel_id |    type     | parcel_geom
----+-----------+-------------+-----------------------------------------------------------
   0 | 001       | commercial  | POLYGON((33 60,0 60,0 98,71 98,71 35,33 60))
   2 | 003       | commercial  | POLYGON((106 98,138 98,137 91,134 84,131 77,126 71,120 66,113 63,106 61,106 98))
   2 | 005       | commercial  | POLYGON((140 148,138 98,72 98,72 148,140 148))
   1 | 007       | commercial  | POLYGON((109 189,116 187,123 183,129 178,134 171,137 164,139 156,140 148,109 148,109
     189))
   0 | 002       | residential | POLYGON((71 98,106 98,106 61,98 60,88 60,71 35,71 98))
   2 | 004       | residential | POLYGON((0 98,0 140,45 140,67 172,72 178,72 98,0 98))
   1 | 006       | residential | POLYGON((72 178,78 183,85 187,93 189,101 190,109 189,109 148,72 148,72 178))
```

Figure



Example: Clustering a preaggregated planetary-scale data population dataset using 3D clustering and weighting. Identify at least 20 regions based on Kontur Population Data that do not span more than 3000 km from their center:

Code

```
CREATE TABLE kontur_population_3000km_clusters AS
SELECT
    geom,
    ST_ClusterKMeans(
        ST_Force4D(
            ST_Transform(ST_Force3D(geom), 4978),
            mvalue => population
        ),
        20,
        max_radius => 3000000
    ) OVER () AS cid
FROM kontur_population;
```

The clustering produces 46 regions. Clusters are centered at well-populated regions such as New York and Moscow. Greenland forms one cluster, several island clusters span the antimeridian, and cluster edges follow the Earth's curvature.



Kontur population clustered with 3000 km maximum radius.

Cluster weighted city populations on the Earth in geocentric coordinates. The maximum radius is specified in meters, while the returned geometries stay in WGS 84 for display.

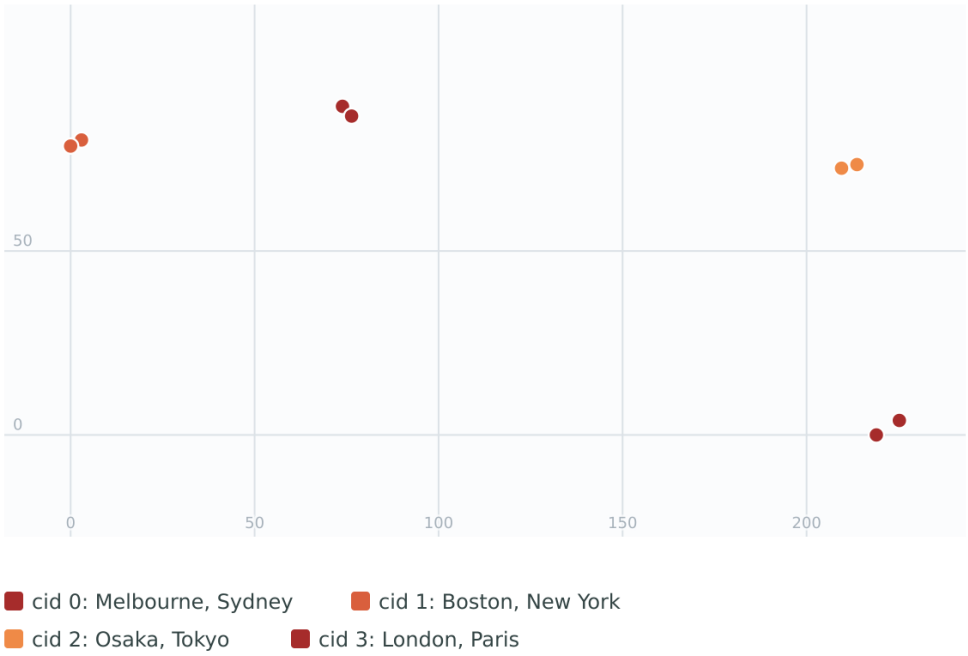
Code

```
WITH population(place, population, geom) AS (
  VALUES
    ('Boston', 5.0, ST_SetSRID(ST_Point(-71.06, 42.36), 4326)),
    ('New York', 20.0, ST_SetSRID(ST_Point(-74.01, 40.71), 4326)),
    ('London', 15.0, ST_SetSRID(ST_Point(-0.13, 51.51), 4326)),
    ('Paris', 11.0, ST_SetSRID(ST_Point(2.35, 48.86), 4326)),
    ('Osaka', 19.0, ST_SetSRID(ST_Point(135.50, 34.69), 4326)),
    ('Tokyo', 37.0, ST_SetSRID(ST_Point(139.69, 35.68), 4326)),
    ('Melbourne', 5.0, ST_SetSRID(ST_Point(144.96, -37.81), 4326)),
    ('Sydney', 5.0, ST_SetSRID(ST_Point(151.21, -33.87), 4326))
), clustered AS (
  SELECT place,
         geom,
         ST_ClusterKMeans(
           ST_Force4D(
             ST_Transform(ST_Force3D(geom), 4978),
             mvalue => population
           ),
           4,
           max_radius => 3000000
         ) OVER (ORDER BY place) AS cid
  FROM population
)
SELECT cid,
       string_agg(place, ', ' ORDER BY place) AS places,
       ST_Collect(geom ORDER BY place) AS cluster_geom
FROM clustered
GROUP BY cid
ORDER BY cid;
```

Output

```
0 | Melbourne, Sydney | MULTIPOINT((144.96 -37.81),(151.21 -33.87))
1 | Boston, New York | MULTIPOINT((-71.06 42.36),(-74.01 40.71))
2 | Osaka, Tokyo | MULTIPOINT((135.5 34.69),(139.69 35.68))
3 | London, Paris | MULTIPOINT((-0.13 51.51),(2.35 48.86))
```

Figure



Voir aussi

ST_ClusterDBSCAN, ST_ClusterIntersectingWin, ST_ClusterWithinWin, ST_ClusterIntersecting, ST_ClusterWithin, ST_Subdivide, ST_Force3D, ST_Force4D,

7.17.7 ST_ClusterWithin

ST_ClusterWithin — Fonction agrégée qui regroupe les géométries en fonction de la distance de séparation.

Synopsis

geometry[] **ST_ClusterWithin**(geometry set g, float8 distance);

Description

Une fonction agrégée qui renvoie un tableau de GeometryCollections, où chaque collection est un cluster contenant certaines géométries d'entrée. Le regroupement partitionne les géométries d'entrée en ensembles dans lesquels chaque géométrie se trouve dans la *distance* spécifiée d'au moins une autre géométrie dans le même cluster. Les distances sont des distances cartésiennes dans les unités de l'ISDR.

ST_ClusterWithin est équivalent à l'exécution de ST_ClusterDBSCAN avec `minpoints => 0`.

Disponibilité : 2.2.0

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

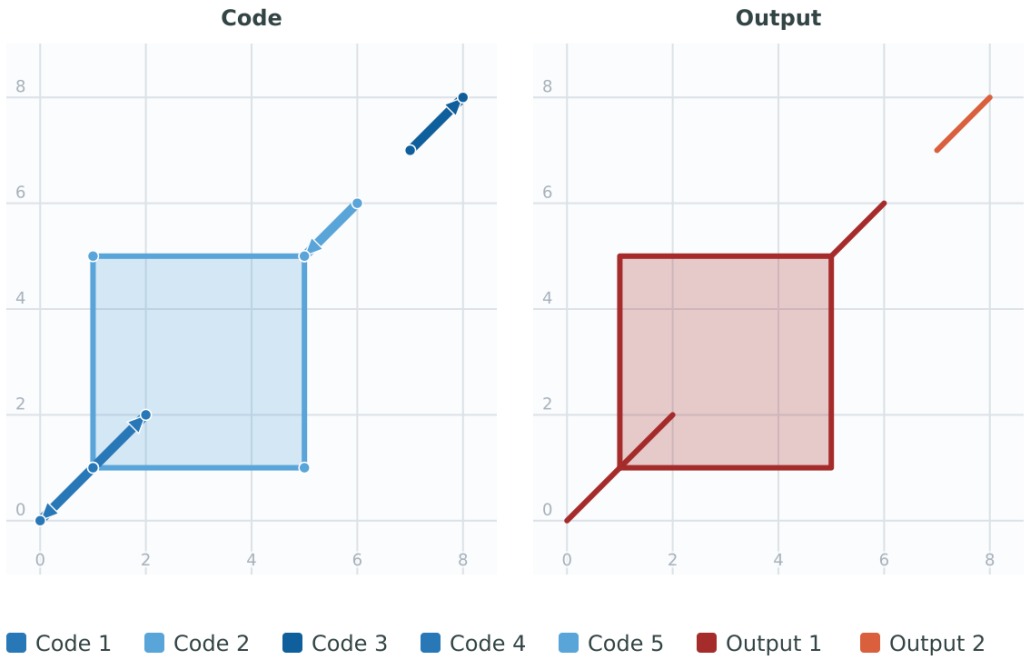
```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0,1 1)'::geometry,
                'LINESTRING (5 5,4 4)'::geometry,
                'LINESTRING (6 6,7 7)'::geometry,
                'LINESTRING (0 0,-1 -1)'::geometry,
                'POLYGON ((0 0,4 0,4 4,0 4,0 0))'::geometry]) AS geom)

SELECT unnest(ST_ClusterWithin(geom, 1.4)) FROM testdata;
```

Output

```
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

Figure



Voir aussi

ST_ClusterWithinWin, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterIntersectingWin

7.17.8 ST_ClusterWithinWin

ST_ClusterWithinWin — Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, regroupement en utilisant la distance de séparation.

Synopsis

integer **ST_ClusterWithinWin**(geometry winset geom, float8 distance);

Description

Une fonction window qui renvoie un numéro de cluster pour chaque géométrie d'entrée. Le regroupement partitionne les géométries en ensembles dans lesquels chaque géométrie se trouve dans la `distance` spécifiée d'au moins une autre géométrie dans le même groupe. Les distances sont des distances cartésiennes dans les unités du SRID.

ST_ClusterWithinWin est équivalent à l'exécution de ST_ClusterDBSCAN avec `minpoints => 0`.

Disponibilité : 3.4.0

Cette méthode prend en charge les types Circular String et Curve.

Exemples

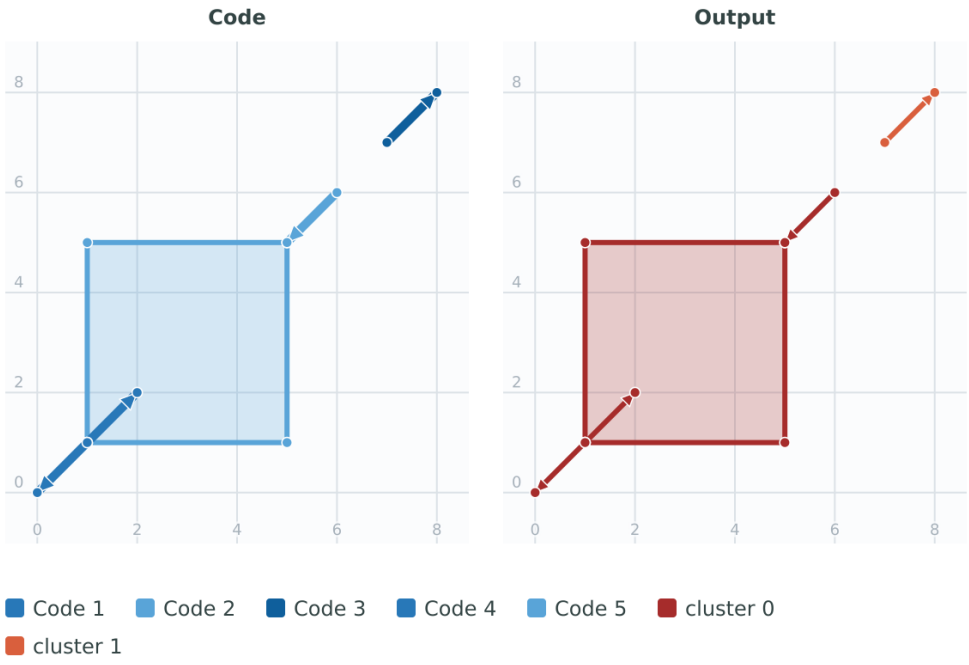
Code

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
  VALUES  (1, 'LINESTRING (0 0,1 1)'),
          (2, 'LINESTRING (5 5,4 4)'),
          (3, 'LINESTRING (6 6,7 7)'),
          (4, 'LINESTRING (0 0,-1 -1)'),
          (5, 'POLYGON ((0 0,4 0,4 4,0 4,0 0))')) AS t(id, geom)
)
SELECT id,
  geom,
  ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
FROM testdata;
```

Output

```
id |           st_astext             | cluster
---+---------------------------------+---------
  1 | LINESTRING(0 0,1 1)             |       0
  2 | LINESTRING(5 5,4 4)             |       0
  3 | LINESTRING(6 6,7 7)             |       1
  4 | LINESTRING(0 0,-1 -1)           |       0
  5 | POLYGON((0 0,4 0,4 4,0 4,0 0)) |       0
```

Figure



Voir aussi

ST_ClusterWithin, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterIntersectingWin,

7.18 Fonctions des boîtes de délimitation

7.18.1 Box2D

Box2D — Renvoie une BOX2D représentant l'étendue 2D d'une géométrie.

Synopsis

box2d **Box2D**(geometry geom);

Description

Renvoie une box2d représentant l'étendue 2D de la géométrie.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

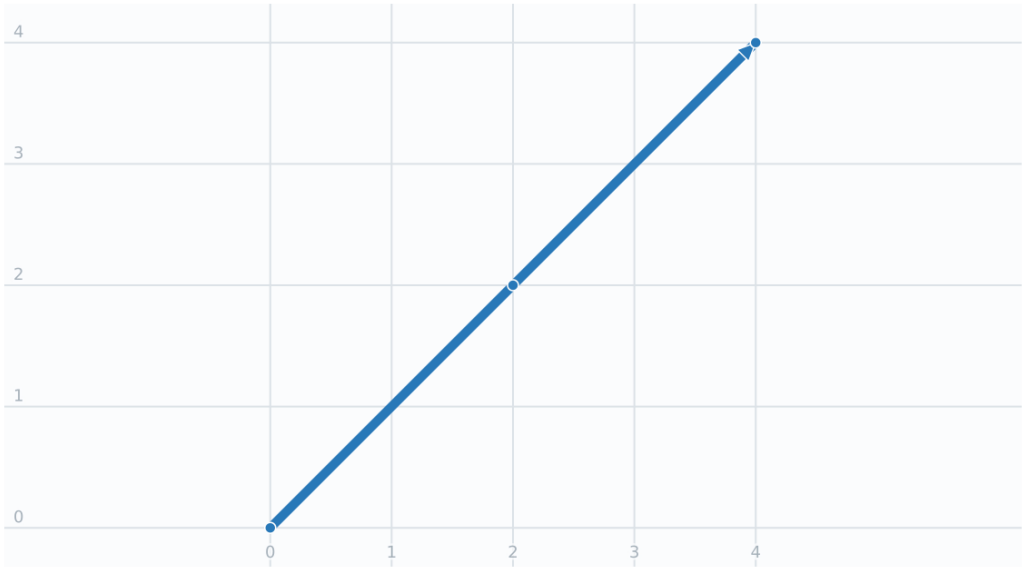
Code

```
SELECT Box2D('LINESTRING(1 2,3 4,5 6)'::geometry);
```

Output

```
BOX(1 2,5 6)
```

Figure



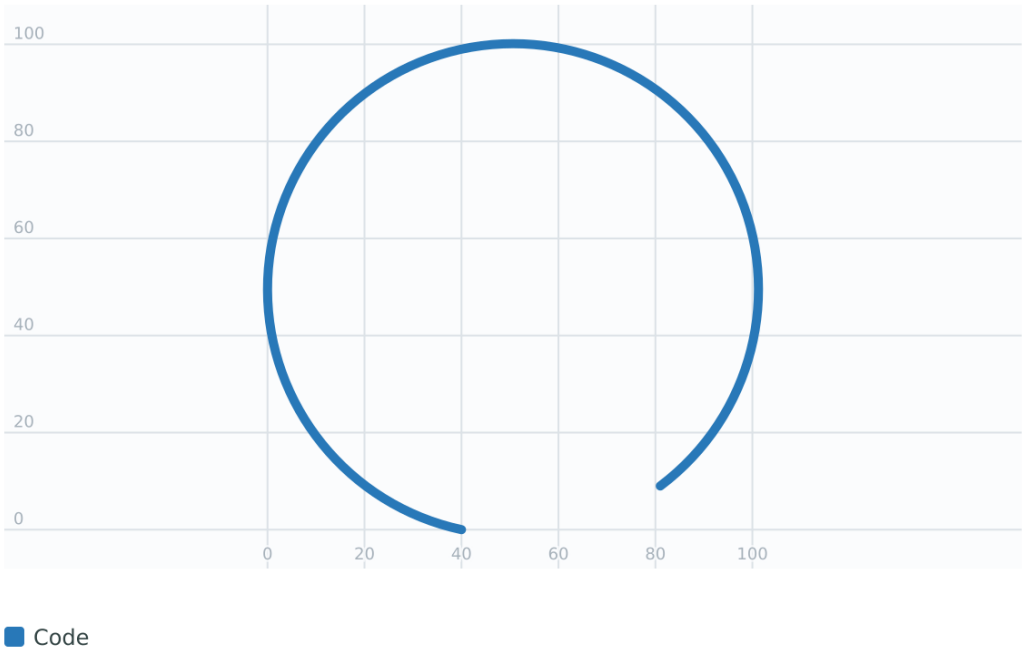
Code

```
SELECT Box2D('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'::geometry);
```

Output

```
BOX(220186.99512189245 150406,220288.24878054656 150506.12682932706)
```

Figure



Voir aussi

Box3D, ST_GeomFromText

7.18.2 Box3D

Box3D — Renvoie une BOX3D représentant l'étendue 3D d'une géométrie.

Synopsis

box3d **Box3D**(geometry geom);

Description

Renvoie une box3d représentant l'étendue 3D de la géométrie.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Cette méthode prend en charge les types Circular String et Curve.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

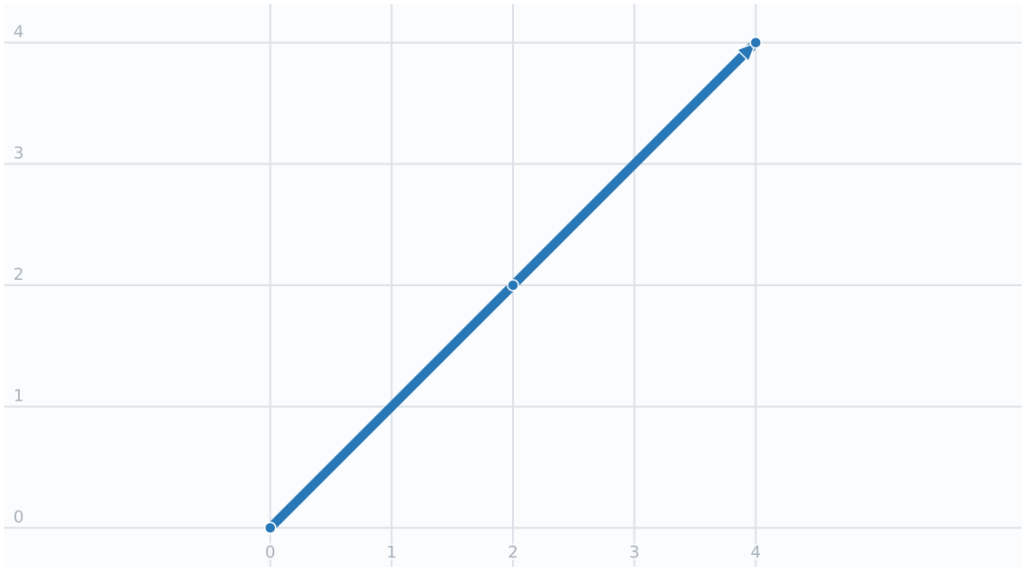
Code

```
SELECT Box3D('LINESTRING(1 2 3,3 4 5,5 6 5)'::geometry);
```

Output

```
BOX3D(1 2 3,5 6 5)
```

Figure



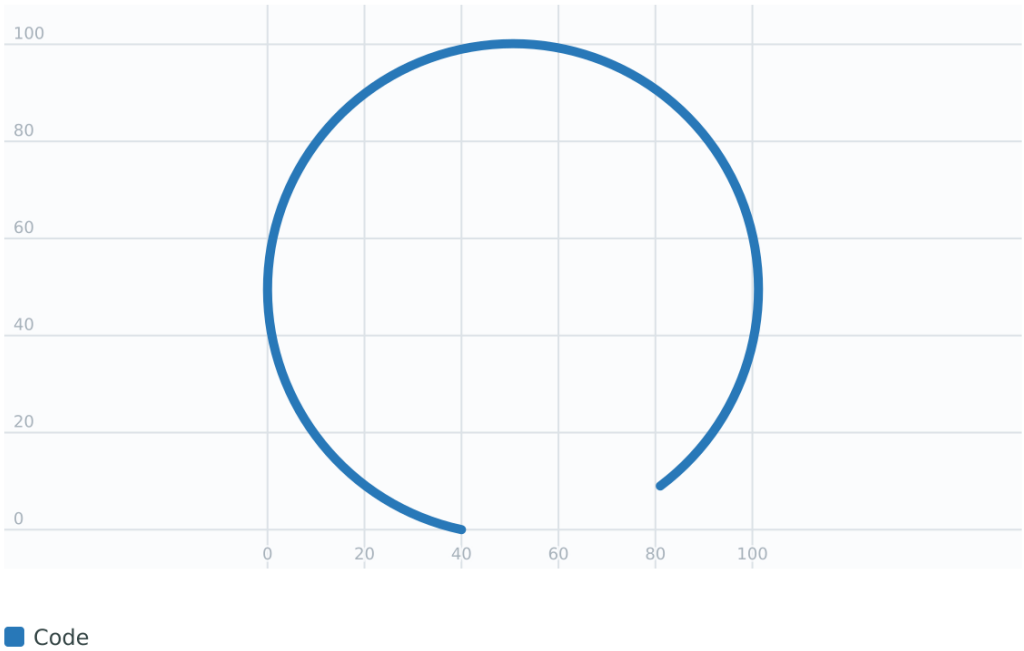
Code

```
SELECT Box3D('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'::geometry);
```

Output

```
BOX3D(220186.99512189245 150406 1,220288.24878054656 150506.12682932706 1)
```

Figure



Voir aussi

Box2D, ST_GeomFromEWKT

7.18.3 ST_EstimatedExtent

ST_EstimatedExtent — Renvoie l'étendue estimée d'une table spatiale.

Synopsis

box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name);
box2d **ST_EstimatedExtent**(text table_name, text geocolumn_name);

Description

Renvoie l'étendue estimée d'une table spatiale sous la forme d'une box2d. Le schéma actuel est utilisé s'il n'est pas spécifié. L'étendue estimée est tirée des statistiques de la colonne géométrique. Cette méthode est généralement plus rapide que le calcul de l'étendue exacte de la table à l'aide de ST_Extent ou ST_3DExtent.

Le comportement par défaut est d'utiliser également les statistiques collectées à partir des tables enfants (tables avec INHERITS) si elles sont disponibles. Si `parent_only` est défini à VRAI, seules les statistiques de la table donnée sont utilisées et les tables enfants sont ignorées.

Pour PostgreSQL >= 8.0.0 les statistiques sont collectées par VACUUM ANALYZE et l'étendue du résultat sera d'environ 95% de l'étendue réelle. Pour PostgreSQL < 8.0.0 les statistiques sont collectées en exécutant `update_geometry_stats()` et l'étendue du résultat sera exacte.



Note
En l'absence de statistiques (table vide ou pas d'ANALYZE), cette fonction renvoie NULL. Avant la version 1.5.4, une exception était levée.



Note
L'échappement des noms de tables et/ou d'espaces de noms qui comprennent des caractères spéciaux et des guillemets peut nécessiter un traitement particulier. Un utilisateur note : "Pour les schémas et les tables, utiliser les règles d'échappement des identificateurs pour produire une chaîne de caractères entre guillemets, puis supprimer le premier et le dernier caractère entre guillemets. Pour les colonnes géométriques, laissez-les telles quelles.

Disponibilité : 1.0.0

Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Estimated_Extent.



Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
```

Output

```
st_estimatedextent
---------------------------------------------------
 BOX(-8877653 4912316,-8010225.5 5589284)
(1 row)
```

Code

```
SELECT ST_EstimatedExtent('feature_poly', 'geom');
```

Output

```
st_estimatedextent
-----------------------------------------------------------------------
 BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
(1 row)
```

Voir aussi

ST_Extent, ST_3DExtent

7.18.4 ST_Expand

ST_Expand — Renvoie une boîte de délimitation développée à partir d'une autre boîte de délimitation ou d'une géométrie.

Synopsis

geometry **ST_Expand**(geometry geom, float units_to_expand);
geometry **ST_Expand**(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d **ST_Expand**(box2d box, float units_to_expand);
box2d **ST_Expand**(box2d box, float dx, float dy);
box3d **ST_Expand**(box3d box, float units_to_expand);
box3d **ST_Expand**(box3d box, float dx, float dy, float dz=0);

Description

Renvoie une boîte de délimitation développée à partir de la boîte de délimitation de l'entrée, soit en spécifiant une distance unique avec laquelle la boîte doit être développée sur les deux axes, soit en spécifiant une distance de développement pour chaque axe. Utilise la double précision. Peut être utilisé pour les requêtes de distance, ou pour ajouter un filtre de boîte de délimitation à une requête afin de tirer parti d'un index spatial.

Outre la version de ST_Expand acceptant et renvoyant une géométrie, des variantes sont fournies qui acceptent et renvoient des types de données box2d et box3d.

Les distances sont exprimées dans les unités du système de référence spatiale de l'entrée.

ST_Expand est similaire à ST_Buffer, mais alors que la mise en mémoire tampon étend une géométrie dans toutes les directions, ST_Expand étend la boîte de délimitation le long de chaque axe.



Note
Avant la version 1.3, ST_Expand était utilisé en conjonction avec ST_Distance pour effectuer des requêtes de distance indexables. Par exemple, `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. Cette fonction a été remplacée par la fonction ST_DWithin, plus simple et plus efficace.

Disponibilité : 1.5.0 comportement modifié pour afficher les coordonnées en double précision au lieu des coordonnées float4.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Amélioration : 2.3.0 : prise en charge de l'expansion d'une boîte par différentes quantités dans différentes dimensions.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Note
Les exemples ci-dessous utilisent l'Atlas national américain Equal Area (SRID=2163) qui est une projection en mètres

10-meter expanded box around bbox of a linestring.

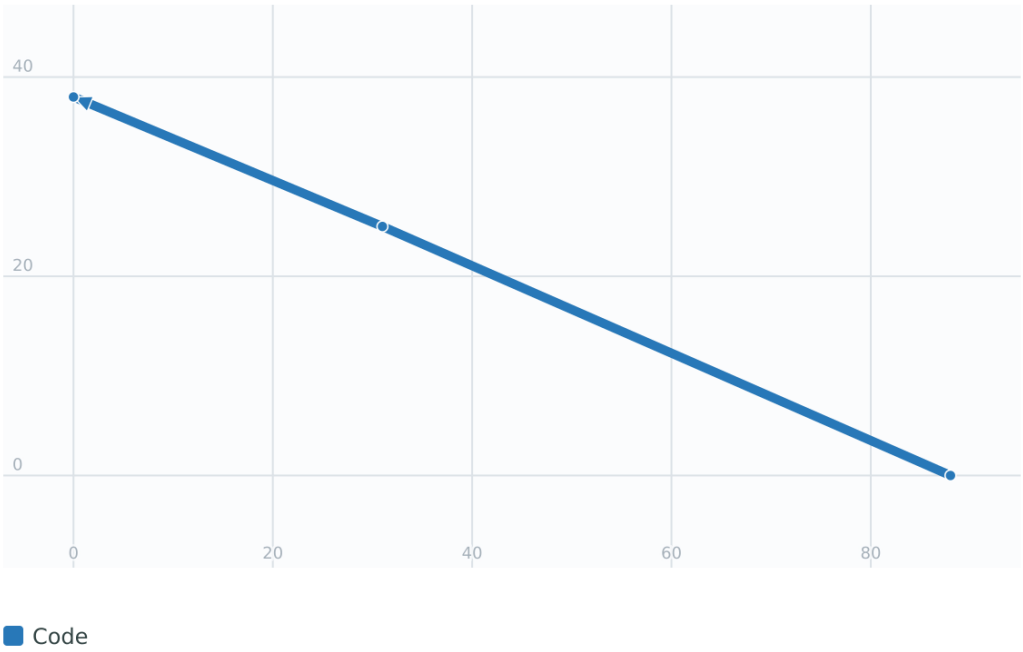
Code

```
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 ↩
    110714)', 2163), 10) As box2d);
```

Output

```
BOX(2312882 110666,2312990 110724)
```

Figure



This example expands a 3D box by 10 meters.

Code

```
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' AS box3d), 10)
```

Output

```
BOX3D(778773 2951731 -9,794885 2970052.61545891 20)
```

This example renders the text representation of a 10-meter expanded box around a point geometry.

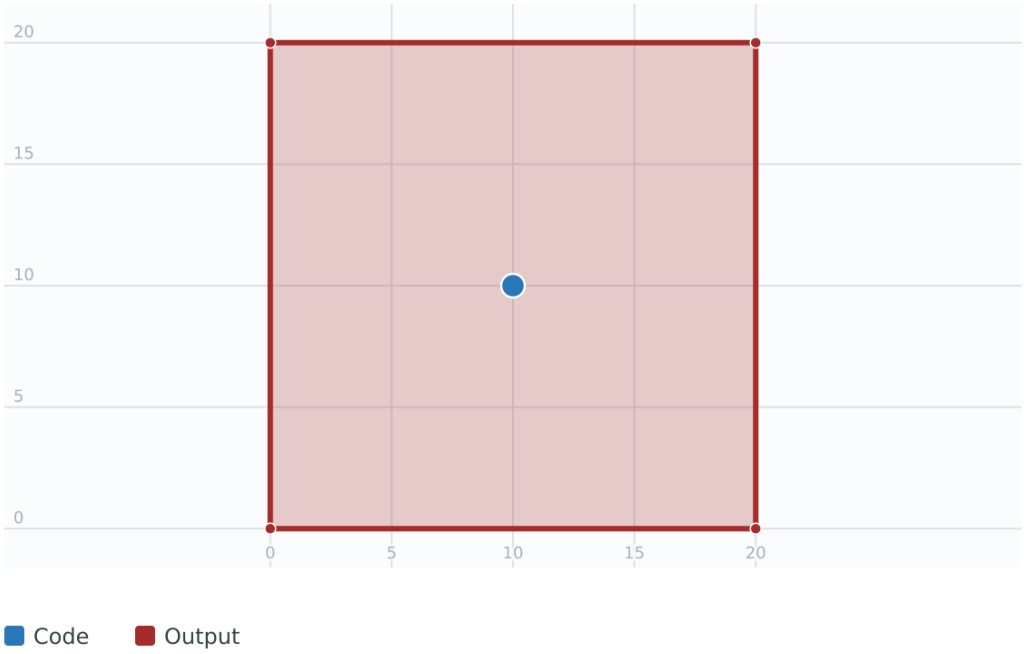
Code

```
SELECT ST_Expand('SRID=2163;POINT(2312980 110676)'::geometry, 10);
```

Output

```
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 110666))
```

Figure



Voir aussi

ST_Buffer, ST_DWithin, ST_SRID

7.18.5 ST_Extent

ST_Extent — Fonction agrégée qui renvoie la boîte de délimitation des géométries.

Synopsis

box2d **ST_Extent**(geometry set geomfield);

Description

Une fonction agrégée qui renvoie une box2d boîte de délimitation qui délimite un ensemble de géométries.

Les coordonnées de la boîte de délimitation sont dans le système de référence spatiale des géométries d'entrée.

Le concept de ST_Extent est similaire à celui de SDO_AGGR_MBR d'Oracle Spatial/Locator.



Note
ST_Extent renvoie des boîtes avec seulement les coordonnées X et Y, même avec des géométries 3D. Pour renvoyer les coordonnées XYZ, utilisez ST_3DExtent.



Note
La valeur `box3d` renvoyée ne contient pas de SRID. Utilisez ST_SetSRID pour la convertir en une géométrie avec des métadonnées SRID. Le SRID est le même que celui des géométries d'entrée.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples



Note
Les exemples ci-dessous utilisent le plan de l'État du Massachusetts (SRID=2249)

Code

```
SELECT ST_Extent(geom) AS bextent FROM sometable;
```

Output

```
BOX(739651.875 2908247.25,794875.8125 2970042.75)
```

Return extent of each geometry category.

Code

```
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;
```

Output

```
bextent                        |         name
-----------------------------------------------------+-----------------
 BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
 BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
 BOX(739651.875 2917394.75,756688.375 2935866)      | C
```

This example converts the extent back into a geometry and renders its extended text representation.

Code

```
SELECT ST_SetSRID(ST_Extent(geom), 2249) AS bextent FROM sometable;
```

Output

```
SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

Voir aussi

ST_EstimatedExtent, ST_3DExtent, ST_SetSRID

7.18.6 ST_3DExtent

ST_3DExtent — Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

Description

Une fonction d'agrégation qui renvoie une box3d (y compris l'ordonnée Z) qui délimite un ensemble de géométries.

Les coordonnées de la boîte de délimitation sont dans le système de référence spatiale des géométries d'entrée.



Note
La valeur `box3d` renvoyée ne contient pas de SRID. Utilisez ST_SetSRID pour la convertir en une géométrie avec des métadonnées SRID. Le SRID est le même que celui des géométries d'entrée.

Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques.

Modifié : 2.0.0 Dans les versions précédentes, cette fonction était appelée ST_Extent3D

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Code

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x, y, z) As geom
    FROM generate_series(1, 3) As x
        CROSS JOIN generate_series(1, 2) As y
        CROSS JOIN generate_series(0, 2) As Z) As foo;
```

Output

```
BOX3D(1 1 0,3 2 2)
```

This example returns the extent of several elevated CircularStrings.

Code

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x, y), 1))), 0, 0, z ↩
    ) As geom
    FROM generate_series(1, 3) As x
        CROSS JOIN generate_series(1, 2) As y
        CROSS JOIN generate_series(0, 2) As Z) As foo;
```

Output

```
BOX3D(0 0 0,4 3 2)
```

Voir aussi

ST_Extent, ST_Force3DZ, ST_SetSRID

7.18.7 ST_MakeBox2D

ST_MakeBox2D — Crée un BOX2D défini par deux géométries de points 2D.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

Description

Crée une box2d définie par deux géométries de points. Ceci est utile pour effectuer des requêtes sur des intervalles.

Exemples

Return all features that fall within or partly reside in a US National Atlas coordinate bounding box. It is assumed here that the geometries are stored with SRID=2163 (US National Atlas equal area).

Code

```
SELECT feature_id, feature_name, geom
FROM features
WHERE geom && ST_SetSRID(
  ST_MakeBox2D(
    ST_Point(-989502.1875, 528439.5625),
    ST_Point(-987121.375, 529933.1875)
  ),
  2163
);
```

Voir aussi

ST_Point, ST_SetSRID, ST_SRID

7.18.8 ST_3DMakeBox

ST_3DMakeBox — Crée un BOX3D défini par deux géométries de points 3D.

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

Description

Crée une box3d définie par deux géométries de points 3D.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Modifié : 2.0.0 Dans les versions précédentes, cette fonction s'appelait ST_MakeBox3D

Exemples

Code

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
    ST_MakePoint(-987121.375 , 529933.1875, 10))
```

Output

```
BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

Voir aussi

ST_MakePoint, ST_SetSRID, ST_SRID

7.18.9 ST_XMax

ST_XMax — Retourne les maxima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

Synopsis

float **ST_XMax**(box3d aGeomorBox2DorBox3D);

Description

Retourne les maxima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car ces dernières ne sont pas soumises à la conversion automatique.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_XMax('BOX3D(1 2 3,4 5 6)');
```

Output

```
4
```

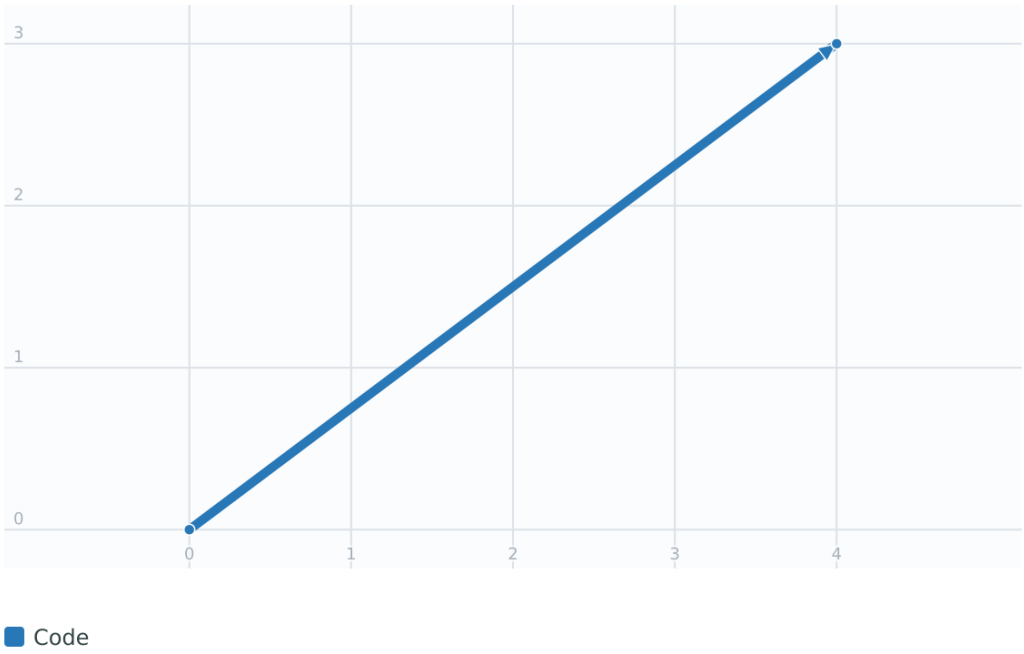
Code

```
SELECT ST_XMax('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
5
```

Figure



Code

```
SELECT ST_XMax(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_XMax('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

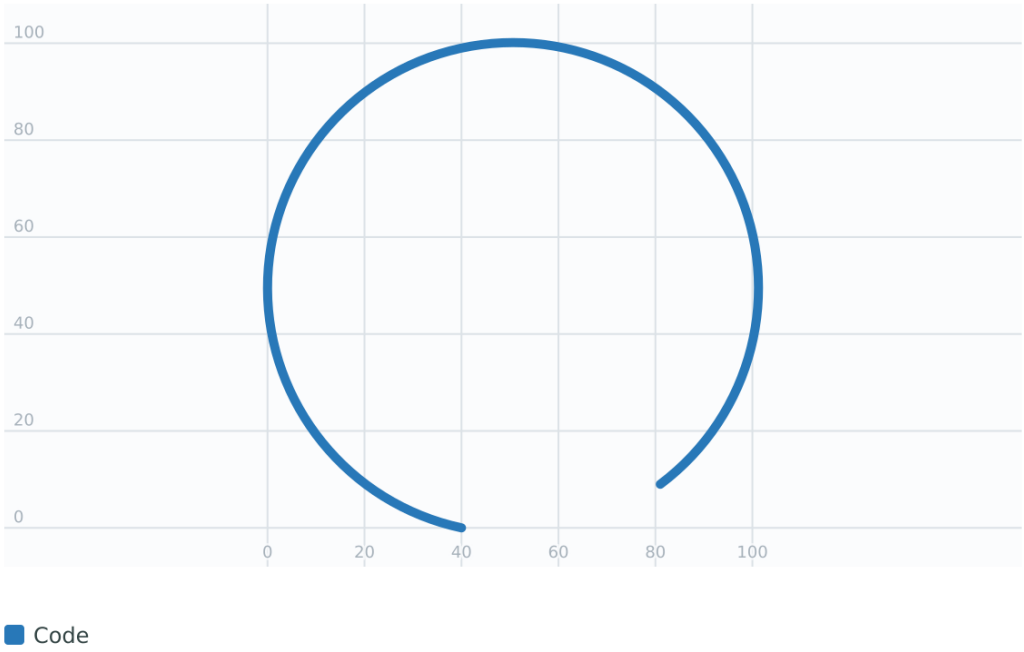
Code

```
SELECT ST_XMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
220288.248780547
```

Figure



Voir aussi

ST_XMin, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.10 ST_XMin

ST_XMin — Retourne les minima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

Description

Retourne les minima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_XMin('BOX3D(1 2 3,4 5 6)');
```

Output

```
1
```

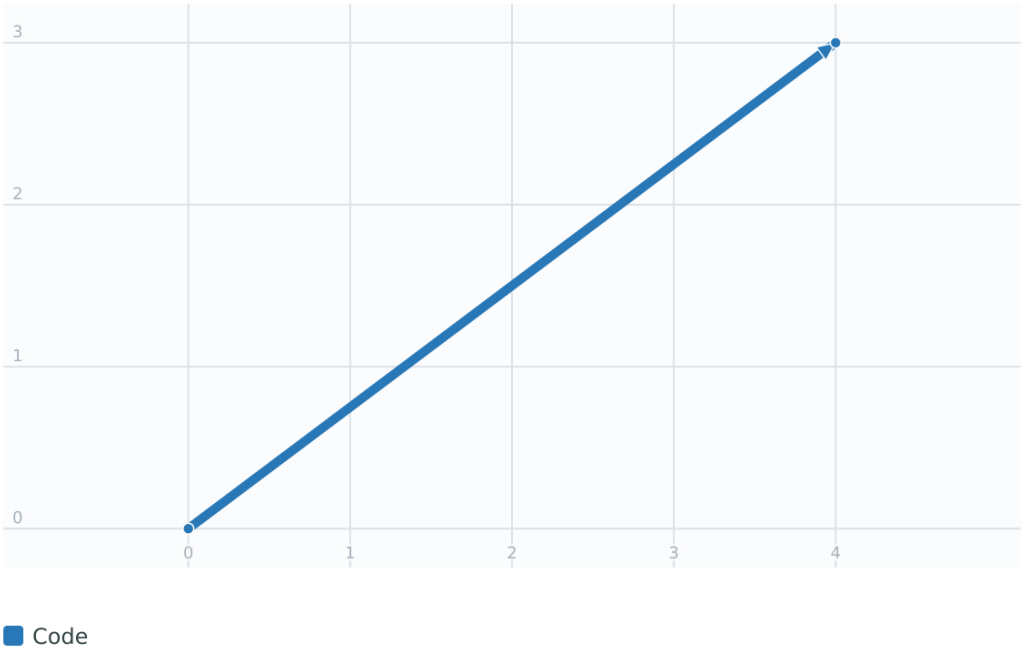
Code

```
SELECT ST_XMin('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
1
```

Figure



Code

```
SELECT ST_XMin(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
-3
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_XMin('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

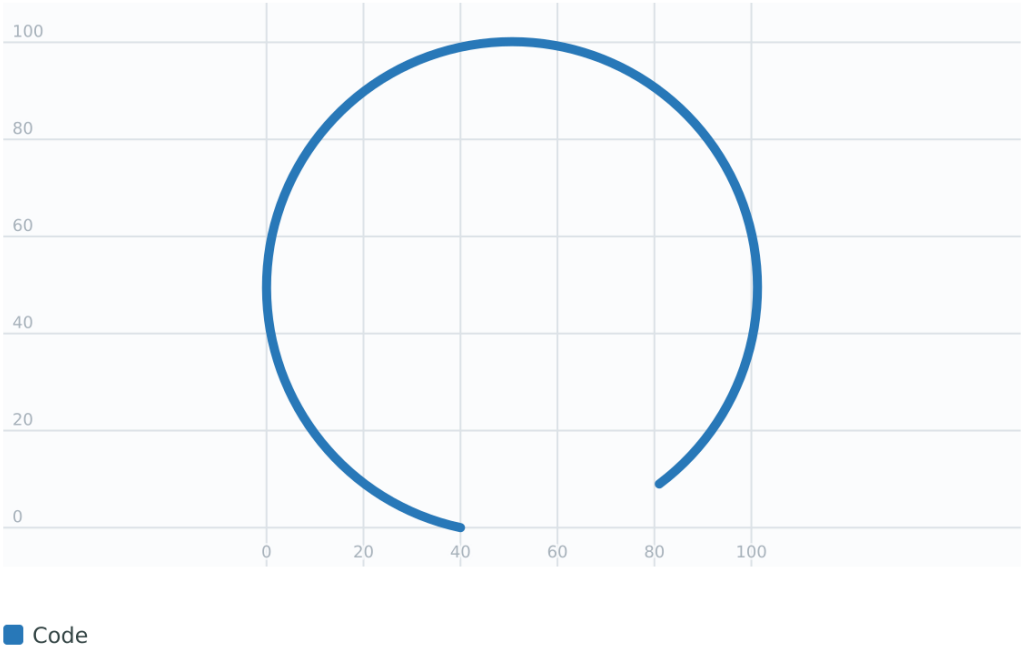
Code

```
SELECT ST_XMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
220186.995121892
```

Figure



Voir aussi

ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.11 ST_YMax

ST_YMax — Retourne les maxima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

Synopsis

float **ST_YMax**(box3d aGeomorBox2DorBox3D);

Description

Retourne les maxima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_YMax('BOX3D(1 2 3,4 5 6)');
```

Output

```
5
```

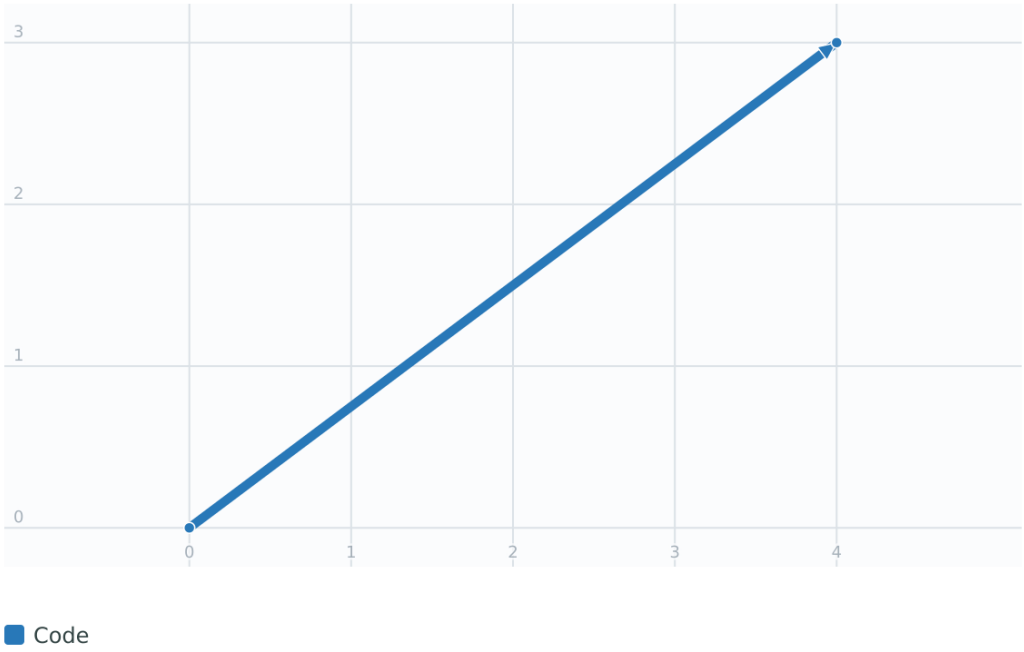
Code

```
SELECT ST_YMax('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
6
```

Figure



Code

```
SELECT ST_YMax(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
4
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_YMax('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

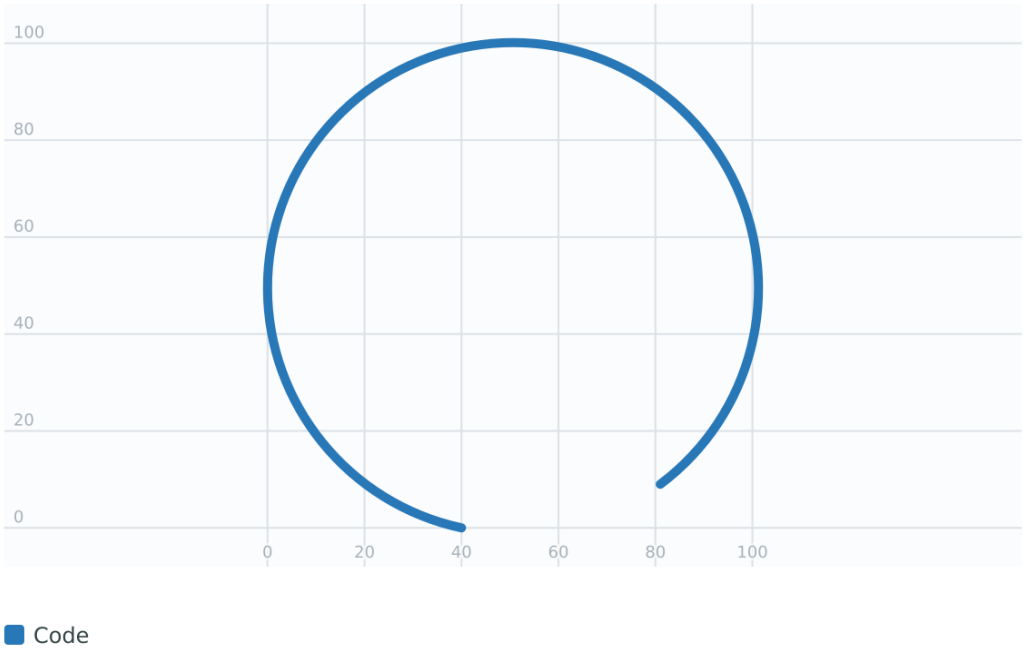
Code

```
SELECT ST_YMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
150506.126829327
```

Figure



Voir aussi

ST_XMin, ST_XMax, ST_YMin, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.12 ST_YMin

ST_YMin — Retourne les minima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

Description

Retourne les minima Y d'une boîte englobante 2D ou 3D ou d'une géométrie.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_YMin('BOX3D(1 2 3,4 5 6)');
```

Output

```
2
```

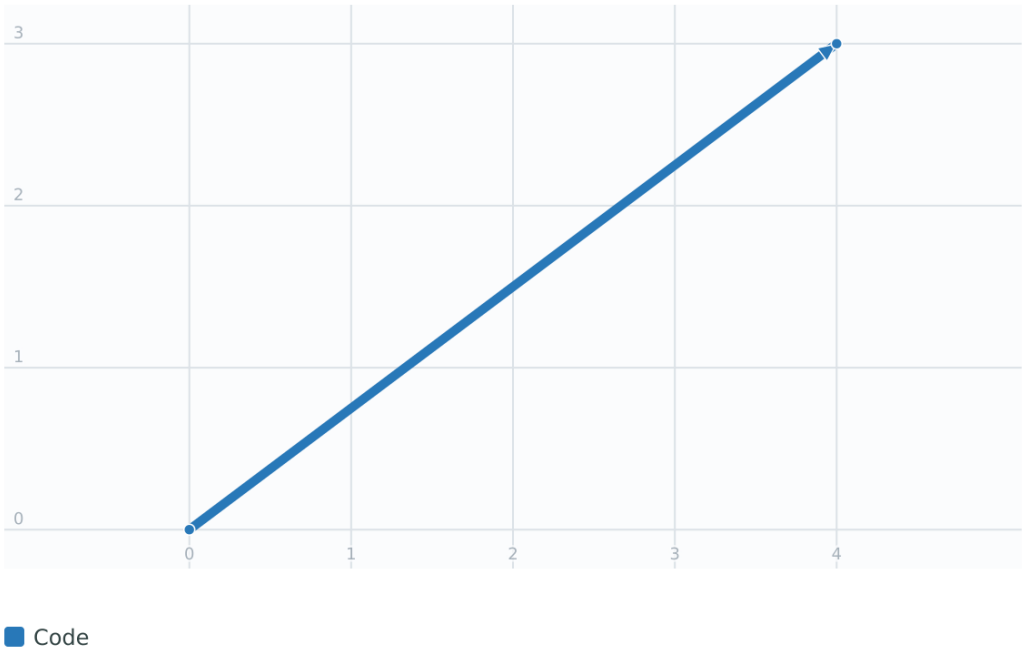
Code

```
SELECT ST_YMin('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
3
```

Figure



Code

```
SELECT ST_YMin(CAST('BOX(-3 2,3 4)' AS box2d));
```

Output

```
2
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_YMin('LINESTRING(1 3,5 6)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

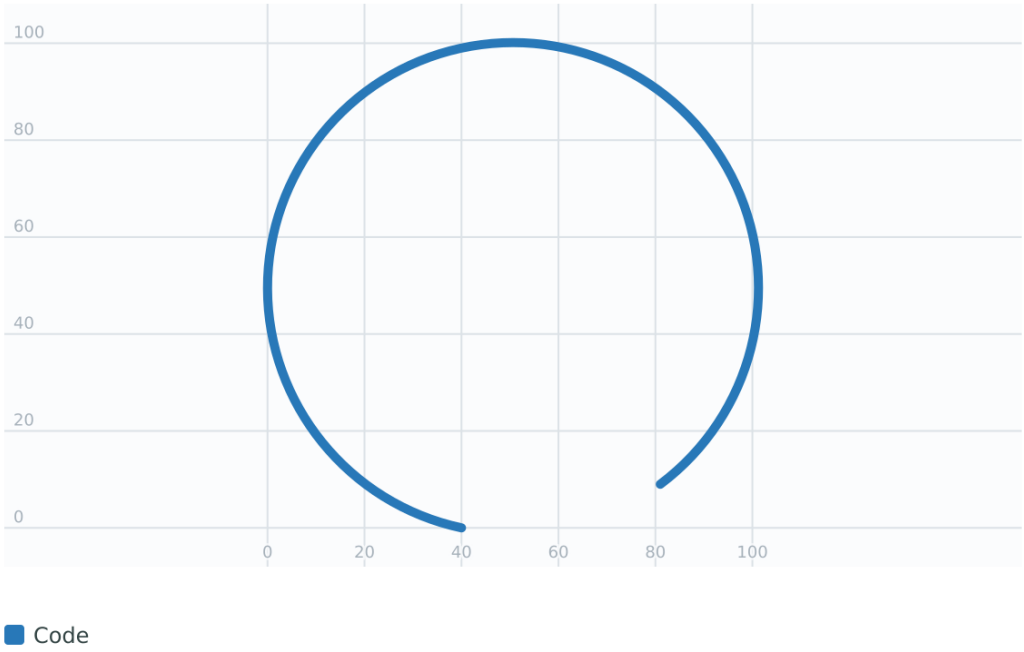
Code

```
SELECT ST_YMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
150406
```

Figure



Voir aussi

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MMax

7.18.13 ST_ZMax

ST_ZMax — Retourne les maxima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

Synopsis

float **ST_ZMax**(box3d aGeomorBox2DorBox3D);

Description

Retourne les maxima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_ZMax('BOX3D(1 2 3,4 5 6)');
```

Output

```
6
```

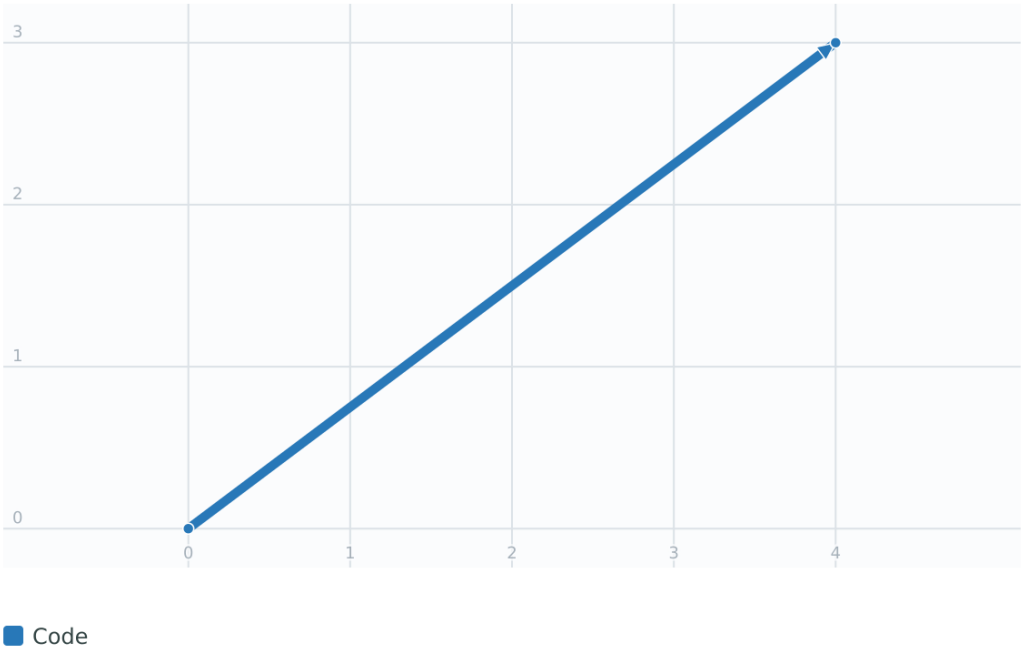
Code

```
SELECT ST_ZMax('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
7
```

Figure



Code

```
SELECT ST_ZMax('BOX3D(-3 2 1,3 4 1)');
```

Output

```
1
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_ZMax('LINESTRING(1 3 4,5 6 7)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

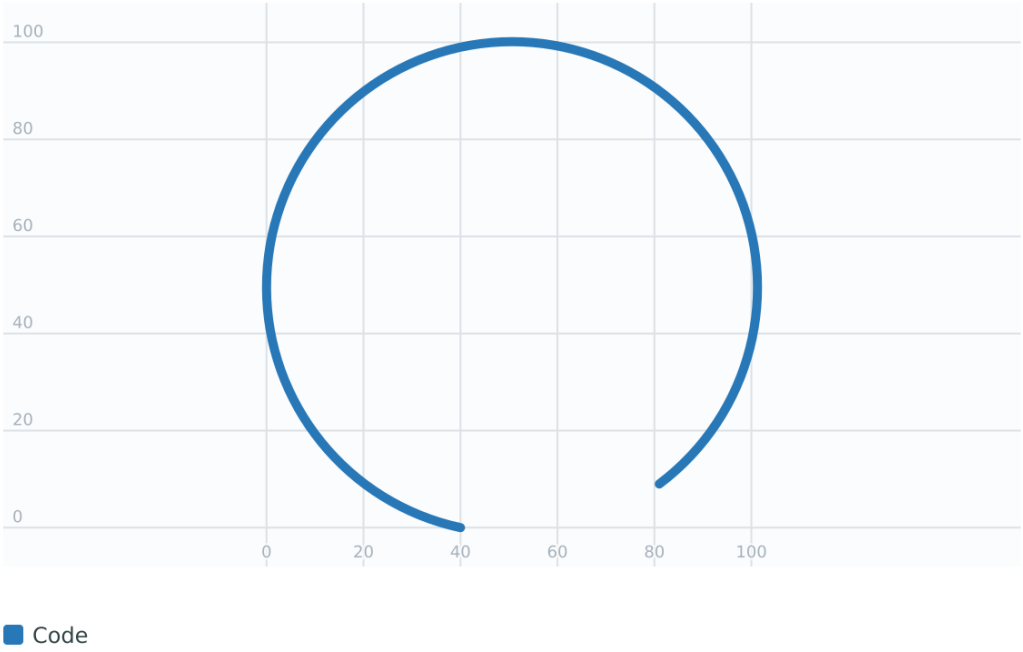
Code

```
SELECT ST_ZMax('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
3
```

Figure



Voir aussi

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_MMin, ST_MMax

7.18.14 ST_ZMin

ST_ZMin — Retourne les minima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

Synopsis

float **ST_ZMin**(box3d aGeomorBox2DorBox3D);

Description

Retourne les minima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_ZMin('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

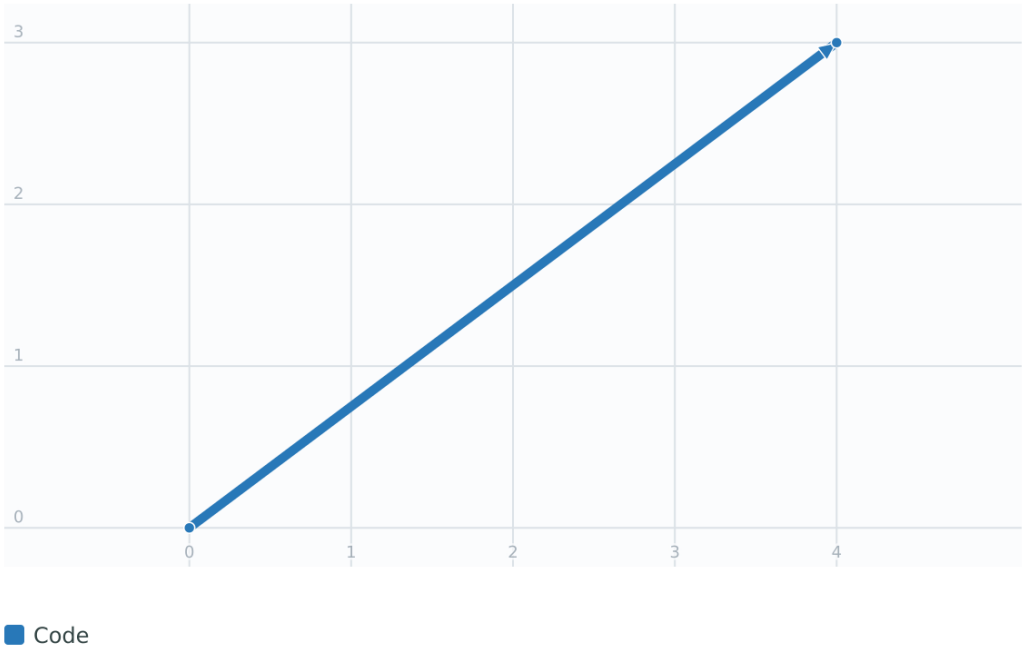
Code

```
SELECT ST_ZMin('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
4
```

Figure



Code

```
SELECT ST_ZMin('BOX3D(-3 2 1,3 4 1)');
```

Output

```
1
```

Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D.

Code

```
SELECT ST_ZMin('LINESTRING(1 3 4,5 6 7)');
```

Output

```
ERROR: BOX3D parser - doesn't start with BOX3D(
```

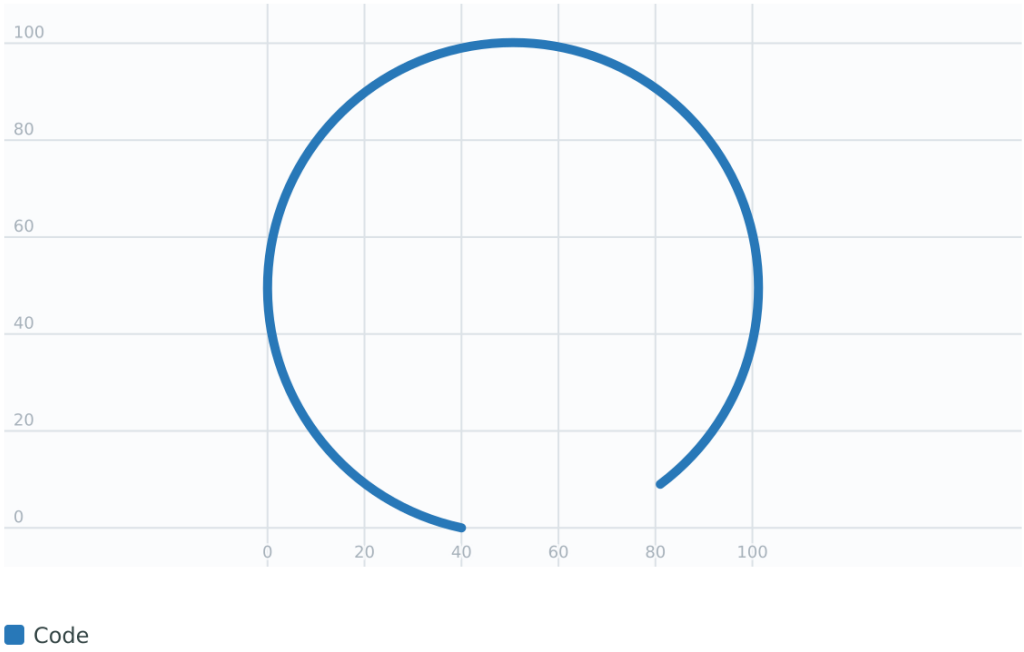
Code

```
SELECT ST_ZMin('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'::geometry);
```

Output

```
1
```

Figure



Voir aussi

ST_GeomFromEWKT, ST_GeomFromText, ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMax, ST_MMin, ST_MMax

7.18.15 ST_XSize

ST_XSize — Returns the X size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XSize**(box3d aGeomorBox2DorBox3D);

Description

Returns the X size of a 2D or 3D bounding box or a geometry, computed as `ST_XMax(aGeomorBox2DorBox3D) - ST_XMin(aGeomorBox2DorBox3D)`.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Availability: 3.7.0

Exemples

Code

```
SELECT ST_XSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

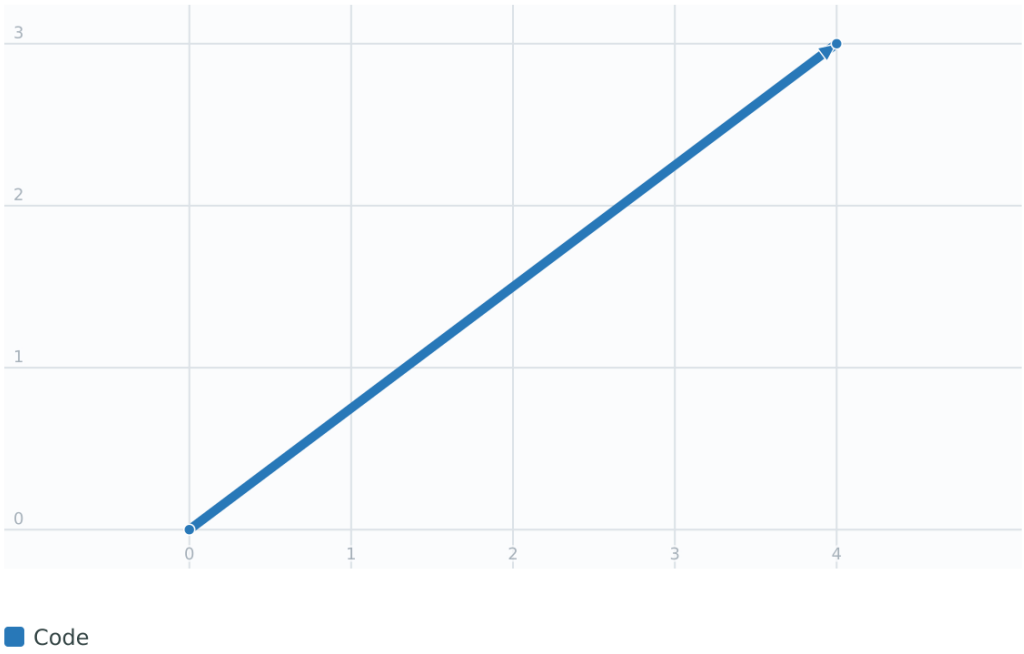
Code

```
SELECT ST_XSize('LINESTRING(1 3,5 6)'::geometry);
```

Output

```
4
```

Figure



Voir aussi

ST_XMin, ST_XMax, ST_YSize, ST_ZSize, ST_MSize

7.18.16 ST_YSize

ST_YSize — Returns the Y size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YSize**(box3d aGeomorBox2DorBox3D);

Description

Returns the Y size of a 2D or 3D bounding box or a geometry, computed as `ST_YMax(aGeomorBox2DorBox3D) - ST_YMin(aGeomorBox2DorBox3D)`.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Availability: 3.7.0

Exemples

Code

```
SELECT ST_YSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

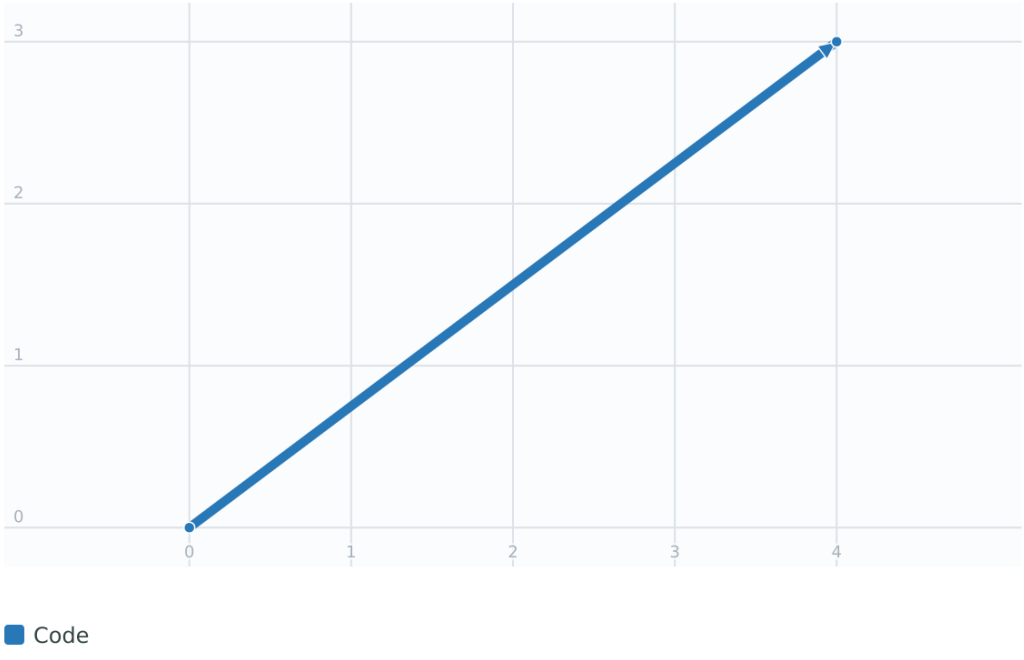
Code

```
SELECT ST_YSize('LINESTRING(1 3,5 6)'::geometry);
```

Output

```
3
```

Figure



Voir aussi

ST_YMin, ST_YMax, ST_XSize, ST_ZSize, ST_MSize

7.18.17 ST_ZSize

ST_ZSize — Returns the Z size of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_ZSize**(box3d aGeomorBox2DorBox3D);

Description

Returns the Z size of a 2D or 3D bounding box or a geometry, computed as `ST_ZMax(aGeomorBox2DorBox3D) - ST_ZMin(aGeomorBox2DorBox3D)`.



Note
Bien que cette fonction ne soit définie que pour box3d, elle fonctionne également pour box2d et les valeurs géométriques en raison du moulage automatique. Cependant, elle n'acceptera pas de représentation textuelle de la géométrie ou de la box2d, car celles-ci ne sont pas coulées automatiquement.

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

 Cette méthode prend en charge les types Circular String et Curve.

Availability: 3.7.0

Exemples

Code

```
SELECT ST_ZSize('BOX3D(1 2 3,4 5 6)');
```

Output

```
3
```

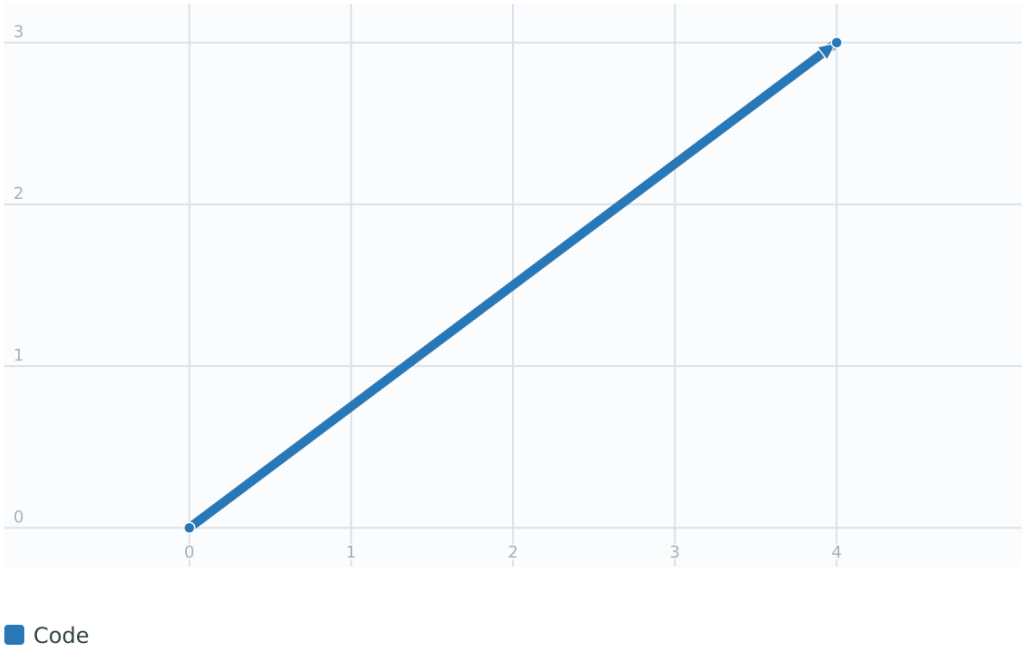
Code

```
SELECT ST_ZSize('LINESTRING(1 3 4,5 6 7)'::geometry);
```

Output

```
3
```

Figure



Voir aussi

ST_ZMin, ST_ZMax, ST_XSize, ST_YSize, ST_MSize

7.18.18 ST_MMin

ST_MMin — Returns the M minima of a geometry.

Synopsis

float **ST_MMin**(geometry geom);

Description

Returns the M minima of a geometry, or null if the geometry lacks M values.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_MMin('POINT M (1 2 3)');
```

Output

```
3
```

Voir aussi

ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMax, ST_MSize

7.18.19 ST_MMax

ST_MMax — Returns the M maxima of a geometry.

Synopsis

float **ST_MMax**(geometry geom);

Description

Returns the M maxima of a geometry, or null if the geometry lacks M values.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT ST_MMax('POINT M (1 2 3)');
```

Output

```
3
```

Voir aussi

ST_XMin, ST_XMax, ST_YMin, ST_YMax, ST_ZMin, ST_ZMax, ST_MMin, ST_MSize

7.18.20 ST_MSize

ST_MSize — Returns the M size of a geometry.

Synopsis

float **ST_MSize**(geometry geom);

Description

Returns the M size of a geometry, computed as `ST_MMax(geom) - ST_MMin(geom)`, or null if the geometry lacks M values.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette méthode prend en charge les types Circular String et Curve.

Availability: 3.7.0

Exemples

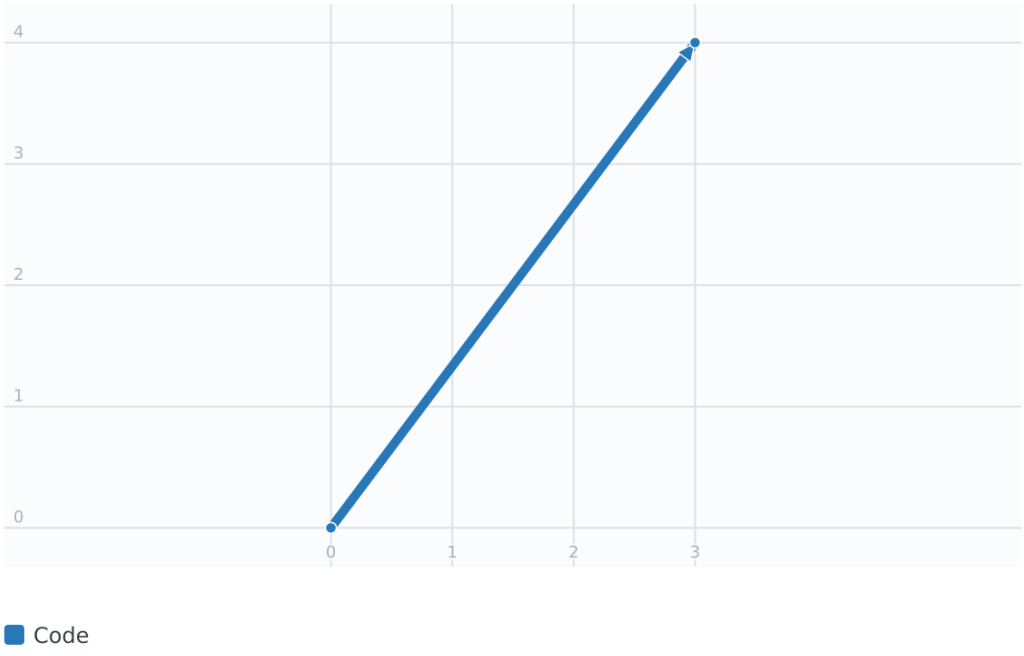
Code

```
SELECT ST_MSize('LINESTRING M (1 2 3,4 6 8)');
```

Output

```
5
```

Figure



Voir aussi

ST_MMin, ST_MMax, ST_XSize, ST_YSize, ST_ZSize

7.19 Référencement linéaire

7.19.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Renvoie un point interpolé le long d'une ligne à un emplacement fractionnaire.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);
geography **ST_LineInterpolatePoint**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);

Description

Retourne un point interpolé sur une ligne. Le premier argument doit être une LINESTRING. Le second argument est un float8 entre 0 et 1 représentant la fraction de la longueur totale de la ligne où le point doit être situé.

Voir ST_LineLocatePoint pour calculer l'emplacement de la ligne la plus proche d'un point.



Note
Cette fonction calcule des points en 2D et interpole ensuite les valeurs de Z et M, tandis que ST_3DLineInterpolatePoint calcule des points en 3D et n'interpole que la valeur M.



Note
Depuis la version 1.1.1 cette fonction interpole aussi les valeurs M et Z (lorsqu'elles sont présentes), les versions précédentes renvoient des valeurs M et Z à 0.0.

Disponibilité : 0.8.2, support de Z et M ajouté en 1.1.1

Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Line_Interpolate_Point.


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

This example returns the point 20 percent along a line.

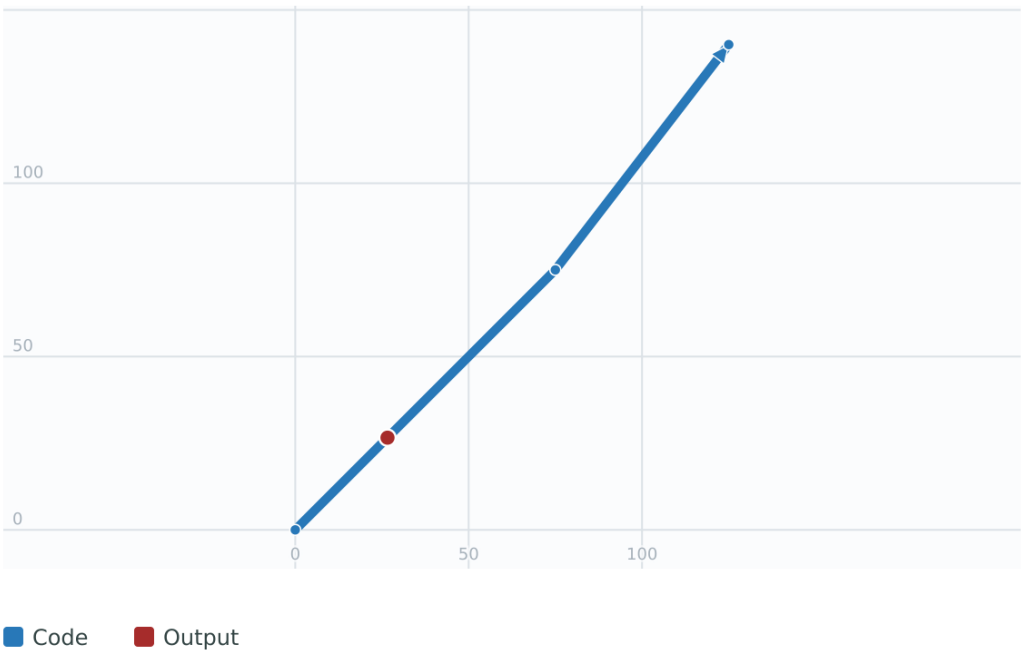
Code

```
SELECT ST_LineInterpolatePoint(
'LINESTRING(25 50,100 125,150 190)',
0.2);
```

Output

```
POINT(51.5974135047432 76.5974135047432)
```

Figure



Le point central d'une ligne 3D :

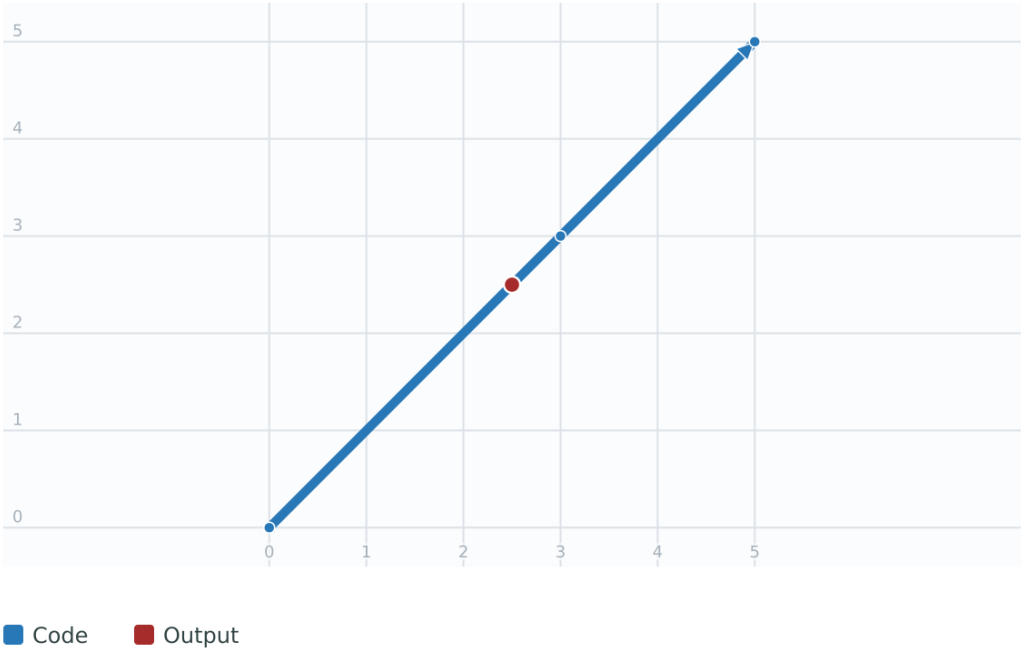
Code

```
SELECT ST_LineInterpolatePoint(
'LINESTRING(1 2 3,4 5 6,6 7 8)',
0.5);
```

Output

```
POINT(3.5 4.5 5.5)
```

Figure



Le point le plus proche d'un point sur une ligne :

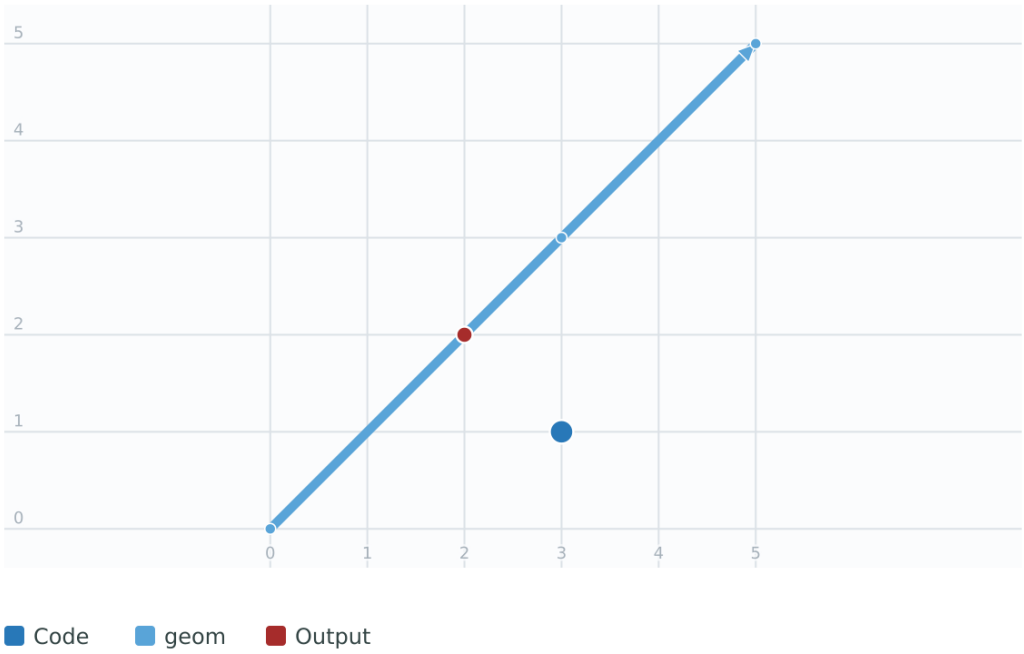
Code

```
SELECT ST_LineInterpolatePoint(
    line.geom,
    ST_LineLocatePoint(line.geom, 'POINT(4 3)'))
FROM (SELECT 'LINESTRING(1 2,4 5,6 7)'::geometry As geom) AS line;
```

Output

```
POINT(3 4)
```

Figure

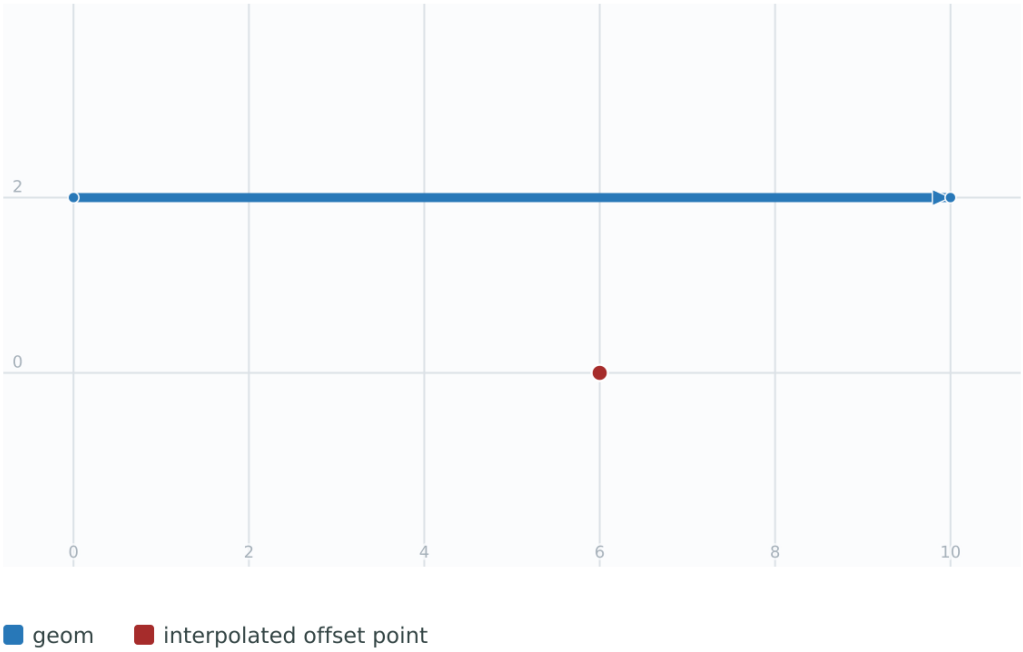


An interpolated point can be offset to the left or right of a line by using ST_OffsetCurve before interpolation. This is useful for locating an address point beside a street centerline.

Code

```
WITH street AS (
    SELECT 'LINESTRING(0 0, 10 0)'::geometry AS geom,
           100 AS from_num,
           200 AS to_num
)
SELECT
    ST_LineInterpolatePoint(
        ST_OffsetCurve(geom, -2),
        (160 - from_num)::float8 / (to_num - from_num)
    ) AS interpolated_offset_point
FROM street;
```

Figure



Voir aussi

ST_LineInterpolatePoints, ST_3DLineInterpolatePoint, ST_LineLocatePoint, ST_OffsetCurve

7.19.2 ST_3DLineInterpolatePoint

ST_3DLineInterpolatePoint — Renvoie un point interpolé le long d'une ligne 3D à un emplacement fractionnaire.

Synopsis

geometry **ST_3DLineInterpolatePoint**(geometry a_linestring, float8 a_fraction);

Description

Retourne un point interpolé sur une ligne. Le premier argument doit être une LINESTRING. Le second argument est un float8 entre 0 et 1 représentant la fraction de la longueur totale de la ligne où le point doit être situé.



Note
ST_LineInterpolatePoint calcule des points en 2D et interpole ensuite les valeurs de Z et M, alors que cette fonction calcule des points en 3D et n'interpole que la valeur M.

Disponibilité : 3.0.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Point de retour 20% le long de la ligne 3D

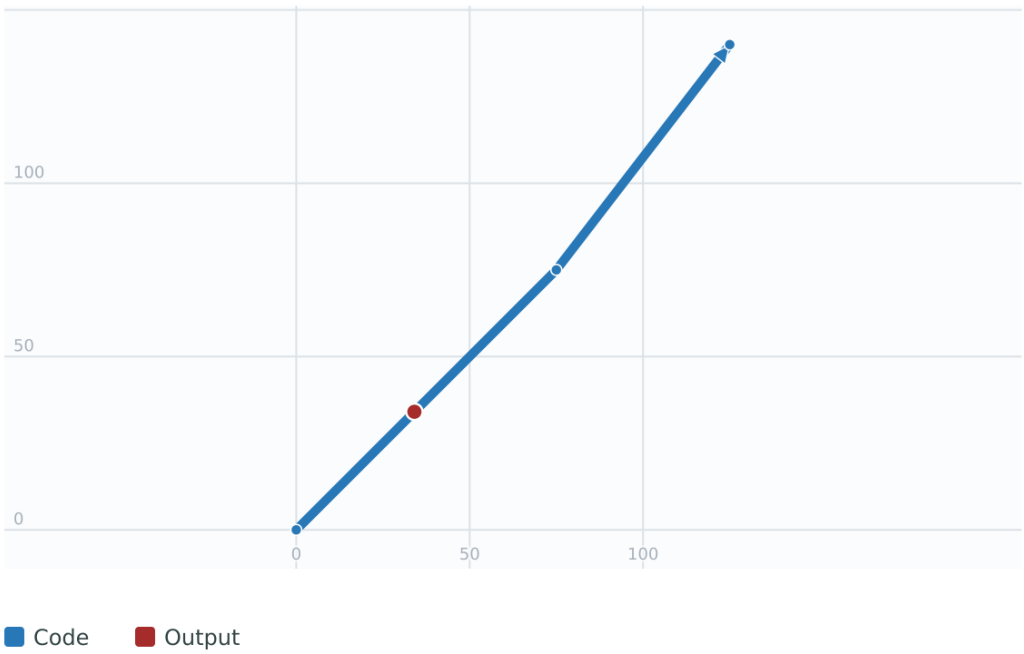
Code

```
SELECT ST_3DLineInterpolatePoint(
'LINESTRING(25 50 70,100 125 90,150 190 200)',
0.20);
```

Output

```
POINT Z (59.06758929108215 84.06758929108216 79.0846904776219)
```

Figure



Voir aussi

ST_LineInterpolatePoint, ST_LineInterpolatePoints, ST_LineLocatePoint

7.19.3 ST_LineInterpolatePoints

ST_LineInterpolatePoints — Renvoie des points interpolés le long d'une ligne à un intervalle fractionnaire.

Synopsis

geometry **ST_LineInterpolatePoints**(geometry a_linestring, float8 a_fraction, boolean repeat);
geography **ST_LineInterpolatePoints**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true, boolean repeat = true);

Description

Renvoie un ou plusieurs points interpolés le long d'une ligne à un intervalle fractionnaire. Le premier argument doit être une LINESTRING. Le second argument est un float8 entre 0 et 1 représentant l'espacement entre les points en tant que fraction de la longueur de la ligne. Si le troisième argument est faux, un seul point sera construit (ce qui est équivalent à ST_LineInterpolatePoint.)

Si le résultat a zéro ou un point, il est retourné sous forme de POINT. S'il a deux points ou plus, il est retourné sous la forme d'un MULTIPOINT.

Disponibilité : 2.5.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Exemples

This example returns points every 20 percent along a two-dimensional line.

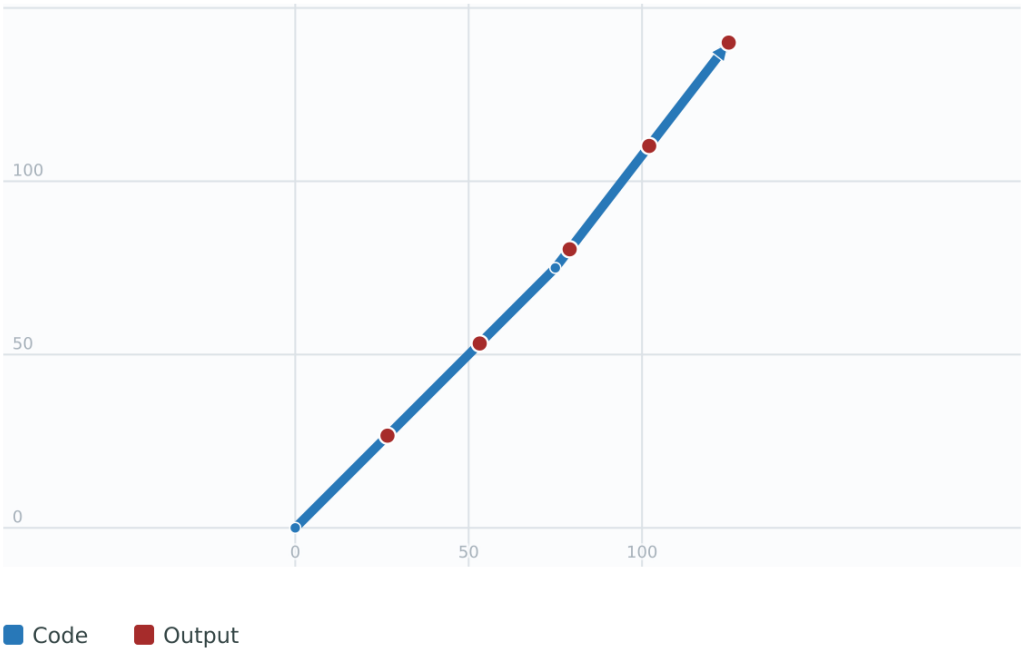
Code

```
SELECT ST_LineInterpolatePoints(
'LINESTRING(25 50,100 125,150 190)',
0.20);
```

Output

```
MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486),(104.132163186446 130.37181214238) ↩
    ,(127.066081593223 160.18590607119),(150 190))
```

Figure



Voir aussi

ST_LineInterpolatePoint, ST_LineLocatePoint

7.19.4 ST_LineLocatePoint

ST_LineLocatePoint — Renvoie l'emplacement fractionnaire du point le plus proche d'un point sur une ligne.

Synopsis

float8 **ST_LineLocatePoint**(geometry a_linestring, geometry a_point);
float8 **ST_LineLocatePoint**(geography a_linestring, geography a_point, boolean use_spheroid = true);

Description

Renvoie un flottant compris entre 0 et 1 représentant l'emplacement du point le plus proche du point donné sur une LineString, en tant que fraction de la ligne 2D longueur.

Vous pouvez utiliser l'emplacement renvoyé pour extraire un point (ST_LineInterpolatePoint) ou une sous-ligne (ST_LineSubstring).

Cette méthode est utile pour calculer approximativement le nombre d'adresses

Disponibilité : 1.1.0

Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Line_Locate_Point.

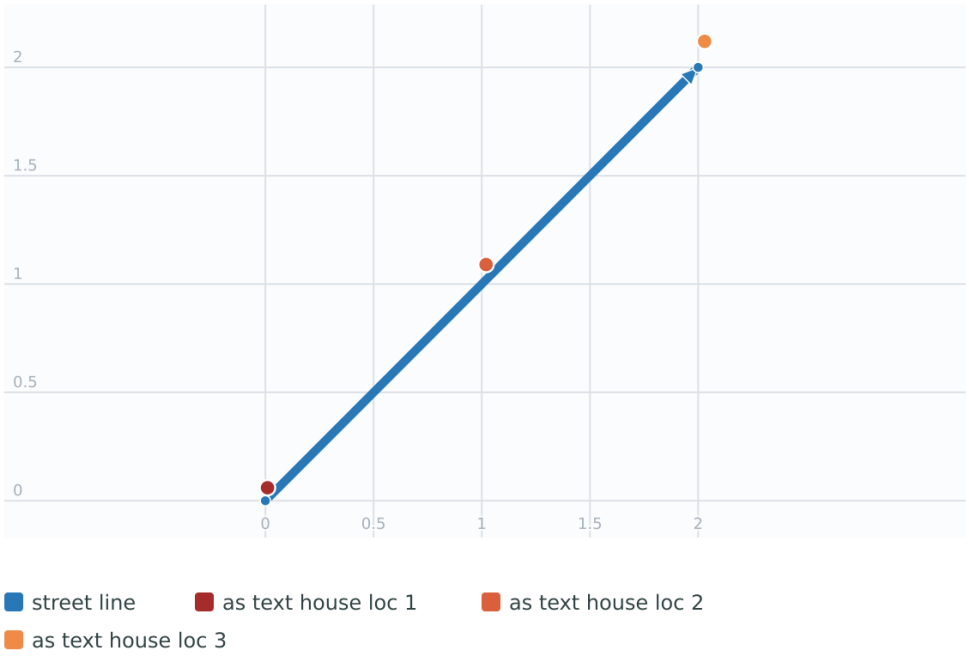
Exemples

This example approximates the street number of a house by locating the closest point on a street line. The subquery generates sample house and street data, and `ST_DWithin` excludes houses that are too far from the street.

Code

```
SELECT house_loc As as_text_house_loc,
    start_house_num +
        CAST( (end_house_num - start_house_num)
            * ST_LineLocatePoint(street_line, house_loc) As integer) As house_num
FROM
(SELECT 'LINESTRING(1 2,3 4)'::geometry As street_line,
    ST_Point(x*1.01, y*1.03) As house_loc, 10 As start_house_num,
        20 As end_house_num
FROM generate_series(1, 3) x CROSS JOIN generate_series(2, 4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);
```

Figure



This example finds the closest point on a line to a point or other geometry.

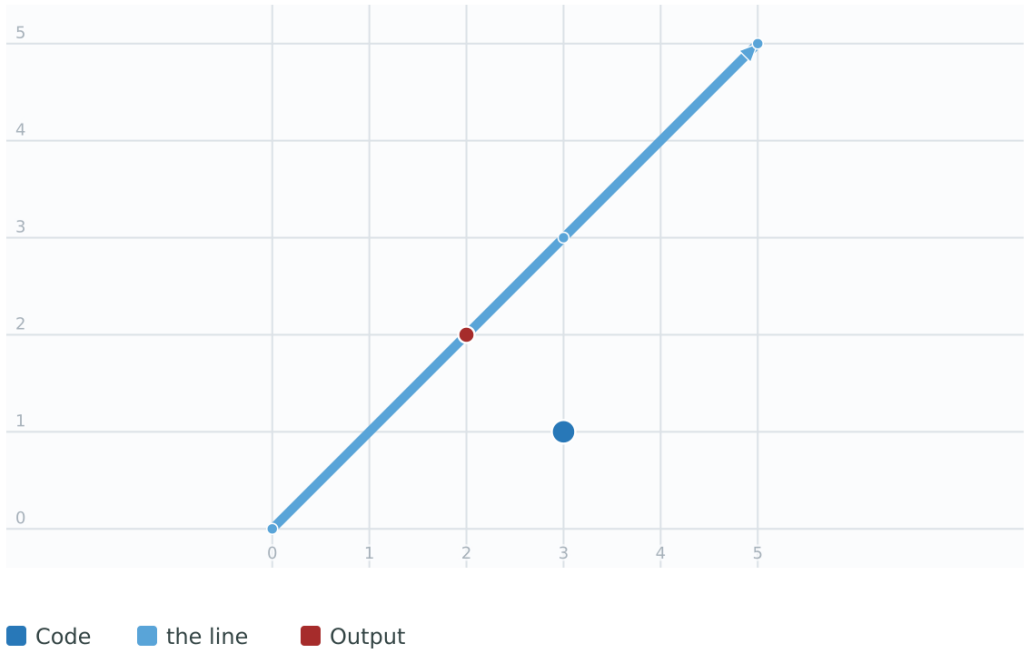
Code

```
SELECT ST_LineInterpolatePoint(
    foo.the_line,
    ST_LineLocatePoint(foo.the_line, 'POINT(4 3)'::geometry))
FROM (SELECT 'LINESTRING(1 2,4 5,6 7)'::geometry As the_line) As foo;
```

Output

```
POINT(3 4)
```

Figure



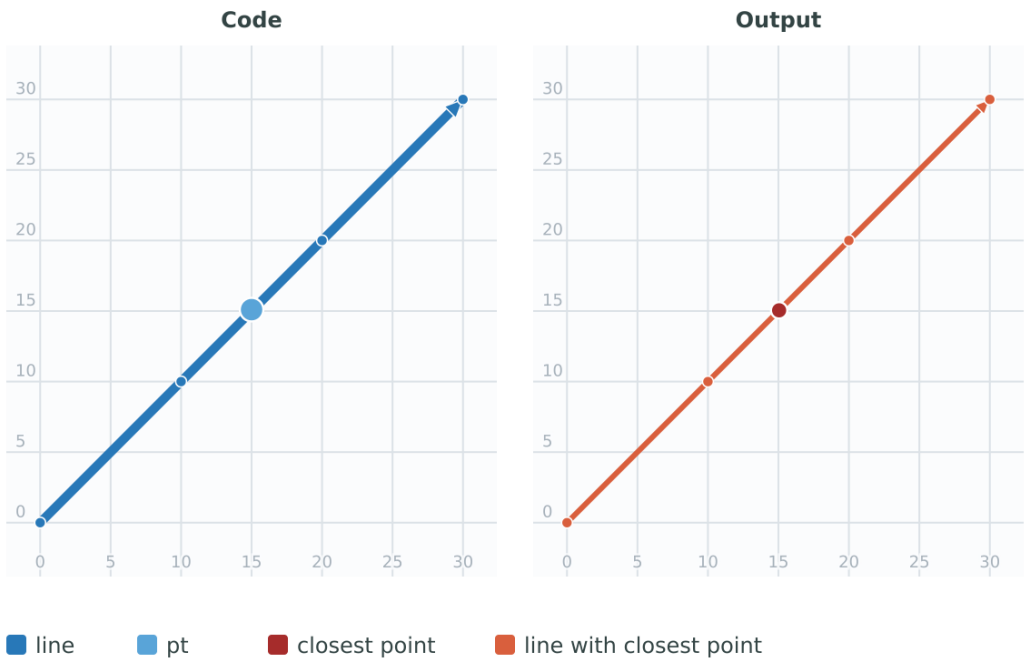
This example locates the closest segment and point on a LineString.

Code

```
WITH data AS (
  SELECT
    'LINESTRING (0 0, 10 10, 20 20, 30 30)'::geometry AS line,
    'POINT(15 15.1)'::geometry AS pt
),
segments AS (
  SELECT
    s.i,
    d.line,
    d.pt,
    ST_MakeLine(ST_PointN(d.line, s.i), ST_PointN(d.line, s.i + 1)) AS segment
  FROM data AS d
  CROSS JOIN LATERAL generate_series(1, ST_NumPoints(d.line) - 1) AS s(i)
)
SELECT
  i AS segment_number,
  round(ST_Distance(segment, pt)::numeric, 3) AS distance,
  ST_ClosestPoint(segment, pt) AS closest_point,
  ST_AddPoint(line, ST_ClosestPoint(segment, pt), i) AS line_with_closest_point
```

```
FROM segments
ORDER BY ST_Distance(segment, pt)
LIMIT 1;
```

Figure



Voir aussi

ST_AddPoint, ST_ClosestPoint, ST_DWithin, ST_Length2D, ST_LineInterpolatePoint, ST_LineSubstring

7.19.5 ST_LineSubstring

ST_LineSubstring — Renvoie la partie d'une ligne située entre deux emplacements fractionnaires.

Synopsis

geometry **ST_LineSubstring**(geometry a_linestring, float8 startfraction, float8 endfraction);
geography **ST_LineSubstring**(geography a_linestring, float8 startfraction, float8 endfraction);

Description

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations. For geometry inputs, the fractions are measured in 2D line length. The Z and M values are interpolated for added endpoints if present.

Si `startfraction` et `endfraction` ont la même valeur, cela équivaut à ST_LineInterpolatePoint.



Note
Cette méthode ne fonctionne qu'avec les LINESTRING. Pour l'utiliser sur des MULTILINESTRINGs contigues, il faut d'abord les joindre avec ST_LineMerge.

Note
Depuis la version 1.1.1, cette fonction interpole les valeurs de M et Z. Les versions précédentes fixent Z et M à des valeurs non spécifiées.

Amélioration : 3.4.0 - La prise en charge de la géographie a été introduite.

Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Line_Substring.

Disponibilité : 1.1.0, prise en charge des Z et M ajoutée dans la 1.1.1

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Extract the middle third of a LineString, from 0.333 to 0.666.

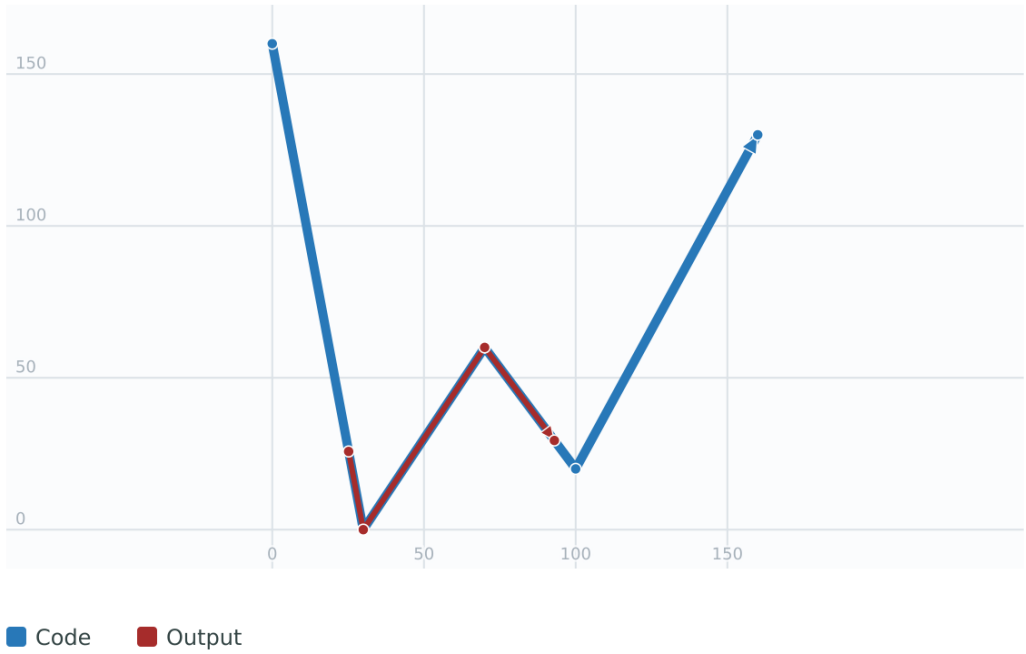
Code

```
SELECT ST_LineSubstring(
'LINESTRING (20 180,50 20,90 80,120 40,180 150)',
0.333,
0.666);
```

Output

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862 49.36542599789519)
```

Figure



Si les points de départ et d'arrivée sont identiques, le résultat est un POINT.

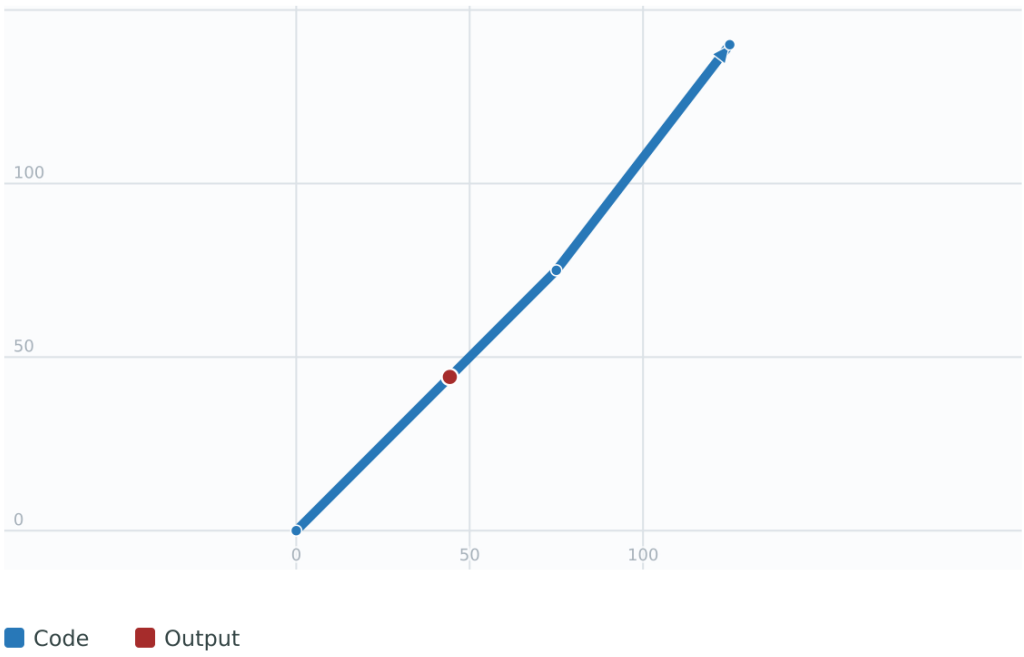
Code

```
SELECT ST_LineSubstring(
'LINESTRING(25 50,100 125,150 190)',
0.333,
0.333);
```

Output

```
POINT(69.28469348539744 94.28469348539744)
```

Figure



Une requête pour découper une LineString en sections de longueur 100 ou moins. Elle utilise `generate_series()` avec un CROSS JOIN LATERAL pour produire l'équivalent d'une boucle FOR.

The final generated step is filtered out when the line length is an exact multiple of the requested section length.

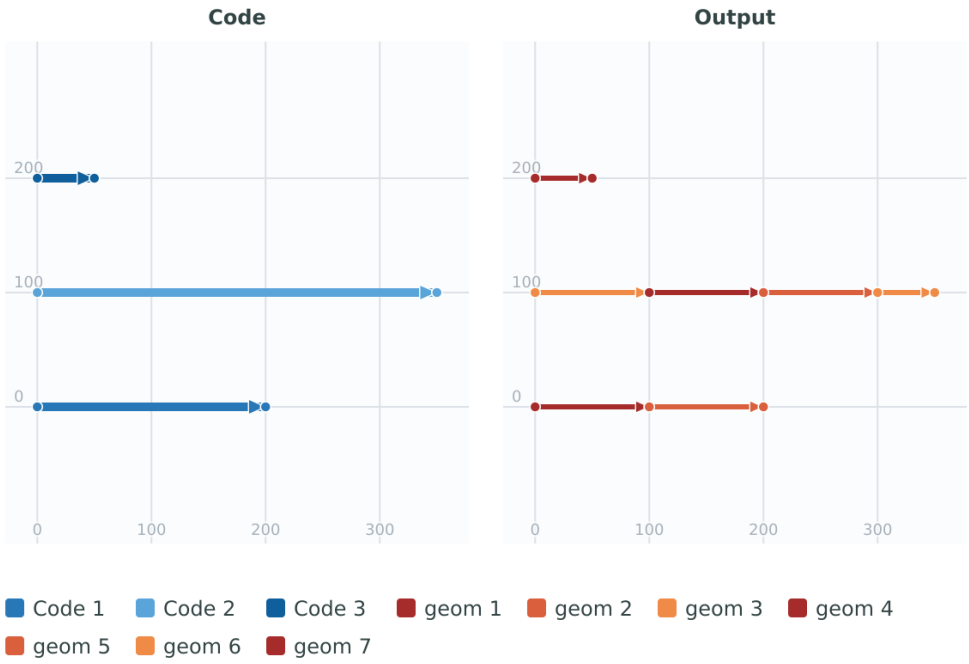
Code

```
WITH data(id, geom) AS (VALUES
        ( 'A', 'LINESTRING( 0 0, 200 0)'::geometry ),
        ( 'B', 'LINESTRING( 0 100, 350 100)'::geometry ),
        ( 'C', 'LINESTRING( 0 200, 50 200)'::geometry )
    )
SELECT id, i,
       ST_LineSubstring(geom, startfrac, LEAST( endfrac, 1 )) AS geom
FROM (
    SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
    ) AS d
CROSS JOIN LATERAL (
    SELECT i, (sublen * i) / len AS startfrac,
              (sublen * (i+1)) / len AS endfrac
    FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
    WHERE (sublen * i) / len <
> 1.0
    ) AS d2;
```

Output

```
id | i |            geom
---+---+-----------------------------
 A  | 0 | LINESTRING(0 0,100 0)
 A  | 1 | LINESTRING(100 0,200 0)
 B  | 0 | LINESTRING(0 100,100 100)
 B  | 1 | LINESTRING(100 100,200 100)
 B  | 2 | LINESTRING(200 100,300 100)
 B  | 3 | LINESTRING(300 100,350 100)
 C  | 0 | LINESTRING(0 200,50 200)
```

Figure



Split a LineString by points by locating the points along the line, adding the 0 and 1 endpoints, and extracting substrings between adjacent locations.

Code

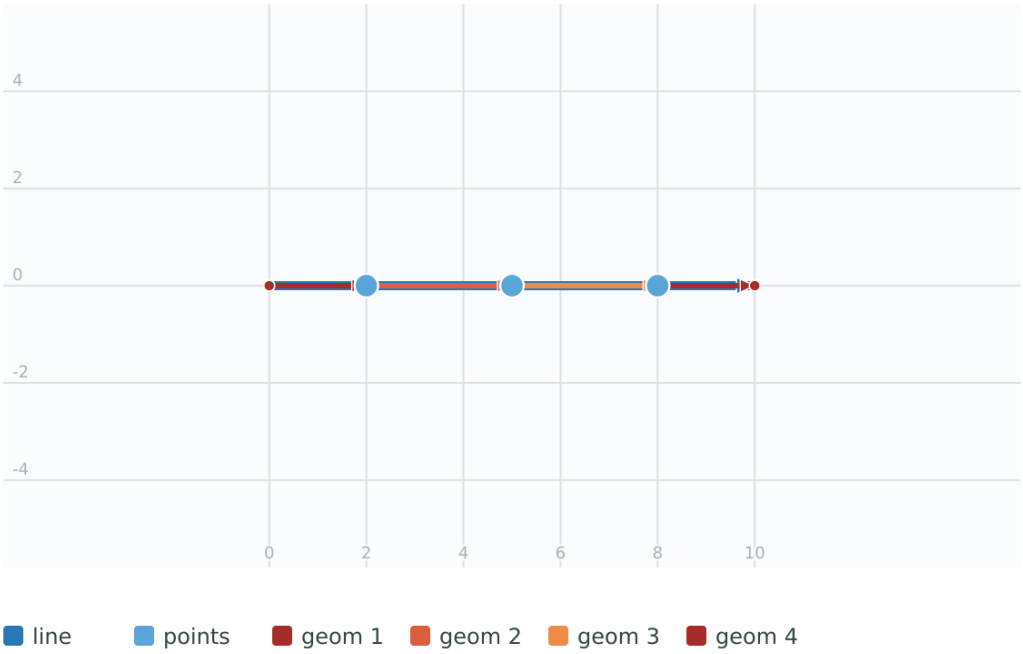
```
WITH data AS (
  SELECT 'LINESTRING(0 0, 10 0)'::geometry AS line,
         'MULTIPOINT((2 0),(5 0),(8 0))'::geometry AS points
),
fractions AS (
  SELECT 0.0 AS fraction
  UNION ALL
  SELECT 1.0
  UNION ALL
  SELECT ST_LineLocatePoint(line, (dp).geom)
  FROM data
  CROSS JOIN LATERAL ST_Dump(points) AS dp
),
ordered AS (
  SELECT DISTINCT fraction
  FROM fractions
),
segments AS (
  SELECT fraction AS start_fraction,
         lead(fraction) OVER (ORDER BY fraction) AS end_fraction
  FROM ordered
)
```

```
SELECT row_number() OVER (ORDER BY start_fraction) AS segment_id,
       ST_LineSubstring(line, start_fraction, end_fraction) AS geom
FROM data
CROSS JOIN segments
WHERE end_fraction IS NOT NULL
  AND start_fraction <
> end_fraction
ORDER BY start_fraction;
```

Output

```
segment_id |         geom
-----------+----------------------
         1 | LINESTRING(0 0,2 0)
         2 | LINESTRING(2 0,5 0)
         3 | LINESTRING(5 0,8 0)
         4 | LINESTRING(8 0,10 0)
```

Figure



Mesures de mise en œuvre de la geography le long d'un sphéroïde, geometry le long d'une ligne

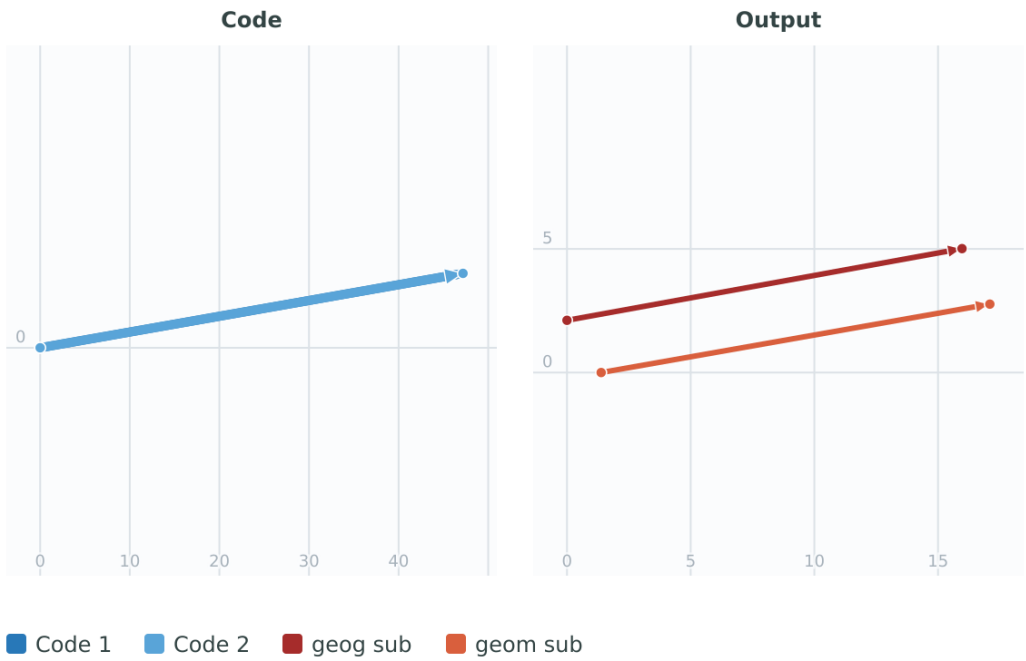
Code

```
SELECT ST_LineSubstring(
'LINESTRING(-118.2436 34.0522,-71.0570 42.3611)'::geography,
0.333,
0.666) AS geog_sub,
ST_LineSubstring(
    'LINESTRING(-118.2436 34.0522,-71.0570 42.3611)'::geometry,
    0.333,
    0.666) AS geom_sub;
```

Output

```
-[ RECORD 1 ]----------
geog_sub | LINESTRING(-103.911641 38.931128,-87.941787 41.831072)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)
```

Figure



For geometry inputs, the fractional locations are based on the 2D length of the line, even when the input has Z values. The resulting Z values are interpolated along the selected 2D location.

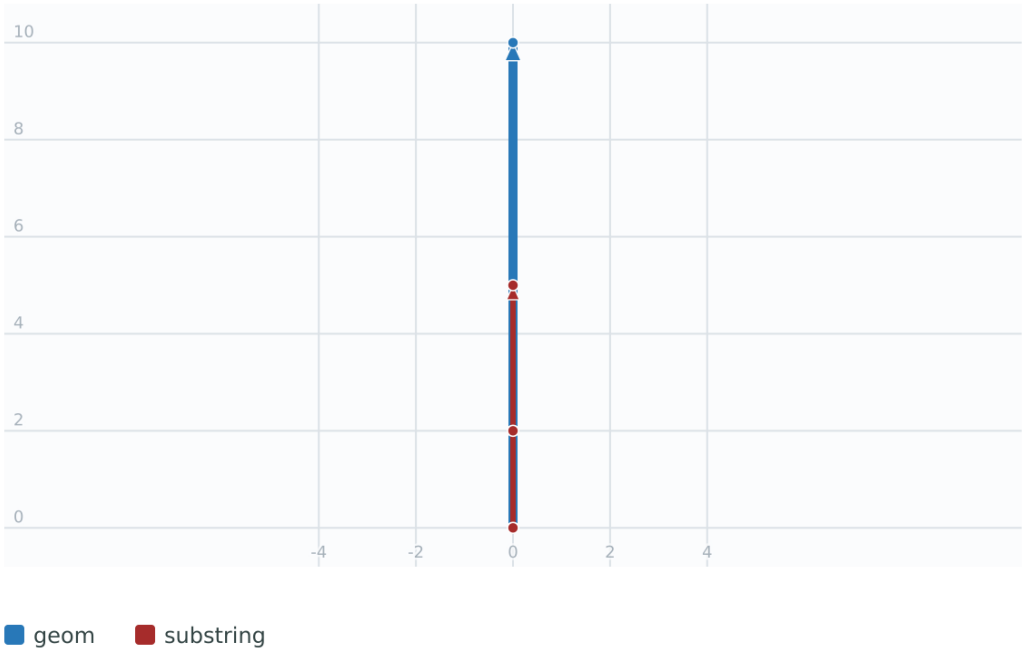
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,0 2 5,0 10 10)'::geometry AS geom
)
SELECT ST_Length(geom) AS length_2d,
       ST_3DLength(geom) AS length_3d,
       ST_LineSubstring(geom, 0, 0.5) AS substring
FROM data;
```

Output

```
length_2d |    length_3d     |              substring
----------+------------------+-------------------------------------
        10 | 14.819145939191106 | LINESTRING Z (0 0 0,0 2 5,0 5 6.875)
```

Figure



Voir aussi

ST_Length, ST_LineExtend, ST_LineInterpolatePoint, ST_LineLocatePoint, ST_LineMerge, ST_Split

7.19.6 ST_LocateAlong

ST_LocateAlong — Renvoie le(s) point(s) d'une géométrie qui correspond(ent) à une valeur de mesure.

Synopsis

geometry **ST_LocateAlong**(geometry geom_with_measure, float8 measure, float8 offset = 0);

Description

Renvoie le(s) emplacement(s) le long d'une géométrie mesurée qui possède(nt) les valeurs de mesure données. Le résultat est un point ou un multipoint. Les entrées polygonales ne sont pas prises en charge.

Si `offset` est fourni, le résultat est décalé à gauche ou à droite de la ligne d'entrée de la distance spécifiée. Un décalage positif se fera vers la gauche, et un décalage négatif vers la droite.



Note
Cette fonction n'est utilisée que pour les géométries linéaires avec une composante M

La sémantique est spécifiée par la norme *ISO/IEC 13249-3 SQL/MM Spatial*.

Disponibilité : 1.1.0 sous l'ancien nom ST_Locate_Along_Measure.

Modifié : 2.0.0 dans les versions précédentes, cette fonction était appelée ST_Locate_Along_Measure.

Cette fonction prend en charge les coordonnées M.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1.13

Exemples

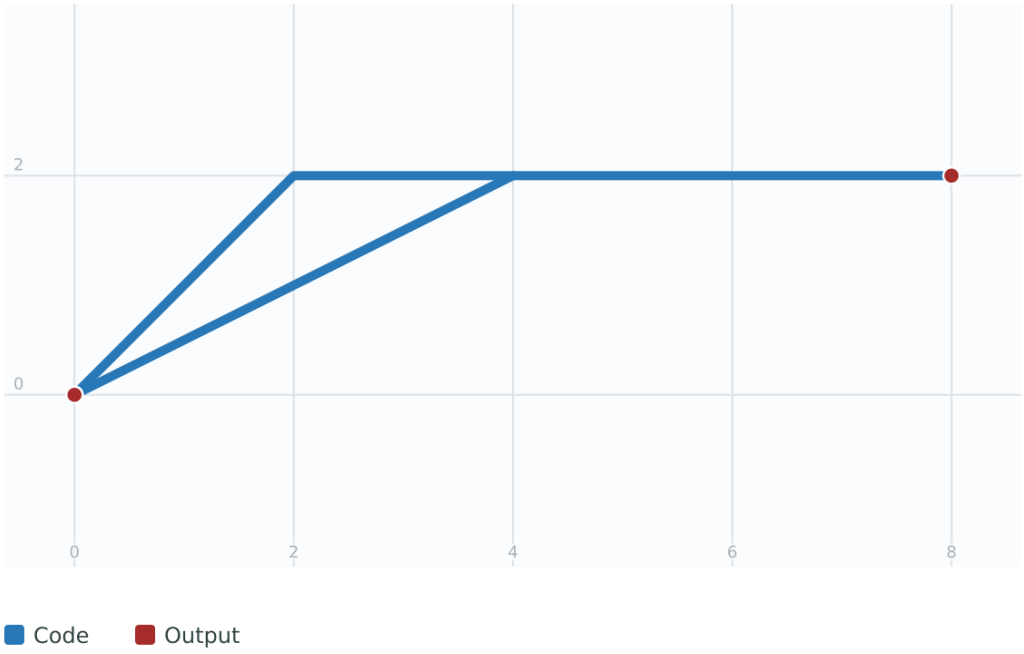
Code

```
SELECT ST_LocateAlong(
'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
3);
```

Output

```
MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

Figure



Voir aussi

ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint

7.19.7 ST_LocateBetween

ST_LocateBetween — Renvoie les parties d'une géométrie qui correspondent à un intervalle de mesure.

Synopsis

geometry **ST_LocateBetween**(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0);

Description

Renvoie une géométrie (collection) contenant les parties de la géométrie mesurée en entrée qui correspondent à l'intervalle de mesure spécifié (inclusivement).

Si le `offset` est fourni, le résultat est décalé à gauche ou à droite de la ligne d'entrée de la distance spécifiée. Un décalage positif se fera vers la gauche, et un décalage négatif vers la droite.

Couper un POLYGONE non convexe peut produire une géométrie non valide.

La sémantique est spécifiée par la norme *ISO/IEC 13249-3 SQL/MM Spatial*.

Disponibilité : 1.1.0 sous l'ancien nom ST_Locate_Between_Measures.

Modifié : 2.0.0 - dans les versions précédentes, cette fonction s'appelait ST_Locate_Between_Measures.

Amélioration : 3.0.0 - ajout de la prise en charge du POLYGONE, du TIN et du TRIANGLE.

Cette fonction prend en charge les coordonnées M.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1

Exemples

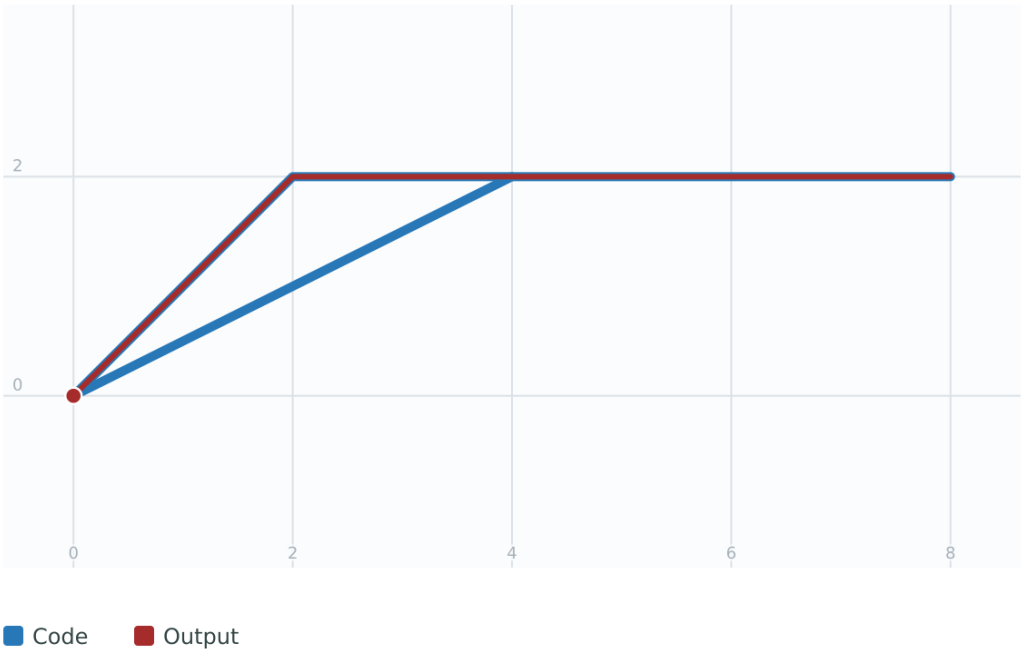
Code

```
SELECT ST_LocateBetween(
'MULTILINESTRING M ((1 2 3,3 4 2,9 4 3),(1 2 3,5 4 5))'::geometry,
1.5,
3);
```

Output

```
GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```

Figure



Extract the section between measures 2 and 8, offset to the left. Because the clipped line and the offset result share the same coordinate space, this example is intended for a single overlay panel rather than separate side-by-side figures.

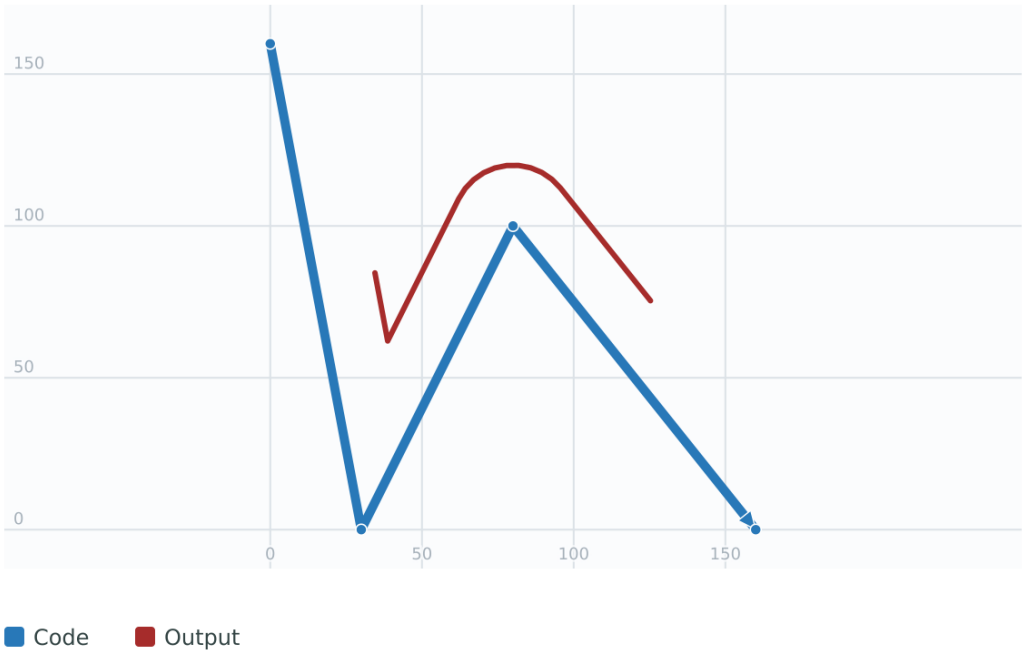
Code

```
SELECT ST_LocateBetween(
ST_AddMeasure(
    'LINESTRING (20 180,50 20,100 120,180 20)',
    0,
    10),
2,
8,
20);
```

Output

```
MULTILINESTRING((54.49835 104.53427,58.700561 82.122481,82.111456 128.944272,84.248937 132.324934,87.01637 ↩
    135.21267,90.303073 137.491987,93.977598 139.071724,97.892984 139.888702,101.892639 139.910247,105.816599 ↩
    139.135495,109.507928 137.595434,112.818995 135.351657,115.617376 132.493901,145.310173 95.377905))
```

Figure



Voir aussi

ST_LocateAlong, ST_LocateBetweenElevations

7.19.8 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Renvoie les parties d'une géométrie qui se trouvent dans un intervalle d'élévation (Z).

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom, float8 elevation_start, float8 elevation_end);

Description

Renvoie une géométrie (collection) contenant les parties d'une géométrie qui se trouvent dans une plage d'élévation (Z).

Couper un POLYGONE non convexe peut produire une géométrie non valide.

Disponibilité : 1.4.0

Amélioration : 3.0.0 - ajout de la prise en charge du POLYGONE, du TIN et du TRIANGLE.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

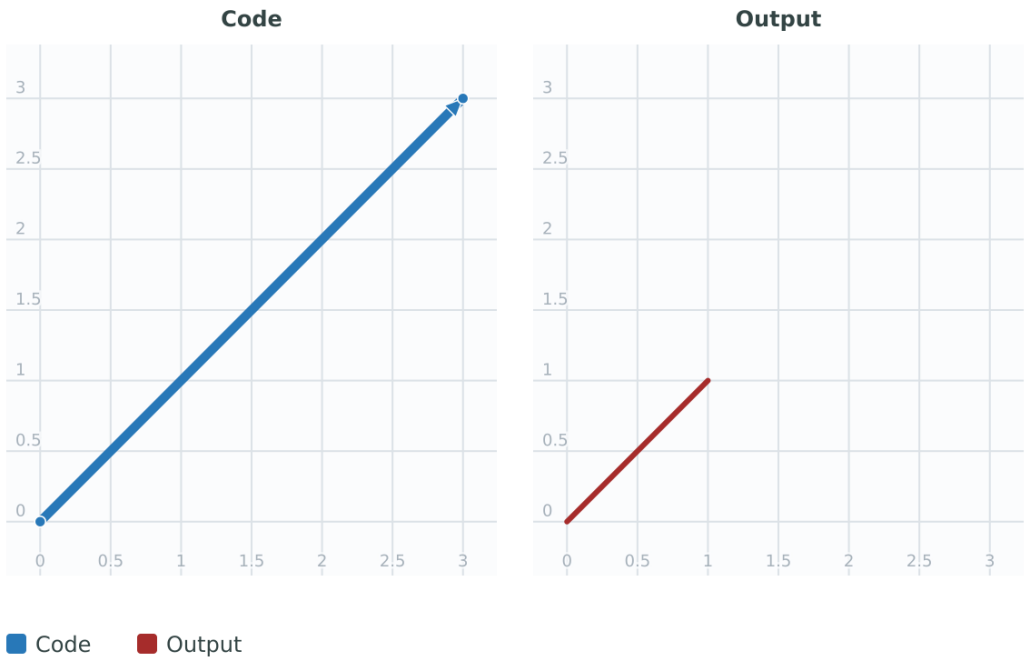
Code

```
SELECT ST_LocateBetweenElevations(
'LINESTRING(1 2 3,4 5 6)'::geometry,
2,
4);
```

Output

```
MULTILINESTRING Z ((1 2 3,2 3 4))
```

Figure



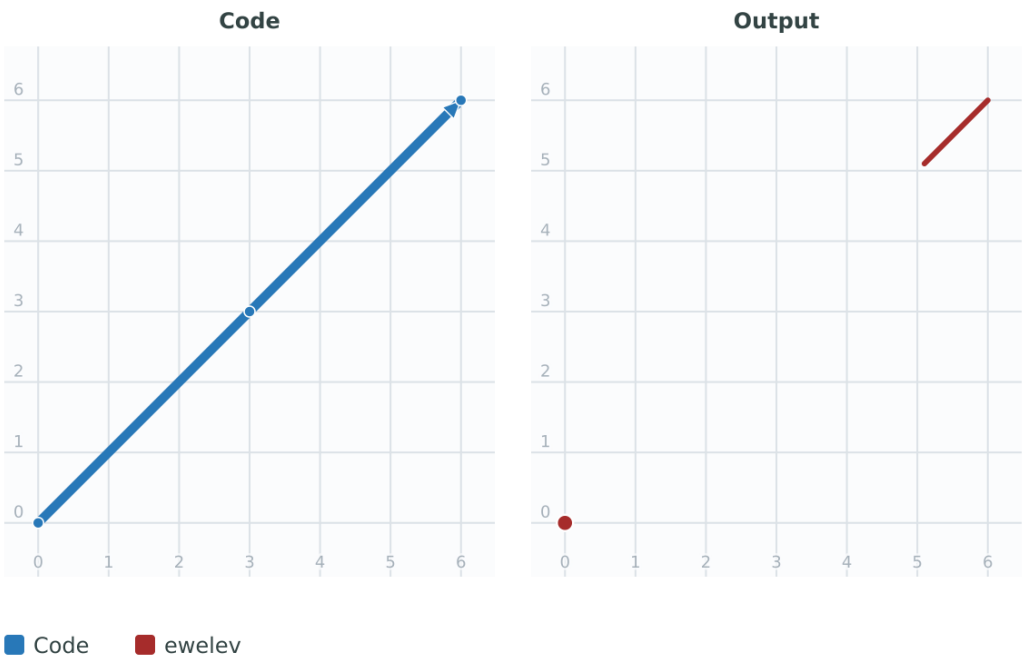
Code

```
SELECT ST_LocateBetweenElevations(
'LINESTRING(1 2 6,4 5 -1,7 8 9)',
6,
9) As ewelev;
```

Output

```
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

Figure



Voir aussi

ST_Dump, ST_LocateBetween

7.19.9 ST_InterpolatePoint

ST_InterpolatePoint — Renvoie la mesure interpolée d'une géométrie la plus proche d'un point.

Synopsis

float8 **ST_InterpolatePoint**(geometry linear_geom_with_measure, geometry point);

Description

Renvoie une valeur de mesure interpolée d'une géométrie linéaire mesurée à l'emplacement le plus proche du point donné.

> **Note**
> Cette fonction n'est utilisée que pour les géométries linéaires avec une composante M

Disponibilité : 2.0.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

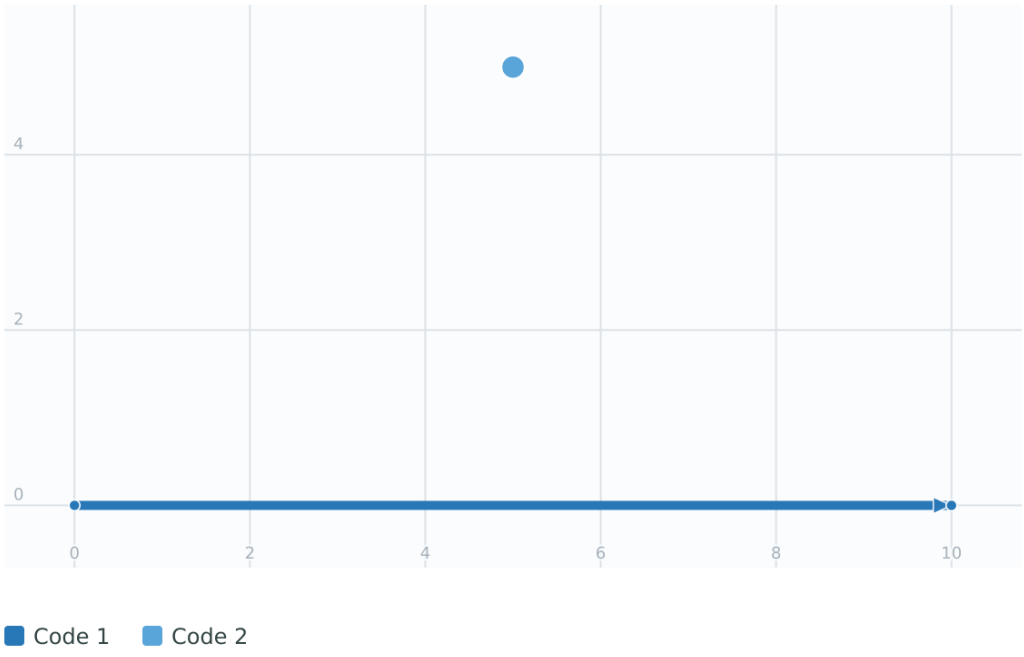
Code

```
SELECT ST_InterpolatePoint(
'LINESTRING M (0 0 0,10 0 20)',
'POINT(5 5)');
```

Output

```
10
```

Figure



Voir aussi

ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.10 ST_3DInterpolatePoint

ST_3DInterpolatePoint — Returns the interpolated measure of a geometry closest to a point in 3D.

Synopsis

float8 **ST_3DInterpolatePoint**(geometry linear_geom_with_measure, geometry point);

Description

Returns the interpolated measure value of a linear ZM geometry at the location closest to the given point, using 3D (XYZ) distance for the projection. Use this function when the geometry has significant Z variation, such as flight trajectories, where ST_InterpolatePoint would give incorrect results by ignoring the Z dimension.



Note
The line must have both Z and M dimensions. The point should have a Z dimension.

Availability: 3.7.0

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

This example interpolates on a line rising diagonally in Z, using a point at the 3D midpoint.

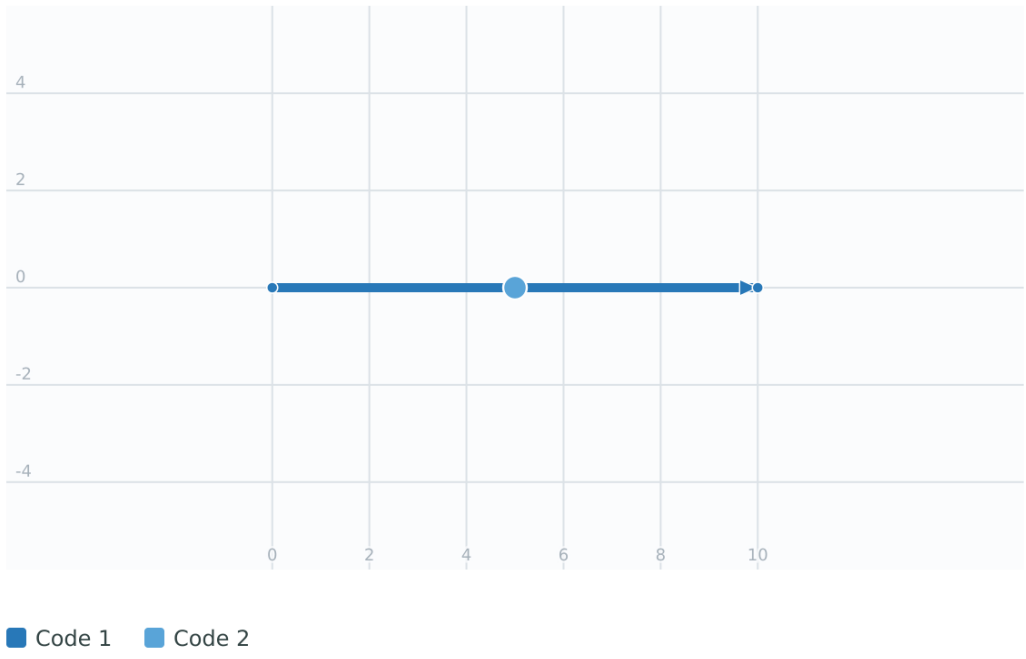
Code

```
SELECT ST_3DInterpolatePoint(
'LINESTRING ZM (0 0 0 0,10 0 10 20)',
'POINT Z (5 0 5)');
```

Output

```
10
```

Figure



This example uses a vertical line with zero XY extent, where 3D projection works but 2D projection does not.

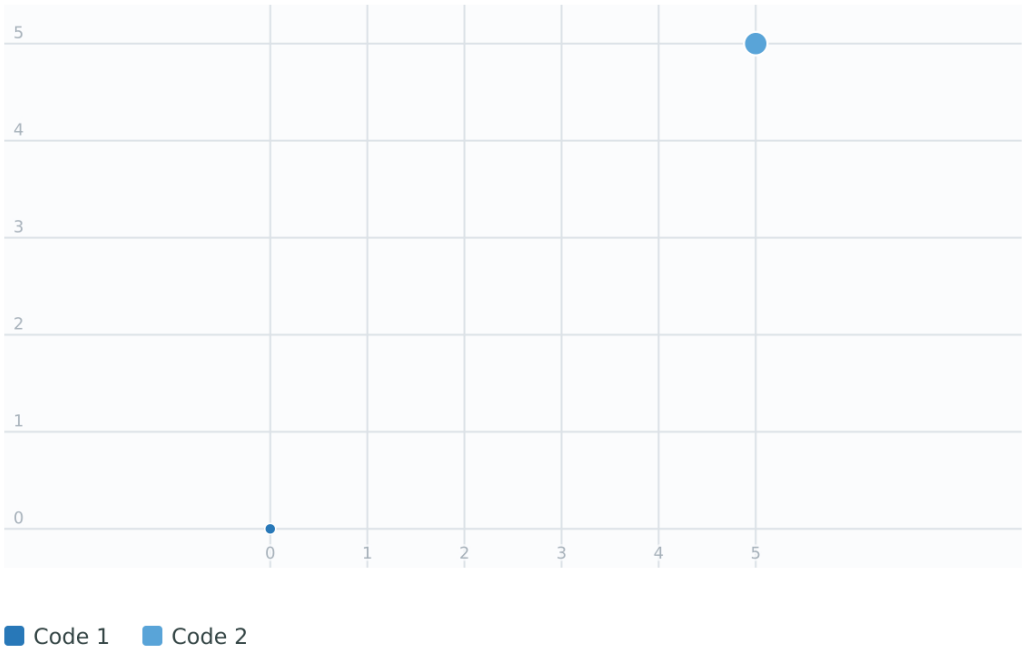
Code

```
SELECT ST_3DInterpolatePoint(
'LINESTRING ZM (0 0 0 0,0 0 10 100)',
'POINT Z (5 5 5)');
```

Output

```
50
```

Figure



Voir aussi

ST_InterpolatePoint, ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.11 ST_AddMeasure

ST_AddMeasure — Interpole les mesures le long d'une géométrie linéaire.

Synopsis

geometry **ST_AddMeasure**(geometry geom_mline, float8 measure_start, float8 measure_end);

Description

Renvoie une géométrie dérivée avec des valeurs de mesure interpolées linéairement entre les points de départ et d'arrivée. Si la géométrie n'a pas de dimension de mesure, une dimension est ajoutée. Si la géométrie a une dimension de mesure, elle est remplacée par de nouvelles valeurs. Seuls les LINESTRINGS et MULTILINESTRINGS sont pris en charge.

Disponibilité : 1.5.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

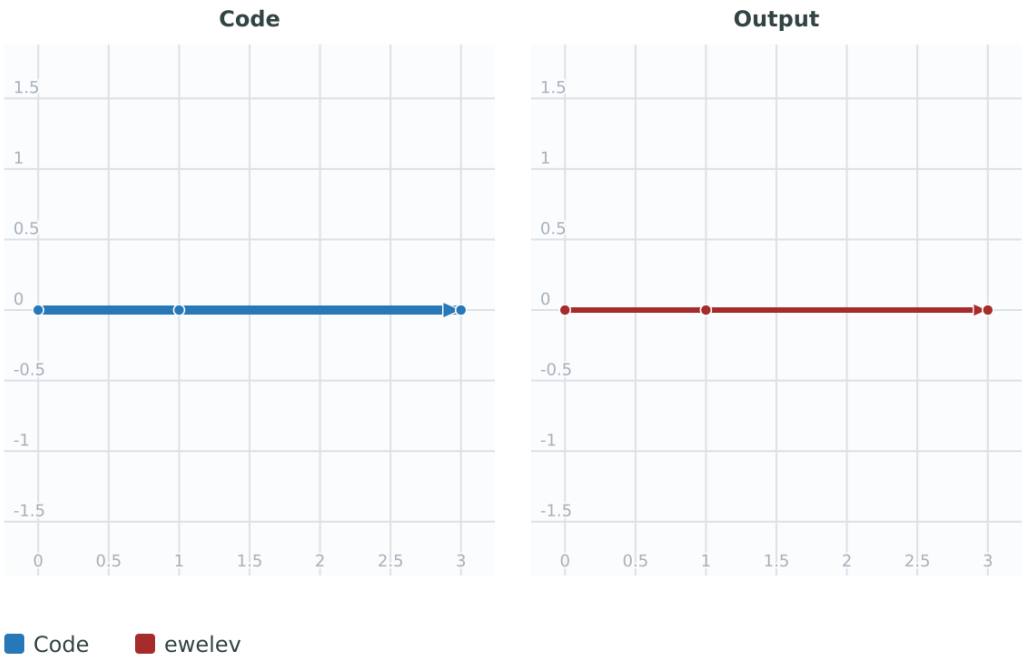
Code

```
SELECT ST_AddMeasure(
'LINESTRING(1 0,2 0,4 0)',
1,
4) As ewelev;
```

Output

```
LINESTRING M (1 0 1,2 0 2,4 0 4)
```

Figure



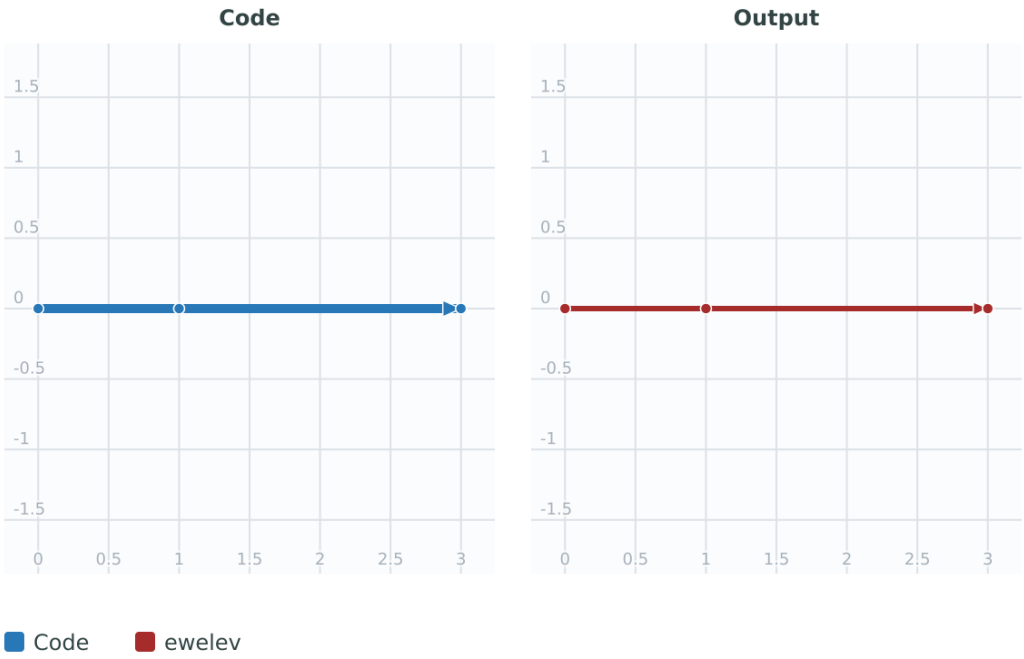
Code

```
SELECT ST_AddMeasure(
'LINESTRING(1 0 4,2 0 4,4 0 4)',
10,
40) As ewelev;
```

Output

```
LINESTRING ZM (1 0 4 10,2 0 4 20,4 0 4 40)
```

Figure



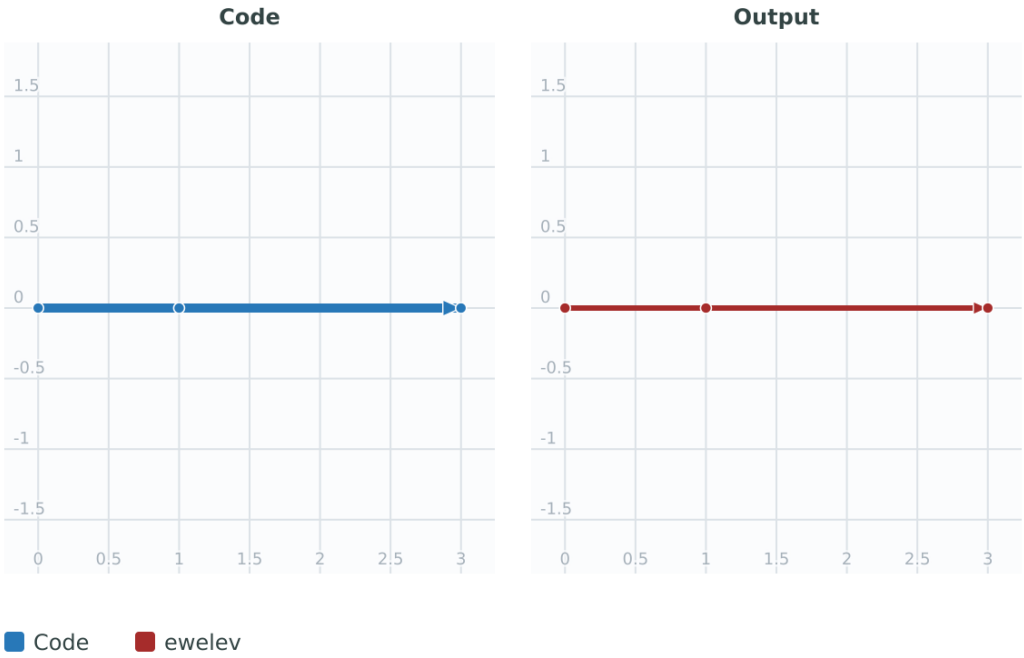
Code

```
SELECT ST_AddMeasure(
'LINESTRINGM(1 0 4, 2 0 4, 4 0 4)',
10,
40) As ewelev;
```

Output

```
LINESTRING M (1 0 10,2 0 20,4 0 40)
```

Figure



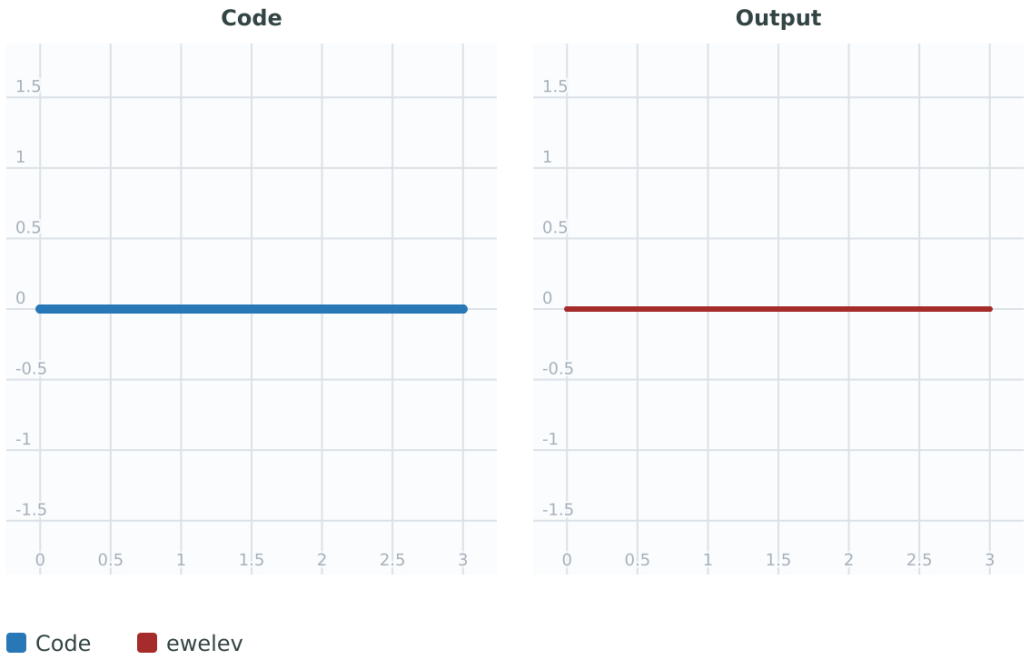
Code

```
SELECT ST_AddMeasure(
'MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4))',
10,
70) As ewelev;
```

Output

```
MULTILINESTRING M ((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

Figure



7.20 Fonctions de trajectoire

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — Teste si la géométrie est une trajectoire valide.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

Description

Teste si une géométrie encode une trajectoire valide. Une trajectoire valide est représentée comme une `LINESTRING` avec des mesures (valeurs M). Les valeurs de mesure doivent augmenter de chaque sommet au suivant.

Les trajectoires valides sont attendues comme entrées pour des fonctions spatio-temporelles telles que ST_ClosestPointOfApproach

Disponibilité : 2.2.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

This example constructs a valid trajectory.

Code

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1),
ST_MakePointM(0, 1, 2)));
```

Output

```
t
```

This example constructs an invalid trajectory.

Code

```
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0, 0, 1), ST_MakePointM(0, 1, 0)));
```

Output

```
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
f
```

Voir aussi

ST_ClosestPointOfApproach

7.20.2 ST_ClosestPointOfApproach

ST_ClosestPointOfApproach — Renvoie une mesure au point d'approche le plus proche de deux trajectoires.

Synopsis

float8 **ST_ClosestPointOfApproach**(geometry track1, geometry track2);

Description

Renvoie la plus petite mesure pour laquelle les points interpolés le long des trajectoires données sont les moins éloignés les uns des autres.

Les entrées doivent être des trajectoires valides comme vérifié par ST_IsValidTrajectory. Null est retourné si les trajectoires ne se chevauchent pas dans leurs plages M.

Pour obtenir les points réels à la mesure calculée, utilisez ST_LocateAlong .

Disponibilité : 2.2.0

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Return the time in which two objects moving between 10:00 and 11:00 are closest to each other and their distance at that point.

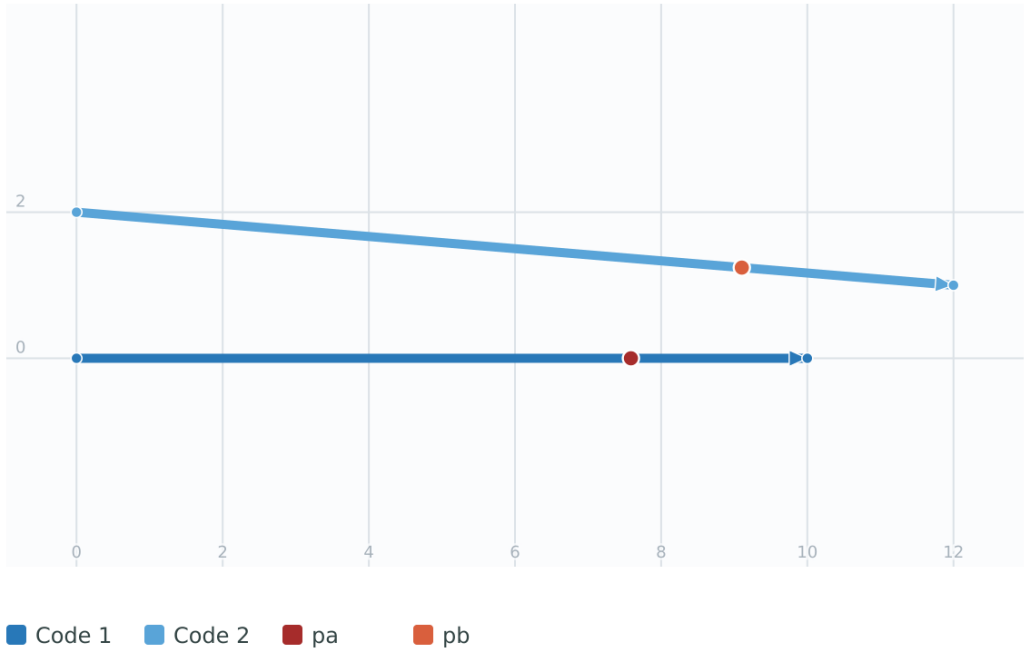
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a, b) m FROM inp
), points AS (
  SELECT ST_GeometryN(ST_LocateAlong(a, m), 1) pa,
         ST_GeometryN(ST_LocateAlong(b, m), 1) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) AT TIME ZONE 'UTC' t,
       ST_3DDistance(pa, pb) distance,
       pa AS pa, pb AS pb
FROM points, cpa;
```

Output

```
-[ RECORD 1 ]----------------------------
t        | 2015-05-26 10:45:31.034483
distance | 1.9652147377620688
pa       | POINT ZM (7.59 0 3.79 1432637131.03)
pb       | POINT ZM (9.1 1.24 3.93 1432637131.03)
```

Figure



Voir aussi

ST_IsValidTrajectory, ST_DistanceCPA, ST_LocateAlong, ST_AddMeasure

7.20.3 ST_DistanceCPA

ST_DistanceCPA — Renvoie la distance entre le point d'approche le plus proche de deux trajectoires.

Synopsis

float8 **ST_DistanceCPA**(geometry track1, geometry track2);

Description

Renvoie la distance (en 2D) entre deux trajectoires à leur point d'approche le plus proche.

Les entrées doivent être des trajectoires valides comme vérifié par ST_IsValidTrajectory. Null est retourné si les trajectoires ne se chevauchent pas dans leurs plages M.

Disponibilité : 2.2.0

 Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Return the minimum distance of two objects moving between 10:00 and 11:00.

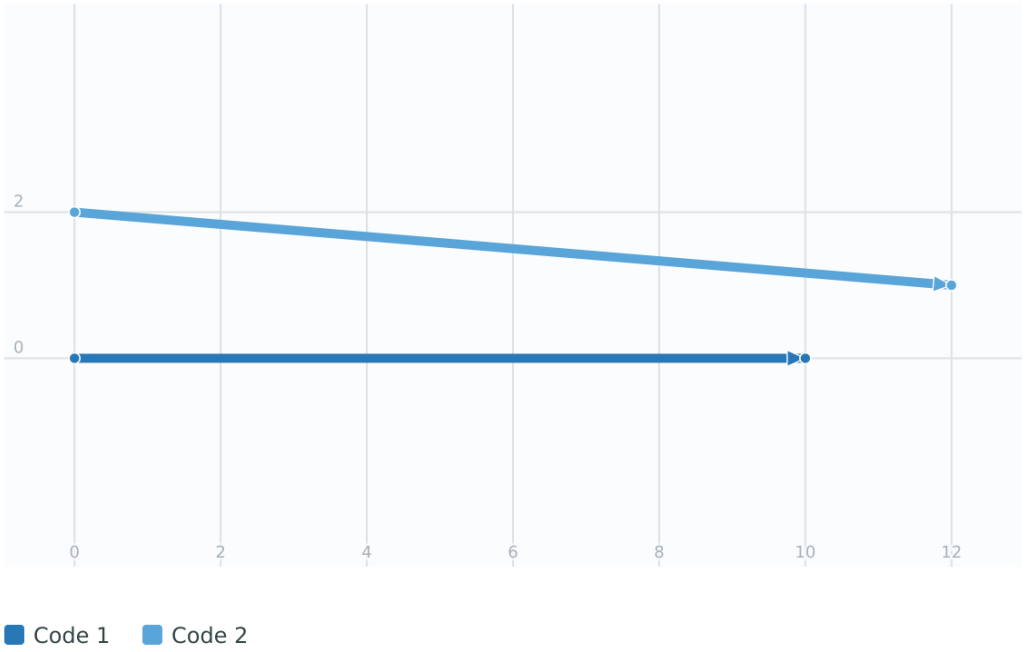
Code

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_DistanceCPA(a, b) distance FROM inp;
```

Output

```
1.9652147377620688
```

Figure



Voir aussi

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_AddMeasure, |=|

7.20.4 ST_CPAWithin

ST_CPAWithin — Teste si le point d'approche le plus proche de deux trajectoires se trouve dans la distance spécifiée.

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

Description

Teste si deux objets en mouvement ont déjà été plus proches que la distance spécifiée.

Les entrées doivent être des trajectoires valides comme vérifié par ST_IsValidTrajectory. False est retourné si les trajectoires ne se chevauchent pas dans leurs plages M.

Disponibilité : 2.2.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

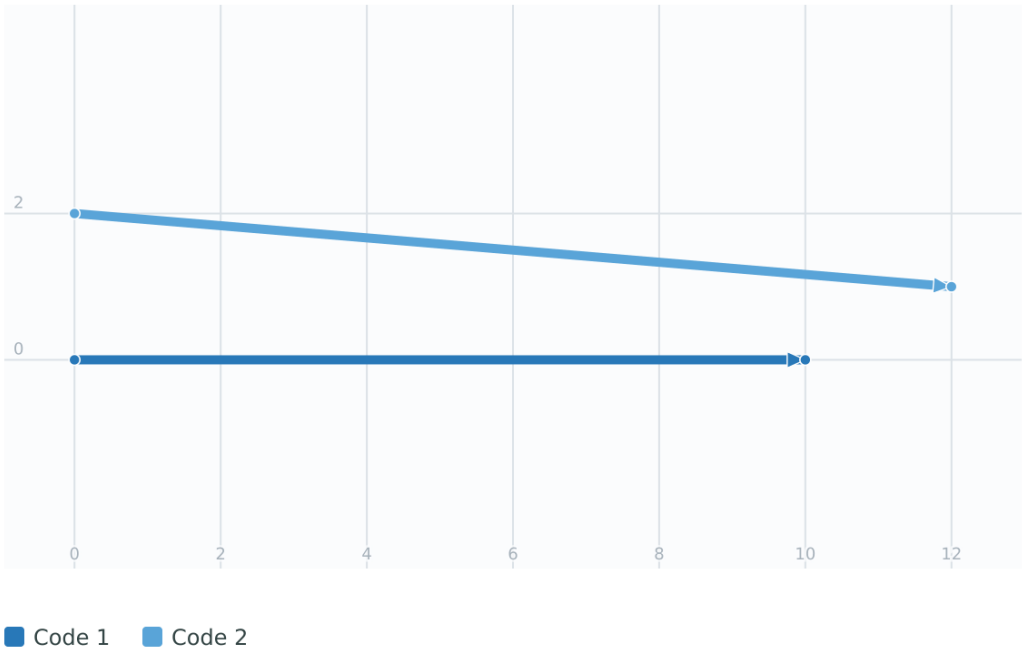
```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0,10 0 5)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
```

```
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10,12 1 2)'::geometry,
    extract(epoch from timestamp '2015-05-26 10:00' AT TIME ZONE 'UTC'),
    extract(epoch from timestamp '2015-05-26 11:00' AT TIME ZONE 'UTC')
  ) b
)
SELECT ST_CPAWithin(a, b, 2), ST_DistanceCPA(a, b) distance FROM inp;
```

Output

```
st_cpawithin |     distance
-------------+------------------
 t           | 1.9652147377620688
```

Figure



Voir aussi

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_DistanceCPA, |=|

7.21 Fonctions de version

7.21.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — Packages et mises à jour des extensions PostGIS (par exemple postgis_raster, postgis_topology, postgis_sfcgal) vers la version donnée ou la plus récente.

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

Description

Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis_raster, postgis_sfcgal, and postgis_topology.

Disponibilité : 2.5.0

Modifié : 3.4.0 pour ajouter l'argument target_version.

Modifié : 3.3.0 support pour les mises à jour à partir de n'importe quelle version de PostGIS. Ne fonctionne pas sur tous les systèmes.

Modifié : 3.0.0 pour repackager les extensions libres et supporter postgis_raster.

Exemples

Code

```
SELECT PostGIS_Extensions_Upgrade();
```

Output

```
NOTICE:  Packaging extension postgis
NOTICE:  Packaging extension postgis_raster
NOTICE:  Packaging extension postgis_sfcgal
NOTICE:  Extension postgis_topology is not available or not packagable for some reason
```

Voir aussi

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.2 PostGIS_Full_Version

PostGIS_Full_Version — Donne des informations complètes sur la version de PostGIS et la configuration du packaging.

Synopsis

text **PostGIS_Full_Version**();

Description

Donne des informations complètes sur la version de PostGIS et la configuration du packaging. Il fournit également des informations sur la synchronisation entre les bibliothèques et les scripts, suggérant des mises à niveau si nécessaire.

Amélioration : 3.4.0 inclut désormais les configurations supplémentaires PROJ NETWORK_ENABLED, URL_ENDPOINT et DATABASE_PATH pour l'emplacement proj.db

Exemples

Code

```
SELECT PostGIS_Full_Version();
```

Output

```
POSTGIS="3.7.0dev 3.7.0alpha1-477-ga9004e778" [EXTENSION] PGSQL="180" GEOS="3.15.0beta1" SFCGAL="2.3.0" PROJ="9.8.1 ↩
     NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL 3.13.1, ↩
     released 2026/06/01" LIBXML="2.15.3"
```

Voir aussi

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Wagyu_Ve
PostGIS_Version

7.21.3 PostGIS_GEOS_Version

PostGIS_GEOS_Version — Renvoie le numéro de version de la librairie GEOS.

Synopsis

text **PostGIS_GEOS_Version**();

Description

Renvoie le numéro de version de la librairie GEOS, ou `NULL` si le support GEOS n'est pas activé.

Exemples

Code

```
SELECT PostGIS_GEOS_Version();
```

Output

```
3.12.0dev-CAPI-1.18.0
```

Voir aussi

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.4 PostGIS_GEOS_Compiled_Version

PostGIS_GEOS_Compiled_Version — Renvoie le numéro de version de la librairie GEOS avec laquelle PostGIS a été construit.

Synopsis

text **PostGIS_GEOS_Compiled_Version**();

Description

Renvoie le numéro de version de la librairie GEOS, ou par rapport à laquelle PostGIS a été construit.

Disponibilité : 3.4.0

Exemples

Code

```
SELECT PostGIS_GEOS_Compiled_Version();
```

Output

```
3.12.0
```

Voir aussi

PostGIS_GEOS_Version, PostGIS_Full_Version

7.21.5 PostGIS_Liblwgeom_Version

PostGIS_Liblwgeom_Version — Renvoie le numéro de version de la librairie liblwgeom. Cela devrait correspondre à la version de PostGIS.

Synopsis

text **PostGIS_Liblwgeom_Version**();

Description

Renvoie le numéro de version de la librairie liblwgeom

Exemples

Code

```
SELECT PostGIS_Liblwgeom_Version();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

Voir aussi

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.6 PostGIS_LibXML_Version

PostGIS_LibXML_Version — Renvoie le numéro de version de la librairie libxml2.

Synopsis

text **PostGIS_LibXML_Version**();

Description

Renvoie le numéro de version de la librairie LibXML2.

Disponibilité : 1.5

Exemples

Code

```
SELECT PostGIS_LibXML_Version();
```

Output

```
2.9.10
```

Voir aussi

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.21.7 PostGIS_LibJSON_Version

PostGIS_LibJSON_Version — Renvoie le numéro de version de la bibliothèque libjson-c.

Synopsis

text **PostGIS_LibJSON_Version**();

Description

Renvoie le numéro de version de la bibliothèque LibJSON-C.

Exemples

Code

```
SELECT PostGIS_LibJSON_Version();
```

Output

```
0.17
```

Voir aussi

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.21.8 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — Retourne la date de packaging de l'extension PostGIS.

Synopsis

text **PostGIS_Lib_Build_Date**();

Description

Retourne la date de packaging de l'extension PostGIS.

Exemples

Code

```
SELECT PostGIS_Lib_Build_Date();
```

Output

```
2023-06-22 03:56:11
```

7.21.9 PostGIS_Lib_Version

PostGIS_Lib_Version — Retourne le numéro de version de l'extension PostGIS.

Synopsis

text **PostGIS_Lib_Version**();

Description

Retourne le numéro de version de l'extension PostGIS.

Exemples

Code

```
SELECT PostGIS_Lib_Version();
```

Output

```
3.4.0dev
```

Voir aussi

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.21.10 PostGIS_PROJ_Version

PostGIS_PROJ_Version — Renvoie le numéro de version de la librairie PROJ4.

Synopsis

text **PostGIS_PROJ_Version**();

Description

Retourne le numéro de version de la bibliothèque PROJ et certaines options de configuration de proj.

Amélioration : 3.4.0 inclut désormais PROJ NETWORK_ENABLED, URL_ENDPOINT et DATABASE_PATH pour l'emplacement proj.db

Exemples

Code

```
SELECT PostGIS_PROJ_Version();
```

Output

```
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↩
    proj DATABASE_PATH=/usr/share/proj/proj.db
```

Voir aussi

PostGIS_PROJ_Compiled_Version, PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Versio
PostGIS_Version

7.21.11 PostGIS_PROJ_Compiled_Version

PostGIS_PROJ_Compiled_Version — Renvoie le numéro de version de la bibliothèque PROJ avec laquelle PostGIS a été construit.

Synopsis

text **PostGIS_PROJ_Compiled_Version**();

Description

Renvoie le numéro de version de la bibliothèque PROJ, ou par rapport à laquelle PostGIS a été buildé.

Disponibilité : 3.5.0

Exemples

Code

```
SELECT PostGIS_PROJ_Compiled_Version();
```

Output

```
9.1.1
```

Voir aussi

PostGIS_PROJ_Version, PostGIS_Full_Version

7.21.12 PostGIS_Wagyu_Version

PostGIS_Wagyu_Version — Renvoie le numéro de version de la librairie interne de Wagyu.

Synopsis

text **PostGIS_Wagyu_Version**();

Description

Renvoie le numéro de version de la librairie interne Wagyu, ou `NULL` si le support Wagyu n'est pas activé.

Exemples

Code

```
SELECT PostGIS_Wagyu_Version();
```

Output

```
0.5.0 (Internal)
```

Voir aussi

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_PROJ_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.21.13 PostGIS_Scripts_Build_Date

PostGIS_Scripts_Build_Date — Retourne la date de packaging des scripts PostGIS.

Synopsis

text **PostGIS_Scripts_Build_Date**();

Description

Retourne la date de packaging des scripts PostGIS.

Disponibilité : 1.0.0RC1

Exemples

Code

```
SELECT PostGIS_Scripts_Build_Date();
```

Output

```
2023-06-22 03:56:11
```

Voir aussi

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.21.14 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Retourne la version des scripts PostGIS installés dans cette base de données.

Synopsis

text **PostGIS_Scripts_Installed**();

Description

Retourne la version des scripts PostGIS installés dans cette base de données.



Note
Si la sortie de cette fonction ne correspond pas à la sortie de PostGIS_Scripts_Released, vous avez probablement manqué de mettre correctement à niveau une base de données existante. Consultez la section Upgrading pour plus d'informations.

Disponibilité : 0.9.0

Exemples

Code

```
SELECT PostGIS_Scripts_Installed();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

Voir aussi

PostGIS_Full_Version, PostGIS_Scripts_Released, PostGIS_Version

7.21.15 PostGIS_Scripts_Released

PostGIS_Scripts_Released — Retourne le numéro de version du script postgis.sql publié avec la librairie PostGIS installée.

Synopsis

text **PostGIS_Scripts_Released**();

Description

Retourne le numéro de version du script postgis.sql publié avec la librairie PostGIS installée.



Note
À partir de la version 1.1.0, cette fonction renvoie la même valeur de PostGIS_Lib_Version. Conservée pour des raisons de rétrocompatibilité.

Disponibilité : 0.9.0

Exemples

Code

```
SELECT PostGIS_Scripts_Released();
```

Output

```
3.4.0dev 3.3.0rc2-993-g61bdf43a7
```

Voir aussi

PostGIS_Full_Version, PostGIS_Scripts_Installed, PostGIS_Lib_Version

7.21.16 PostGIS_Version

PostGIS_Version — Retourne le numéro de version de PostGIS et les options de compilation.

Synopsis

text **PostGIS_Version**();

Description

Retourne le numéro de version de PostGIS et les options de compilation.

Exemples

Code

```
SELECT PostGIS_Version();
```

Output

```
3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
```

Voir aussi

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version

7.22 Variables PostGIS GUC (Grand Unified Custom Variables)

7.22.1 postgis.gdal_datapath

postgis.gdal_datapath — Une option de configuration pour régler la valeur de l'option GDAL_DATA de GDAL. Si elle n'est pas assignée, la valeur de la variable d'environnement GDAL_DATA est utilisée.

Description

Une variable GUC PostgreSQL pour régler la valeur de l'option GDAL_DATA de GDAL. La valeur `postgis.gdal_datapath` devrait être le chemin physique complet vers les fichiers de données de GDAL.

Cette option de configuration est principalement destinée aux plateformes Windows où le chemin des fichiers de données de GDAL n'est pas codé en dur. Cette option devrait aussi être réglée lorsque les fichiers de données GDAL ne sont pas situés dans le chemin attendu par GDAL.



Note
Cette option peut être réglée dans le fichier de configuration postgresql.conf de PostgreSQL. Elle peut aussi être réglée par connexion ou par transaction.

Disponibilité : 2.2.0



Note
Des informations complémentaires sur GDAL_DATA sont disponibles dans la description des Options de configuration.

Exemples

Régler et remettre à la valeur par défaut `postgis.gdal_datapath`

Code

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

Réglage pour une base de données spécifique, sous Windows

Code

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

Voir aussi

PostGIS_GDAL_Version, ST_Transform

7.22.2 postgis.gdal_enabled_drivers

postgis.gdal_enabled_drivers — Option de configuration permettant de définir les drivers GDAL activés dans l'environnement PostGIS. Affecte la variable de configuration GDAL GDAL_SKIP.

Description

Option de configuration permettant de définir les drivers GDAL activés dans l'environnement PostGIS. Affecte la variable de configuration GDAL GDAL_SKIP. Cette option peut être définie dans le fichier de configuration de PostgreSQL : postgresql.conf. Elle peut également être définie par connexion ou transaction.

La valeur initiale de `postgis.gdal_enabled_drivers` peut aussi être définie en passant la variable d'environnement `POSTGIS_GDAL_ENABLED_DRIVERS` avec la liste des drivers activés au processus de démarrage de PostgreSQL.

Les drivers GDAL activés peuvent être spécifiés par leur nom abrégé ou leur code. Les noms courts ou les codes des drivers sont disponibles sur GDAL Raster Formats. Il est possible de spécifier plusieurs drivers en insérant un espace entre chacun d'eux.



Note

Trois codes spéciaux sont disponibles pour `postgis.gdal_enabled_drivers`. Les codes sont sensibles à la casse.

- `DISABLE_ALL` désactive tous les drivers GDAL. S'il est présent, `DISABLE_ALL` remplace toutes les autres valeurs de `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` active tous les drivers GDAL.
- `VSICURL` active le système de fichiers virtuels `/vsicurl/` de GDAL.

Lorsque `postgis.gdal_enabled_drivers` est défini sur DISABLE_ALL, les tentatives d'utilisation des rasters out-db, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() et ST_AsPNG() se traduisent par des messages d'erreur.



Note

Dans l'installation standard de PostGIS, `postgis.gdal_enabled_drivers` est défini sur DISABLE_ALL.



Note

Des informations supplémentaires sur GDAL_SKIP sont disponibles sur le site GDAL Configuration Options.

Disponibilité : 2.2.0

Exemples

Set and reset `postgis.gdal_enabled_drivers` for the current session.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.gdal_enabled_drivers = default;
```

Set specific drivers for all new connections to a database.

Code

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Set the default for the whole database cluster to enable all drivers. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

Code

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Reload the PostgreSQL configuration so the changed setting is applied.

Code

```
SELECT pg_reload_conf();
```

Voir aussi

ST_FromGDALRaster, ST_AsGDALRaster, ST_AsTIFF, ST_AsPNG, ST_AsJPEG, postgis.enable_outdb_rasters

7.22.3 postgis.enable_outdb_rasters

postgis.enable_outdb_rasters — Une option de configuration booléenne pour permettre l'accès aux bandes matricielles de out-db.

Description

Une option de configuration booléenne pour permettre l'accès aux bandes matricielles de out-db. Cette option peut être définie dans le fichier de configuration de PostgreSQL : postgresql.conf. Elle peut également être définie par connexion ou transaction.

La valeur initiale de `postgis.enable_outdb_rasters` peut également être définie en passant la variable d'environnement `POSTGIS_ENABLE_OUTDB_RASTERS` avec une valeur non nulle au processus de démarrage de PostgreSQL.



Note
Même si `postgis.enable_outdb_rasters` est True, le GUC `postgis.gdal_enabled_drivers` détermine les formats raster accessibles.



Note
Dans l'installation standard de PostGIS, `postgis.enable_outdb_rasters` est défini sur False.

Disponibilité : 2.2.0

Exemples

Set and reset `postgis.enable_outdb_rasters` for the current session.

Code

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set the default for all new connections to a database.

Code

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Set the default for the whole database cluster. This requires superuser access. Also note that database, session, and user settings override this.

This example writes the setting to postgres.auto.conf.

Code

```
ALTER SYSTEM SET postgis.enable_outdb_rasters = true;
```

Reload the PostgreSQL configuration so the changed setting is applied.

Code

```
SELECT pg_reload_conf();
```

Voir aussi

postgis.gdal_enabled_drivers postgis.gdal_vsi_options

7.22.4 postgis.gdal_vsi_options

postgis.gdal_vsi_options — Une chaîne de configuration pour définir les options utilisées lors de l'utilisation d'un raster out-db.

Description

Une chaîne de configuration pour définir les options utilisées lors de l'utilisation d'un raster out-db. Les options de configuration contrôlent des éléments tels que l'espace alloué par GDAL au cache des données locales, la lecture ou non des aperçus et les clés d'accès à utiliser pour les sources de données out-db distantes.

Disponibilité : 3.2.0

Exemples

Définit `postgis.gdal_vsi_options` pour la session actuelle :

Code

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↩
    yyyyyyyyyyyyyyyyyyyyyyyyy';
```

Définissez `postgis.gdal_vsi_options` uniquement pour la *transaction courante* en utilisant le mot clé `LOCAL` :

Code

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx ↩
    AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Voir aussi

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.22.5 postgis.gdal_cpl_debug

postgis.gdal_cpl_debug — A boolean configuration to turn logging of GDAL debug messages on and off.

Description

By default, GDAL logging is printed to stderr, and lower level debug messages are not printed at all. Turning this GUC to true will cause detailed GDAL logging to be sent into the PostgreSQL server log. PostgreSQL clients receive only generic GDAL notice, debug, warning, or error messages.

Disponibilité : 3.6.0

Voir aussi

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.23 Fonctions de dépannage

7.23.1 PostGIS_AddBBox

PostGIS_AddBBox — Ajoute une bounding box à la géométrie.

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

Description

Ajouter une boîte de délimitation à la géométrie. Cela rendrait les requêtes basées sur la boîte de délimitation plus rapides, mais augmenterait la taille de la géométrie.



Note
Les boîtes de délimitation sont automatiquement ajoutées aux géométries, de sorte qu'en général, cela n'est pas nécessaire, à moins que la boîte de délimitation générée ne soit corrompue d'une manière ou d'une autre ou que vous ayez une ancienne installation qui ne contient pas les boîtes de délimitation. Dans ce cas, vous devez supprimer l'ancienne et ajouter la nouvelle.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
UPDATE sometable
SET geom =  PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

Voir aussi

PostGIS_DropBBox, PostGIS_HasBBox

7.23.2 PostGIS_DropBBox

PostGIS_DropBBox — Supprime le cache de la boîte de délimitation de la géométrie.

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

Description

Supprime le cache de la boîte de délimitation de la géométrie. Cela réduit la taille de la géométrie, mais rend les requêtes basées sur les boîtes de délimitation plus lentes. Elle est également utilisée pour supprimer une boîte de délimitation corrompue. Un signe révélateur d'une boîte de délimitation corrompue dans le cache est lorsque vos ST_Intersects et autres requêtes de relations laissent de côté des géométries qui devraient légitimement renvoyer un résultat vrai.



Note
Les boîtes de délimitation sont automatiquement ajoutées aux géométries et améliorent la vitesse des requêtes. En général, cela n'est donc pas nécessaire, sauf si le boîte de délimitation générée est corrompue d'une manière ou d'une autre ou si vous avez une ancienne installation qui ne contient pas de boîtes de délimitation. Dans ce cas, il est nécessaire de supprimer l'ancienne installation et de la remplacer par une nouvelle. Ce type de corruption a été observé dans les séries 8.3-8.3.6 où les bboxes mises en cache n'étaient pas toujours recalculées lorsqu'une géométrie changeait et lors des mises à jour vers une version plus récente où le dump non rechargé ne corrigeait pas les bboxes déjà corrompues. La mise à jour vers une version plus récente sans rechargement du dump ne corrigera pas les boîtes déjà corrompues. Il est donc possible de corriger manuellement en utilisant la méthode ci-dessous et de lire la bbox ou de recharger le dump.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

This example drops cached bounding boxes where the stored box is wrong. Applying `Box2D` to `ST_AsBinary` forces a fresh box calculation, while `Box2D` on the table geometry reads the cached box.

Code

```
UPDATE sometable
SET geom =  PostGIS_DropBBox(geom)
WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));
```

Output

```
UPDATE ...
```

After dropping corrupt cached boxes, re-add them where needed.

Code

```
UPDATE sometable
SET geom =  PostGIS_AddBBox(geom)
WHERE Not PostGIS_HasBBOX(geom);
```

Output

```
UPDATE ...
```

Voir aussi

PostGIS_AddBBox, PostGIS_HasBBox, Box2D

7.23.3 PostGIS_HasBBox

PostGIS_HasBBox — Renvoie TRUE si la bbox de cette géométrie est en cache, sinon FALSE.

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

Description

Renvoie TRUE si la boîte de délimitation de cette géométrie est mise en cache, FALSE dans le cas contraire. Utilisez PostGIS_AddBBox et PostGIS_DropBBox pour contrôler la mise en cache.

 Cette méthode prend en charge les types Circular String et Curve.

Exemples

Code

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

Voir aussi

PostGIS_AddBBox, PostGIS_DropBBox

Chapitre 8

Référence des fonctions SFCGAL

SFCGAL est une bibliothèque C++ autour de CGAL qui fournit des fonctions spatiales avancées en 2D et 3D. Pour des raisons de robustesse, les coordonnées géométriques sont représentées par des nombres rationnels exacts.

Les instructions d'installation de la bibliothèque se trouvent sur la page d'accueil de SFCGAL (http://www.sfcgal.org). Pour activer les fonctions, utilisez `create extension postgis_sfcgal`.

8.1 Fonctions de gestion de SFCGAL

8.1.1 postgis_sfcgal_version

postgis_sfcgal_version — Retourne la version de SFCGAL utilisée

Synopsis

text **postgis_sfcgal_version**(void);

Description

Retourne la version de SFCGAL utilisée

Disponibilité : 2.1.0

Cette méthode nécessite le backend SFCGAL.

Voir aussi

postgis_sfcgal_full_version

8.1.2 postgis_sfcgal_full_version

postgis_sfcgal_full_version — Retourne la version complète de SFCGAL en cours d'utilisation, y compris les versions CGAL et Boost

Synopsis

text **postgis_sfcgal_full_version**(void);

Description

Retourne la version complète de SFCGAL utilisée, y compris les versions CGAL et Boost

Disponibilité : 3.3.0

Cette méthode nécessite le backend SFCGAL.

Voir aussi

postgis_sfcgal_version

8.2 Fonctions d'accès et de modifications SFCGAL

8.2.1 CG_ForceLHR

CG_ForceLHR — Force l'orientation LHR d'un objet

Synopsis

geometry **CG_ForceLHR**(geometry geom);

Description

Returns an equivalent geometry whose polygonal components follow the left-hand rule: exterior rings are counter-clockwise and interior rings are clockwise. Rings are reversed as needed; coordinates and dimensionality are preserved. The input must be valid.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Reorient a polygon and its hole to the left-hand rule.

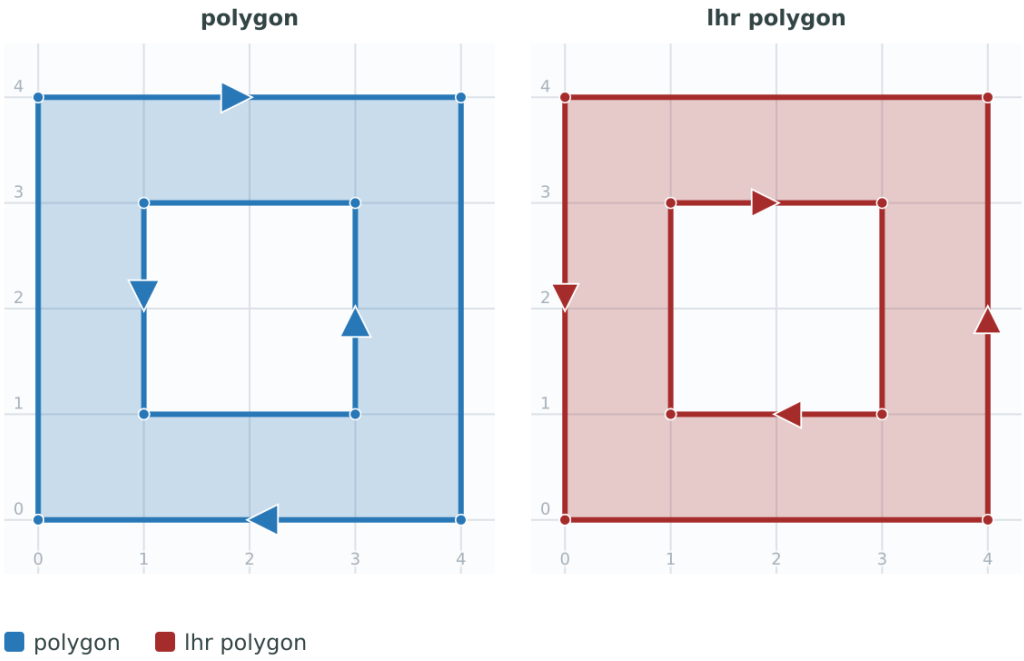
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,0 4,4 4,4 0,0 0),
                  (1 1,3 1,3 3,1 3,1 1))'::geometry AS geom
)
SELECT geom AS input_polygon,
       CG_ForceLHR(geom) AS lhr_polygon,
       ST_IsPolygonCCW(geom) AS input_is_lhr,
       ST_IsPolygonCCW(CG_ForceLHR(geom)) AS output_is_lhr
FROM data;
```

Output

```
POLYGON((0 0,0 4,4 4,4 0,0 0),(1 1,3 1,3 3,1 3,1 1)) | POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1,1 3,3 3,3 1,1 1)) | f | t
```

Figure



8.2.2 CG_IsPlanar

CG_IsPlanar — Vérifie si une surface est planaire ou non

Synopsis

boolean **CG_IsPlanar**(geometry geom);

Description

Tests whether the input geometry lies in a single plane in 3D.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

8.2.3 CG_IsSolid

CG_IsSolid — Teste si la géométrie est un solide. Aucun contrôle de validité n'est effectué.

Synopsis

boolean **CG_IsSolid**(geometry geom1);

Description

Returns `true` when the input carries the solid flag, such as a geometry produced by CG_MakeSolid. A closed `PolyhedralSurface` remains a surface until it is explicitly converted. This tests the representation, not shell closure or validity; no validity check is performed.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Compare the same closed shell before and after converting it to a solid.

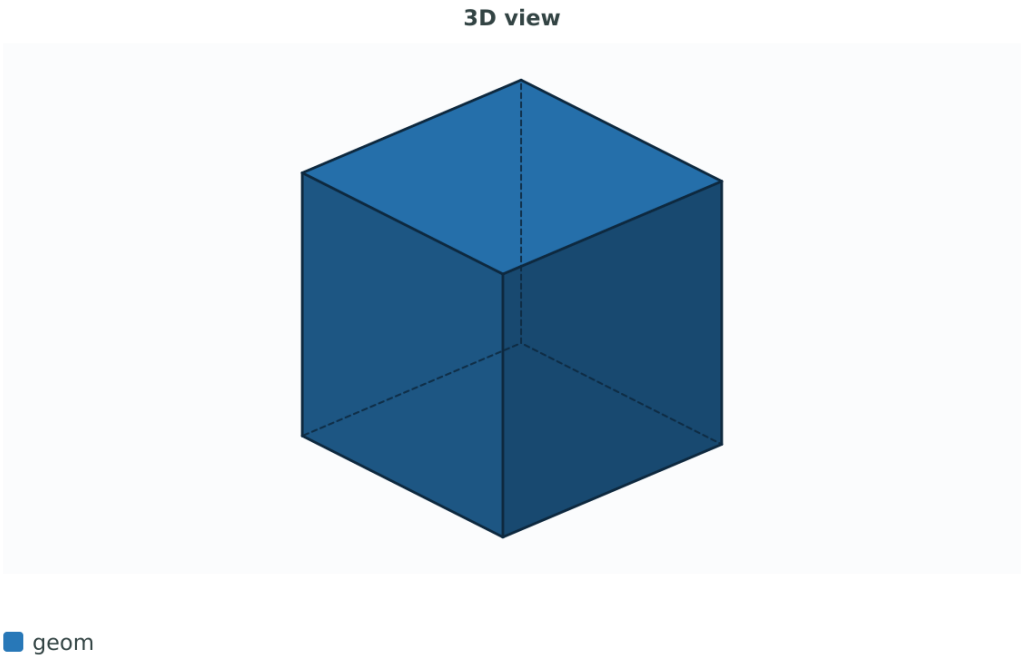
Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z (
    ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
    ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
    ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
  )'::geometry AS geom
)
SELECT CG_IsSolid(geom) AS shell_is_solid,
       CG_IsSolid(CG_MakeSolid(geom)) AS solid_is_solid
FROM data;
```

Output

```
f | t
```

Figure



8.2.4 CG_MakeSolid

CG_MakeSolid — Transformer la géométrie dans un solide. Aucune vérification n'est effectuée. Pour obtenir un solide valide, la géométrie d'entrée doit être une surface polyédrique fermée ou un TIN fermé.

Synopsis

geometry **CG_MakeSolid**(geometry geom1);

Description

Marks a PolyhedralSurface or TIN as a solid. The input is expected to describe a closed shell; the function sets the solid representation and does not validate the shell.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

8.2.5 CG_Orientation

CG_Orientation — Détermine l'orientation d'une surface

Synopsis

integer **CG_Orientation**(geometry geom);

Description

Cette fonction ne s'applique qu'aux polygones. Elle renvoie -1 si le polygone est orienté dans le sens inverse des aiguilles d'une montre et 1 s'il est orienté dans le sens des aiguilles d'une montre.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

8.2.6 CG_Area

CG_Area — Calcule la surface d'une géométrie

Synopsis

double precision **CG_Area**(geometry geom);

Description

Calcule la surface d'une géométrie.

Réalisé par le module SFCGAL

> **Note**
> NOTE : cette fonction renvoie une valeur en double précision représentant la surface.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Exemples de géométrie

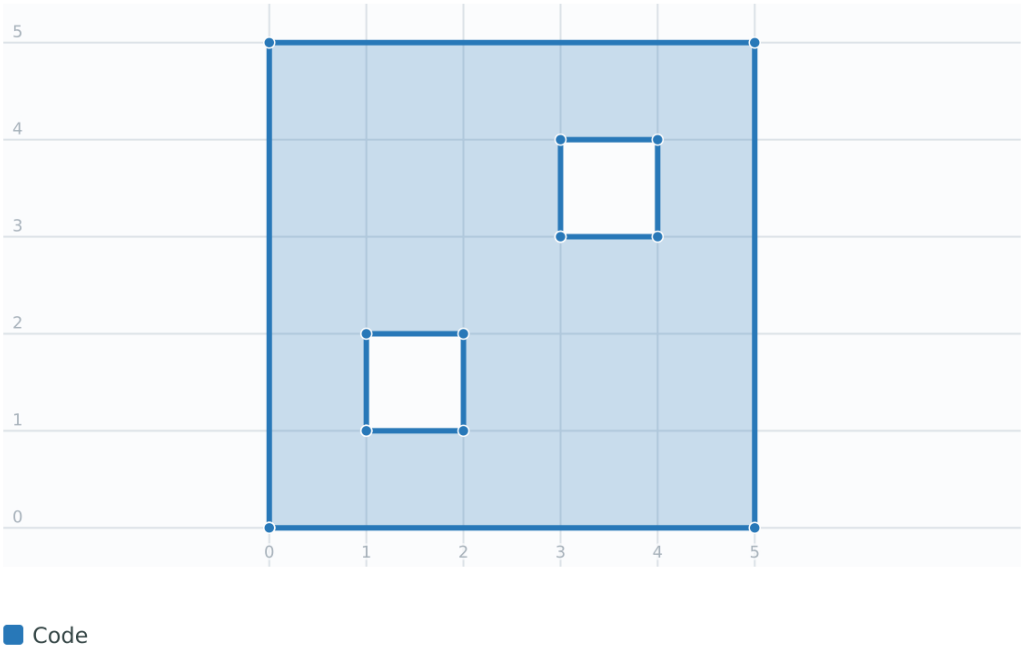
Code

```
SELECT CG_Area('Polygon ((0 0,0 5,5 5,5 0,0 0),(1 1,2 1,2 2,1 2,1 1),(3 3,4 3,4 4,3 4,3 3))' ↩
    );
```

Output

```
23
```

Figure



Voir aussi

CG_3DArea, ST_Area

8.2.7 CG_3DArea

CG_3DArea — Calcule la surface des géométries de surface 3D. Retourne 0 pour les solides.

Synopsis

float**CG_3DArea**(geometry geom1);

Description

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 8.1, 10.5

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Remarque : par défaut, une PolyhedralSurface construite à partir de WKT est une géométrie de surface, et non un solide. Elle possède donc une surface. Une fois convertie en solide, elle n'a plus de surface.

Code

```
SELECT CG_3DArea(geom) As cube_surface_area,
CG_3DArea(CG_MakeSolid(geom)) As solid_surface_area
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry) As f(geom);
```

Output

```
cube_surface_area | solid_surface_area
------------------+--------------------
6                 | 0
```

Voir aussi

CG_Area, CG_MakeSolid, CG_IsSolid, CG_Area

8.2.8 CG_Volume

CG_Volume — Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.

Synopsis

float **CG_Volume**(geometry geom1);

Description

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3 : 9.1 (identique à CG_3DVolume)

Closed surface geometries can have non-zero volume. Use CG_MakeSolid when the geometry value itself must carry the solid flag.

Exemple

A closed polyhedral surface has volume. CG_MakeSolid makes the solid representation explicit.

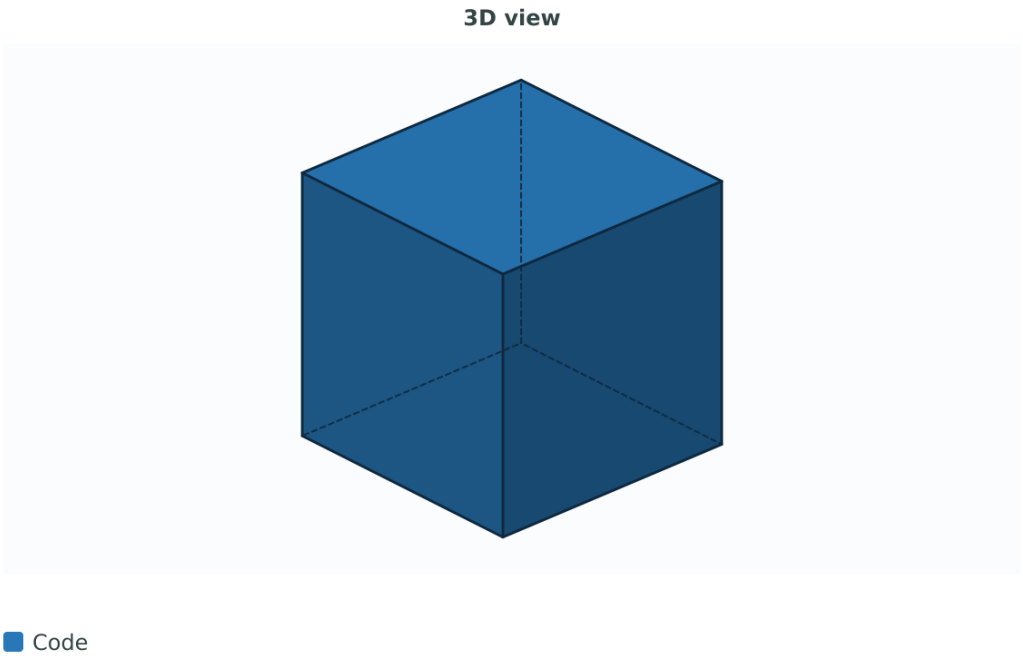
Code

```
SELECT CG_Volume(geom) As cube_surface_vol,
CG_Volume(CG_MakeSolid(geom)) As solid_surface_vol
FROM (SELECT 'POLYHEDRALSURFACE( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry) As f(geom);
```

Output

```
cube_surface_vol | solid_surface_vol
-----------------+------------------
1                | 1
```

Figure



Voir aussi

CG_3DArea, CG_MakeSolid, CG_IsSolid

8.3 Fonctions de traitement et de relation SFCGAL

8.3.1 CG_Intersection

CG_Intersection — Calcul l'intersection de deux géométries

Synopsis

geometry **CG_Intersection**(geometry geomA , geometry geomB);

Description

Calcule l'intersection de deux géométries.

Réalisé par le module SFCGAL

Note
NOTE : cette fonction renvoie une géométrie représentant l'intersection.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Exemples de géométrie

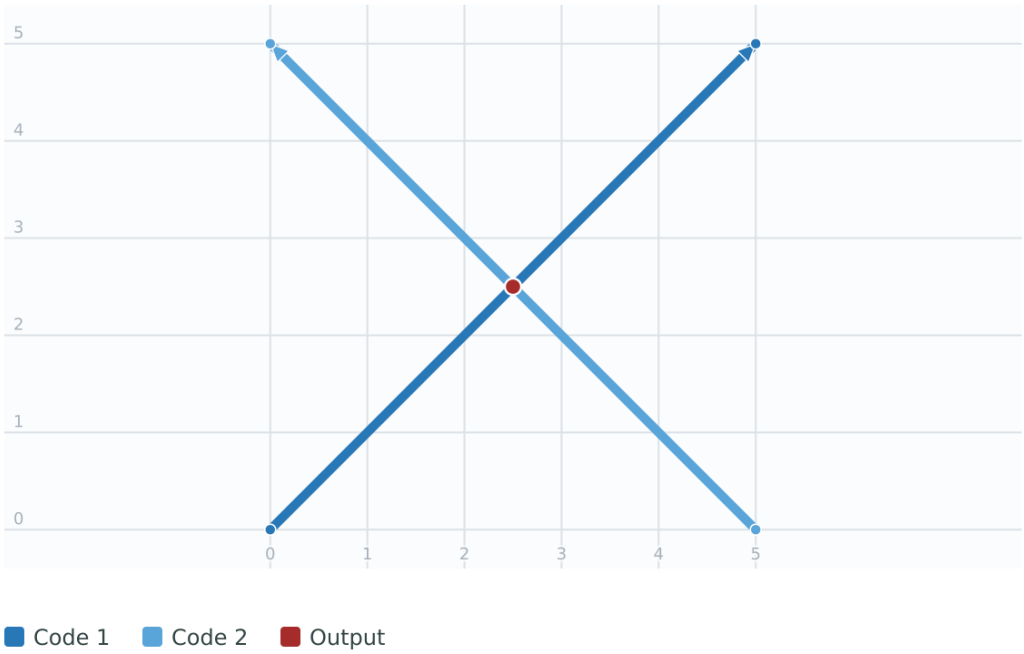
Code

```
SELECT CG_Intersection('LINESTRING(0 0,5 5)', 'LINESTRING(5 0,0 5)');
```

Output

```
POINT(2.5 2.5)
```

Figure



Voir aussi

CG_3DIntersection, ST_Intersection

8.3.2 CG_Intersects

CG_Intersects — Teste si deux géométries se croisent (elles ont au moins un point en commun)

Synopsis

boolean **CG_Intersects**(geometry geomA , geometry geomB);

Description

Renvoie `true` si deux géométries se croisent. Les géométries se croisent si elles ont un point commun.

Réalisé par le module SFCGAL



Note
NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

Disponibilité : 3.5.0

 Cette méthode nécessite le backend SFCGAL.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples de géométrie

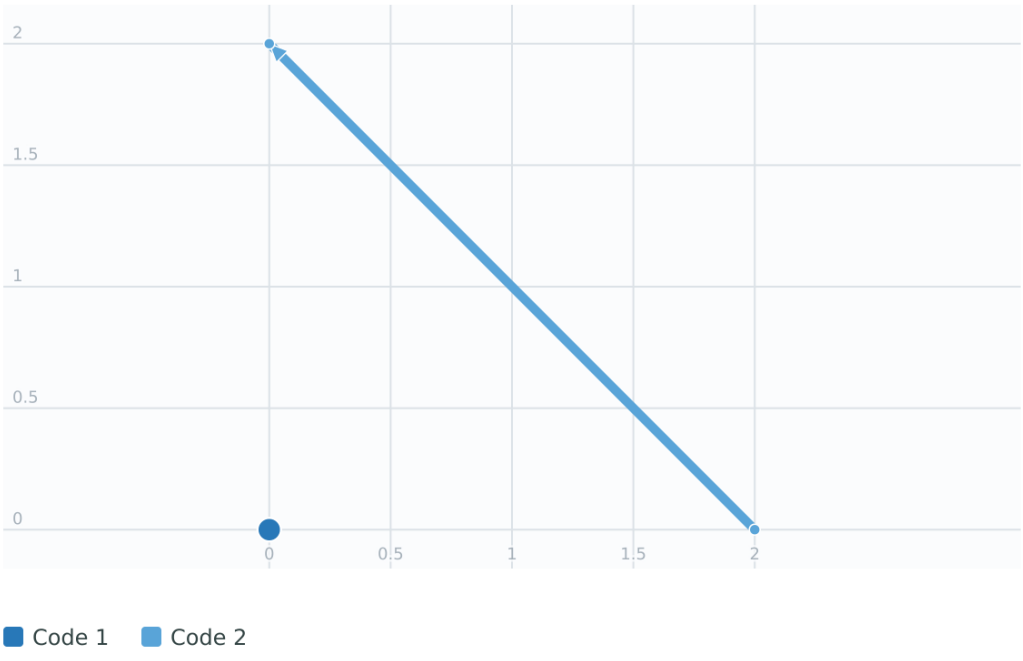
Code

```
SELECT CG_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 2 0,0 2 )'::geometry);
```

Output

```
f
```

Figure



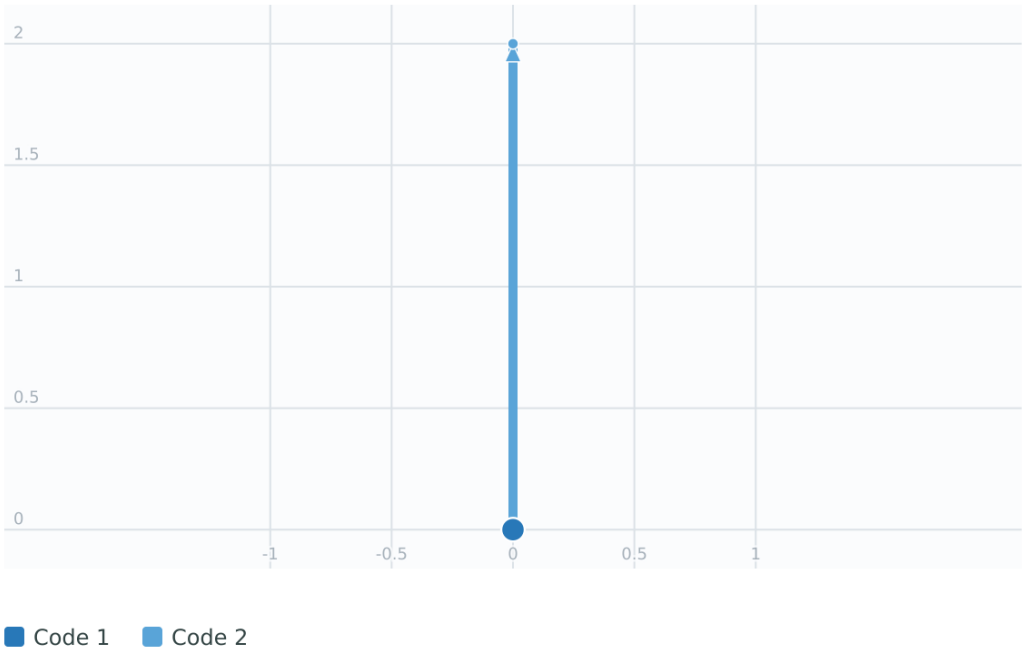
Code

```
SELECT CG_Intersects('POINT(0 0)'::geometry, 'LINESTRING ( 0 0,0 2 )'::geometry);
```

Output

```
t
```

Figure



Voir aussi

CG_3DIntersects, ST_3DIntersects, ST_Intersects, ST_Disjoint

8.3.3 CG_3DIntersects

CG_3DIntersects — Teste si deux géométries 3D intersectent

Synopsis

boolean **CG_3DIntersects**(geometry geomA , geometry geomB);

Description

Teste si deux géométries 3D intersectent. Les géométries 3D intersectent si elles ont au moins un point commun dans l'espace tridimensionnel.

Réalisé par le module SFCGAL



Note
NOTE : il s'agit de la version "autorisée" qui renvoie un booléen et non un entier.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples de géométrie

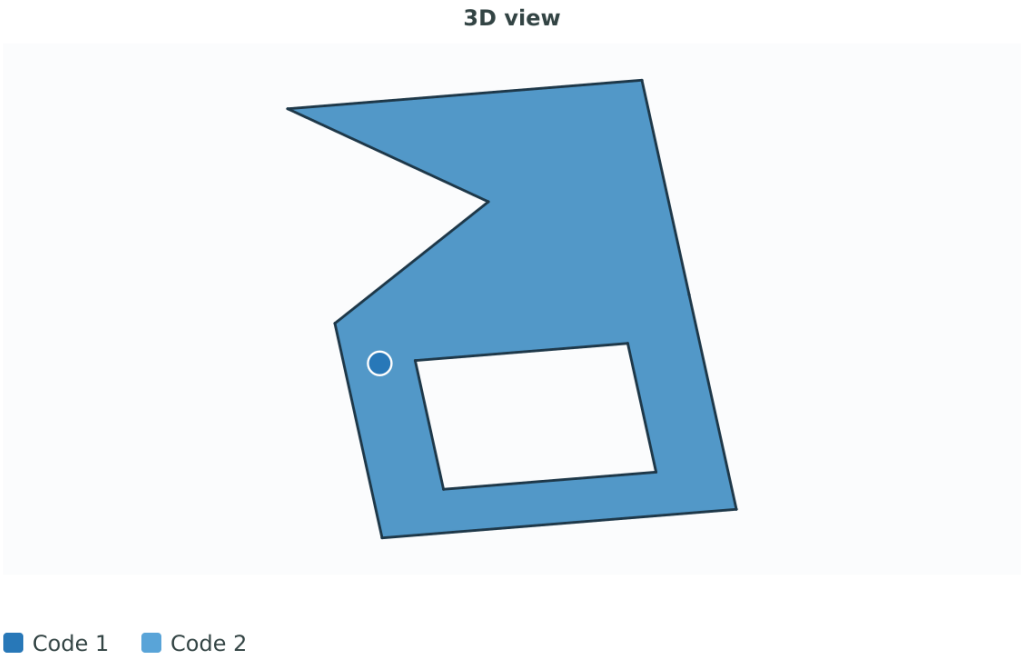
Code

```
SELECT CG_3DIntersects('POINT(1.2 0.1 0)', 'POLYHEDRALSURFACE(((0 0 0,0.5 0.5 0,1 0 0,1 1 0,0 1 0,0 0 0)),((1 0 0,2 0 0,2 1 0,1 1 0,1 0 0),(1.2 0.2 0,1.2 0.8 0,1.8 0.8 0,1.8 0.2 0,1.2 0.2 0)))');
```

Output

```
t
```

Figure



Voir aussi

CG_Intersects, ST_3DIntersects, ST_Intersects, ST_Disjoint

8.3.4 CG_Difference

CG_Difference — Calcul la différence géométrique entre deux géométries

Synopsis

geometry **CG_Difference**(geometry geomA , geometry geomB);

Description

Calcule la différence géométrique entre deux géométries. La géométrie résultante est un ensemble de points présents dans la géométrie A mais pas dans la géométrie B.

Réalisé par le module SFCGAL



Note
NOTE : cette fonction renvoie une géométrie.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples de géométrie

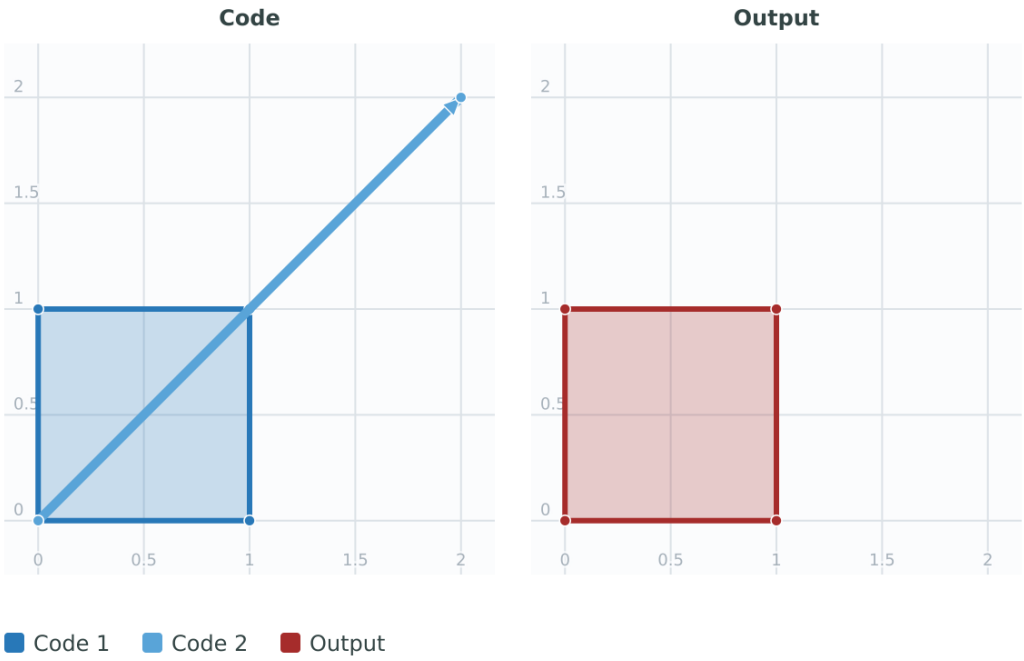
Code

```
SELECT CG_Difference('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry, 'LINESTRING(0 0,2 2)':: ↩
    geometry);
```

Output

```
POLYGON((0 0,1 0,1 1,0 1,0 0))
```

Figure



Voir aussi

CG_3DDifference, ST_Difference

8.3.5 CG_3DDifference

CG_3DDifference — Effectuer une différence 3D

Synopsis

geometry **CG_3DDifference**(geometry geom1, geometry geom2);

Description

Renvoie la partie de geom1 qui ne fait pas partie de geom2.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

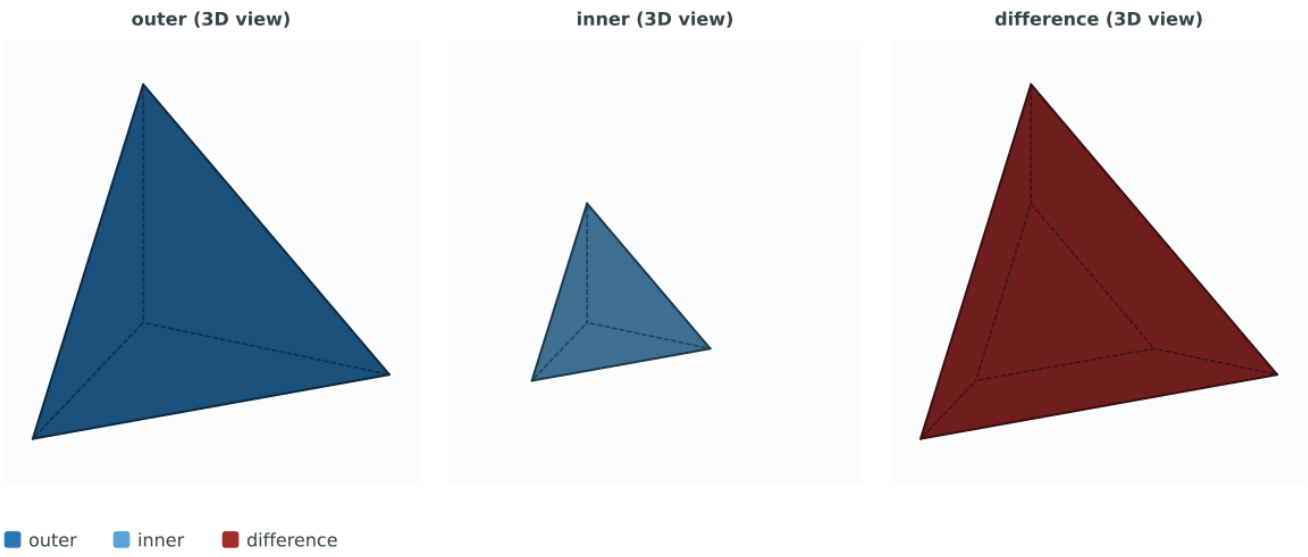
Exemples

Subtract a smaller tetrahedral solid and render the two inputs and the actual remaining solid separately.

Code

```
WITH data AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
    ((0 0 0,0 4 0,4 0 0,0 0 0)),
    ((0 0 0,4 0 0,0 0 4,0 0 0)),
    ((0 0 0,0 0 4,0 4 0,0 0 0)),
    ((4 0 0,0 4 0,0 0 4,4 0 0))
  )'::geometry) AS outer_solid
), solids AS (
  SELECT outer_solid,
         ST_Scale(outer_solid, 0.5, 0.5, 0.5) AS inner_solid
  FROM data
)
SELECT outer_solid AS input_outer,
       inner_solid AS input_inner,
       CG_3DDifference(outer_solid, inner_solid) AS difference
FROM solids;
```

Figure



Voir aussi

CG_Extrude, ST_AsX3D, CG_3DIntersection CG_3DUnion

8.3.6 CG_Distance

CG_Distance — Calcule la distance minimale entre deux géométries

Synopsis

double precision **CG_Distance**(geometry geomA , geometry geomB);

Description

Calcule la distance minimale entre deux géométries.

Réalisé par le module SFCGAL

> **Note**
> NOTE : cette fonction renvoie une valeur double précision représentant la distance.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples de géométrie

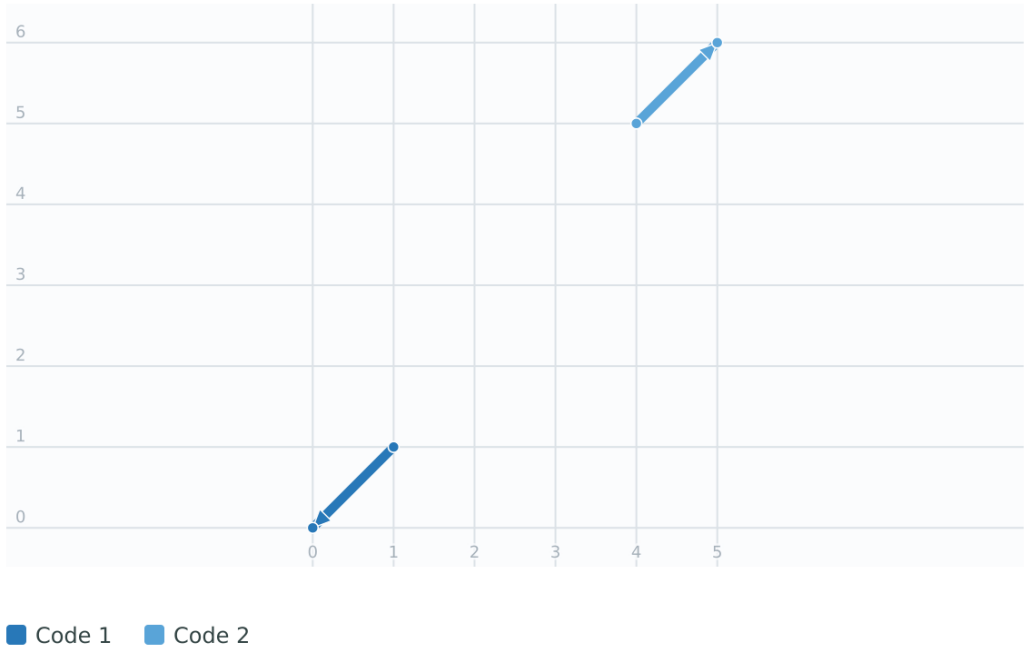
Code

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
```

Output

```
5
```

Figure



Voir aussi

CG_3DDistance, CG_Distance

8.3.7 CG_3DDistance

CG_3DDistance — Calcule la distance 3D minimale entre deux géométries

Synopsis

double precision **CG_3DDistance**(geometry geomA , geometry geomB);

Description

Calcule la distance 3D minimale entre deux géométries.

Réalisé par le module SFCGAL



Note
NOTE : cette fonction renvoie une valeur double précision représentant la distance 3D.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples de géométrie

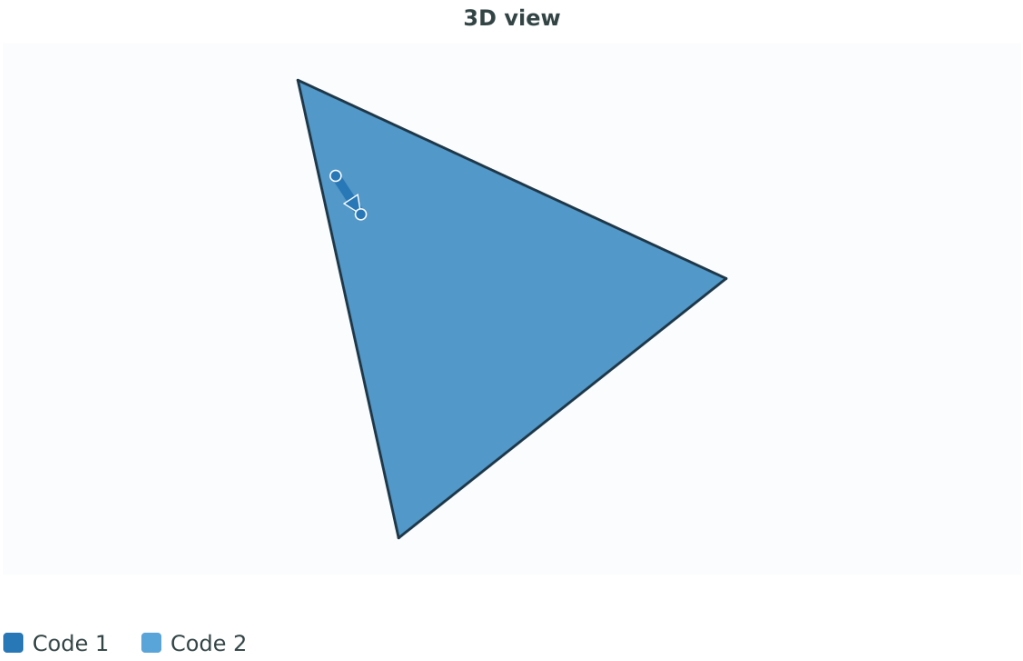
Code

```
SELECT CG_3DDistance('LINESTRING(-1.0 0.0 2.0,1.0 0.0 3.0)', 'TRIANGLE((-4.0 0.0 1.0,4.0 0.0 ↩
    1.0,0.0 4.0 1.0,-4.0 0.0 1.0))');
```

Output

```
1
```

Figure



Voir aussi

CG_Distance, ST_3DDistance

8.3.8 CG_3DConvexHull

CG_3DConvexHull — Calcule l'enveloppe convexe 3D d'une géométrie.

Synopsis

geometry **CG_3DConvexHull**(geometry geom1);

Description

Computes the smallest convex three-dimensional geometry that contains the input. Unlike ST_ConvexHull, this function uses Z coordinates when constructing the hull.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Compute the 3D convex hull of a 3D LineString.

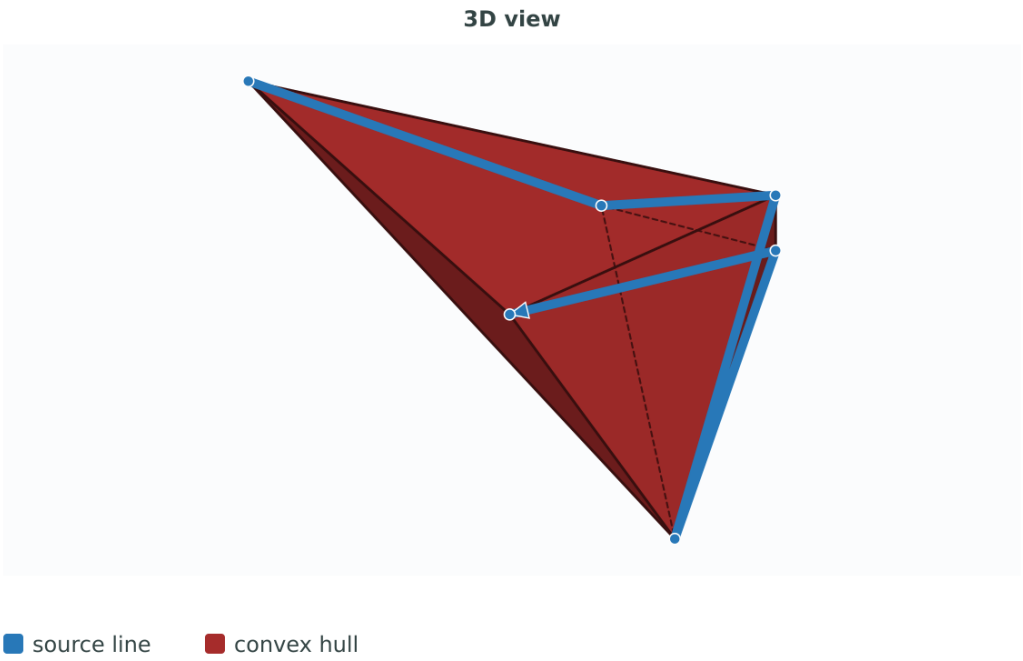
Code

```
WITH data AS (
  SELECT 'LINESTRING Z(0 0 5,1 5 3,5 7 6,9 5 3,5 7 5,6 3 5)'::geometry AS source_line
)
SELECT CG_3DConvexHull(source_line) AS convex_hull
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7 5,1 5 3,5 7 6)),((0 0 5,6 3 ↩
    5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6 3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 ↩
    5,9 5 3,1 5 3)))
```

Figure



Compare the extruded source geometry with its 3D convex hull.

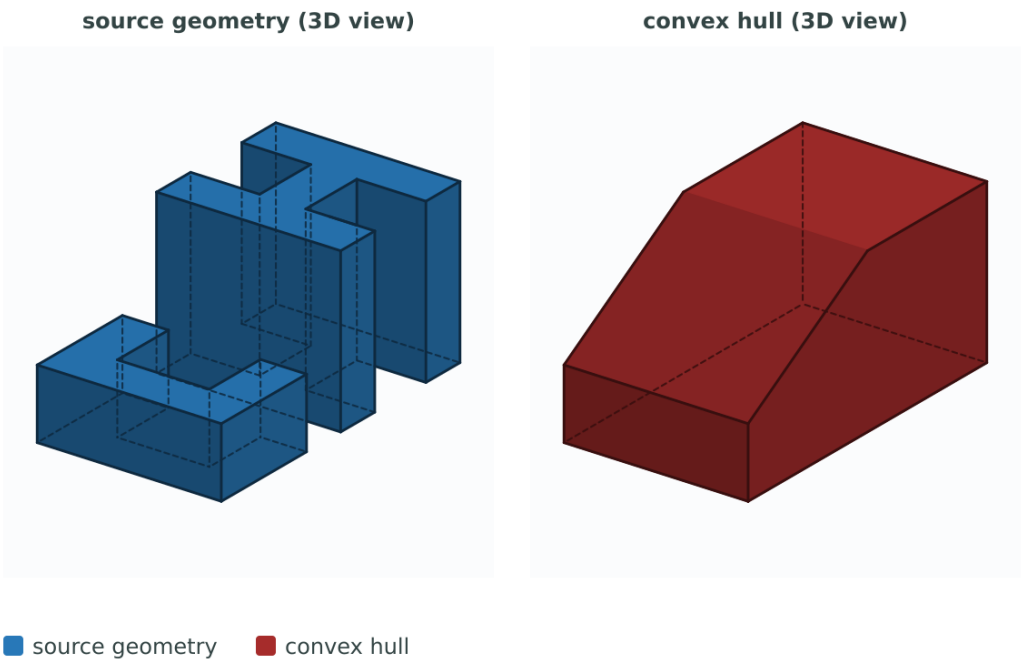
Code

```
WITH footprints(label, geom, height) AS (VALUES
  ('C', 'POLYGON((0 0,0 8,5 8,5 6,2 6,2 2,5 2,5 0,0 0))'::geometry, 3),
  ('H', 'POLYGON((7 0,9 0,9 3,12 3,12 0,14 0,14 8,12 8,12 5,9 5,9 8,7 8,7 0))'::geometry, 7)
), parts AS (
  SELECT label, CG_Extrude(geom, 0, 0, height) AS geom
  FROM footprints
), shapes AS (
  SELECT ST_Collect(geom ORDER BY label) AS source_geometry
  FROM parts
)
SELECT
  source_geometry AS input_source_geometry,
  CG_3DConvexHull(source_geometry) AS convex_hull
FROM shapes;
```

Output

```
-[ RECORD 1 ]----------
input_source_geometry | GEOMETRYCOLLECTION(POLYHEDRALSURFACE(((0 0 0,0 8 0,5 8 0,5 6 0,2 6 0,2 2 0,5 2 0,5 0 0,0 0 0)) ↩
     ,((0 0 3,5 0 3,5 2 3,2 2 3,2 6 3,5 6 3,5 8 3,0 8 3,0 0 3)),((0 0 0,0 0 3,0 8 3,0 8 0,0 0 0)),((0 8 0,0 8 3,5 8 3,5 ↩
      8 0,0 8 0)),((5 8 0,5 8 3,5 6 3,5 6 0,5 8 0)),((5 6 0,5 6 3,2 6 3,2 6 0,5 6 0)),((2 6 0,2 6 3,2 2 3,2 2 0,2 6 0)) ↩
     ,((2 2 0,2 2 3,5 2 3,5 2 0,2 2 0)),((5 2 0,5 2 3,5 0 3,5 0 0,5 2 0)),((5 0 0,5 0 3,0 0 3,0 0 0,5 0 0))),  ↩
     POLYHEDRALSURFACE(((7 0 0,7 8 0,9 8 0,9 5 0,12 5 0,12 8 0,14 8 0,14 0 0,12 0 0,12 3 0,9 3 0,9 0 0,7 0 0)),((7 0  ↩
     7,9 0 7,9 3 7,12 3 7,12 0 7,14 0 7,14 8 7,12 8 7,12 5 7,9 5 7,9 8 7,7 8 7,7 0 7)),((7 0 0,7 0 7,7 8 7,7 8 0,7 0 0) ↩
     ),((7 8 0,7 8 7,9 8 7,9 8 0,7 8 0)),((9 8 0,9 8 7,9 5 7,9 5 0,9 8 0)),((9 5 0,9 5 7,12 5 7,12 5 0,9 5 0)),((12 5  ↩
     0,12 5 7,12 8 7,12 8 0,12 5 0)),((12 8 0,12 8 7,14 8 7,14 8 0,12 8 0)),((14 8 0,14 8 7,14 0 7,14 0 0,14 8 0)),((14 ↩
      0 0,14 0 7,12 0 7,12 0 0,14 0 0)),((12 0 0,12 0 7,12 3 7,12 3 0,12 0 0)),((12 3 0,12 3 7,9 3 7,9 3 0,12 3 0)),((9 ↩
      3 0,9 3 7,9 0 7,9 0 0,9 3 0)),((9 0 0,9 0 7,7 0 7,7 0 0,9 0 0))))
convex_hull           | POLYHEDRALSURFACE(((14 8 7,14 0 0,14 8 0,14 8 7)),((14 0 7,7 8 7,7 0 7,14 0 7)),((0 8 3,7 8 7,0 ↩
      8 0,0 8 3)),((0 0 3,7 8 7,0 8 3,0 0 3)),((0 0 3,0 8 3,0 0 0,0 0 3)),((7 0 7,7 8 7,0 0 3,7 0 7)),((14 0 7,14 0  ↩
     0,14 8 7,14 0 7)),((7 0 7,14 0 0,14 0 7,7 0 7)),((14 8 7,7 8 7,14 0 7,14 8 7)),((0 0 0,0 8 3,0 8 0,0 0 0)),((0 0  ↩
     0,14 0 0,0 0 3,0 0 0)),((0 0 3,14 0 0,7 0 7,0 0 3)),((0 8 0,14 0 0,0 0 0,0 8 0)),((14 8 0,14 0 0,0 8 0,14 8 0)) ↩
     ,((14 8 0,7 8 7,14 8 7,14 8 0)),((0 8 0,7 8 7,14 8 0,0 8 0)))
```

Figure



Voir aussi

ST_Letters, ST_AsX3D

8.3.9 CG_3DIntersection

CG_3DIntersection — Réaliser une intersection 3D

Synopsis

geometry **CG_3DIntersection**(geometry geom1, geometry geom2);

Description

Renvoie une géométrie qui est la partie partagée entre geom1 et geom2.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

3D linestrings and polygons can intersect in a 3D segment.

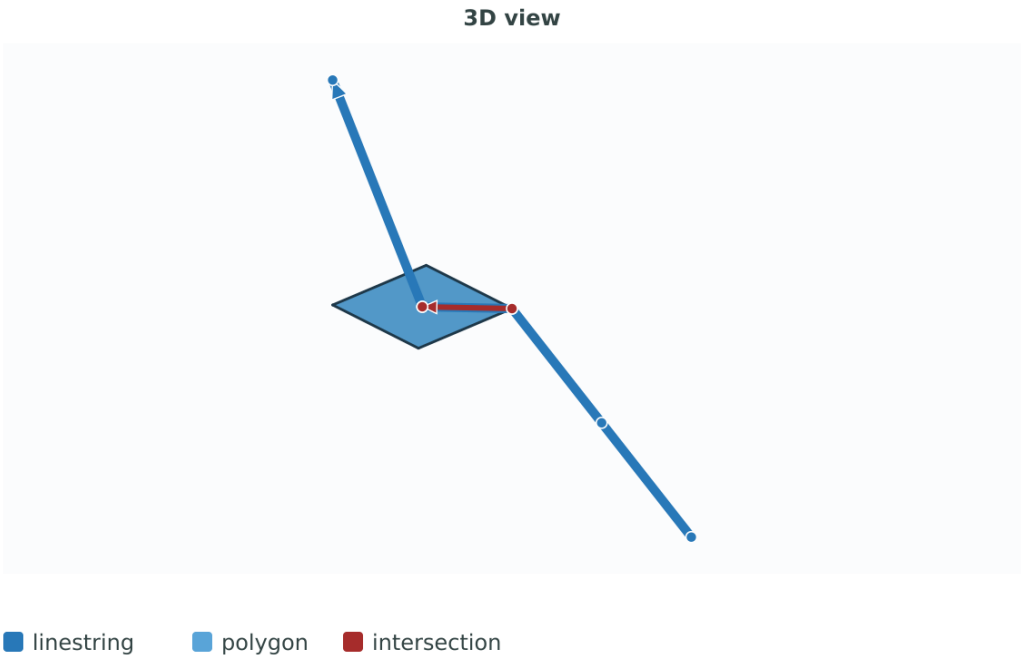
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)'::geometry AS ↩
    input_linestring,
         'POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8))'::geometry AS input_polygon
)
SELECT input_linestring AS input_linestring,
       input_polygon AS input_polygon,
       CG_3DIntersection(input_linestring, input_polygon) AS intersection
FROM data;
```

Output

```
LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10) | POLYGON Z ((0 0 8,0 1 8,1 1 8,1 0 8,0 0 8)) | LINESTRING Z (1 1 ↩
     8,0.5 0.5 8)
```

Figure



Intersect a cube shell with a vertical polygon.

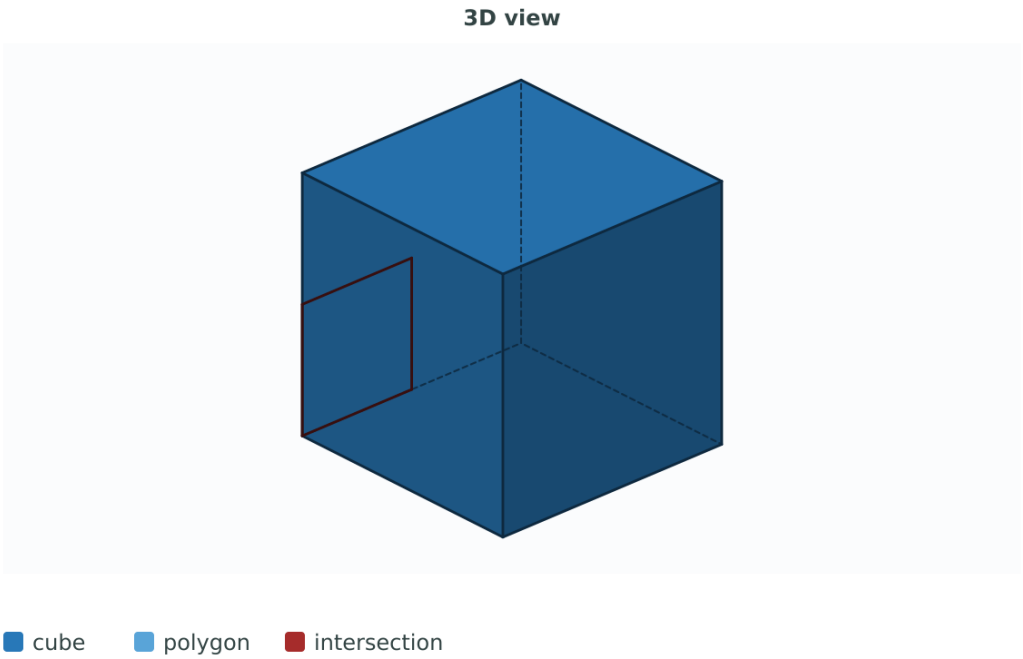
Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z(
    ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
    ((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),
    ((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
    ((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
    ((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),
    ((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))
  )'::geometry AS input_cube,
  'POLYGON Z ((0 0 0,0 0 0.5,0 0.5 0.5,0 0.5 0,0 0 0))'::geometry AS input_polygon
)
SELECT input_cube AS input_cube,
       input_polygon AS input_polygon,
       CG_3DIntersection(input_cube, input_polygon) AS intersection
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 ↩
    0)),((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))) | ↩
    POLYGON Z ((0 0 0,0 0 0.5,0 0.5 0.5,0 0.5 0,0 0 0)) | TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 ↩
    0.5 0.5,0 0.5 0)))
```

Figure



A smaller solid contained by a larger solid remains volumetric after intersection.

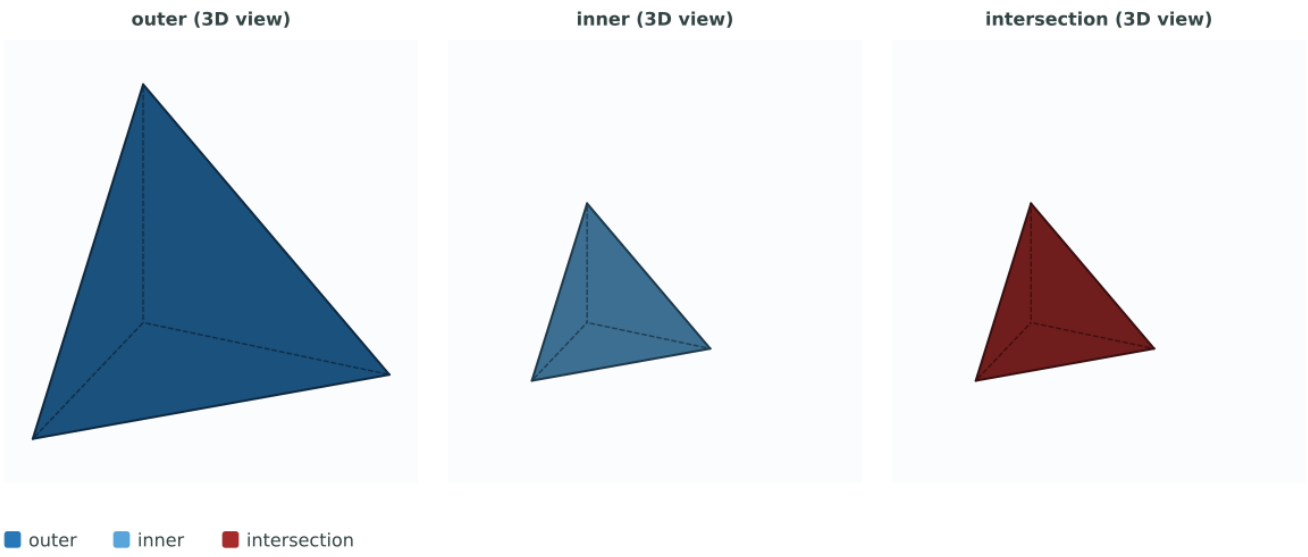
Code

```
WITH data AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
    ((0 0 0,0 4 0,4 0 0,0 0 0)),
    ((0 0 0,4 0 0,0 0 4,0 0 0)),
    ((0 0 0,0 0 4,0 4 0,0 0 0)),
    ((4 0 0,0 4 0,0 0 4,4 0 0))
  )'::geometry) AS outer_solid
), solids AS (
  SELECT outer_solid,
         ST_Scale(outer_solid, 0.5, 0.5, 0.5) AS inner_solid
  FROM data
)
SELECT outer_solid AS input_outer,
       inner_solid AS input_inner,
       CG_3DIntersection(outer_solid, inner_solid) AS intersection
FROM solids;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 0 0,0 0 0)),((0 0 0,4 0 0,0 0 4,0 0 0)),((0 0 0,0 0 4,0 4 0,0 0 0)),((4 0 0,0 4 ↩
    0,0 0 4,4 0 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 2 0,2 0 0,0 0 0)),((0 0 0,2 0 0,0 0 2,0 0 0)),((0 0 0,0 0 2,0 2 ↩
    0,0 0 0)),((2 0 0,0 2 0,0 0 2,2 0 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 2 0,2 0 0,0 0 0)),((0 0 0,0 0 2,0 2 0,0 0 ↩
    0)),((2 0 0,0 0 2,0 0 0,2 0 0)),((2 0 0,0 2 0,0 0 2,2 0 0)))
```

Figure



8.3.10 CG_Union

CG_Union — Calcule l'union de deux géométries

Synopsis

geometry **CG_Union**(geometry geomA , geometry geomB);

Description

Calcule l'union de deux géométries.

Réalisé par le module SFCGAL



Note
NOTE : cette fonction renvoie une géométrie représentant l'union.

Disponibilité : 3.5.0


Cette méthode nécessite le backend SFCGAL.

Exemples de géométrie

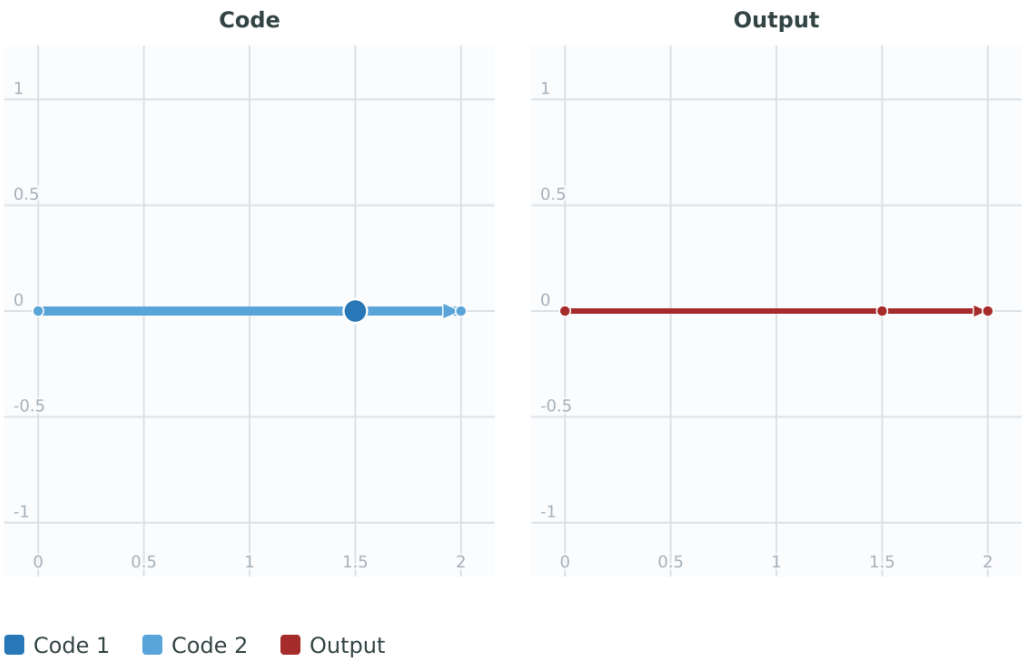
Code

```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
```

Output

```
LINESTRING(-1 0,0.5 0,1 0)
```

Figure



Voir aussi

CG_3DUnion, ST_Union

8.3.11 CG_3DUnion

CG_3DUnion — Réaliser l'union 3D en utilisant postgis_sfcgal.

Synopsis

geometry **CG_3DUnion**(geometry geom1, geometry geom2);
geometry **CG_3DUnion**(geometry set g1field);

Description

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette méthode implémente la spécification SQL/MM. SQL-MM IEC 13249-3: 5.1

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Variante aggregate : renvoie une géométrie qui est l'union 3D d'un ensemble de géométries. La fonction CG_3DUnion() est une fonction "aggregate" dans la terminologie de PostgreSQL. Cela signifie qu'elle opère sur des lignes de données, de la même manière que les fonctions SUM() et AVG() et, comme la plupart des agrégats, elle ignore les géométries NULL.

Exemples

Union two overlapping tetrahedral solids.

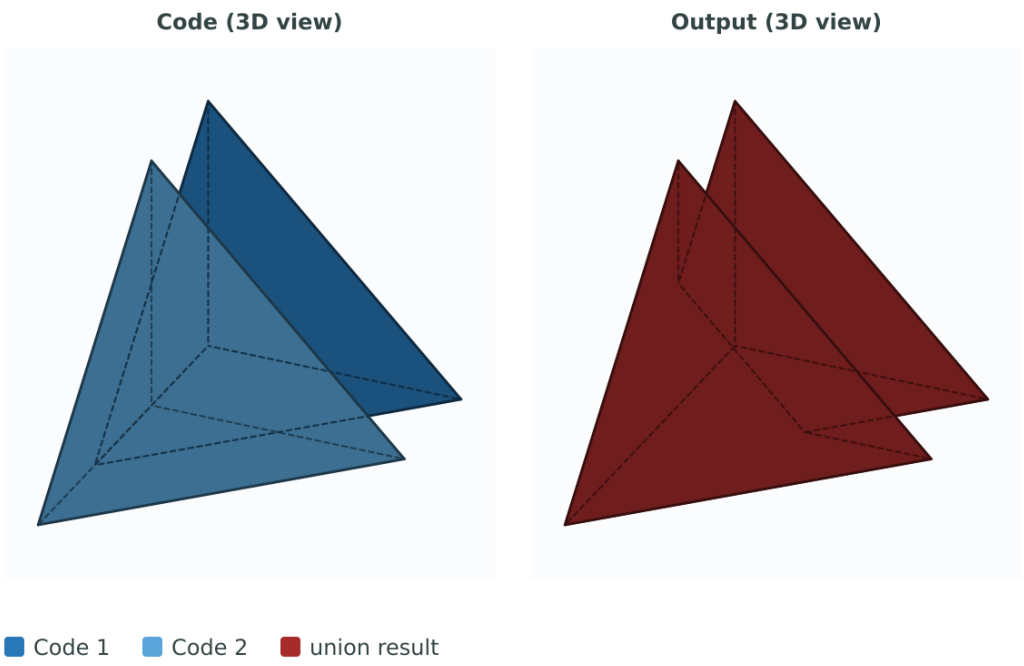
Code

```
WITH solids AS (
  SELECT CG_MakeSolid('POLYHEDRALSURFACE Z (
            ((0 0 0,4 0 0,0 4 0,0 0 0)),
            ((0 0 0,0 0 4,4 0 0,0 0 0)),
            ((0 0 0,0 4 0,0 0 4,0 0 0)),
            ((4 0 0,0 0 4,0 4 0,4 0 0)))'::geometry) AS geom1,
         CG_MakeSolid('POLYHEDRALSURFACE Z (
            ((2 0 0,6 0 0,2 4 0,2 0 0)),
            ((2 0 0,2 0 4,6 0 0,2 0 0)),
            ((2 0 0,2 4 0,2 0 4,2 0 0)),
            ((6 0 0,2 0 4,2 4 0,6 0 0)))'::geometry) AS geom2
)
SELECT CG_3DUnion(geom1, geom2) AS union_result
FROM solids;
```

Output

```
POLYHEDRALSURFACE Z (((2 0 0,0 0 0,2 2 0,2 0 0)),((2 0 2,0 0 0,2 0 0,2 0 2)),((0 0 4,0 4 0,0 0 0,0 0 4)),((2 0 2,0 4 ↩
    0,0 0 4,2 0 2)),((0 4 0,2 2 0,0 0 0,0 4 0)),((2 2 0,0 4 0,2 0 2,2 2 0)),((0 0 0,2 0 2,0 0 4,0 0 0)),((4 0 0,2 0 ↩
    0,2 2 0,4 0 0)),((4 0 0,2 0 2,2 0 0,4 0 0)),((4 0 0,2 2 0,6 0 0,4 0 0)),((2 0 2,4 0 0,2 0 4,2 0 2)),((2 0 2,2 4 ↩
    0,2 2 0,2 0 2)),((2 4 0,2 0 4,6 0 0,2 4 0)),((2 2 0,2 4 0,6 0 0,2 2 0)),((2 4 0,2 0 2,2 0 4,2 4 0)),((4 0 0,6 0 ↩
    0,2 0 4,4 0 0)))
```

Figure



Voir aussi

CG_Extrude, ST_AsX3D, CG_3DIntersection CG_3DDifference

8.3.12 CG_AlphaShape

CG_AlphaShape — Calcul d'une forme Alpha entourant une géométrie

Synopsis

geometry **CG_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

Description

Calcule la forme alpha des points d'une géométrie. Une forme alpha est une géométrie polygonale (généralement) concave qui contient tous les sommets de l'entrée et dont les sommets sont un sous-ensemble des sommets de l'entrée. Une forme alpha permet de se rapprocher davantage de la forme de l'entrée que la forme produite par l' enveloppe convexe.

La "proximité de l'ajustement" est contrôlée par le paramètre `alpha`, qui peut prendre des valeurs comprises entre 0 et l'infini. Des valeurs alpha plus petites produisent des résultats plus concaves. Les valeurs alpha supérieures à une certaine valeur dépendant des données produisent l'enveloppe convexe de l'entrée.



Note
Conformément à l'implémentation du CGAL, la valeur alpha est le *carré* du rayon du disque utilisé dans l'algorithme Alpha-Shape pour "éroder" la triangulation de Delaunay des points d'entrée. Voir CGAL Alpha-Shapes pour plus d'informations. Cela diffère de la définition originale des formes alpha, qui définit alpha comme le rayon du disque d'érosion.

La forme calculée ne contient pas de trous, sauf si l'argument facultatif `allow_holes` est spécifié comme true.

Cette fonction calcule effectivement une enveloppe concave d'une géométrie d'une manière similaire à ST_ConcaveHull, mais utilise CGAL et un algorithme différent.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.4.1.


Cette méthode nécessite le backend SFCGAL.

Exemples

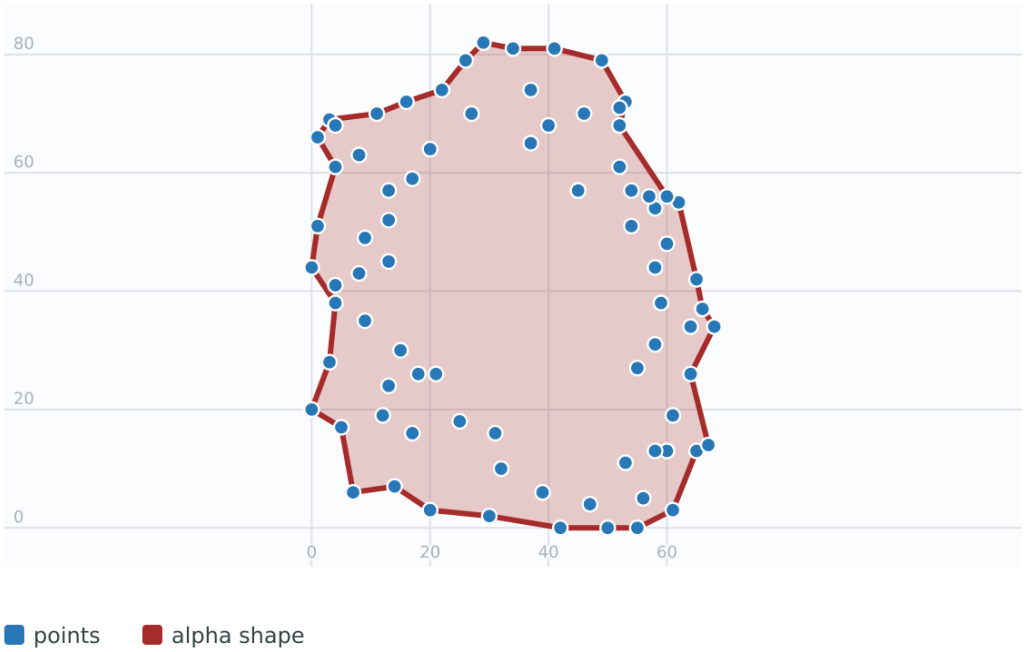
The original point cloud and the alpha-shape share one comparison frame.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 ↩
    65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52 ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31 ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_AlphaShape(input_points, 80.2) AS alpha_shape
FROM data;
```

Figure



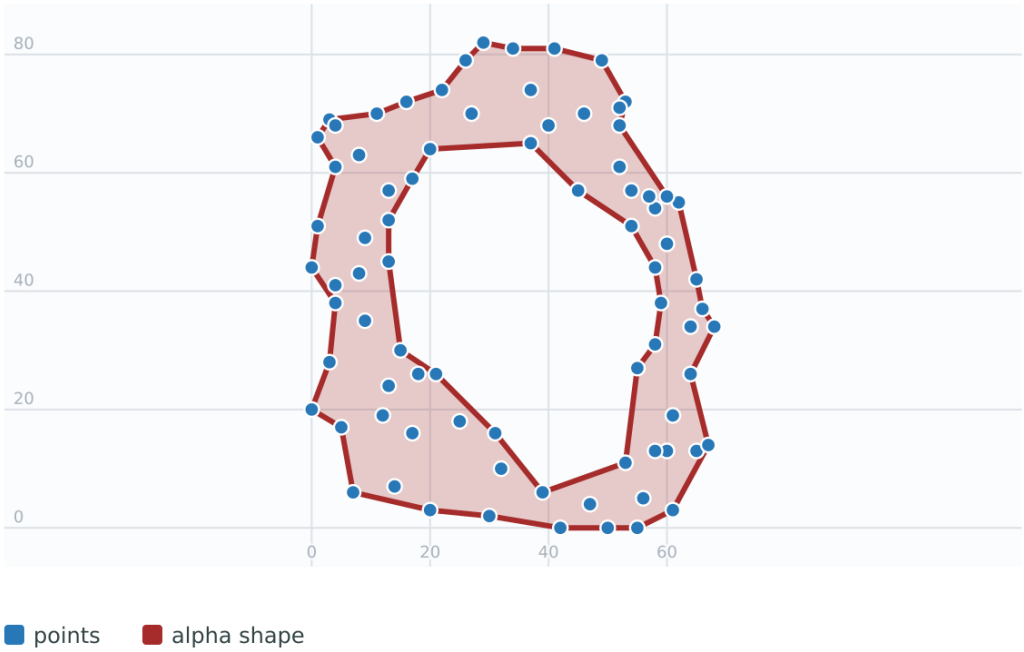
The same point cloud can produce a shape with a hole when holes are allowed.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 51)
    ,(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 58)
    ,(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_AlphaShape(input_points, 100.1, true) AS alpha_shape
FROM data;
```

Figure



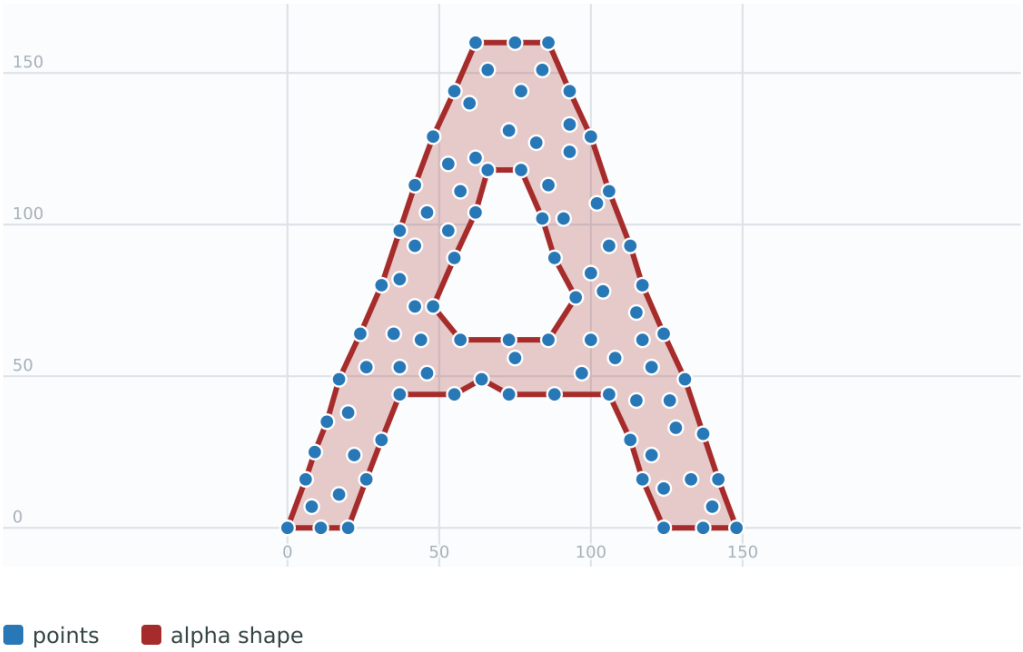
The second documented point cloud also produces a holed alpha-shape.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT ((132 64),(114 64),(99 64),(81 64),(63 64),(57 49),(52 36),(46 20),(37
    20),(26 20),(32 36),(39 55),(43 69),(50 84),(57 100),(63 118),(68 133),(74 149),(81 164)
    ,(88 180),(101 180),(112 180),(119 164),(126 149),(132 131),(139 113),(143 100),(150 84)
    ,(157 69),(163 51),(168 36),(174 20),(163 20),(150 20),(143 36),(139 49),(132 64),(99
    151),(92 138),(88 124),(81 109),(74 93),(70 82),(83 82),(99 82),(112 82),(126 82),(121
    96),(114 109),(110 122),(103 138),(99 151),(34 27),(43 31),(48 44),(46 58),(52 73),(63
    73),(61 84),(72 71),(90 69),(101 76),(123 71),(141 62),(166 27),(150 33),(159 36),(146
    44),(154 53),(152 62),(146 73),(134 76),(143 82),(141 91),(130 98),(126 104),(132 113)
    ,(128 127),(117 122),(112 133),(119 144),(108 147),(119 153),(110 171),(103 164),(92
    171),(86 160),(88 142),(79 140),(72 124),(83 131),(79 118),(68 113),(63 102),(68 93),(35
     45))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_AlphaShape(input_points, 102.2, true) AS alpha_shape
FROM data;
```

Figure



Voir aussi

ST_ConcaveHull, CG_OptimalAlphaShape

8.3.13 CG_ApproxConvexPartition

CG_ApproxConvexPartition — Calcul de la partition convexe approximative de la géométrie du polygone

Synopsis

geometry **CG_ApproxConvexPartition**(geometry geom);

Description

Calcule la partition convexe approximative de la géométrie du polygone (à l'aide d'une triangulation).



Note
Une partition d'un polygone P est un ensemble de polygones tels que les intérieurs des polygones ne se coupent pas et que l'union des polygones est égale à l'intérieur du polygone original P. Les fonctions CG_ApproxConvexPartition et CG_GreeneApproxConvexPartition produisent des partitions convexes approximativement optimales. Ces deux fonctions produisent des décompositions convexes en décomposant d'abord le polygone en polygones plus simples ; CG_ApproxConvexPartition utilise une triangulation et CG_GreeneApproxConvexPartition une partition monotone. Ces deux fonctions garantissent toutes deux qu'elles ne produiront pas plus de quatre fois le nombre optimal de pièces convexes, mais elles diffèrent par leur complexité d'exécution. Bien que l'algorithme d'approximation basé sur la triangulation produise souvent moins de pièces convexes, ce n'est pas toujours le cas.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0.

Nécessite SFCGAL >= 1.5.0

 Cette méthode nécessite le backend SFCGAL.

Exemples

Approximate convex partition of the same polygon used by the other partition examples.

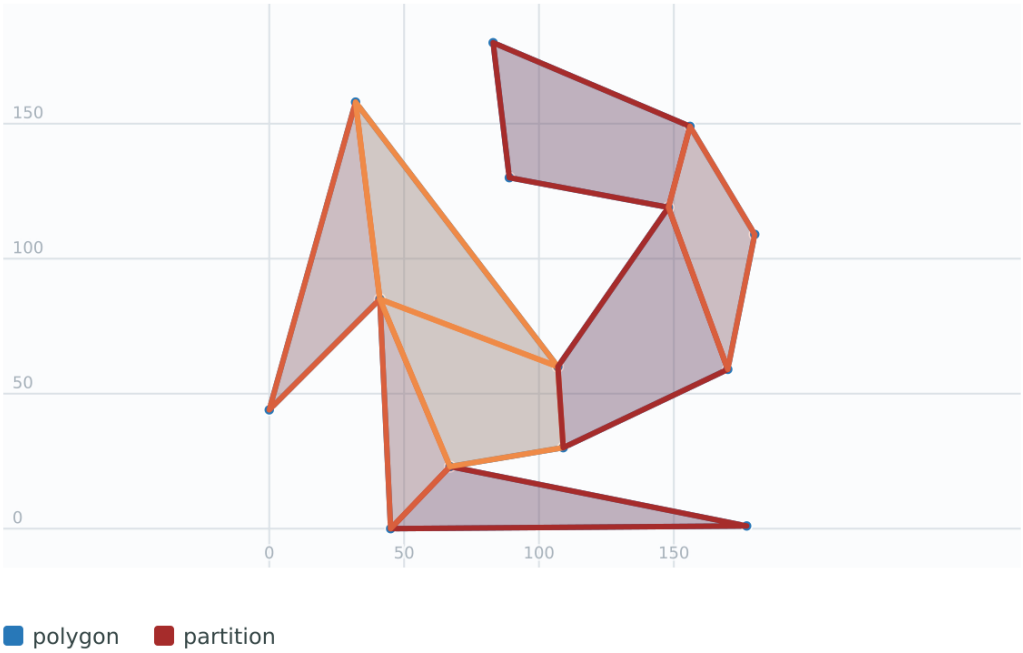
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_ApproxConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)), POLYGON((32 159,0 45,41 86,32 159)), POLYGON ↩
    ((107 61,32 159,41 86,107 61)), POLYGON((45 1,177 2,67 24,45 1)), POLYGON((41 86,45 1,67 24,41 86)), POLYGON((107 ↩
    61,41 86,67 24,109 31,107 61)), POLYGON((148 120,107 61,109 31,170 60,148 120)), POLYGON((156 150,148 120,170 ↩
    60,180 110,156 150)))
```

Figure



Voir aussi

CG_YMonotonePartition, CG_GreeneApproxConvexPartition, CG_OptimalConvexPartition

8.3.14 CG_ApproximateMedialAxis

CG_ApproximateMedialAxis — Calculer l'axe médian approximatif d'une géométrie aréolaire.

Synopsis

geometry **CG_ApproximateMedialAxis**(geometry geom);
geometry **CG_ApproximateMedialAxis**(geometry geom, boolean projected);

Description

Retourne un axe médian approximatif pour l'entrée aréolaire en se basant sur son squelette (straight skeleton). Utilise une API spécifique à SFCGAL lorsque compilée avec une version compatible (1.2.0+). Sinon, la fonction est juste un wrapper de CG_StraightSkeleton (cas le plus lent).

When `projected` is `true`, free endpoints of the medial axis are extended to reach the polygon boundary (projected medial axis). Requires SFCGAL 2.3.0+. When built against an older SFCGAL version, a notice is emitted and the non-projected result is returned instead.

Disponibilité : 3.5.0

Availability: 3.7.0 - `projected` parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions.



Note
Cette fonction ignore la dimension Z. Elle donne toujours un résultat en 2D, même lorsqu'elle est utilisée sur une géométrie en 3D.

 Cette méthode nécessite le backend SFCGAL.

 Cette fonction prend en charge les surfaces Polyhedral.

 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

A polygon and its approximate medial axis.

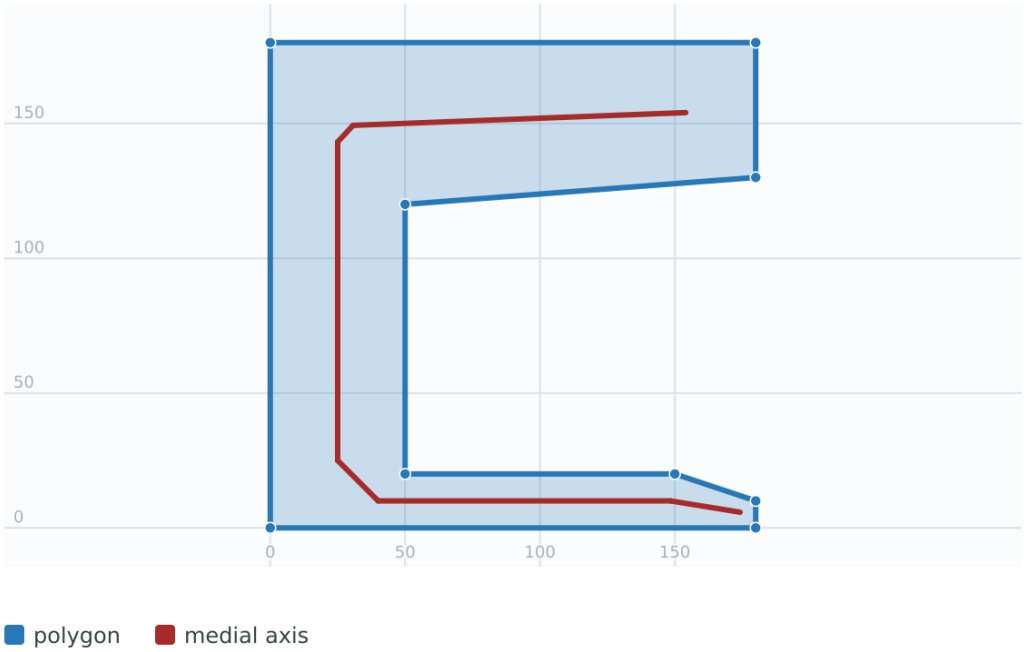
Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' ↩
    ::geometry AS polygon
)
SELECT polygon AS input_polygon,
       CG_ApproximateMedialAxis(polygon) AS medial_axis
FROM data;
```

Output

```
POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190)) | MULTILINESTRING((184.189 ↩
    15.811,158.377 20),(50 20,158.377 20),(50 20,35 35),(35 153.151,35 35),(35 153.151,40.697 159.303),(164.04 ↩
    164.04,40.697 159.303))
```

Figure



This example computes the projected medial axis with free endpoints extended to the polygon boundary.

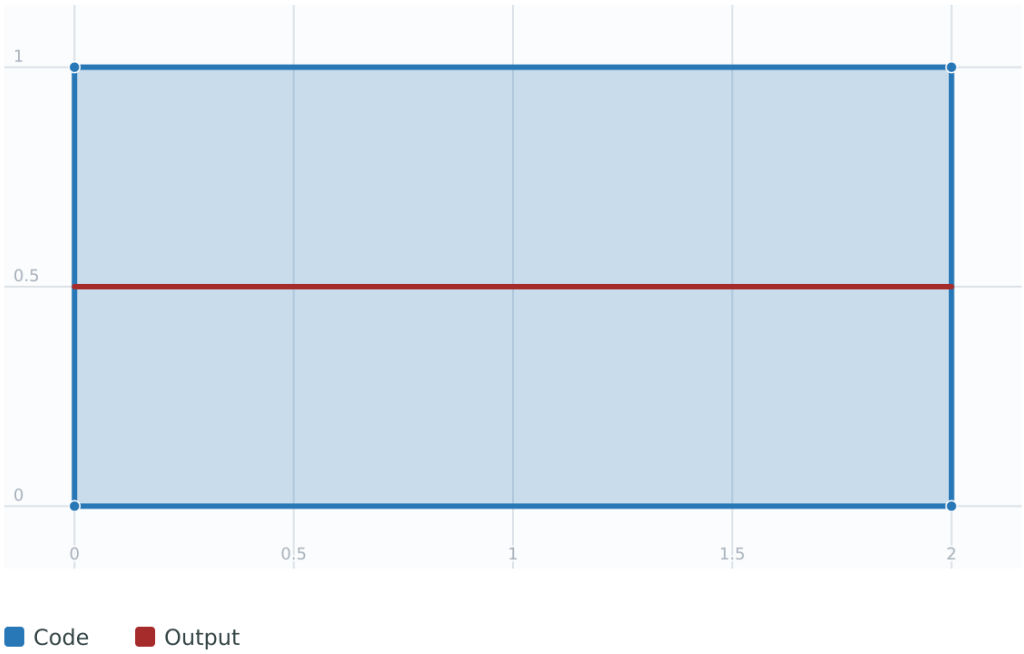
Code

```
SELECT CG_ApproximateMedialAxis('POLYGON((0 0,2 0,2 1,0 1,0 0))', true);
```

Output

```
MULTILINESTRING((0 0.5,0.5 0.5,1.5 0.5,2 0.5))
```

Figure



Voir aussi

CG_StraightSkeleton, CG_StraightSkeletonPartition

8.3.15 CG_GenerateFlatRoof

CG_GenerateFlatRoof — Generate a flat (box) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateFlatRoof**(geometry geom, float8 height);

Description

Generate a flat (box-shaped) roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The roof height above the ground plane is given by `height` (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Generate a flat roof two units above a rectangular footprint.

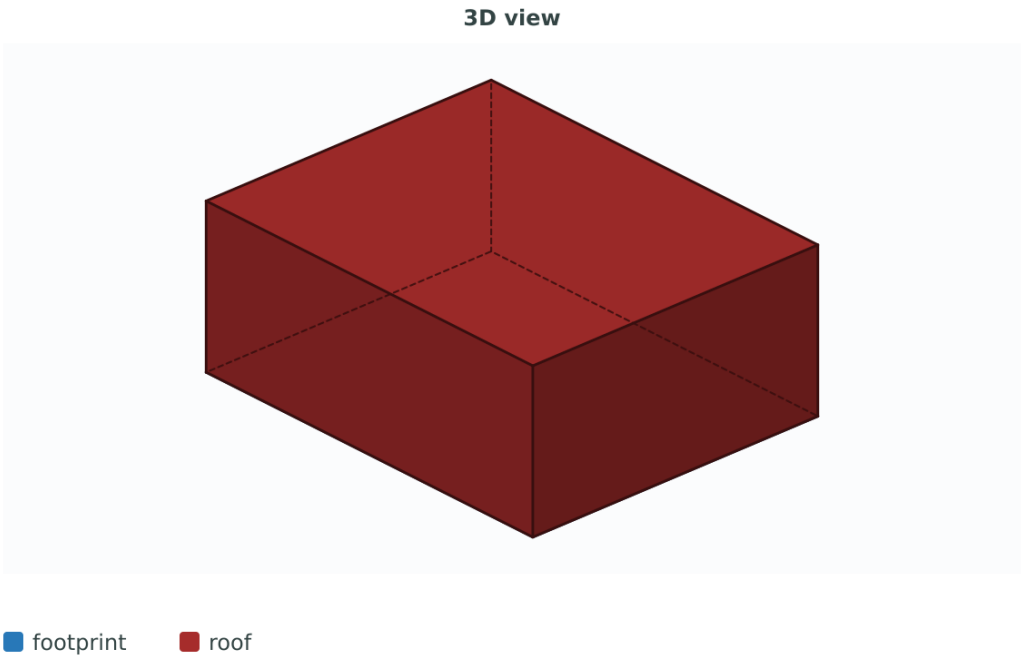
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateFlatRoof(footprint, 2.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,5 4 0,5 0 0,0 0 0)),((0 0 2,5 0 2,5 4 2,0 4 2,0 0 2)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 ↩
    0)),((0 4 0,0 4 2,5 4 2,5 4 0,0 4 0)),((5 4 0,5 4 2,5 0 2,5 0 0,5 4 0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)))
```

Figure



Voir aussi

CG_GenerateHippedRoof, CG_GenerateGableRoof, CG_GenerateSkillionRoof, CG_GenerateRoof

8.3.16 CG_GenerateHippedRoof

CG_GenerateHippedRoof — Generate a hipped roof from a footprint polygon.

Synopsis

geometry **CG_GenerateHippedRoof**(geometry geom, float8 height);

Description

Generate a hipped roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. All sides slope upward to meet at a ridge or apex. The roof height is given by `height` (default 3.0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

The footprint and hipped roof are shown side-by-side so the generated 3D faces do not hide behind the input footprint.

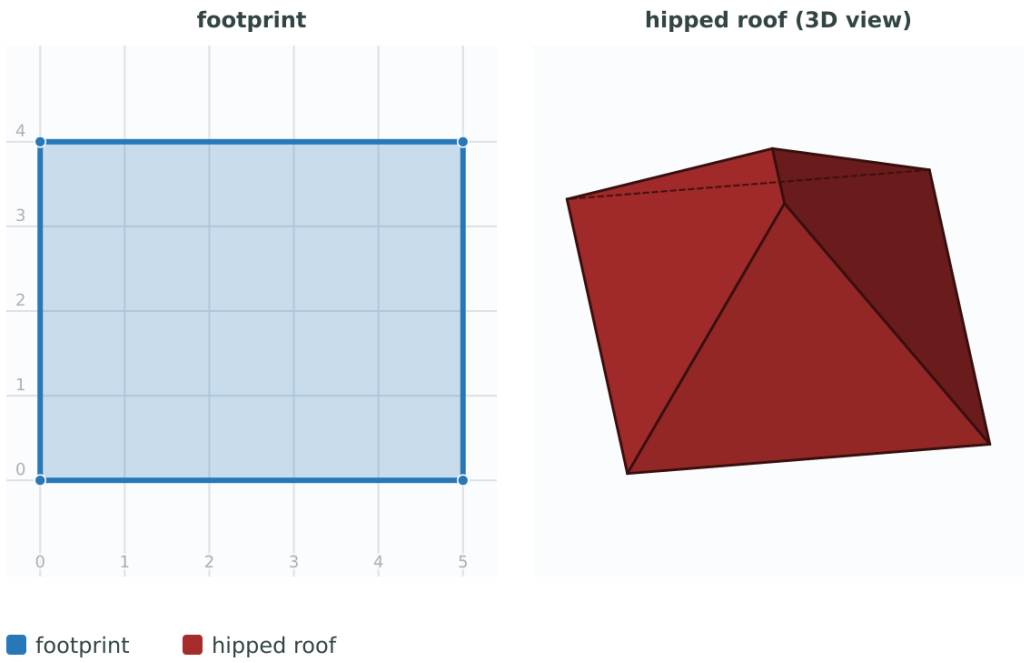
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS input_footprint
)
SELECT input_footprint AS input_footprint,
       CG_GenerateHippedRoof(input_footprint, 2.0) AS hipped_roof
FROM data;
```

Output

```
POLYGON((0 0,5 0,5 4,0 4,0 0)) | POLYHEDRALSURFACE Z (((0 4 0,5 4 0,5 0 0,0 0 0,0 4 0)),((0 4 0,0 0 0,2 2 2,0 4 0)),((0 ↩
      0 0,5 0 0,3 2 2,2 2 2,0 0 0)),((5 0 0,5 4 0,3 2 2,5 0 0)),((5 4 0,0 4 0,2 2 2,3 2 2,5 4 0)))
```

Figure



Voir aussi

CG_GenerateFlatRoof, CG_GenerateGableRoof, CG_GenerateSkillionRoof, CG_GenerateRoof

8.3.17 CG_GenerateGableRoof

CG_GenerateGableRoof — Generate a gable roof from a footprint polygon.

Synopsis

geometry **CG_GenerateGableRoof**(geometry geom, float8 height, float8 slope_angle);

Description

Generate a gable roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. Two opposite sides are vertical gable ends; the other two sides slope to a horizontal ridge. `height` sets the ridge height above the ground (default 3.0) and `slope_angle` controls the pitch in degrees (default 30.0). The SRID of the input geometry is preserved.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Generate a gable roof using a height of 2 and a 30 degree slope.

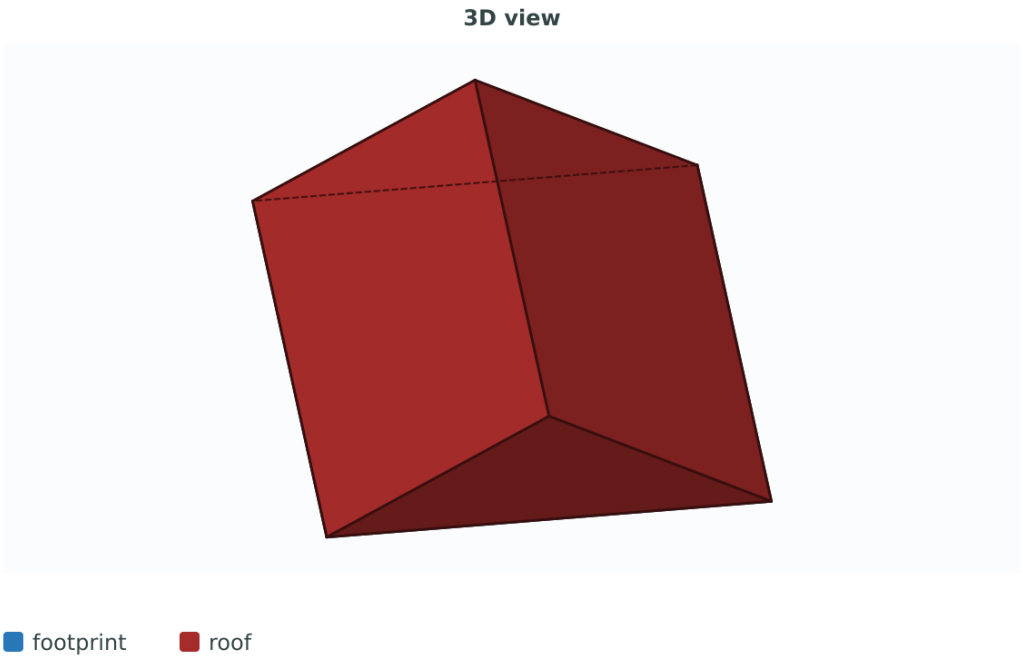
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateGableRoof(footprint, 2.0, 30.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((5 4 0,5 0 0,0 0 0,0 4 0,5 4 0)),((0 4 0,0 0 0,0 2 1.155,0 4 0)),((0 0 0,5 0 0,5 2 1.155,0 2 ↩
    1.155,0 0 0)),((5 0 0,5 4 0,5 2 1.155,5 0 0)),((5 4 0,0 4 0,0 2 1.155,5 2 1.155,5 4 0)))
```

Figure



Voir aussi

CG_GenerateFlatRoof, CG_GenerateHippedRoof, CG_GenerateSkillionRoof, CG_GenerateRoof

8.3.18 CG_GenerateSkillionRoof

CG_GenerateSkillionRoof — Generate a skillion (single-slope) roof from a footprint polygon.

Synopsis

geometry **CG_GenerateSkillionRoof**(geometry geom, float8 height, float8 slope_angle, integer primary_edge_index);

Description

Generate a skillion (single-slope, shed-style) roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The entire roof surface slopes in one direction. `height` sets the maximum ridge height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects which polygon edge defines the high side (0-based, default 0).

When the combination of `height` and `slope_angle` would cause a side wall to vanish (the slope reaches the far edge before the full height is achieved), that face is omitted from the result. This means the number of output patches can vary depending on the parameters — a lower height relative to the footprint depth produces more faces than a taller, steeper configuration.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Height 2.0 produces 6 faces (all walls present); height 4.0 produces 5 faces (one side wall is filtered out because the slope reaches the far edge before the full height).

Code

```
SELECT
  ST_NumPatches(CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))', 2.0)) AS "height 2",
  ST_NumPatches(CG_GenerateSkillionRoof(
    'POLYGON((0 0,5 0,5 4,0 4,0 0))', 4.0)) AS "height 4";
```

Output

```
6 | 5
```

The height 2.0 result is shown with its footprint.

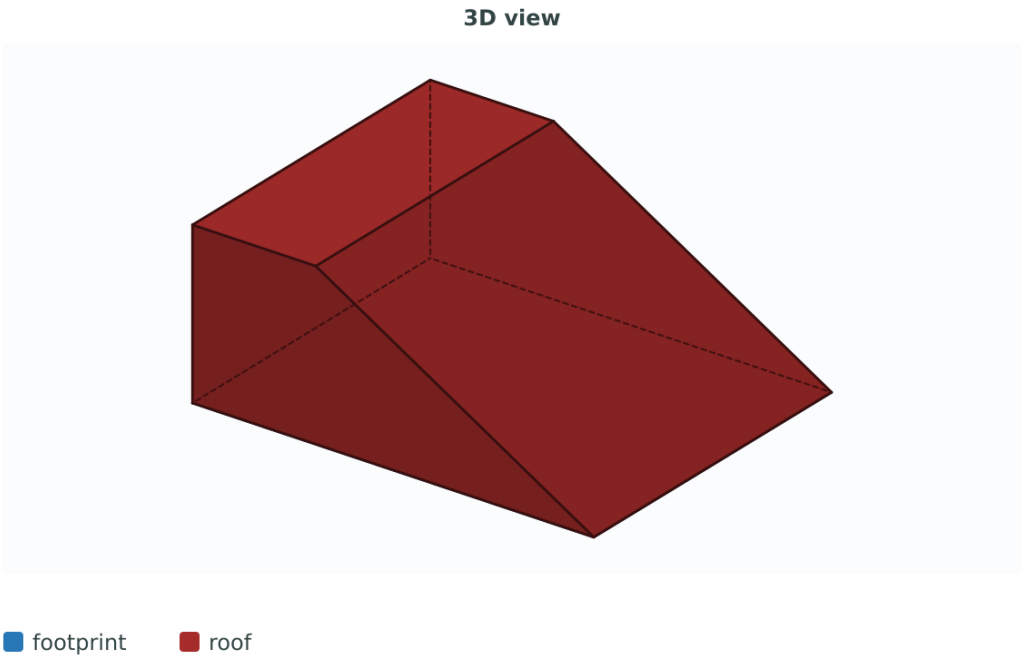
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,5 0,5 4,0 4,0 0))'::geometry AS footprint
)
SELECT CG_GenerateSkillionRoof(footprint, 2.0) AS roof
FROM data;
```

Output

```
POLYHEDRALSURFACE Z (((0 4 0,5 4 0,5 0 0,0 0 0,0 4 0)),((5 0 2,5 4 2,3.464 4 2,3.464 0 2,5 0 2)),((0 4 0,0 0 0,3.464 0 ↩
    2,3.464 4 2,0 4 0)),((0 0 0,5 0 0,5 0 2,3.464 0 2,0 0 0)),((5 0 0,5 4 0,5 4 2,5 0 2,5 0 0)),((5 4 0,0 4 0,3.464 4 ↩
    2,5 4 2,5 4 0)))
```

Figure



Voir aussi

CG_GenerateFlatRoof, CG_GenerateHippedRoof, CG_GenerateGableRoof, CG_GenerateRoof

8.3.19 CG_GenerateRoof

CG_GenerateRoof — Generate a roof of the requested type from a footprint polygon.

Synopsis

geometry **CG_GenerateRoof**(geometry geom, text roof_type, float8 height, float8 slope_angle, integer primary_edge_index);

Description

Generate a roof as a 3D `PolyhedralSurface Z` from a 2D footprint polygon. The `roof_type` parameter selects the roof style (default `'HIPPED'`):

— `FLAT` — flat box roof.
— `HIPPED` — hipped roof (all sides slope to apex).
— `GABLE` — gable roof (two sloping sides, two vertical ends).
— `SKILLION` — single-slope (shed) roof.

`height` sets the roof height (default 3.0), `slope_angle` sets the pitch in degrees (default 30.0), and `primary_edge_index` selects the reference edge for skillion roofs (default 0).

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

The generic dispatcher returns the same 3D roofs as the specialized helpers for the same footprint.

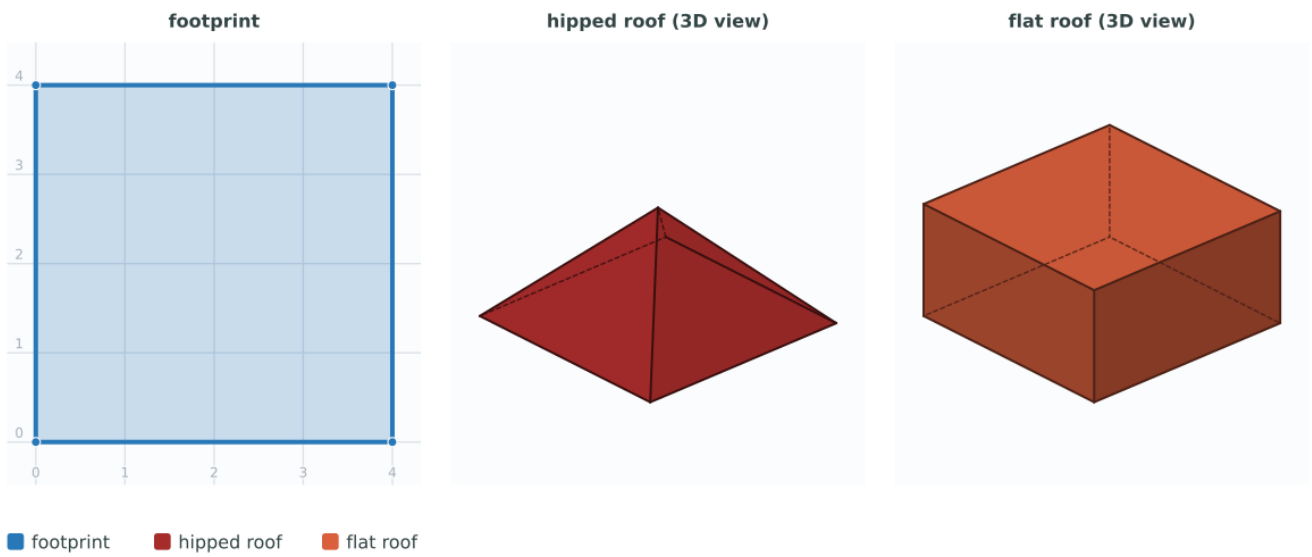
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,4 0,4 4,0 4,0 0))'::geometry AS input_footprint
)
SELECT input_footprint AS input_footprint,
       CG_GenerateRoof(input_footprint, 'HIPPED', 2.0, 30.0, 0) AS hipped_roof,
       CG_GenerateRoof(input_footprint, 'FLAT', 2.0, 30.0, 0) AS flat_roof
FROM data;
```

Output

```
POLYGON((0 0,4 0,4 4,0 4,0 0)) | POLYHEDRALSURFACE Z (((0 4 0,4 4 0,4 0 0,0 0 0,0 4 0)),((0 4 0,0 0 0,2 2 2,0 4 0)),((0 ↩
      0 0,4 0 0,2 2 2,0 0 0)),((4 0 0,4 4 0,2 2 2,4 0 0)),((4 4 0,0 4 0,2 2 2,4 4 0))) | POLYHEDRALSURFACE Z (((0 0 0,0 ↩
      4 0,4 4 0,4 0 0,0 0 0)),((0 0 2,4 0 2,4 4 2,0 4 2,0 0 2)),((0 0 0,0 0 2,0 4 2,0 4 0,0 0 0)),((0 4 0,0 4 2,4 4 2,4 ↩
      4 0,0 4 0)),((4 4 0,4 4 2,4 0 2,4 0 0,4 4 0)),((4 0 0,4 0 2,0 0 2,0 0 0,4 0 0)))
```

Figure



Voir aussi

CG_GenerateFlatRoof, CG_GenerateHippedRoof, CG_GenerateGableRoof, CG_GenerateSkillionRoof

8.3.20 CG_PolygonRepair

CG_PolygonRepair — Repair an invalid polygon or multipolygon.

Synopsis

geometry **CG_PolygonRepair**(geometry geom, text rule);

Description

Repair an invalid polygon or multipolygon using CGAL's 2D Polygon Repair algorithm. Returns a `MultiPolygon`.

The `rule` parameter controls the repair strategy:

— `EVEN_ODD` (default) — areas covered an odd number of times are kept. Available with CGAL 6.0+.
— `NON_ZERO` — areas with non-zero winding number are kept. Requires CGAL 6.1+.
— `UNION` — union of all input polygons. Requires CGAL 6.1+.
— `INTERSECTION` — intersection of all input polygons. Requires CGAL 6.1+.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge les surfaces Polyhedral.

Exemples

Repair a self-intersecting bowtie polygon with the default `EVEN_ODD` rule.

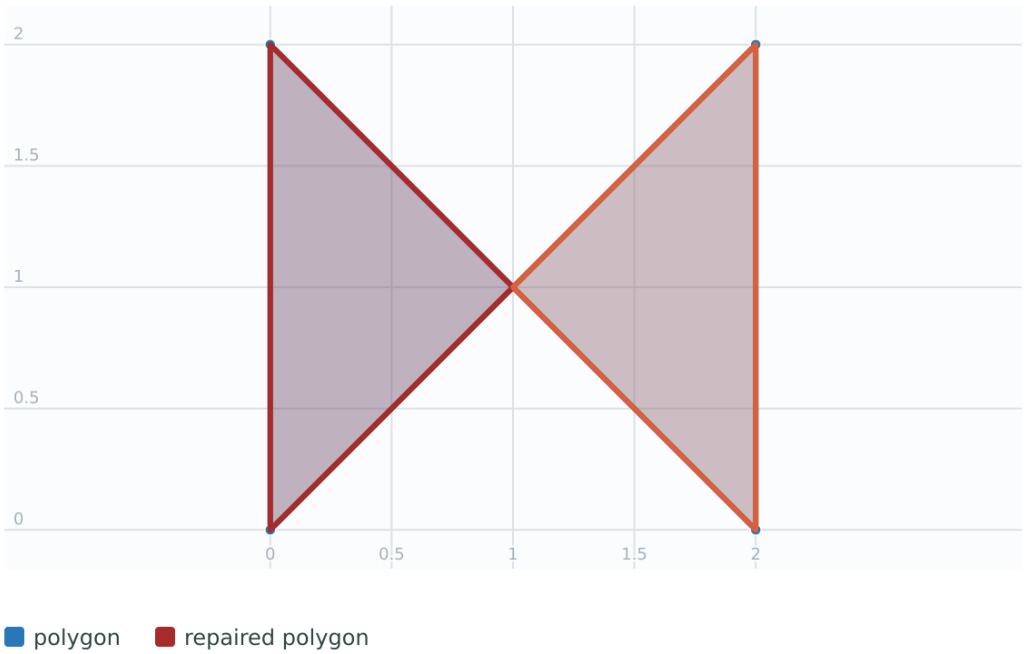
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_PolygonRepair(input_polygon) AS repaired_polygon
FROM data;
```

Output

```
POLYGON((0 0,2 2,2 0,0 2,0 0)) | MULTIPOLYGON(((0 0,1 1,0 2,0 0)),((1 1,2 0,2 2,1 1)))
```

Figure



With the `UNION` rule, all winding area is kept. This rule requires CGAL 6.1 or later.

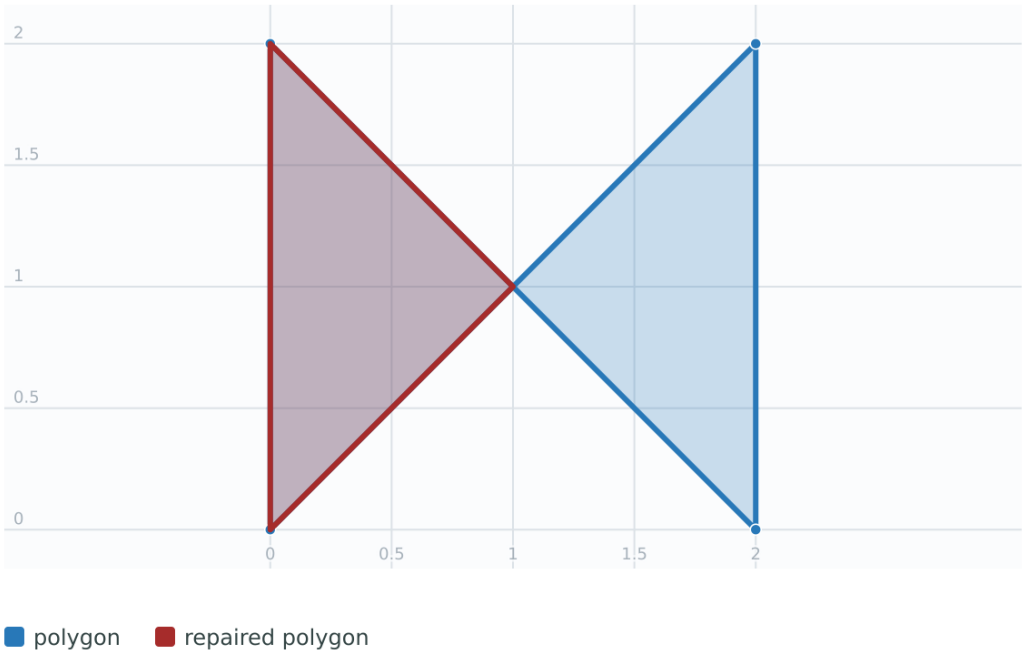
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,2 2,2 0,0 2,0 0))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_PolygonRepair(input_polygon, 'UNION') AS repaired_polygon
FROM data;
```

Output

```
POLYGON((0 0,2 2,2 0,0 2,0 0)) | MULTIPOLYGON(((0 0,1 1,0 2,0 0)))
```

Figure



Voir aussi

ST_IsValid, ST_MakeValid

8.3.21 CG_ConstrainedDelaunayTriangles

CG_ConstrainedDelaunayTriangles — Renvoie une triangulation de Delaunay contrainte autour de la géométrie d'entrée donnée.

Synopsis

geometry **CG_ConstrainedDelaunayTriangles**(geometry g1);

Description

Renvoie un Constrained Delaunay triangulation autour des sommets de la géométrie d'entrée. Le résultat est un TIN.

Cette méthode nécessite le backend SFCGAL.

Disponibilité : 3.0.0

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

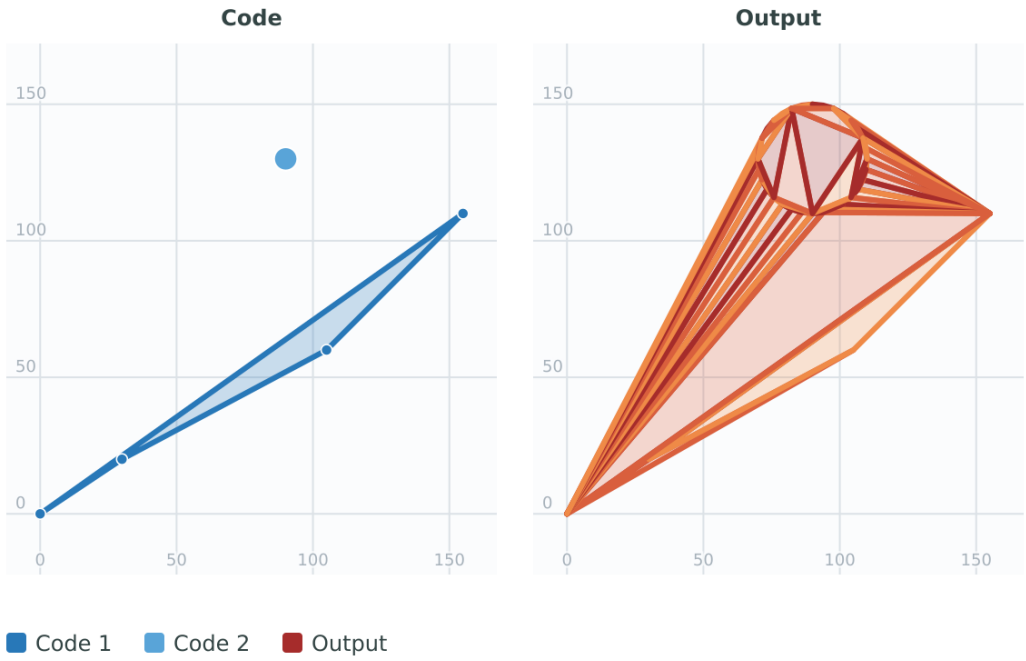
Exemples

Constrained Delaunay triangulation of the polygon and buffered point.

Code

```
SELECT
  CG_ConstrainedDelaunayTriangles(
    ST_Union(
      'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry,
      ST_Buffer('POINT(110 170)'::geometry, 20)
    )
  );
```

Figure

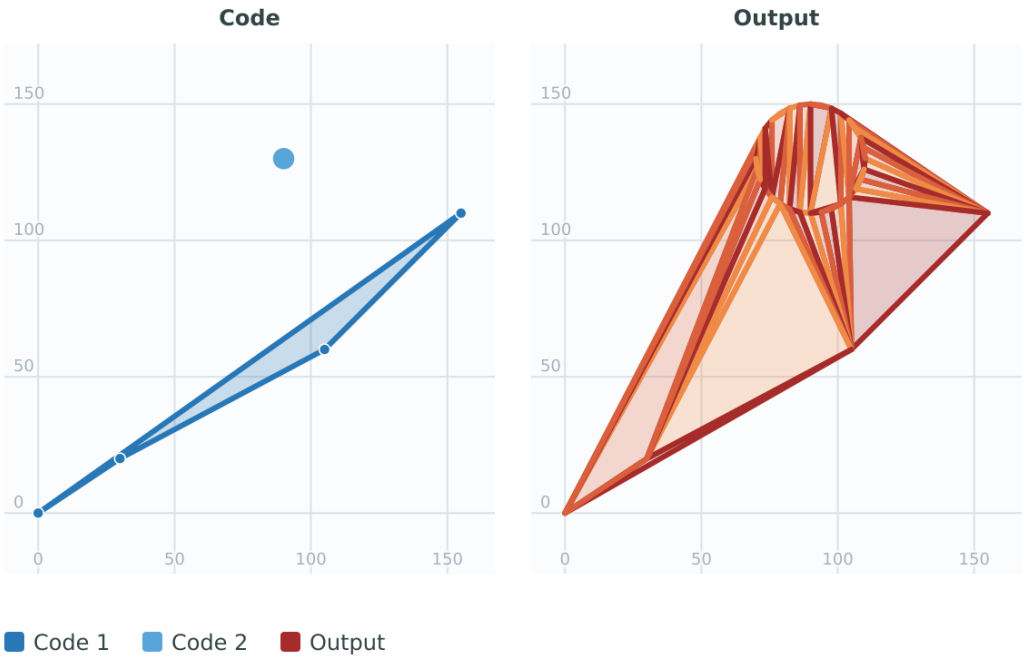


ST_DelaunayTriangles applied to the same input. Triangle edges cross polygon boundaries.

Code

```
SELECT
  ST_DelaunayTriangles(
    ST_Union(
      'POLYGON((175 150,20 40,50 60,125 100,175 150))'::geometry,
      ST_Buffer('POINT(110 170)'::geometry, 20)
    )
  );
```

Figure



Voir aussi

ST_DelaunayTriangles, ST_TriangulatePolygon, CG_Tesselate, ST_ConcaveHull, ST_Dump

8.3.22 CG_Extrude

CG_Extrude — Extruder une surface vers un volume

Synopsis

geometry **CG_Extrude**(geometry geom, float x, float y, float z);

Description

Extrudes the input along the vector (`x`, `y`, `z`) and connects it to its translated copy. Extrusion can raise the dimensionality of the result, for example from a surface to a volume.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

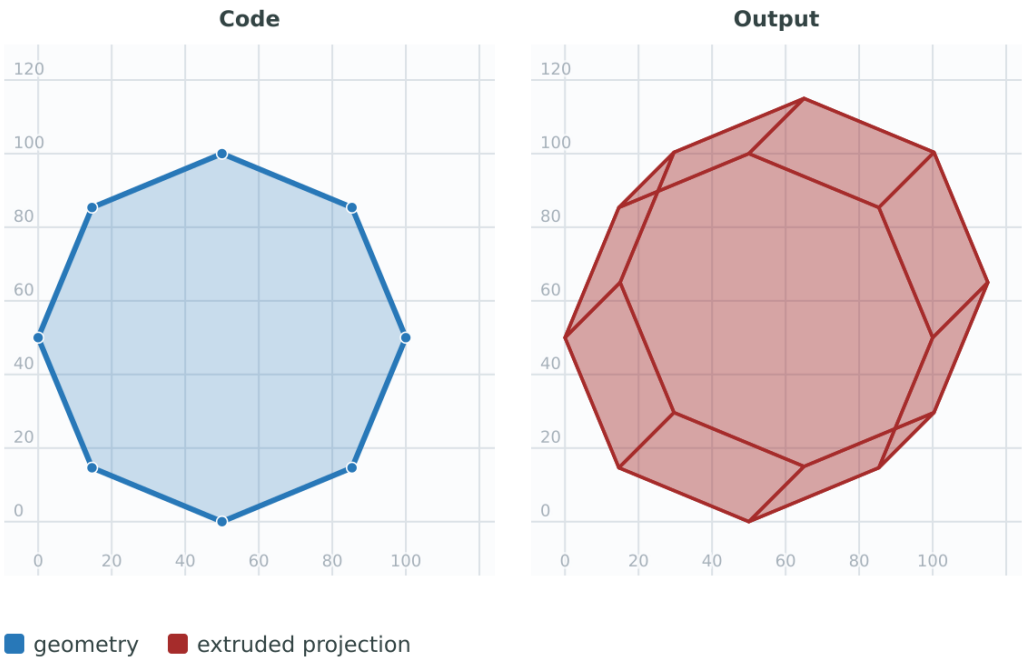
The figures use ST_Affine to project Z diagonally into XY after extrusion. The returned geometry from `CG_Extrude` remains a 3D PolyhedralSurface.

Extruding an octagonal Polygon 30 units along Z.

Code

```
WITH data AS (
  SELECT ST_Buffer('POINT(100 90)'::geometry, 50, 'quad_segs=2') AS geom
), projected AS (
  SELECT ST_Force2D(ST_Affine(geom,
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS input_geom,
         ST_Force2D(ST_Affine(CG_Extrude(geom, 0, 0, 30),
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS output_geom
  FROM data
)
SELECT input_geom AS input_geometry,
       output_geom AS extruded_projection
FROM projected;
```

Figure



Extruding a LineString 10 units along Z.

Code

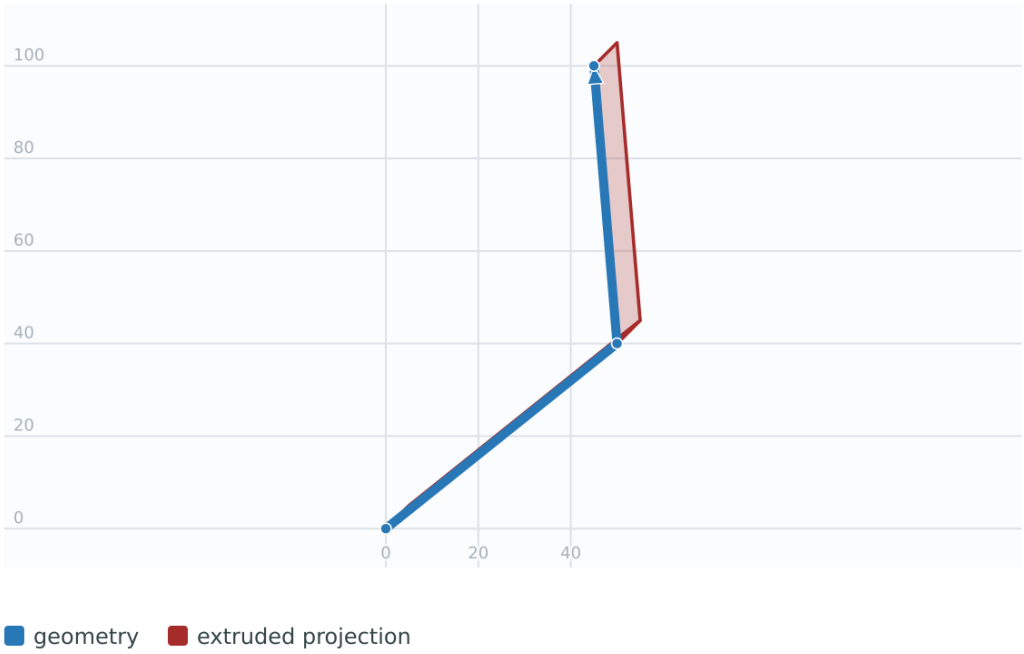
```
WITH data AS (
  SELECT 'LINESTRING(50 50,100 90,95 150)'::geometry AS geom
), projected AS (
  SELECT ST_Force2D(ST_Affine(geom,
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS input_geom,
         ST_Force2D(ST_Affine(CG_Extrude(geom, 0, 0, 10),
           1, 0, 0.5, 0, 1, 0.5, 0, 0, 0, 0, 0, 0)) AS output_geom
  FROM data
)
SELECT input_geom AS input_geometry,
       output_geom AS extruded_projection
```

```
FROM projected;
```

Output

```
LINESTRING(50 50,100 90,95 150) | POLYHEDRALSURFACE(((50 50,100 90,105 95,55 55,50 50)),((100 90,95 150,100 155,105 ↩
    95,100 90)))
```

Figure



Voir aussi

ST_AsX3D, CG_ExtrudeStraightSkeleton

8.3.23 CG_ExtrudeStraightSkeleton

CG_ExtrudeStraightSkeleton — Extrusion de squelette droit

Synopsis

geometry **CG_ExtrudeStraightSkeleton**(geometry geom, float roof_height, float body_height = 0);

Description

Calcule une extrusion avec une hauteur maximale de la géométrie du polygone.



Note
Il s'agit peut-être du premier cas d'utilisation (historique) des squelettes droits : étant donné un toit polygonal, le squelette droit donne directement la disposition de chaque tente. Si chaque arête du squelette est soulevée du plan d'une hauteur égale à sa distance de décalage, le toit résultant est "correct" en ce sens que l'eau tombera toujours vers les bords du contour (la bordure du toit), quel que soit l'endroit où elle tombe sur le toit. La fonction calcule cette extrusion ou "toit" sur un polygone. Si l'argument body_height > 0, alors le polygone est extrudé comme avec CG_Extrude(polygon, 0, 0, body_height). Le résultat est une union de ces polyedres surfaciques.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0.

Nécessite SFCGAL >= 1.5.0

Cette méthode nécessite le backend SFCGAL.

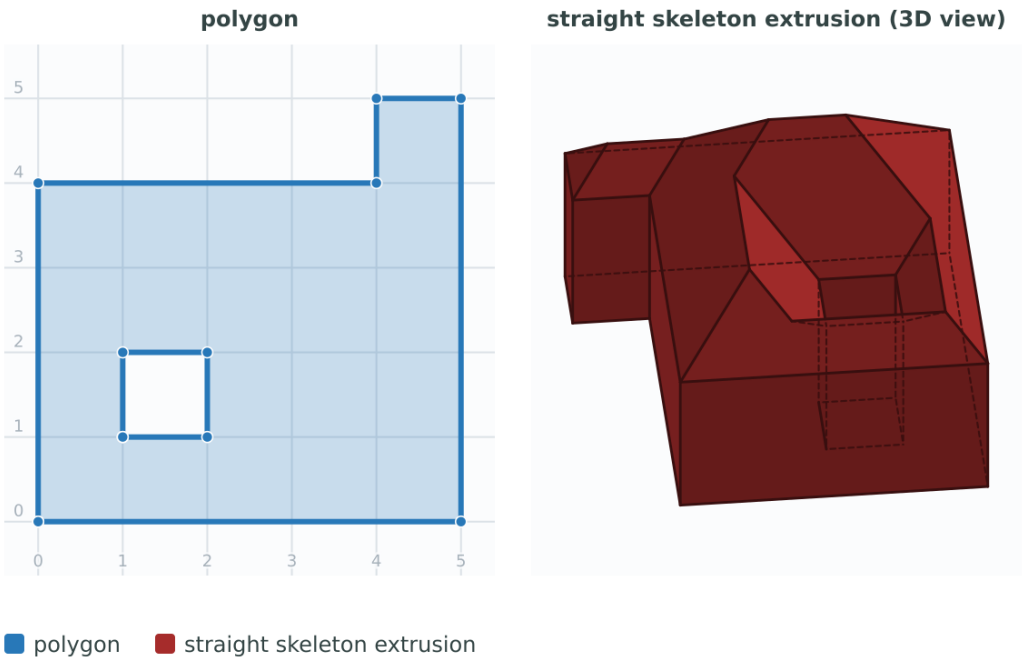
Exemples

Extrude a polygon with a hole into a body and straight-skeleton roof.

Code

```
WITH data AS (
  SELECT 'POLYGON ((0 0,5 0,5 5,4 5,4 4,0 4,0 0),(1 1,1 2,2 2,2 1,1 1))'::geometry AS ↩
    input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_ExtrudeStraightSkeleton(input_polygon, 3.0, 2.0) AS straight_skeleton_extrusion
FROM data;
```

Figure



Voir aussi

CG_Extrude, CG_ExtrudeStraightSkeleton, CG_StraightSkeleton, CG_StraightSkeletonPartition

8.3.24 CG_GreeneApproxConvexPartition

CG_GreeneApproxConvexPartition — Calcul de la partition convexe approximative de la géométrie du polygone

Synopsis

geometry **CG_GreeneApproxConvexPartition**(geometry geom);

Description

Calcule une partition convexe approximative monotone de la géométrie du polygone.



Note
Une partition d'un polygone P est un ensemble de polygones tels que les intérieurs des polygones ne se coupent pas et que l'union des polygones est égale à l'intérieur du polygone original P. Les fonctions CG_ApproxConvexPartition et CG_GreeneApproxConvexPartition produisent des partitions convexes approximativement optimales. Ces deux fonctions produisent des décompositions convexes en décomposant d'abord le polygone en polygones plus simples ; CG_ApproxConvexPartition utilise une triangulation et CG_GreeneApproxConvexPartition une partition monotone. Ces deux fonctions garantissent toutes deux qu'elles ne produiront pas plus de quatre fois le nombre optimal de pièces convexes, mais elles diffèrent par leur complexité d'exécution. Bien que l'algorithme d'approximation basé sur la triangulation produise souvent moins de pièces convexes, ce n'est pas toujours le cas.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0.

Nécessite SFCGAL >= 1.5.0

 Cette méthode nécessite le backend SFCGAL.

Exemples

Greene's monotone-based approximation uses the same polygon and stays directly comparable in one panel.

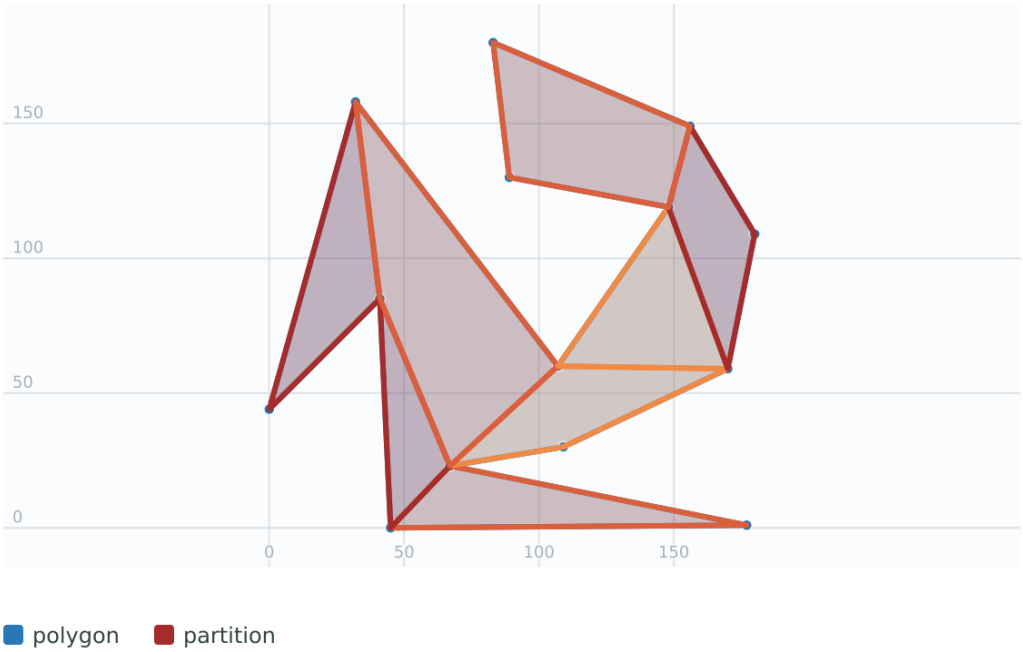
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67  ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_GreeneApproxConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)), POLYGON((45 1,177 2,67 24,45 1)), POLYGON((67 24,109 ↩
    31,170 60,107 61,67 24)), POLYGON((41 86,45 1,67 24,41 86)), POLYGON((107 61,32 159,41 86,67 24,107 61)), POLYGON ↩
    ((148 120,107 61,170 60,148 120)), POLYGON((148 120,170 60,180 110,156 150,148 120)), POLYGON((156 150,83 181,89 ↩
    131,148 120,156 150)))
```

Figure



Voir aussi

CG_YMonotonePartition, CG_ApproxConvexPartition, CG_OptimalConvexPartition

8.3.25 CG_MinkowskiSum

CG_MinkowskiSum — Effectue la somme de Minkowski

Synopsis

geometry **CG_MinkowskiSum**(geometry geom1, geometry geom2);

Description

Cette fonction effectue une somme de minkowski en 2D d'un point, d'une ligne ou d'un polygone avec un polygone.

Une somme de minkowski de deux géométries A et B est l'ensemble de tous les points qui sont la somme de n'importe quel point de A et B. Les sommes de minkowski sont souvent utilisées dans la planification des mouvements et la conception assistée par ordinateur. Plus de détails sur Wikipedia Addition de Minkowski.

Le premier paramètre peut être n'importe quelle géométrie 2D (point, ligne, polygone). Si une géométrie 3D est transmise, elle sera convertie en 2D en forçant Z à 0, ce qui peut entraîner des cas d'invalidité. Le second paramètre doit être un polygone 2D.

La mise en œuvre utilise CGAL 2D Minkowskisum.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

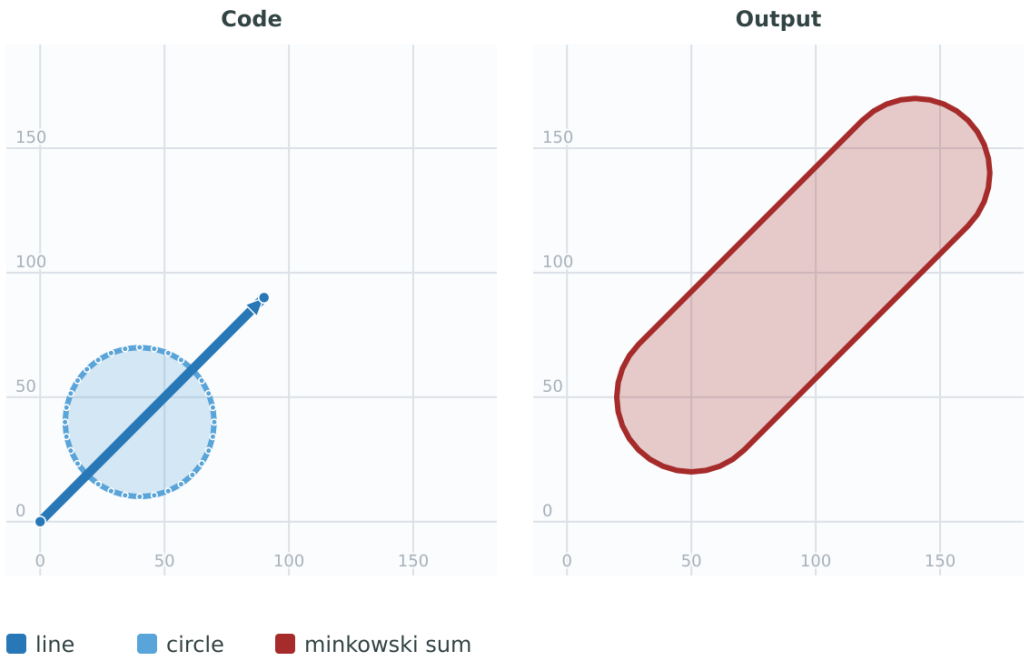
Exemples

Minkowski sum of a LineString and a circular polygon. The line crosses the polygon, and the result sweeps the polygon along the line.

Code

```
WITH data AS (
  SELECT ST_MakeLine(ST_Point(10, 10), ST_Point(100, 100)) AS line,
         ST_Buffer('POINT(50 50)'::geometry, 30) AS circle
)
SELECT line AS input_line,
       circle AS input_circle,
       CG_MinkowskiSum(line, circle) AS minkowski_sum
FROM data;
```

Figure



Minkowski sum of a MultiPoint and a polygon. The result contains one translated copy of the polygon for each point.

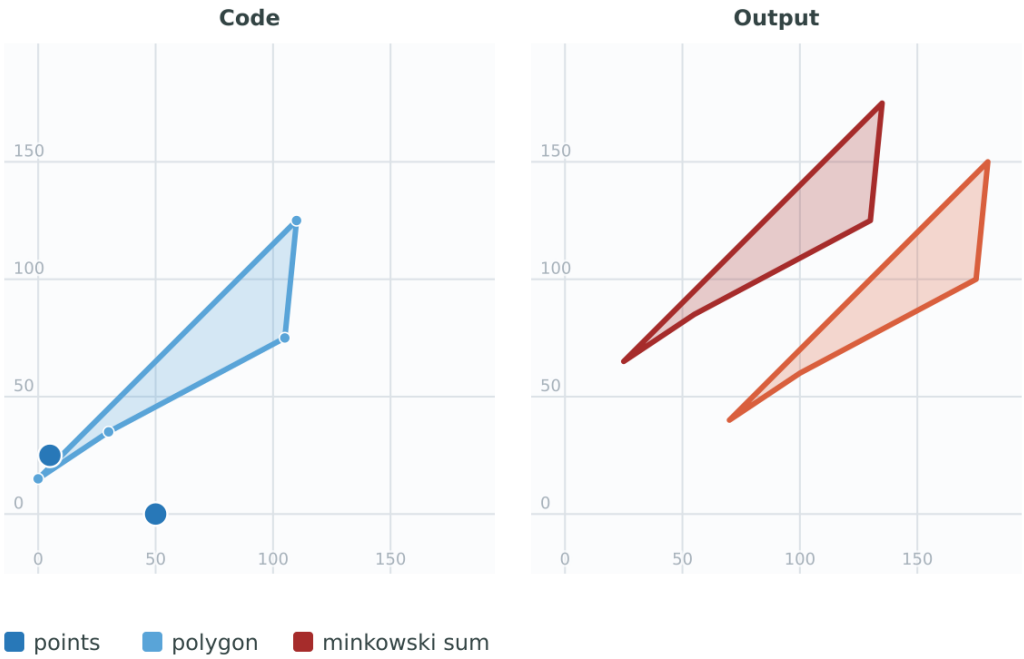
Code

```
WITH data AS (
  SELECT 'MULTIPOINT(25 50,70 25)'::geometry AS points,
         'POLYGON((130 150,20 40,50 60,125 100,130 150))'::geometry AS polygon
)
SELECT points AS input_points,
       polygon AS input_polygon,
       CG_MinkowskiSum(points, polygon) AS minkowski_sum
FROM data;
```

Output

```
MULTIPOINT((25 50),(70 25)) | POLYGON((130 150,20 40,50 60,125 100,130 150)) | MULTIPOLYGON(((45 90,75 110,150 150,155 ↩
    200,45 90)),((90 65,120 85,195 125,200 175,90 65)))
```

Figure



8.3.26 CG_OptimalAlphaShape

CG_OptimalAlphaShape — Calcule une forme alpha entourant une géométrie en utilisant une valeur alpha "optimale".

Synopsis

geometry **CG_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

Description

Calcule la forme alpha "optimale" des points d'une géométrie. La forme alpha est calculée en utilisant une valeur de α choisie de telle sorte que :

1. le nombre d'éléments du polygone est égal ou inférieur à `nb_components` (qui vaut 1 par défaut)
2. tous les points d'entrée sont contenus dans la forme

Le résultat ne contiendra pas de trous à moins que l'argument facultatif `allow_holes` ne soit spécifié comme true.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.4.1.

Cette méthode nécessite le backend SFCGAL.

Exemples

The original point cloud and the optimal alpha-shape share one comparison frame.

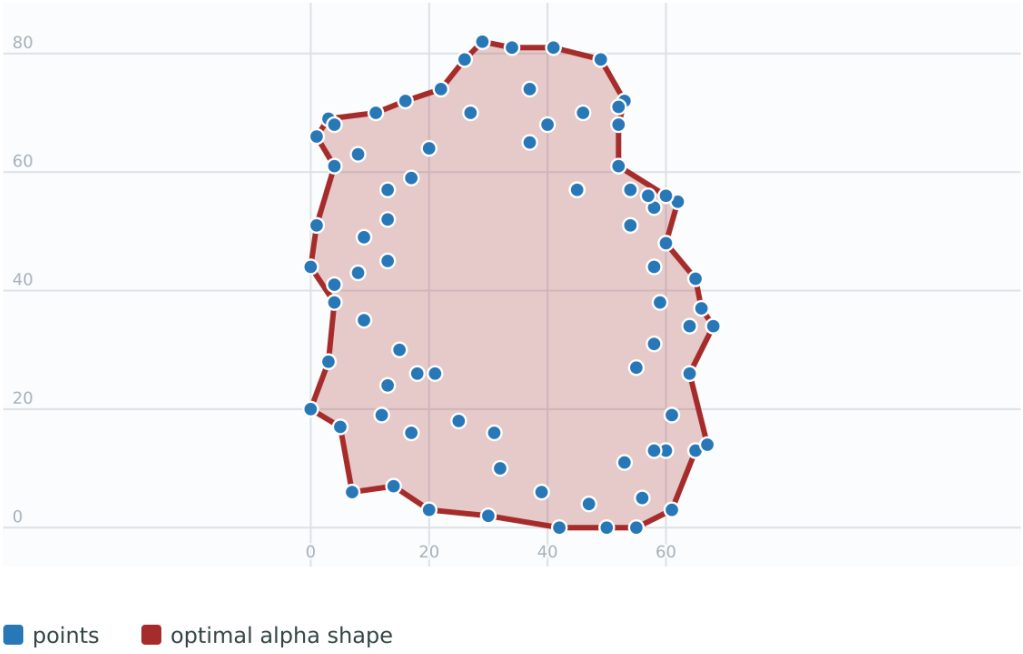
Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 ↩
    65),
(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
```

```
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52 ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31 ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_OptimalAlphaShape(input_points) AS optimal_alpha_shape
FROM data;
```

Figure



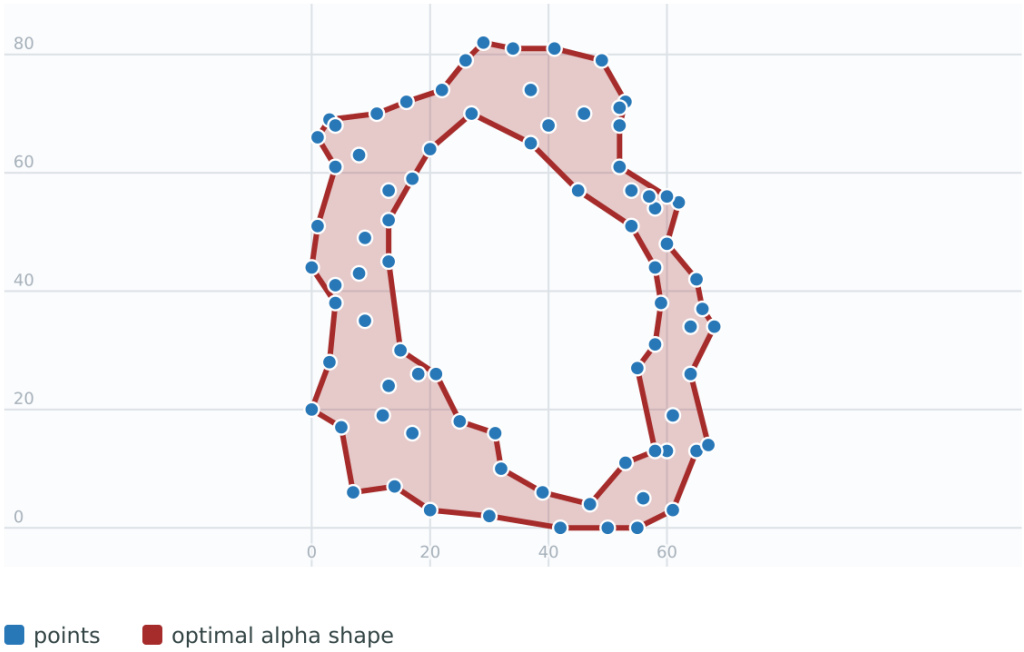
Allowing holes keeps the same outer shell but carves a central hole.

Code

```
WITH data AS (
  SELECT 'MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),(88 29),(24 82),(32 51) ↩
    ,(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61),(32 65),(81 47),(88 58) ↩
    ,(68 73),(49 95),(81 60),(87 50),
(78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 29),(27 84),(52 ↩
    98),(72 95),(85 71),
(75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 97),(27 77),(39 ↩
    88),(60 81),
(80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 64),(69 86),(60 ↩
    90),(50 86),(43 80),(36 73),
(36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 16),(38 46),(31 ↩
    59),(34 86),(45 90),(64 97))'::geometry AS input_points
)
SELECT input_points AS input_points,

         CG_OptimalAlphaShape(input_points, allow_holes => true) AS optimal_alpha_shape
FROM data;
```

Figure



Voir aussi

ST_ConcaveHull, CG_AlphaShape

8.3.27 CG_OptimalConvexPartition

CG_OptimalConvexPartition — Calcul d'une partition convexe optimale de la géométrie du polygone

Synopsis

geometry **CG_OptimalConvexPartition**(geometry geom);

Description

Calcul d'une partition convexe optimale de la géométrie du polygone.



Note
Une partition d'un polygone P est un ensemble de polygones tels que les intérieurs des polygones ne se croisent pas et que l'union des polygones est égale à l'intérieur du polygone original P. CG_OptimalConvexPartition produit une partition optimale en termes de nombre de morceaux.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0.

Nécessite SFCGAL >= 1.5.0


Cette méthode nécessite le backend SFCGAL.

Exemples

The optimal convex partition uses the same polygon and produces the smallest documented piece count.

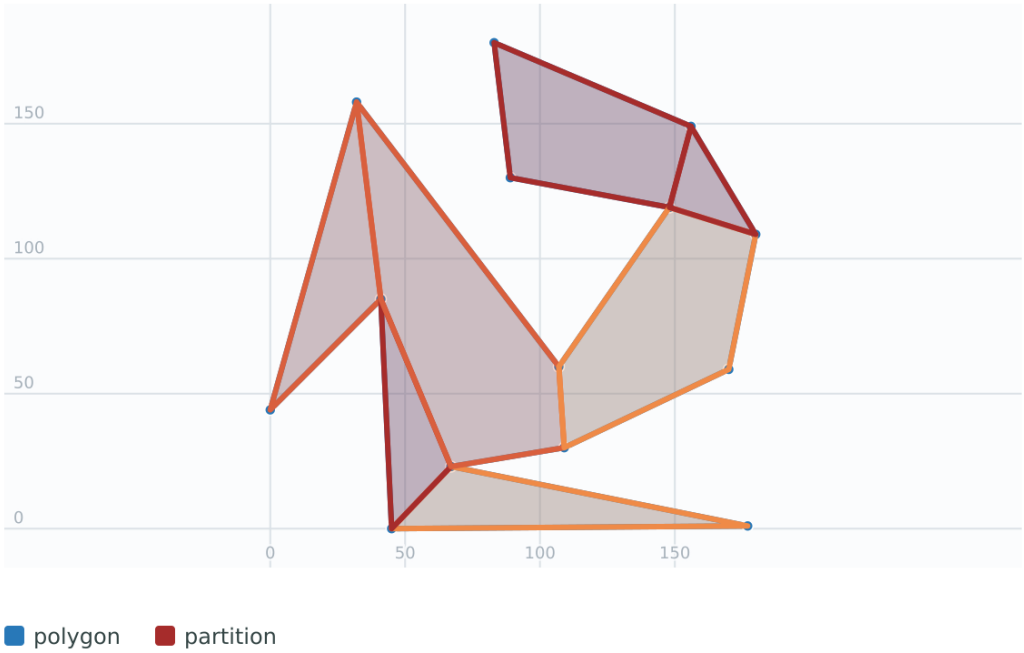
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_OptimalConvexPartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((156 150,83 181,89 131,148 120,156 150)), POLYGON((32 159,0 45,41 86,32 159)), POLYGON ↩
    ((45 1,177 2,67 24,45 1)), POLYGON((41 86,45 1,67 24,41 86)), POLYGON((107 61,32 159,41 86,67 24,109 31,107 61)), ↩
    POLYGON((148 120,107 61,109 31,170 60,180 110,148 120)), POLYGON((156 150,148 120,180 110,156 150)))
```

Figure



Voir aussi

CG_YMonotonePartition, CG_ApproxConvexPartition, CG_GreeneApproxConvexPartition

8.3.28 CG_StraightSkeleton

CG_StraightSkeleton — Calcule un squelette (straight skeleton) à partir d'une géométrie

Synopsis

geometry **CG_StraightSkeleton**(geometry geom, boolean use_distance_as_m = false);

Description

Computes the straight skeleton of a polygon by tracing the inward propagation of its edges. When *use_distance_as_m* is `true`, the M coordinate of each output point stores its distance from the input boundary.

Disponibilité : 3.5.0

Nécessite SFCGAL >= 1.3.8 pour l'option use_distance_as_m



Note
Cette fonction ignore la dimension Z. Elle donne toujours un résultat en 2D, même lorsqu'elle est utilisée sur une géométrie en 3D.

 Cette méthode nécessite le backend SFCGAL.

 Cette fonction prend en charge les surfaces Polyhedral.

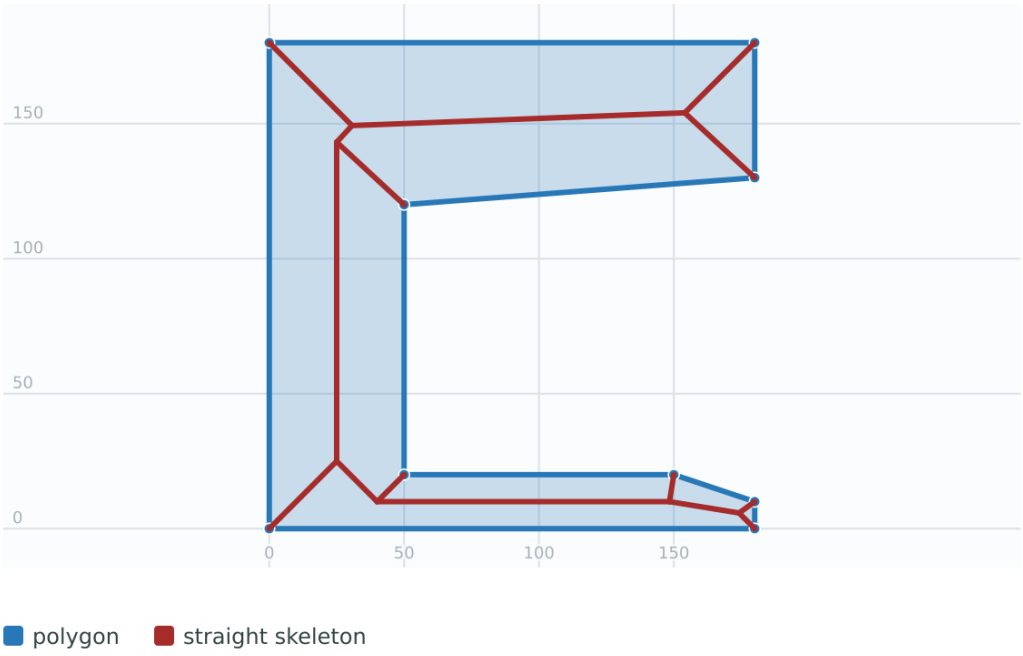
 Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' ↩
    ::geometry AS polygon
)
SELECT CG_StraightSkeleton(polygon) AS straight_skeleton
FROM data;
```

Figure



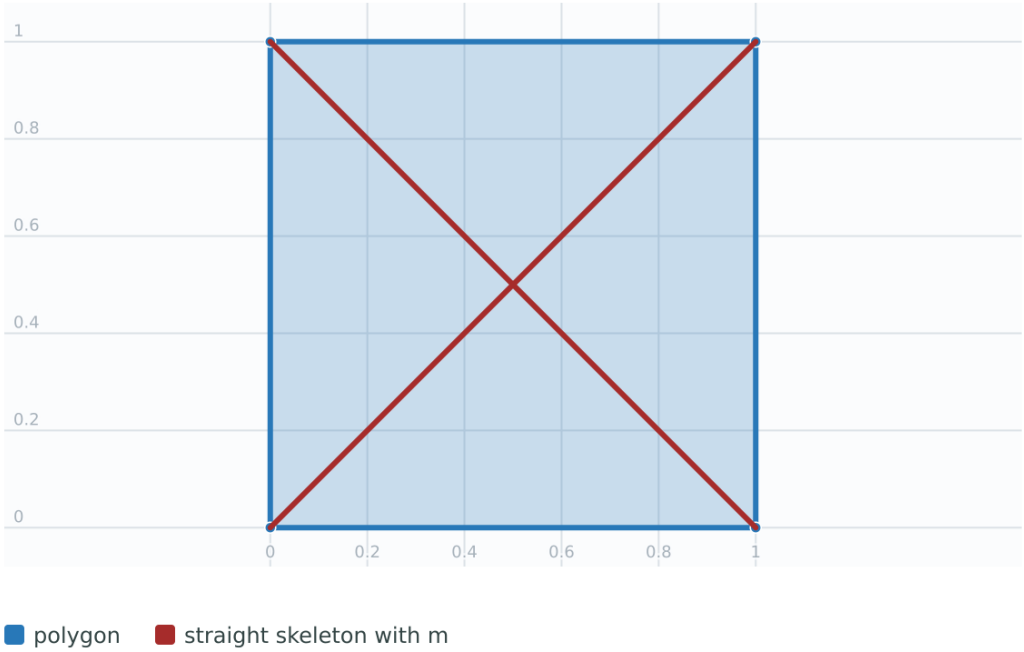
Code

```
WITH data AS (
  SELECT 'POLYGON((0 0,1 0,1 1,0 1,0 0))'::geometry AS polygon
)
SELECT CG_StraightSkeleton(polygon, true) AS straight_skeleton_with_m
FROM data;
```

Output

```
MULTILINESTRING M ((0 0 0,0.5 0.5 0.5),(1 0 0,0.5 0.5 0.5),(1 1 0,0.5 0.5 0.5),(0 1 0,0.5 0.5 0.5))
```

Figure



Note that valid inputs with rings that touch at a single point will raise an error.

Code

```
SELECT CG_StraightSkeleton('POLYGON((0 0,3 0,3 3,0 3,0 0),(0 0,1 2,2 1,0 0))'));
```

Output

```
NOTICE:  During straight_skeleton(A) :
NOTICE:    with A: POLYGON((0/1 0/1,3/1 0/1,3/1 3/1,0/1 3/1,0/1 0/1),(0/1 0/1,1/1 2/1,2/1 1/1,0/1 0/1))
ERROR:  straight skeleton of Polygon with point touching rings is not implemented.
```

Voir aussi

CG_StraightSkeletonPartition, CG_ExtrudeStraightSkeleton

8.3.29 CG_Tesselate

CG_Tesselate — Effectue la tessellation de la surface d'un polygone ou d'une surface polyédrique et renvoie un TIN ou une collection de TINS

Synopsis

geometry **CG_Tesselate**(geometry geom);

Description

Prend en entrée une surface telle que MULTI(POLYGON) ou POLYHEDRALSURFACE et renvoie une représentation TIN via le processus de tessellation à l'aide de triangles.



Note
ST_TriangulatePolygon est similaire à cette fonction, sauf qu'elle renvoie une collection géométrique de polygones au lieu d'un TIN et qu'elle ne fonctionne qu'avec des géométries 2D.

Disponibilité : 3.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

Exemples

Tessellation divides each face of a cube into triangles.

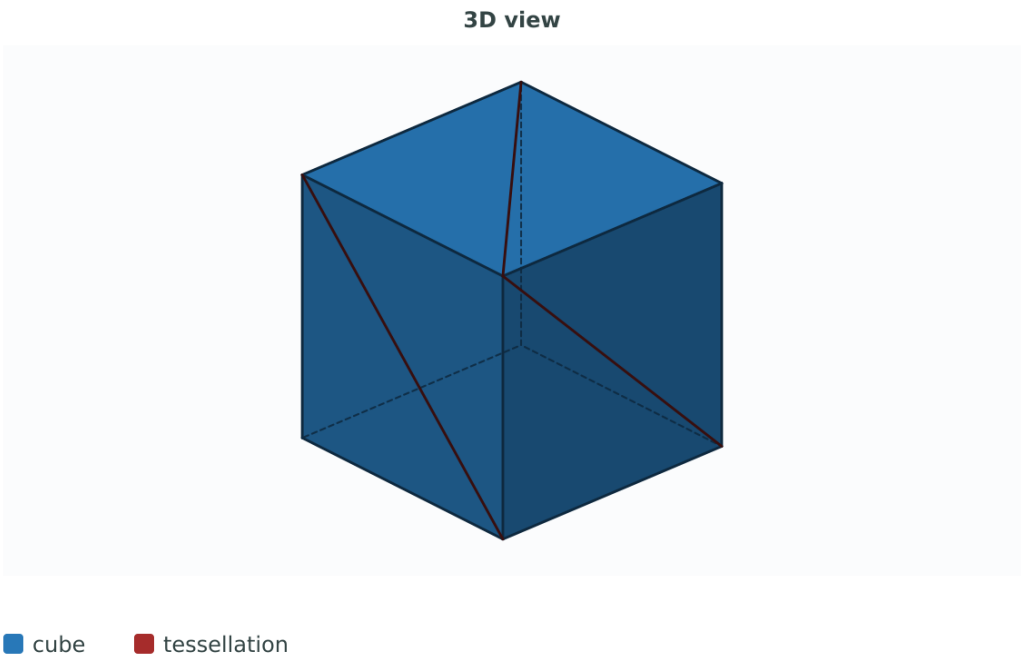
Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z( ((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),
((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),
((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),
((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)) )'::geometry AS cube
)
SELECT CG_Tesselate(cube) AS tessellation
FROM data;
```

Output

```
TIN Z (((0 0 0,0 0 1,0 1 1,0 0 0)),((0 1 0,0 0 0,0 1 1,0 1 0)),((0 0 0,0 1 0,1 1 0,0 0 0)),((1 0 0,0 0 0,1 1 0,1 0 0)) ↩
    ,((0 0 1,1 0 0,1 0 1,0 0 1)),((0 0 1,0 0 0,1 0 0,0 0 1)),((1 1 0,1 1 1,1 0 1,1 1 0)),((1 0 0,1 1 0,1 0 1,1 0 0)) ↩
    ,((0 1 0,0 1 1,1 1 1,0 1 0)),((1 1 0,0 1 0,1 1 1,1 1 0)),((0 1 1,1 0 1,1 1 1,0 1 1)),((0 1 1,0 0 1,1 0 1,0 1 1)))
```

Figure



Tessellation of a Polygon.

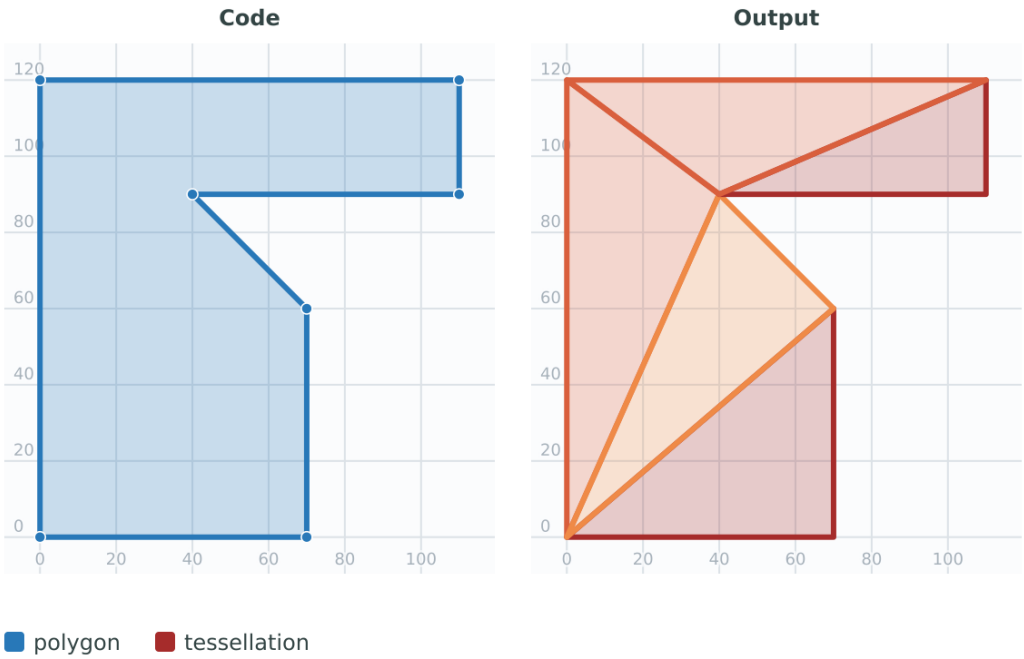
Code

```
WITH data AS (
  SELECT 'POLYGON((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190))'::geometry AS
    polygon
)
SELECT polygon AS input_polygon,
       CG_Tesselate(polygon) AS tessellation
FROM data;
```

Output

```
POLYGON((10 190,10 70,80 70,80 130,50 160,120 160,120 190,10 190)) | TIN(((80 130,10 70,80 70,80 130)),((50 160,10
    190,10 70,50 160)),((80 130,50 160,10 70,80 130)),((120 160,120 190,50 160,120 160)),((120 190,10 190,50 160,120
    190)))
```

Figure



Voir aussi

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.30 CG_Triangulate

CG_Triangulate — Triangule une géométrie polygonale

Synopsis

geometry **CG_Triangulate**(geometry geom);

Description

Triangule une géométrie polygonale.

Réalisé par le module SFCGAL



Note
NOTE : cette fonction renvoie une géométrie représentant le résultat de la triangulation.

Disponibilité : 3.5.0


Cette méthode nécessite le backend SFCGAL.

Exemples de géométrie

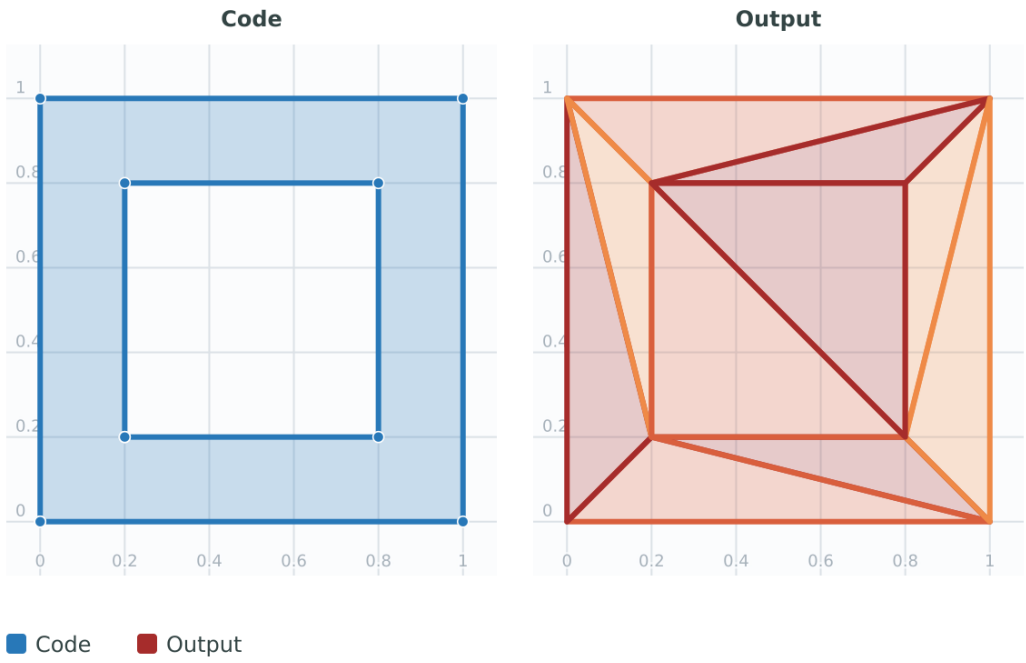
Code

```
SELECT CG_Triangulate(
  'POLYGON((0 0,1 0,1 1,0 1,0 0),
           (0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))'
);
```

Output

```
TIN(((0.8 0.2,0.2 0.2,1 0,0.8 0.2)),((0.2 0.2,0 0,1 0,0.2 0.2)),((1 1,0.8 0.8,0.8 0.2,1 1)),((0 1,0 0,0.2 0.2,0 1)),((0 ↩
     1,0.2 0.8,1 1,0 1)),((0 1,0.2 0.2,0.2 0.8,0 1)),((0.2 0.8,0.8 0.8,1 1,0.2 0.8)),((0.2 0.8,0.2 0.2,0.8 0.2,0.2  ↩
    0.8)),((1 1,0.8 0.2,1 0,1 1)),((0.8 0.8,0.2 0.8,0.8 0.2,0.8 0.8)))
```

Figure



Voir aussi

CG_ConstrainedDelaunayTriangles, ST_DelaunayTriangles, ST_TriangulatePolygon

8.3.31 CG_Visibility

CG_Visibility — Calculer un polygone de visibilité à partir d'un point ou d'un segment dans une géométrie polygonale

Synopsis

geometry **CG_Visibility**(geometry polygon, geometry point);
geometry **CG_Visibility**(geometry polygon, geometry pointA, geometry pointB);

Description

Returns the portion of a polygon visible from an observer point, or from the observer segment defined by *pointA* and *pointB*.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0.

Nécessite SFCGAL >= 1.5.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les surfaces Polyhedral.

Cette fonction prend en charge les triangles et les réseaux irréguliers triangulés (TIN).

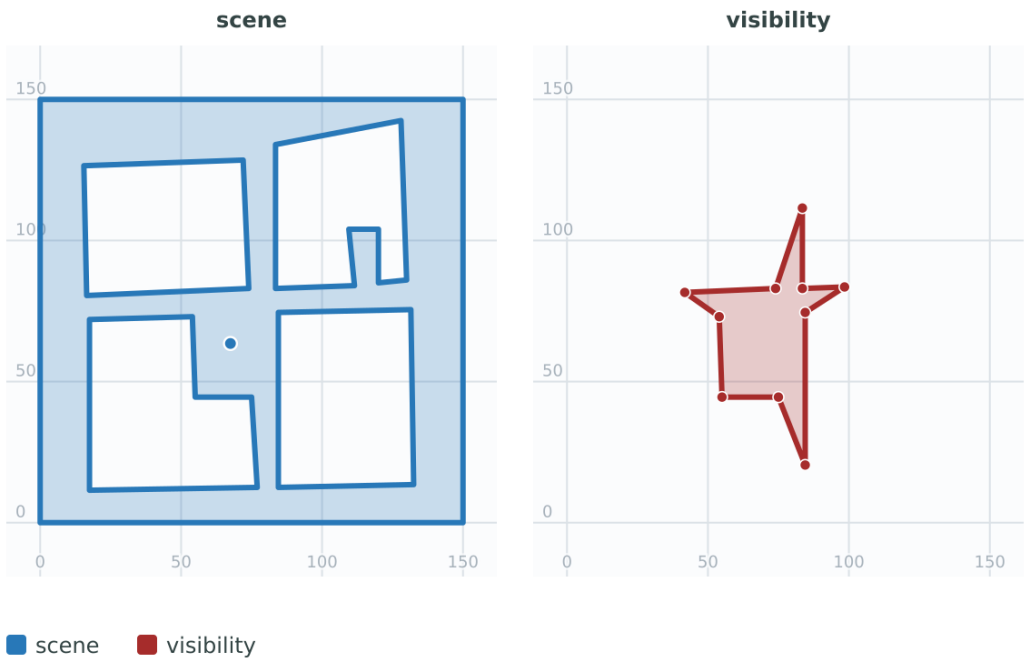
Exemples

Visibility from a point inside a polygon with holes.

Code

```
WITH data AS (
  SELECT 'POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 ↩
    37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 ↩
   108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 ↩
   96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry AS polygon,
        'POINT(91 87)'::geometry AS observer
)
SELECT ST_Collect(polygon, observer) AS input_scene,
       CG_Visibility(polygon, observer) AS visibility
FROM data;
```

Figure



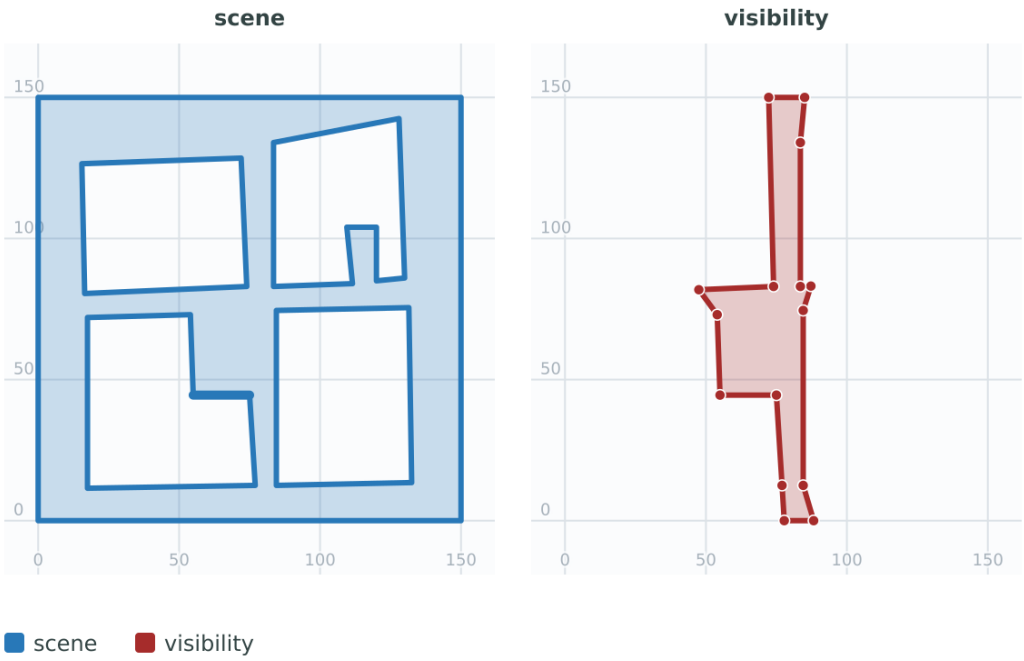
Visibility from a segment inside the same polygon.

Code

```
WITH data AS (
  SELECT 'POLYGON((23.5 23.5,23.5 173.5,173.5 173.5,173.5 23.5,23.5 23.5),(108 98,108 36,156 ↩
    37,155 99,108 98),(107 157.5,107 106.5,135 107.5,133 127.5,143.5 127.5,143.5 ↩
   108.5,153.5 109.5,151.5 166,107 157.5),(41 95.5,41 35,100.5 36,98.5 68,78.5 68,77.5 ↩
   96.5,41 95.5),(39 150,40 104,97.5 106.5,95.5 152,39 150))'::geometry AS polygon,
```

```
        'LINESTRING(78.5 68,98.5 68)'::geometry AS observer
)
SELECT ST_Collect(polygon, observer) AS input_scene,
       CG_Visibility(polygon,
                     ST_StartPoint(observer),
                     ST_EndPoint(observer)) AS visibility
FROM data;
```

Figure



8.3.32 CG_YMonotonePartition

CG_YMonotonePartition — Calcul de la partition y-monotone de la géométrie du polygone

Synopsis

geometry **CG_YMonotonePartition**(geometry geom);

Description

Calcul de la partition y-monotone de la géométrie du polygone.

> **Note**
> Une partition d'un polygone P est un ensemble de polygones tels que les intérieurs des polygones ne se croisent pas et que l'union des polygones est égale à l'intérieur du polygone original P. Un polygone y-monotone est un polygone dont les sommets v1,...,vn peuvent être divisés en deux chaînes v1,...,vk et vk,...,vn,v1, de sorte que toute ligne horizontale croise l'une ou l'autre chaîne au plus une fois. Cet algorithme ne garantit pas de limite au nombre de polygones produits par rapport au nombre optimal.

Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0.

Nécessite SFCGAL >= 1.5.0

Cette méthode nécessite le backend SFCGAL.

Exemples

The same polygon can also be partitioned into y-monotone pieces for downstream processing.

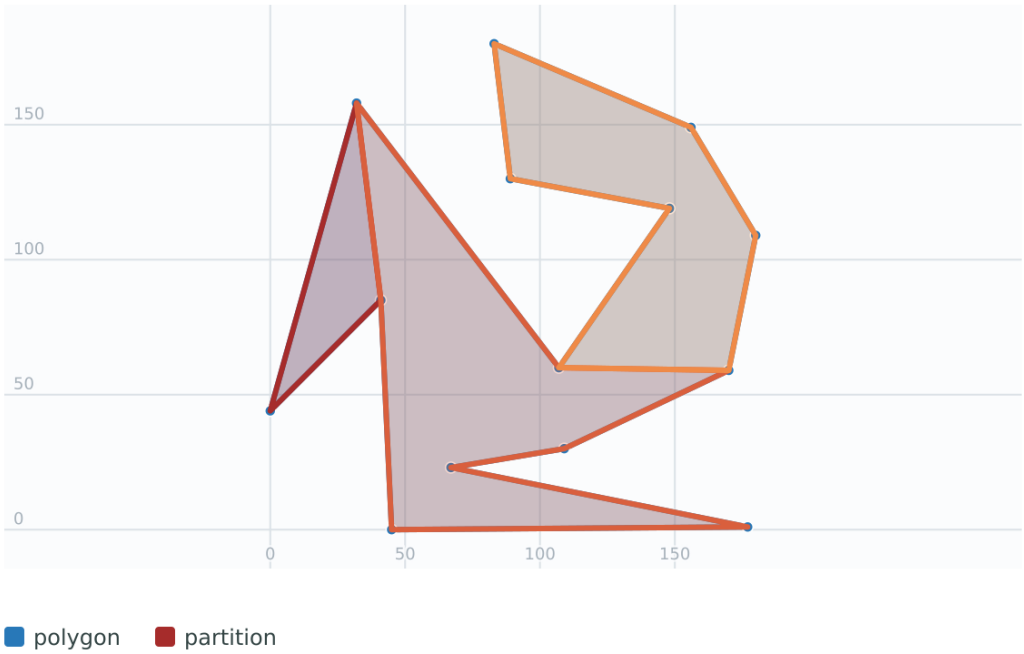
Code

```
WITH data AS (
  SELECT 'POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 ↩
    24,109 31,170 60,180 110,156 150))'::geometry AS input_polygon
)
SELECT input_polygon AS input_polygon,
       CG_YMonotonePartition(input_polygon) AS partition
FROM data;
```

Output

```
POLYGON((156 150,83 181,89 131,148 120,107 61,32 159,0 45,41 86,45 1,177 2,67 24,109 31,170 60,180 110,156 150)) | ↩
    GEOMETRYCOLLECTION(POLYGON((32 159,0 45,41 86,32 159)), POLYGON((107 61,32 159,41 86,45 1,177 2,67 24,109 31,170 ↩
    60,107 61)), POLYGON((156 150,83 181,89 131,148 120,107 61,170 60,180 110,156 150)))
```

Figure



Voir aussi

CG_ApproxConvexPartition, CG_GreeneApproxConvexPartition, CG_OptimalConvexPartition

8.3.33 CG_StraightSkeletonPartition

CG_StraightSkeletonPartition — Calcule la partition du squelette droit d'un polygone.

Synopsis

geometry **CG_StraightSkeletonPartition**(geometry geom, boolean auto_orientation);

Description

Calcule la partition du squelette droit du polygone d'entrée geometry *geom*. Le squelette droit est une partition du polygone en faces formées en traçant l'effondrement de ses arêtes. Si *auto_orientation* vaut true, la fonction ajustera automatiquement l'orientation du polygone d'entrée pour garantir des résultats corrects.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0.

Cette méthode nécessite le backend SFCGAL.

Exemples

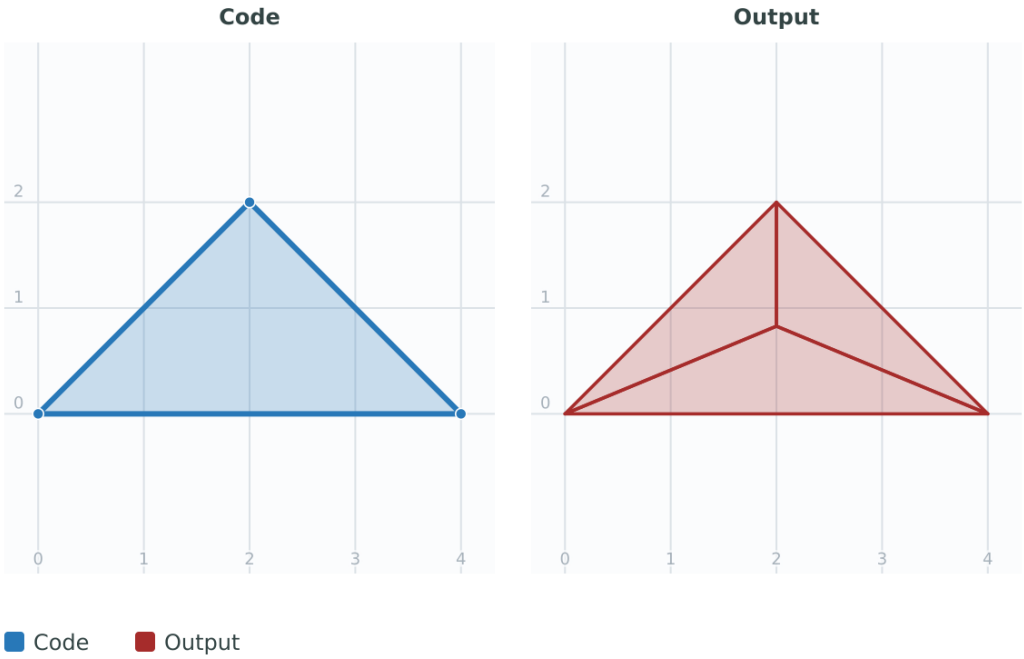
Code

```
SELECT CG_StraightSkeletonPartition('POLYGON((0 0,4 0,2 2,0 0))', true);
```

Output

```
POLYHEDRALSURFACE(((0 0,2 0.82842712474619,2 2,0 0)),((4 0,2 0.82842712474619,0 0,4 0)),((2 2,2 0.82842712474619,4 0,2 ↩
    2)))
```

Figure

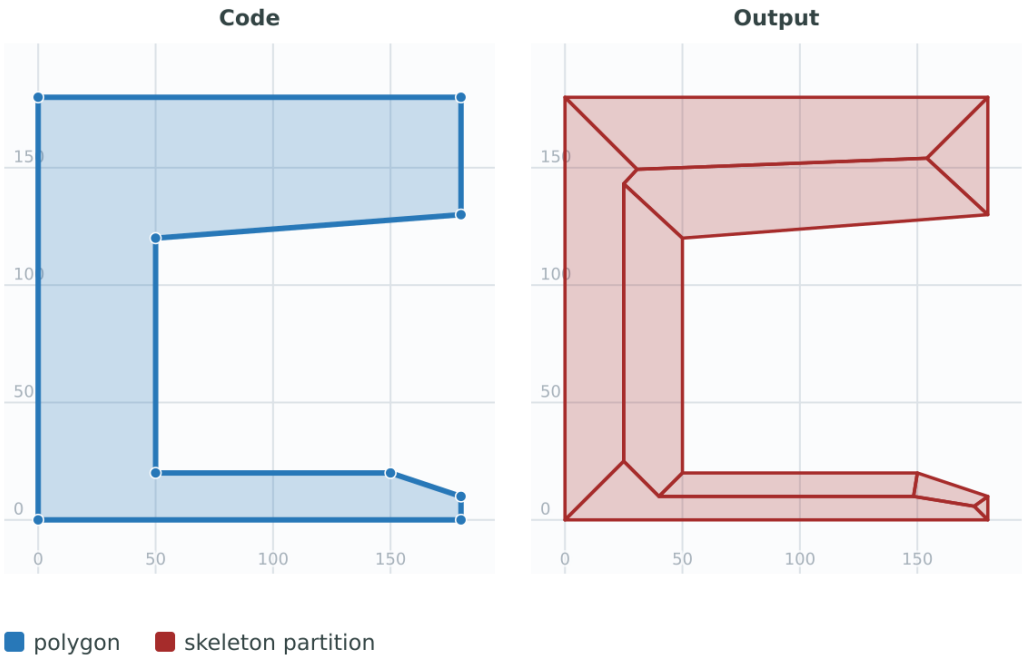


Straight skeleton partition of a polygon.

Code

```
WITH data AS (
  SELECT 'POLYGON((190 190,10 190,10 10,190 10,190 20,160 30,60 30,60 130,190 140,190 190))' ↩
    ::geometry AS polygon
)
SELECT polygon AS input_polygon,
       CG_StraightSkeletonPartition(polygon, true) AS skeleton_partition
FROM data;
```

Figure



Voir aussi

CG_StraightSkeleton, ST_Polygonize

8.3.34 CG_3DBuffer

CG_3DBuffer — Calcule un tampon 3D autour d'une géométrie.

Synopsis

geometry **CG_3DBuffer**(geometry geom, float8 radius, integer segments, integer buffer_type);

Description

Generates a 3D buffer around the input geometry *`geom`* with a specified *`radius`*. The buffer is constructed in 3D space, creating a volumetric representation of the geometry's surroundings. The *`segments`* parameter defines the number of segments used to approximate the curved sections of the buffer, with a minimum value of 4 segments required. The *`buffer_type`* specifies the type of buffer to create: 0: Rounded buffer (default) 1: Flat buffer 2: Square buffer

Input geometry must be a Point or LineString.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Exemples

Code

```
SELECT CG_3DBuffer('POINT(0 0 0)', 1, 8, 0);
```

Output

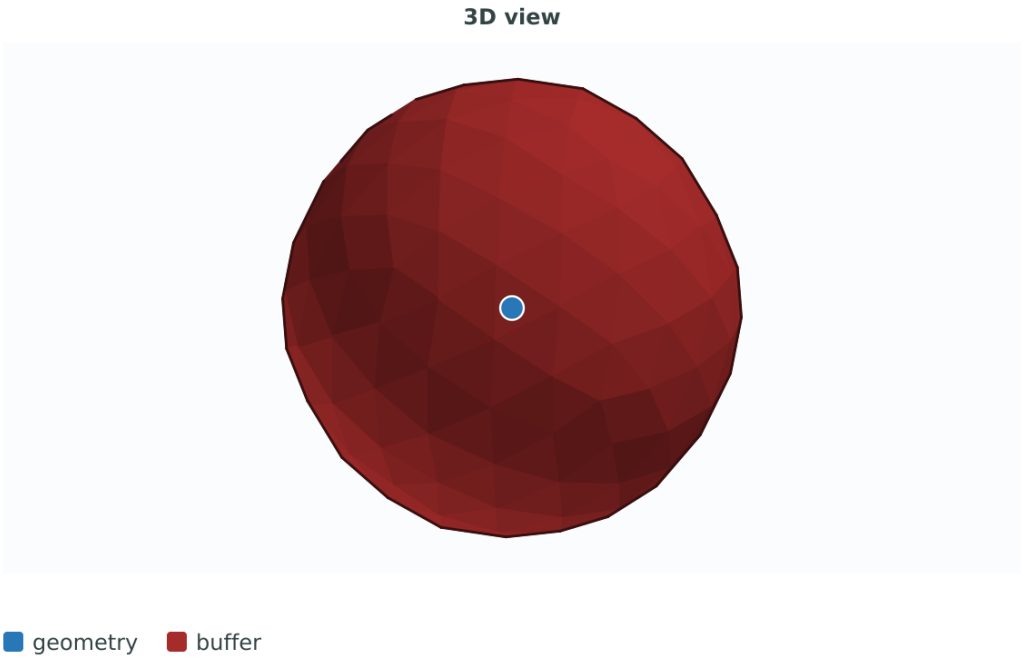
```
POLYHEDRALSURFACE Z (((0 0 1, 0.5 -0.5 0.71, 0 -0.71 0.71, 0 0 1)), ... )
```

A rounded point buffer using 32 segments.

Code

```
SELECT
  'POINT(100 90)'::geometry AS input_geometry,
  CG_3DBuffer('POINT(100 90)'::geometry, 50, 32, 0) AS buffer;
```

Figure

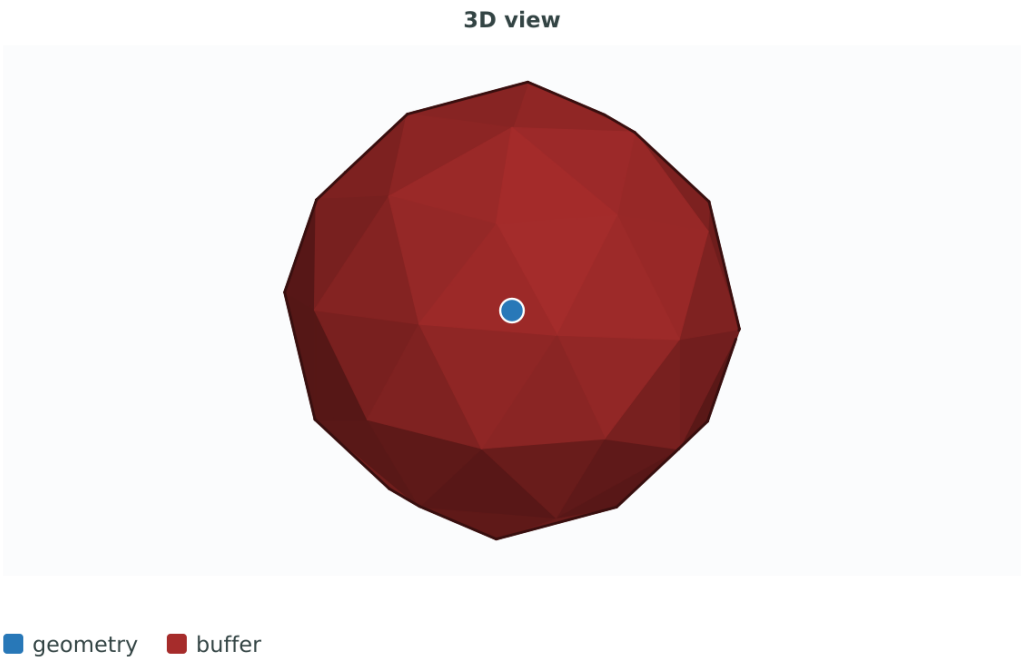


A coarser rounded point buffer using 5 segments.

Code

```
SELECT
  'POINT(100 90)'::geometry AS input_geometry,
  CG_3DBuffer('POINT(100 90)'::geometry, 50, 5, 0) AS buffer;
```

Figure

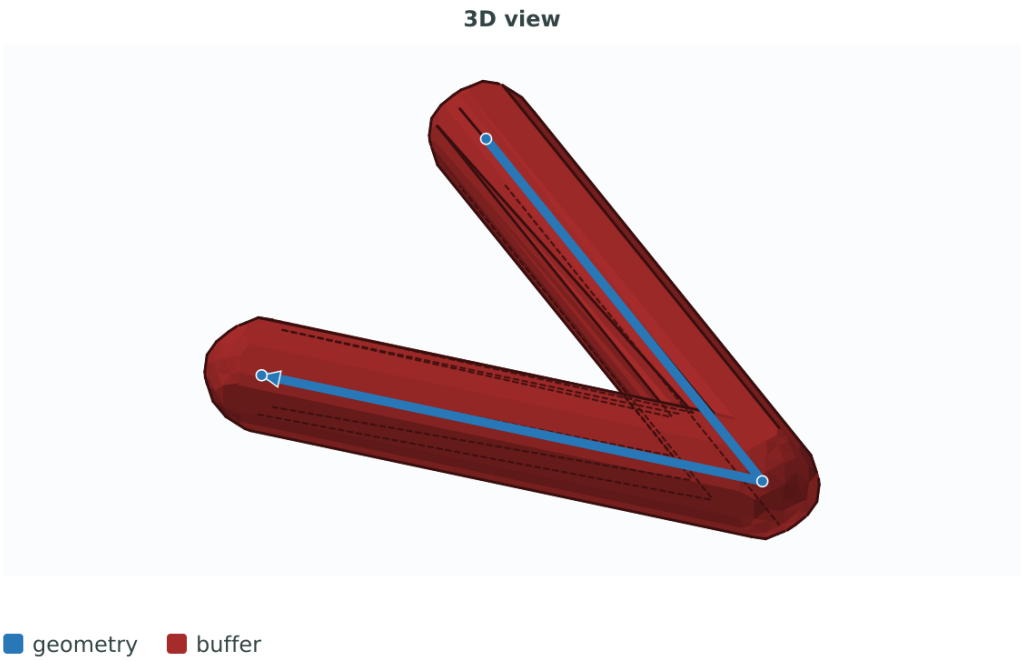


A rounded buffer around a line using 32 segments.

Code

```
SELECT
  'LINESTRING(50 50,150 150,150 50)'::geometry AS input_geometry,
  CG_3DBuffer('LINESTRING(50 50,150 150,150 50)'::geometry, 10, 32, 0) AS buffer;
```

Figure

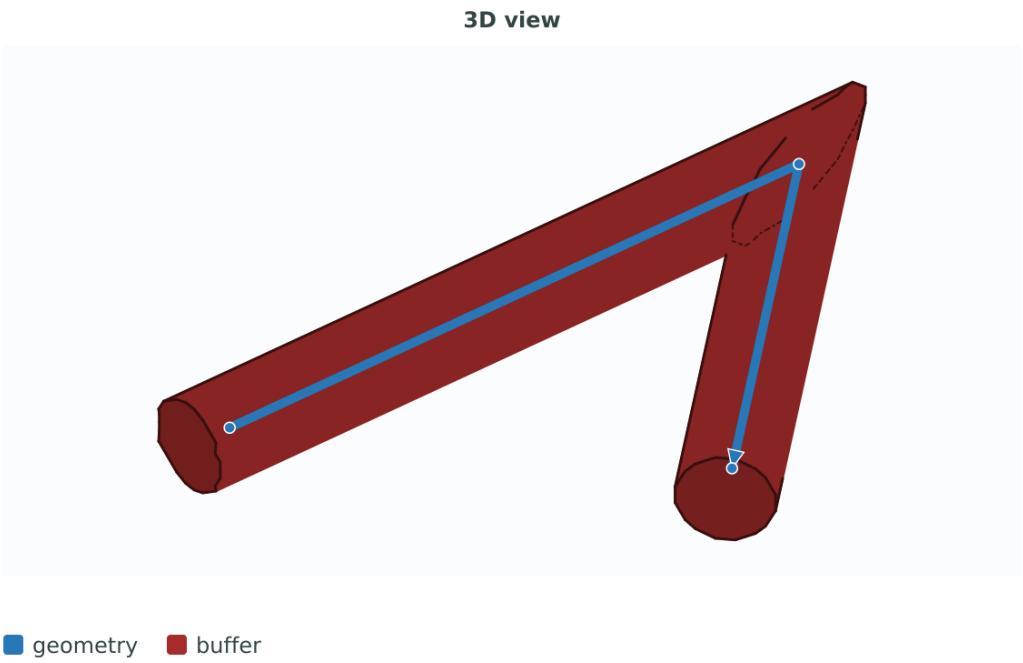


A square buffer around the same line using 32 segments.

Code

```
SELECT
  'LINESTRING(50 50,150 150,150 50)'::geometry AS input_geometry,
  CG_3DBuffer('LINESTRING(50 50,150 150,150 50)'::geometry, 10, 32, 2) AS buffer;
```

Figure



Voir aussi

ST_Buffer, CG_3DConvexHull, ST_AsX3D

8.3.35 CG_Rotate

CG_Rotate — Fait pivoter une géométrie d'un angle donné autour de l'origine (0,0).

Synopsis

geometry **CG_Rotate**(geometry geom, float8 angle);

Description

Fait pivoter la géométrie d'entrée *geom* de *angle* radians autour du point d'origine (0,0). La rotation est effectuée dans l'espace 2D ; les coordonnées Z ne sont pas modifiées. Les angles positifs font tourner la géométrie dans le sens inverse des aiguilles d'une montre.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Exemples

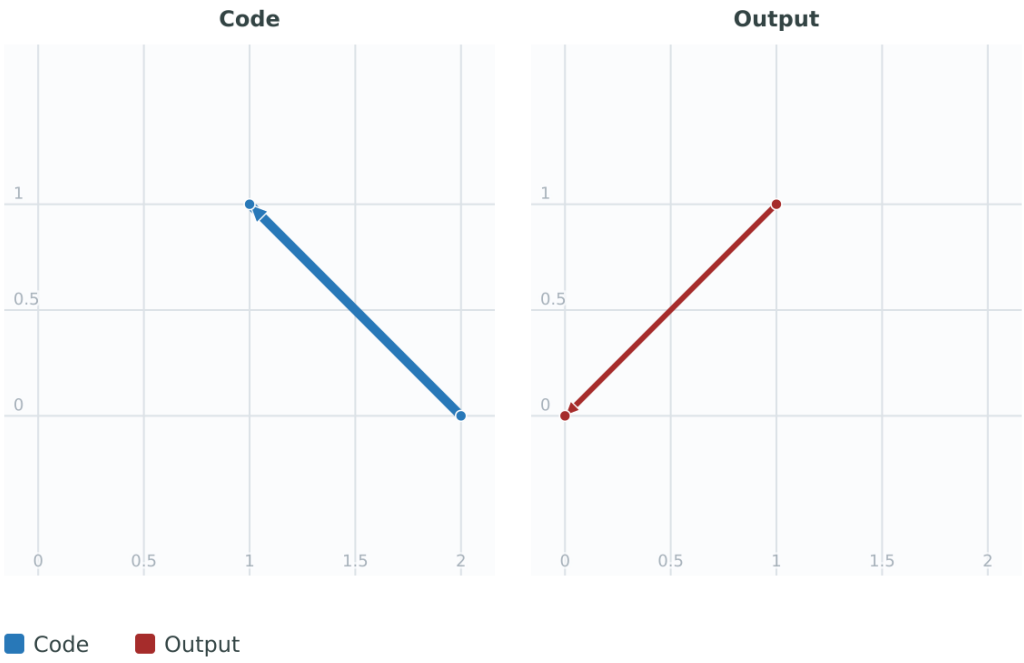
Code

```
SELECT CG_Rotate('LINESTRING(1 0,0 1)', pi()/2);
```

Output

```
LINESTRING(6.123233995736766e-17 1,-1 6.123233995736766e-17)
```

Figure



Voir aussi

CG_2DRotate, ST_Rotate

8.3.36 CG_2DRotate

CG_2DRotate — Fait pivoter une géométrie d'un angle donné autour d'un point spécifié en 2D.

Synopsis

geometry **CG_2DRotate**(geometry geom, float8 angle, float8 cx, float8 cy);

Description

Fait pivoter la géométrie d'entrée *geom* de *angle* radians autour du point (*cx*, *cy*). La rotation est effectuée dans l'espace 2D ; les coordonnées Z ne sont pas prises en compte. Les angles positifs font tourner la géométrie dans le sens inverse des aiguilles d'une montre.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Exemples

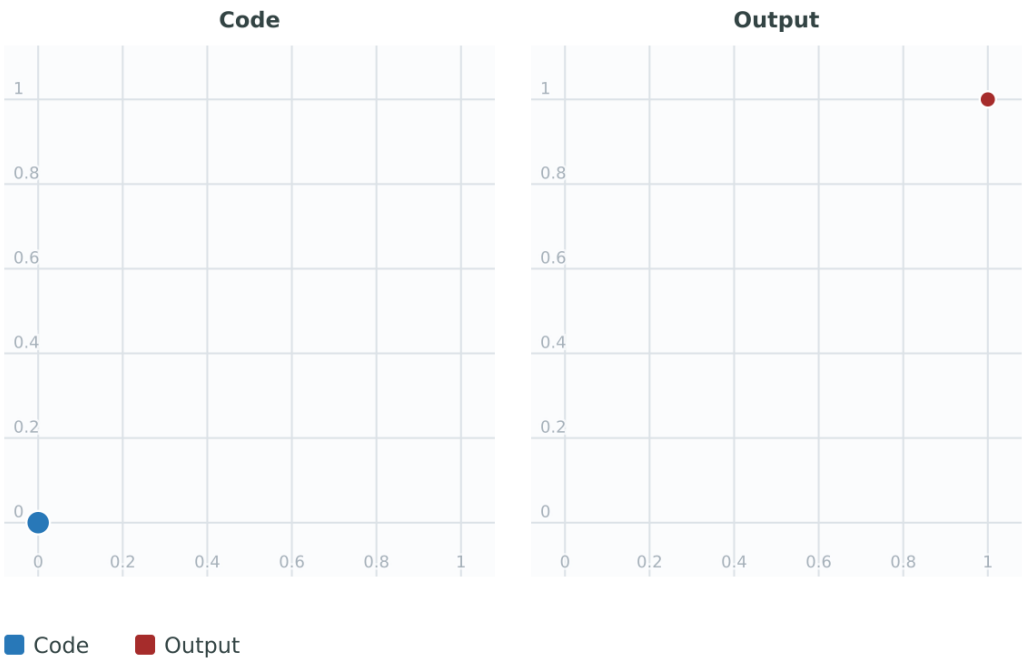
Code

```
SELECT CG_2DRotate('POINT(1 0)', pi()/2, 1, 1);
```

Output

```
POINT(2 1)
```

Figure



Voir aussi

CG_Rotate, CG_3DRotate

8.3.37 CG_3DRotate

CG_3DRotate — Fait pivoter une géométrie dans l'espace 3D autour d'un vecteur d'axe.

Synopsis

geometry **CG_3DRotate**(geometry geom, float8 angle, float8 ax, float8 ay, float8 az);

Description

Fait pivoter la géométrie d'entrée *geom* de *angle* radians autour d'un axe défini par le vecteur (*ax*, *ay*, *az*) passant par l'origine (0,0,0).

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

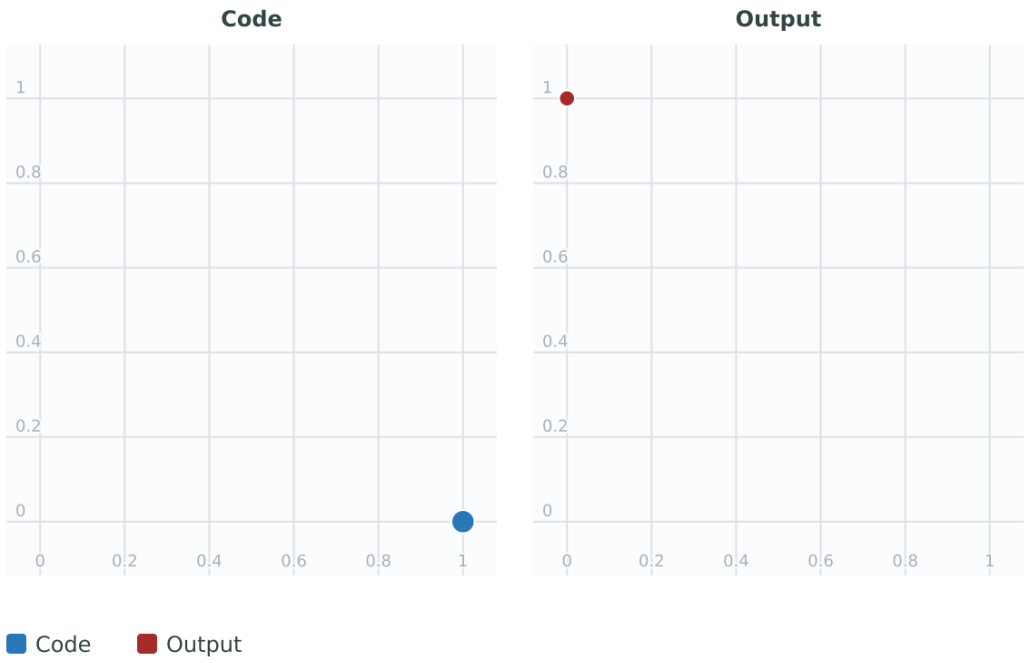
Code

```
SELECT CG_3DRotate('POINT(1 0 0)', pi()/2, 0, 0, 1);
```

Output

```
POINT Z (6.123233995736766e-17 1 0)
```

Figure



Voir aussi

CG_RotateX, CG_RotateY, CG_RotateZ

8.3.38 CG_RotateX

CG_RotateX — Fait pivoter une géométrie autour de l'axe X d'un angle donné.

Synopsis

geometry **CG_RotateX**(geometry geom, float8 angle);

Description

Fait pivoter la géométrie d'entrée *geom* de *angle* radians autour de l'axe X.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

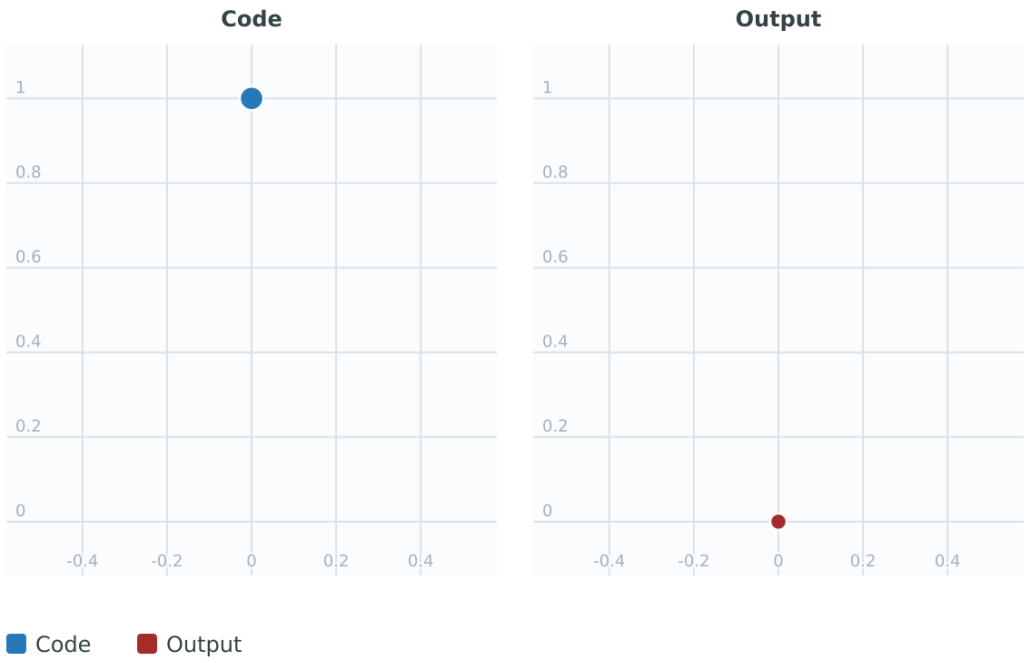
Code

```
SELECT CG_RotateX('POINT(0 1 0)', pi()/2);
```

Output

```
POINT Z (0 6.123233995736766e-17 1)
```

Figure



Voir aussi

CG_RotateY, CG_RotateZ, CG_3DRotate

8.3.39 CG_RotateY

CG_RotateY — Fait pivoter une géométrie autour de l'axe Y d'un angle donné.

Synopsis

geometry **CG_RotateY**(geometry geom, float8 angle);

Description

Fait pivoter la géométrie d'entrée *geom* de *angle* radians autour de l'axe Y.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

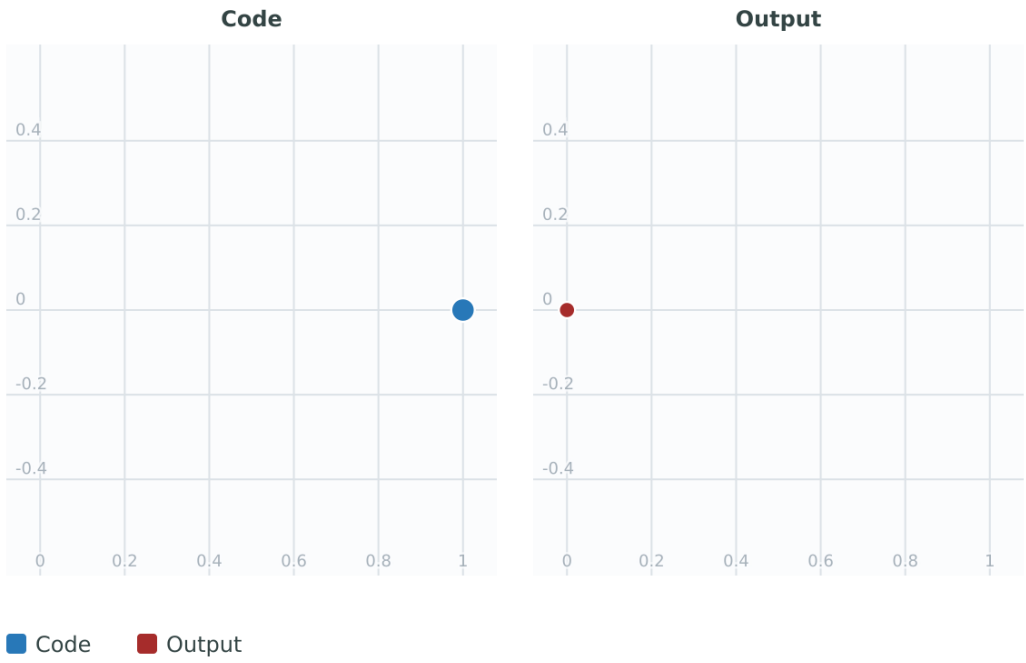
Code

```
SELECT CG_RotateY('POINT(1 0 0)', pi()/2);
```

Output

```
POINT Z (6.123233995736766e-17 0 -1)
```

Figure



Voir aussi

CG_RotateX, CG_RotateZ, CG_3DRotate

8.3.40 CG_RotateZ

CG_RotateZ — Fait pivoter une géométrie autour de l'axe Z d'un angle donné.

Synopsis

geometry **CG_RotateZ**(geometry geom, float8 angle);

Description

Fait pivoter la géométrie d'entrée *geom* de *angle* radians autour de l'axe Z.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

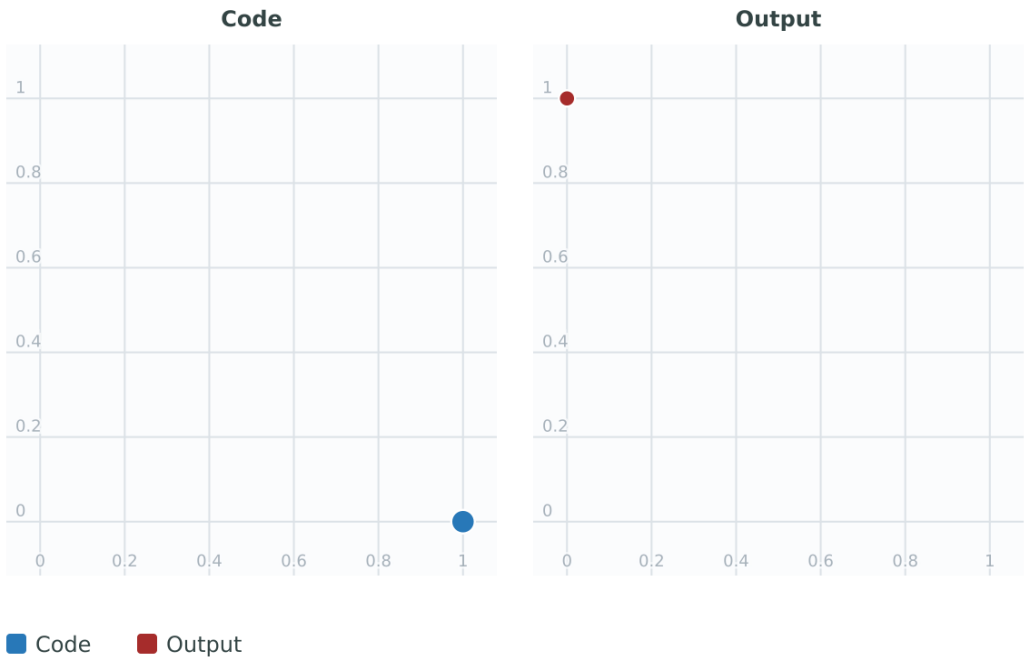
Code

```
SELECT CG_RotateZ('POINT(1 0 0)', pi()/2);
```

Output

```
POINT Z (6.123233995736766e-17 1 0)
```

Figure



Voir aussi

CG_RotateX, CG_RotateY, CG_3DRotate

8.3.41 CG_Scale

CG_Scale — Met à l'échelle une géométrie uniformément dans toutes les dimensions selon un facteur donné.

Synopsis

geometry **CG_Scale**(geometry geom, float8 factor);

Description

Met à l'échelle la géométrie d'entrée *geom* par un facteur d'échelle uniforme dans toutes les dimensions (X, Y et Z). La mise à l'échelle est effectuée par rapport au point d'origine (0,0,0).

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Exemples

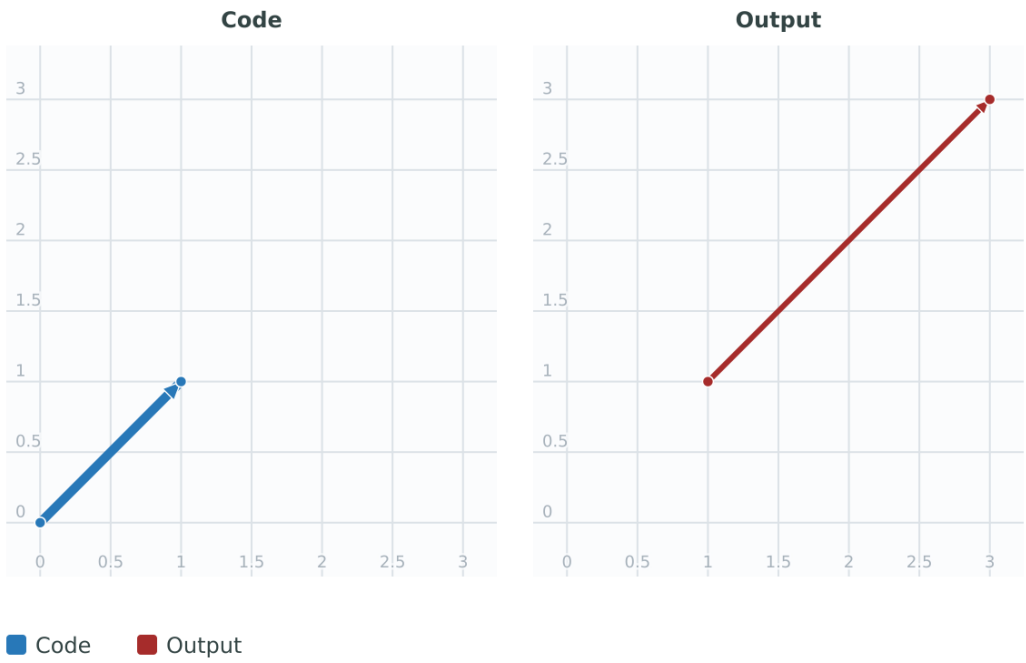
Code

```
SELECT CG_Scale('LINESTRING(1 1,2 2)', 2);
```

Output

```
LINESTRING(2 2,4 4)
```

Figure



Voir aussi

CG_3DScale, CG_3DScaleAroundCenter, ST_Scale

8.3.42 CG_3DScale

CG_3DScale — Met à l'échelle une géométrie en fonction de facteurs distincts le long des axes X, Y et Z.

Synopsis

geometry **CG_3DScale**(geometry geom, float8 factorX, float8 factorY, float8 factorZ);

Description

Met à l'échelle la géométrie d'entrée *geom* en fonction de différents facteurs le long des axes X, Y et Z. La mise à l'échelle est effectuée par rapport au point d'origine (0,0,0). La mise à l'échelle est effectuée par rapport au point d'origine (0,0,0).

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT CG_3DScale('POINT(1 1 1)', 2, 3, 4);
```

Output

```
POINT Z (2 3 4)
```

Figure



Voir aussi

CG_Scale, CG_3DScaleAroundCenter

8.3.43 CG_3DScaleAroundCenter

CG_3DScaleAroundCenter — Met à l'échelle une géométrie dans l'espace 3D autour d'un point central spécifié.

Synopsis

geometry **CG_3DScaleAroundCenter**(geometry geom, float8 factorX, float8 factorY, float8 factorZ, float8 centerX, float8 centerY, float8 centerZ);

Description

Met à l'échelle la géométrie d'entrée *geom* par différents facteurs le long des axes X, Y et Z, par rapport à un point central spécifié (*centerX*, *centerY*, *centerZ*).

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

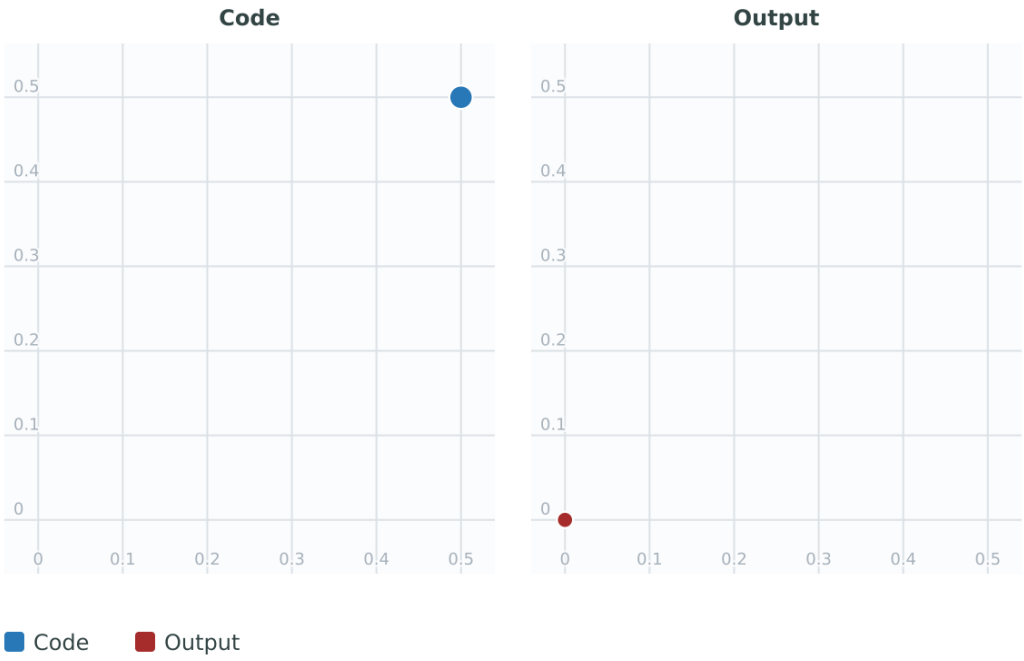
Code

```
SELECT CG_3DScaleAroundCenter('POINT(2 2 2)', 0.5, 0.5, 0.5, 1, 1, 1);
```

Output

```
POINT Z (1.5 1.5 1.5)
```

Figure



Voir aussi

CG_Scale, CG_3DScale

8.3.44 CG_Translate

CG_Translate — Déplace une géométrie en fonction de décalages donnés dans l'espace 2D.

Synopsis

geometry **CG_Translate**(geometry geom, float8 deltaX, float8 deltaY);

Description

Déplace la géométrie d'entrée *`geom`* en ajoutant *`deltaX`* aux coordonnées X et *`deltaY`* aux coordonnées Y. Les coordonnées Z sont supprimées. Les coordonnées Z ne sont pas prises en compte.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Exemples

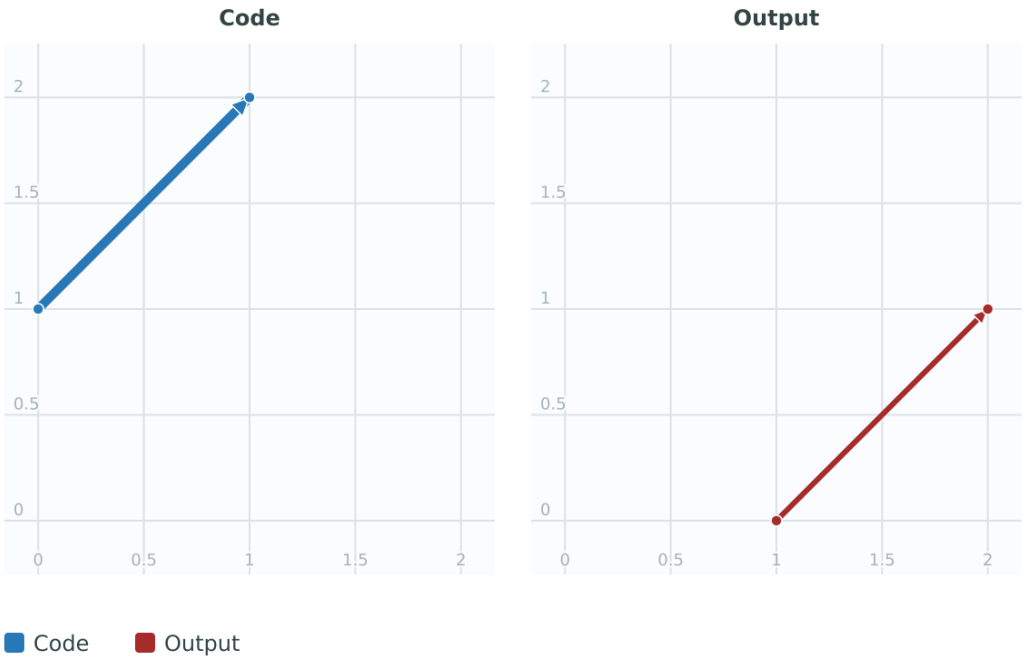
Code

```
SELECT CG_Translate('LINESTRING(1 1,2 2)', 1, -1);
```

Output

```
LINESTRING(2 0,3 1)
```

Figure



Voir aussi

CG_3DTranslate, ST_Translate

8.3.45 CG_3DTranslate

CG_3DTranslate — Déplace une géométrie en fonction de décalages donnés dans l'espace 3D.

Synopsis

geometry **CG_3DTranslate**(geometry geom, float8 deltaX, float8 deltaY, float8 deltaZ);

Description

Déplace la géométrie d'entrée *geom* en ajoutant *deltaX* aux coordonnées X, *deltaY* aux coordonnées Y et *deltaZ* aux coordonnées Z.

Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Code

```
SELECT CG_3DTranslate('POINT(1 1 1)', 1, -1, 2);
```

Output

```
POINT Z (2 0 3)
```

Figure



Voir aussi

CG_Translate, ST_Translate

8.3.46 CG_Simplify

CG_Simplify — Reduces the complexity of a geometry while preserving essential features and Z/M values.

Synopsis

geometry **CG_Simplify**(geometry geom, double precision threshold, boolean preserveTopology = false);

Description

Simplifies a geometry using SFCGAL's simplification algorithm, which reduces the number of points or vertices while preserving the essential features of the geometry. This function preserves Z and M values during simplification.

The algorithm is based on constrained triangulation and uses the CGAL Polyline Simplification 2 library with additional handling to preserve Z and M coordinates. When topology is preserved and geometries intersect, Z and M values are interpolated at intersection points.

This function works well with 3D terrain-like geometries (2.5D) but is not designed for vertical surfaces like walls.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Cette fonction prend en charge les coordonnées M.

Paramètres

geom Input geometry

threshold Maximum distance threshold (in geometry unit) for simplification. The higher this value, the more simplified the resulting geometry will be.

preserveTopology If set to true, the function ensures that the topology of the geometry is preserved. When geometries intersect in this mode, Z and M values at intersection points are interpolated. The default value is false.

Return Value

Returns a simplified geometry with preserved Z and M values.

Exemples

This example simplifies a polygon with a threshold of 0.5.

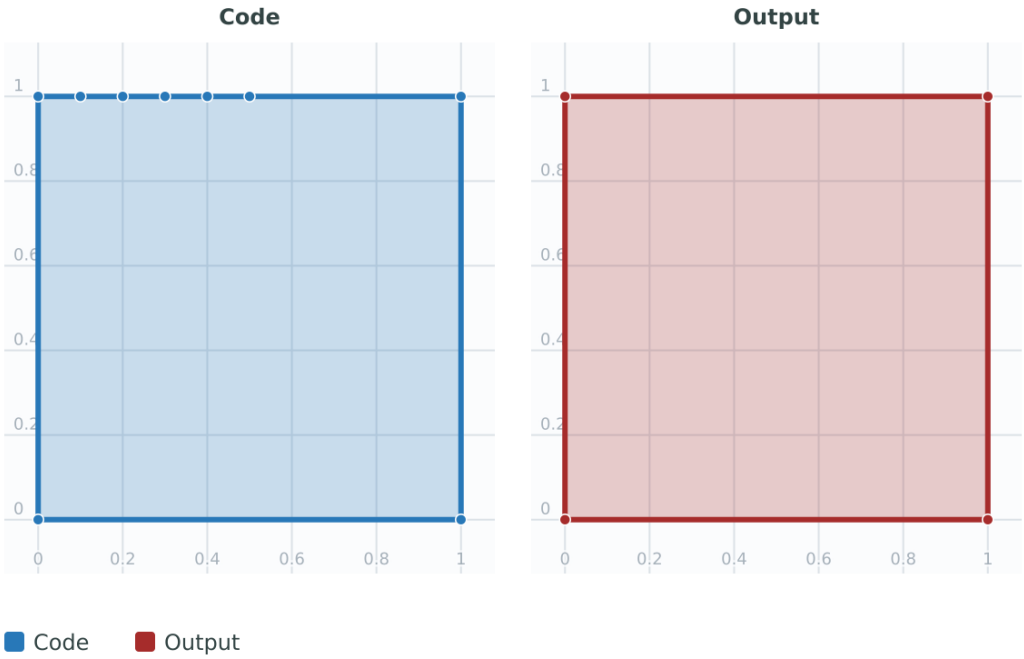
Code

```
SELECT CG_Simplify(
    'POLYGON((0 0,0 1,0.1 1,0.2 1,0.3 1,0.4 1,0.5 1,1 1,1 0,0 0))',
    0.5
  );
```

Output

```
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Figure



Preserving topology can keep additional support points in a mixed ZM collection.

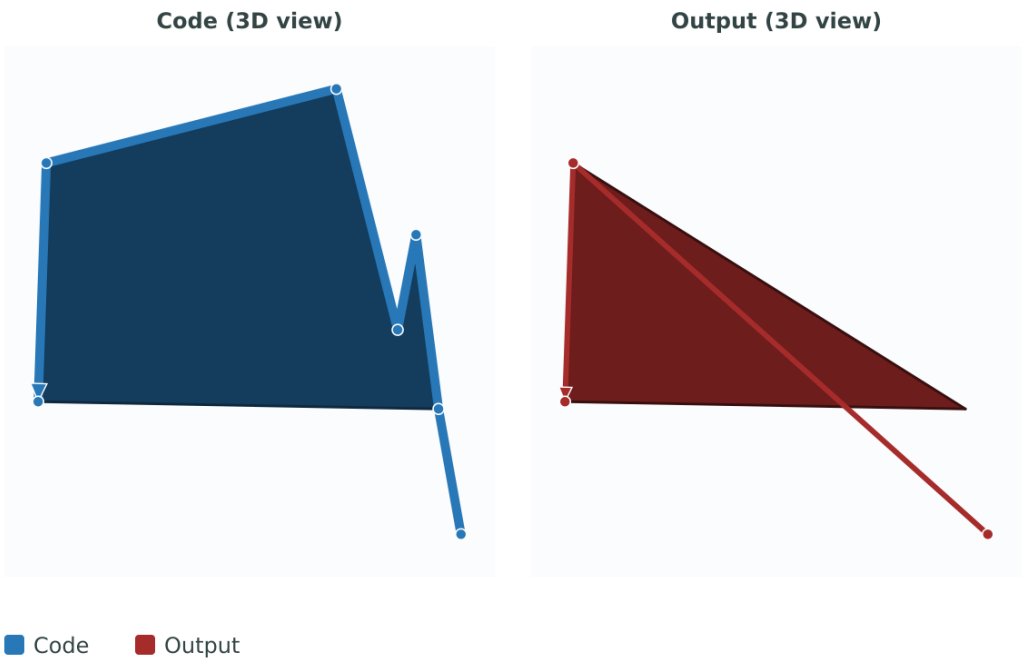
Code

```
SELECT CG_Simplify(
    'GEOMETRYCOLLECTION ZM(
      LINESTRING ZM(-1 -1 3 4,0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 ↩
    15 150),
      POLYGON ZM((0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 15 150,0 0 ↩
    10 100))
    )',
    2,
    false
  );
```

Output

```
GEOMETRYCOLLECTION ZM (LINESTRING ZM (-1 -1 3 4,2 19 25 250,-4 20 15 150),POLYGON ZM ((0 0 10 100,2 19 25 250,-4 20 15 ↩
    150,0 0 10 100)))
```

Figure



The topology-preserving variant keeps the shared anchor point at the collection intersection.

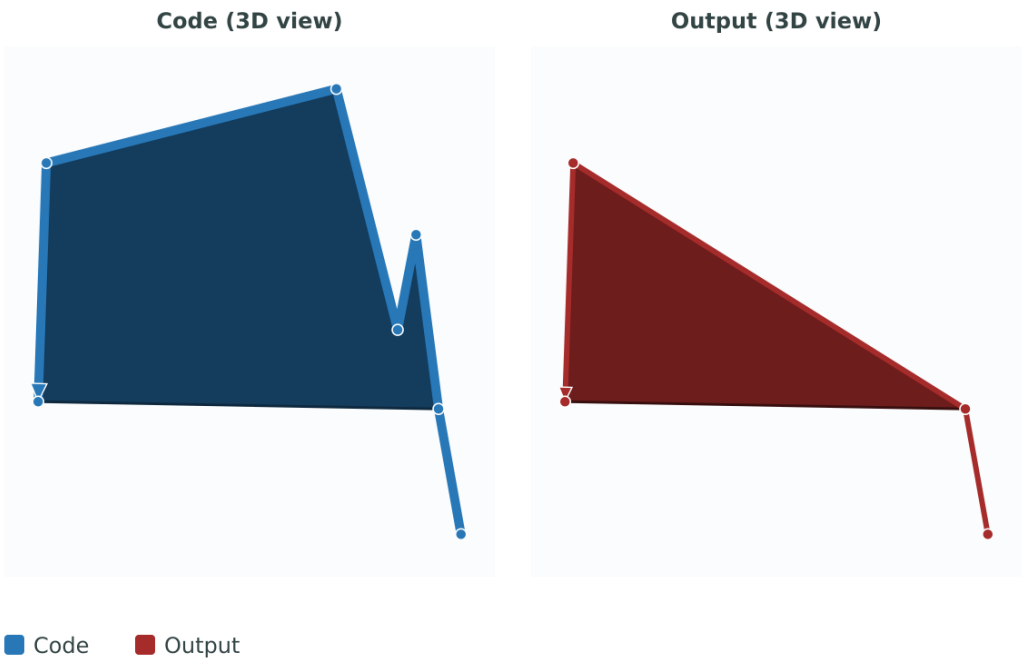
Code

```
SELECT CG_Simplify(
    'GEOMETRYCOLLECTION ZM(
      LINESTRING ZM(-1 -1 3 4,0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 ↩
    15 150),
      POLYGON ZM((0 0 10 100,1 1 20 200,0 2 15 150,0 5 30 300,2 19 25 250,-4 20 15 150,0 0 ↩
    10 100))
    )',
    2,
    true
  );
```

Output

```
GEOMETRYCOLLECTION ZM (LINESTRING ZM (-1 -1 3 4,0 0 10 100,2 19 25 250,-4 20 15 150),POLYGON ZM ((0 0 10 100,2 19 25 ↩
    250,-4 20 15 150,0 0 10 100)))
```

Figure

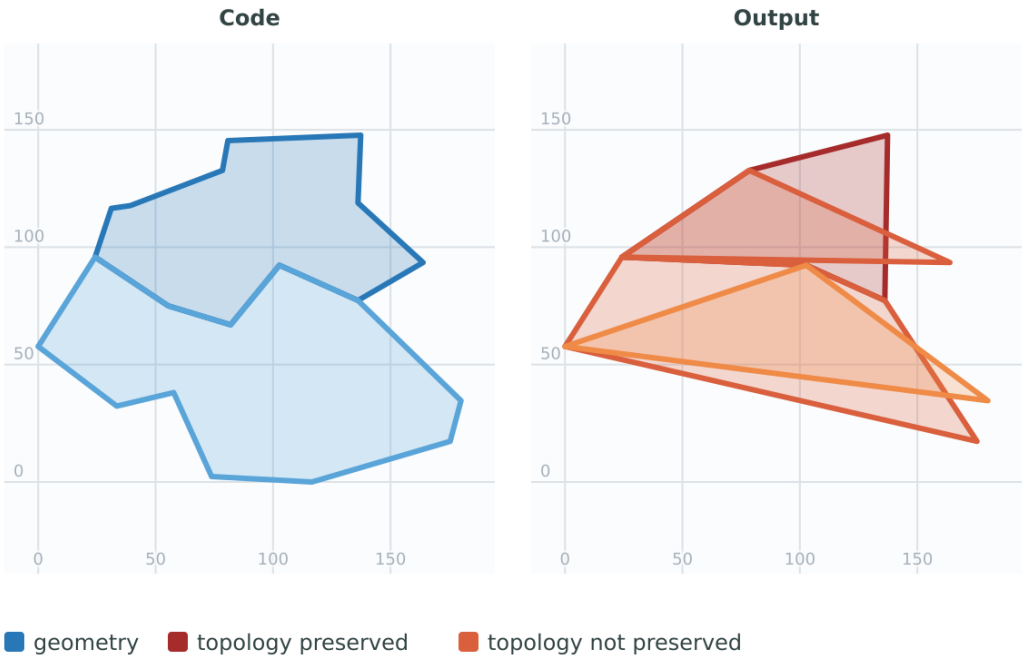


Compare simplification with and without topology preservation, using a threshold of 50.

Code

```
WITH data AS (
  SELECT ST_GeomFromText('GEOMETRYCOLLECTION(
    POLYGON((88.46 158.85,90.77 171.54,147.31 173.85,146.15 145,173.85 119.62,146.15 ↩
    103.46,112.69 118.46,91.92 93.08,65.38 101.15,34.23 121.92,41.15 142.69,49.23 ↩
    143.85,88.46 158.85)),
    POLYGON((112.69 118.46,146.15 103.46,190 60.77,185.38 43.46,126.54 26.15,83.85 ↩
    28.46,67.69 64.23,43.46 58.46,10 83.85,34.23 121.92,65.38 101.15,91.92 93.08,112.69 ↩
    118.46)))
  ') AS geom
)
SELECT geom AS input_geometry,
       CG_Simplify(geom, 50, true) AS topology_preserved,
       CG_Simplify(geom, 50, false) AS topology_not_preserved
FROM data;
```

Figure



Voir aussi

ST_Simplify, ST_SimplifyPreserveTopology

8.3.47 CG_3DAlphaWrapping

CG_3DAlphaWrapping — Computes a 3D Alpha-wrapping strictly enclosing a geometry.

Synopsis

geometry **CG_3DAlphaWrapping**(geometry geom, integer relative_alpha, integer relative_offset);

Description

Computes the 3D Alpha Wrapping of the points in a geometry. An alpha wrapping is a watertight and orientable surface mesh that strictly encloses the input. It can be seen as an extension or refinement of an alpha-shape.

The `relative_alpha` parameter controls which features will appear in the output. It can have values from 0 to infinity. Higher relative_alpha values produce coarser wraps; lower values retain more detail.

The `relative_offset` parameter controls the tightness of the result. It can have values from 0 to infinity. If this parameter is set to 0, its value is automatically determined based on the relative_alpha parameter.

Availability: 3.6.0 - requires SFCGAL >= 2.1.0

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

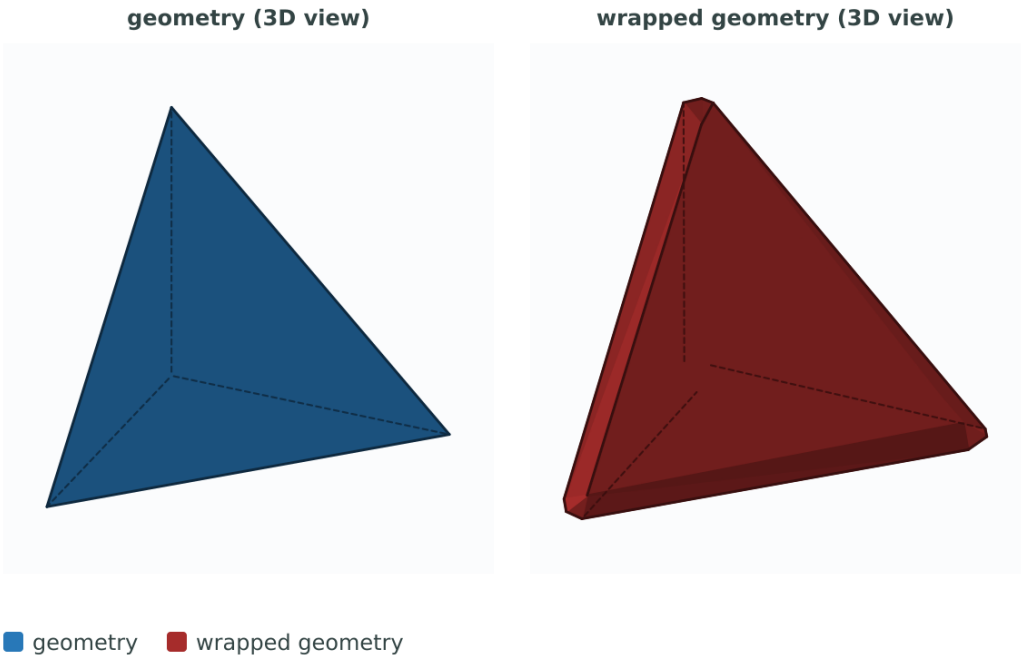
Exemples

Wrap a tetrahedral surface and show the native 3D input beside the actual enclosing mesh.

Code

```
WITH data AS (
  SELECT 'POLYHEDRALSURFACE Z (
    ((0 0 0,4 0 0,0 4 0,0 0 0)),
    ((0 0 0,0 0 4,4 0 0,0 0 0)),
    ((0 0 0,0 4 0,0 0 4,0 0 0)),
    ((4 0 0,0 0 4,0 4 0,4 0 0))
  )'::geometry AS input_geometry
)
SELECT input_geometry AS input_geometry,
       CG_3DAlphaWrapping(input_geometry, 1)
         AS wrapped_geometry
FROM data;
```

Figure



Voir aussi

CG_AlphaShape

8.3.48 CG_NurbsCurveInterpolate

CG_NurbsCurveInterpolate — Creates an interpolating NURBS curve passing through all given data points

Synopsis

geometry **CG_NurbsCurveInterpolate**(geometry data_points, integer degree);

Description

Creates a NURBS curve that interpolates through all provided data points. The resulting curve passes exactly through each point in the input geometry.

Parameters:

— `data_points` - A LINESTRING geometry containing the points to interpolate through.
— `degree` - Polynomial degree of the NURBS curve (typically 2 or 3).
Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Interpolate a degree-3 NURBS curve through four 2D points.

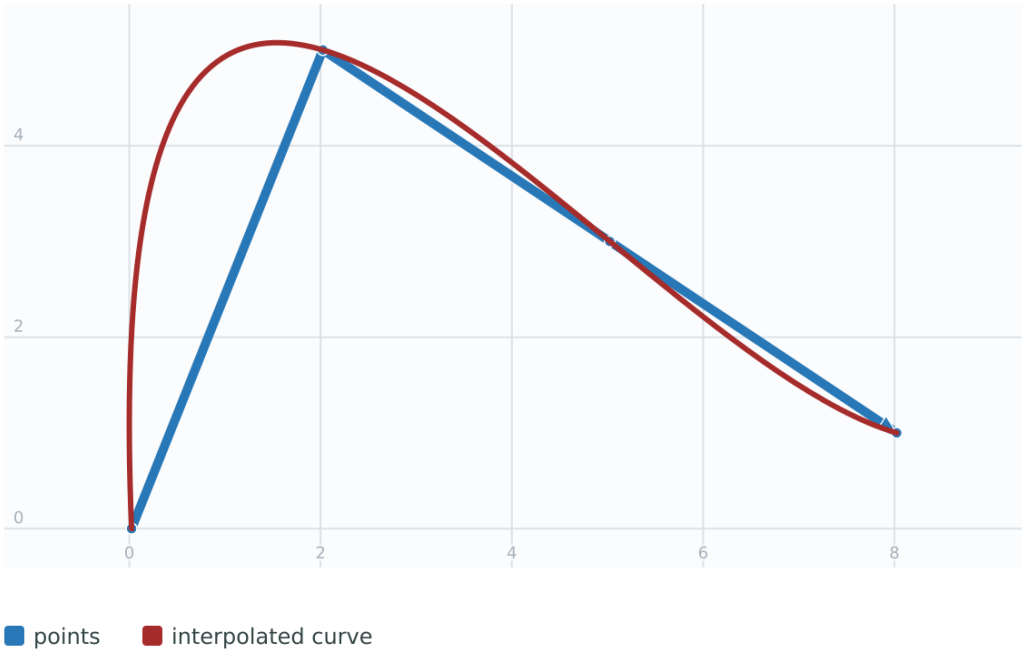
Code

```
WITH data AS (
  SELECT 'LINESTRING(0 0,2 5,5 3,8 1)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveInterpolate(input_points, 3) AS interpolated_curve
FROM data;
```

Output

```
LINESTRING(0 0,2 5,5 3,8 1) | NURBSCURVE(DEGREE 3, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT( ↩
    WEIGHTEDPOINT(-0.43 10.29),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(4.95 1.9),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(8 1), ↩
    WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(1,4)))
```

Figure



Interpolate a 3D degree-3 NURBS curve through the documented control points.

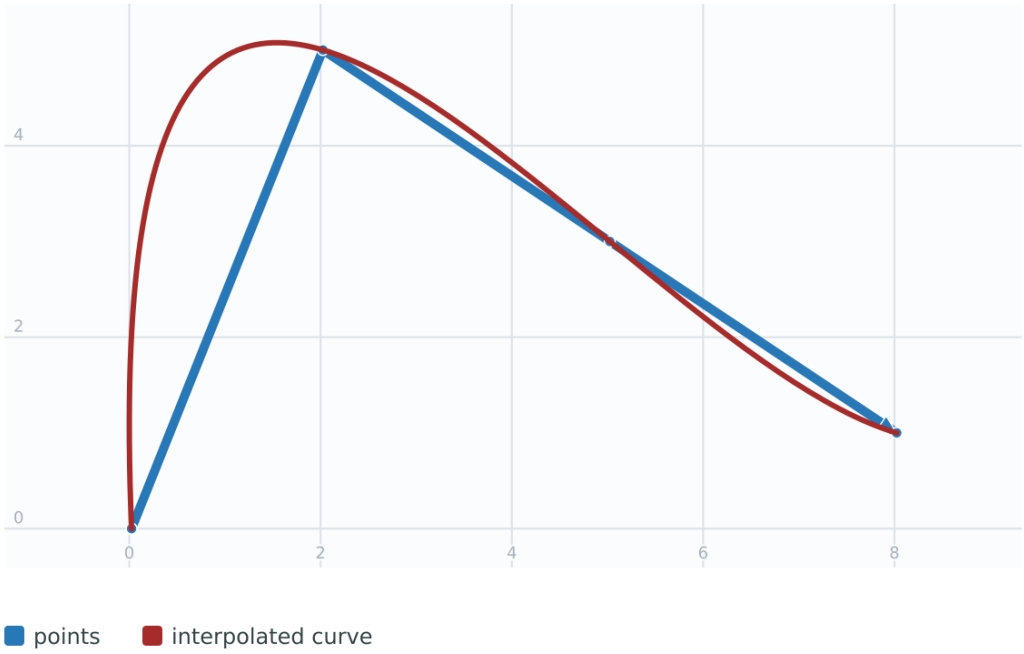
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,2 5 2,5 3 1,8 1 0)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveInterpolate(input_points, 3) AS interpolated_curve
FROM data;
```

Output

```
LINESTRING Z (0 0 0,2 5 2,5 3 1,8 1 0) | NURBSCURVE Z (DEGREE 3,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z (0 0 0), ↩
    WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (-0.43 10.29 4.36),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (4.95 1.9 0.54), ↩
    WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (8 1 0),WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(1,4)))
```

Figure



Voir aussi

CG_NurbsCurveApproximate, ST_MakeNurbsCurve

8.3.49 CG_NurbsCurveApproximate

CG_NurbsCurveApproximate — Creates an approximating NURBS curve fitting data points within a tolerance

Synopsis

geometry **CG_NurbsCurveApproximate**(geometry data_points, integer degree, float8 tolerance, integer max_control_points=100);

Description

Creates a NURBS curve that approximates the given data points within the specified tolerance. Unlike interpolation, the curve does not necessarily pass through all points but provides a smooth fit with fewer control points.

Parameters:
— `data_points` - A LINESTRING geometry containing the points to approximate.
— `degree` - Polynomial degree of the NURBS curve.
— `tolerance` - Maximum allowed distance between the curve and data points.
— `max_control_points` - Optional maximum number of control points (default: 100). Limits curve complexity.
Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

Approximate noisy 2D data with a degree-2 curve and a tolerance of 0.5.

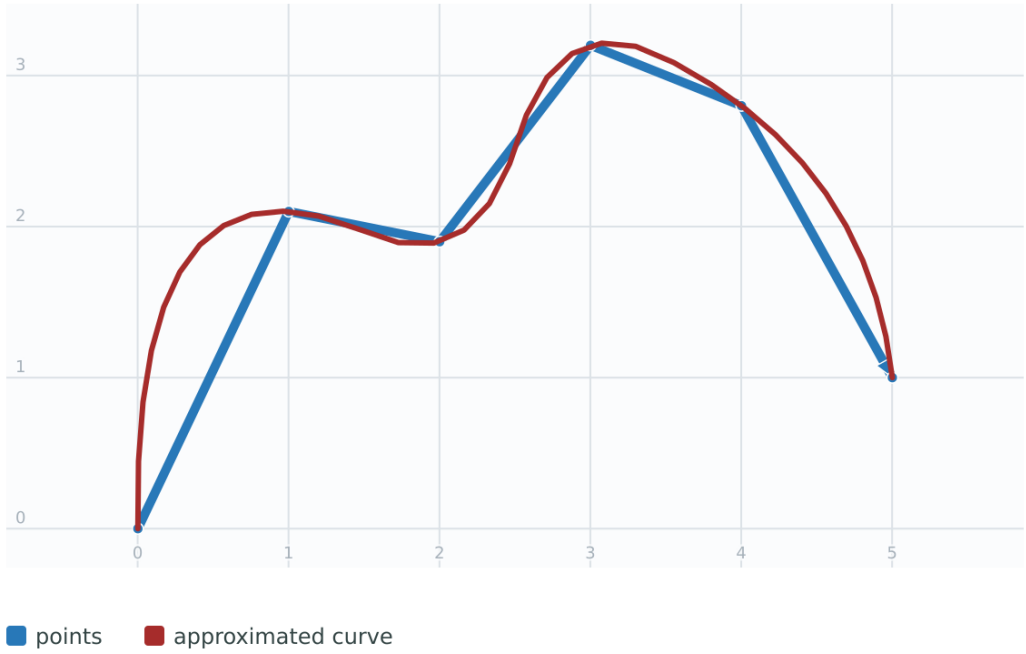
Code

```
WITH data AS (
  SELECT 'LINESTRING(0 0,1 2.1,2 1.9,3 3.2,4 2.8,5 1)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveApproximate(input_points, 2, 0.5) AS approximated_curve
FROM data;
```

Output

```
LINESTRING(0 0,1 2.1,2 1.9,3 3.2,4 2.8,5 1) | NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT
    1), NURBSPOINT(WEIGHTEDPOINT(-0.05 2.64),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2.24 1.65),WEIGHT 1), NURBSPOINT(
    WEIGHTEDPOINT(2.79 3.48),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(4.84 2.43),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 1),
    WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(0.35,1), KNOT(0.51,1), KNOT(0.68,1), KNOT(1,3)))
```

Figure



Approximate a 3D point set while preserving the Z dimension in the fitted curve.

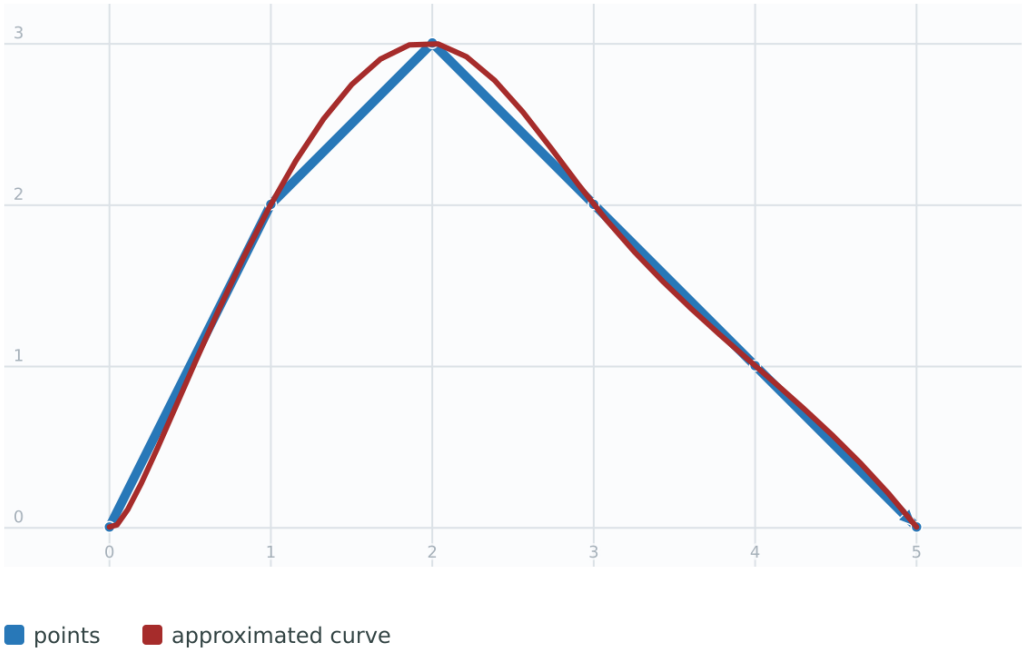
Code

```
WITH data AS (
  SELECT 'LINESTRING Z (0 0 0,1 2 1,2 3 2,3 2 1,4 1 0,5 0 0)'::geometry AS input_points
)
SELECT input_points AS input_points,
       CG_NurbsCurveApproximate(input_points, 3, 0.1) AS approximated_curve
FROM data;
```

Output

```
LINESTRING Z (0 0 0,1 2 1,2 3 2,3 2 1,4 1 0,5 0 0) | NURBSCURVE Z (DEGREE 3,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z
     (0 0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (0.18 -0.15 -1.29),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (1.49 4.36
     3.35),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (3.33 1.33 0.38),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (4.34 0.83
    -0.44),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (5 0 0),WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(0.46,1), KNOT(0.64,1), KNOT
    (1,4)))
```

Figure



Voir aussi

CG_NurbsCurveInterpolate, ST_MakeNurbsCurve

8.3.50 CG_NurbsCurveDerivative

CG_NurbsCurveDerivative — Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter

Synopsis

geometry **CG_NurbsCurveDerivative**(geometry nurbscurve, float8 parameter, integer derivative_order);

Description

Computes the derivative of a NURBS curve at a specific parameter value. The derivative represents the rate of change at that point:
— Order 1: Tangent vector (direction and speed)
— Order 2: Curvature vector (how the tangent changes)
— Order 3: Third-order derivative (supported maximum)
Parameters:
— `nurbscurve` - A NURBS curve geometry.
— `parameter` - Parameter value for evaluation (typically 0.0 to 1.0).
— `derivative_order` - Order of derivative to compute (1, 2, or 3).
Returns a POINT geometry representing the derivative vector at the specified parameter.

Availability: 3.7.0 - requires SFCGAL >= 2.3.0.

Cette méthode nécessite le backend SFCGAL.

Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

The first derivative gives the tangent vector at the evaluation parameter.

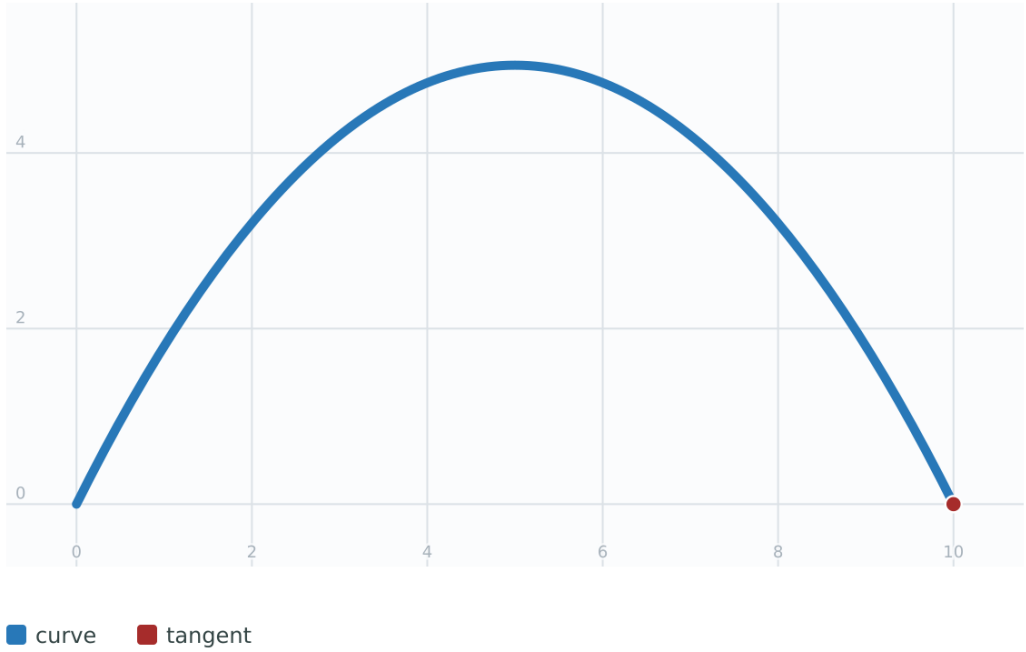
Code

```
WITH data AS (
  SELECT 'NURBSCURVE (2, (0.00 0.00, 5.00 10.00, 10.00 0.00))'::geometry AS input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 1) AS tangent
FROM data;
```

Output

```
NURBSCURVE(DEGREE 2, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(5 10),WEIGHT 1),
    NURBSPOINT(WEIGHTEDPOINT(10 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3))) | POINT Z (10 0 0)
```

Figure



The second derivative exposes curvature change for a higher-degree curve.

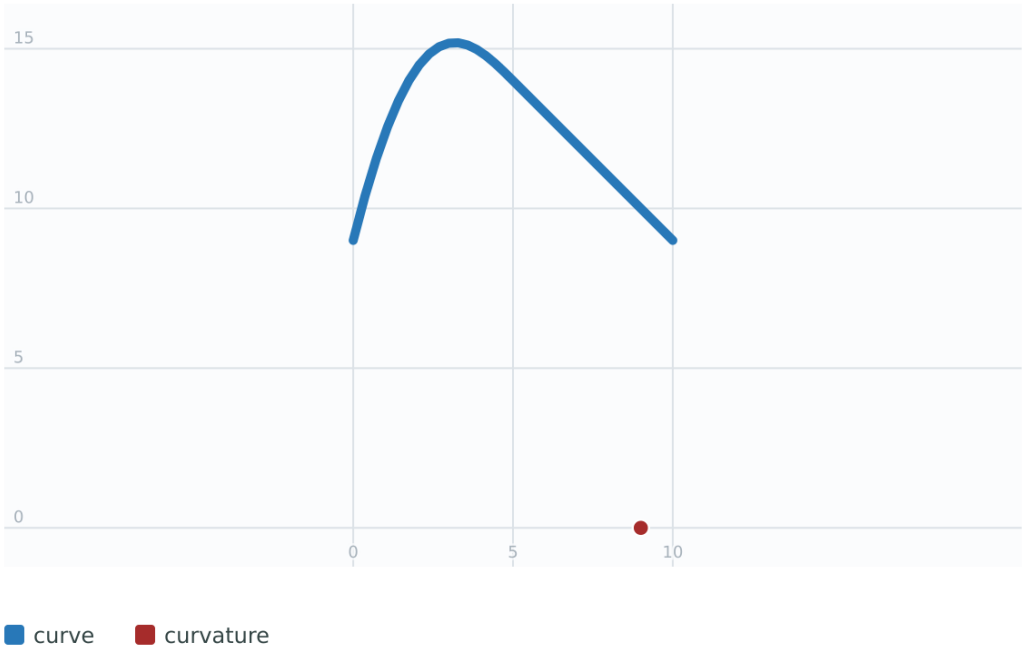
Code

```
WITH data AS (
  SELECT 'NURBSCURVE (3, (0.00 0.00, 2.00 8.00, 5.00 5.00, 8.00 2.00, 10.00 0.00))':: ↩
    geometry AS input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 2) AS curvature
FROM data;
```

Output

```
NURBSCURVE(DEGREE 3, CONTROLPOINTS(NURBSPOINT(WEIGHTEDPOINT(0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(2 8),WEIGHT 1),
    NURBSPOINT(WEIGHTEDPOINT(5 5),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(8 2),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT(10 0),
    WEIGHT 1)),KNOTS (KNOT(0,4), KNOT(0.5,1), KNOT(1,4))) | POINT Z (9 -9 0)
```

Figure



The same derivative call also works for 3D NURBS curves.

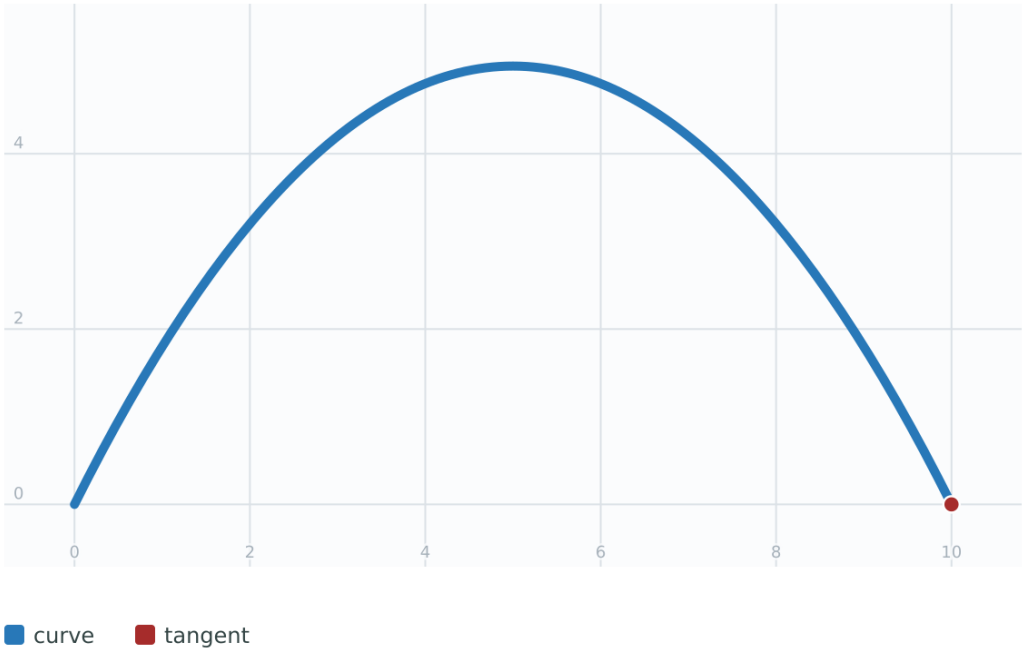
Code

```
WITH data AS (
  SELECT 'NURBSCURVE Z (2, (0.00 0.00 0.00, 5.00 10.00 5.00, 10.00 0.00 0.00))'::geometry AS ↩
    input_curve
)
SELECT input_curve AS input_curve,
       CG_NurbsCurveDerivative(input_curve, 0.5, 1) AS tangent
FROM data;
```

Output

```
NURBSCURVE Z (DEGREE 2,CONTROLPOINTS Z (NURBSPOINT(WEIGHTEDPOINT Z (0 0 0),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (5 10 ↩
    5),WEIGHT 1), NURBSPOINT(WEIGHTEDPOINT Z (10 0 0),WEIGHT 1)),KNOTS (KNOT(0,3), KNOT(1,3))) | POINT Z (10 0 0)
```

Figure



Voir aussi

ST_Evaluate, ST_MakeNurbsCurve

8.4 Deprecated SFCGAL Function Aliases

These `ST_` names remain available for compatibility and emit deprecation notices. Their entries contain only the legacy signature and migration link. Use the corresponding `CG_` pages for behavior, examples, and geometry support details.

8.4.1 ST_3DIntersection

ST_3DIntersection — Deprecated alias for `CG_3DIntersection`.

Synopsis

geometry **ST_3DIntersection**(geometry geom1, geometry geom2);

Description

AVERTISSEMENT
`ST_3DIntersection` is deprecated as of 3.5.0. Use CG_3DIntersection instead.

Availability: 2.1.0

Cette méthode nécessite le backend SFCGAL.

8.4.2 ST_3DDifference

ST_3DDifference — Deprecated alias for `CG_3DDifference`.

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

Description



AVERTISSEMENT
`ST_3DDifference` is deprecated as of 3.5.0. Use CG_3DDifference instead.

Availability: 2.2.0


Cette méthode nécessite le backend SFCGAL.

8.4.3 ST_3DUnion

ST_3DUnion — Deprecated alias for `CG_3DUnion`.

Synopsis

geometry **ST_3DUnion**(geometry geom1, geometry geom2);
geometry **ST_3DUnion**(geometry set g1field);

Description



AVERTISSEMENT
`ST_3DUnion` is deprecated as of 3.5.0. Use CG_3DUnion instead.

Availability: 2.2.0


Cette méthode nécessite le backend SFCGAL.

8.4.4 ST_Tesselate

ST_Tesselate — Deprecated alias for `CG_Tesselate`.

Synopsis

geometry **ST_Tesselate**(geometry geom);

Description



AVERTISSEMENT
`ST_Tesselate` is deprecated as of 3.5.0. Use CG_Tesselate instead.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.5 ST_3DArea

ST_3DArea — Deprecated alias for `CG_3DArea`.

Synopsis

float**ST_3DArea**(geometry geom1);

Description



AVERTISSEMENT
`ST_3DArea` is deprecated as of 3.5.0. Use CG_3DArea instead.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.6 ST_Extrude

ST_Extrude — Deprecated alias for `CG_Extrude`.

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

Description



AVERTISSEMENT
`ST_Extrude` is deprecated as of 3.5.0. Use CG_Extrude instead.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.7 ST_ForceLHR

ST_ForceLHR — Deprecated alias for `CG_ForceLHR`.

Synopsis

geometry **ST_ForceLHR**(geometry geom);

Description



AVERTISSEMENT
`ST_ForceLHR` is deprecated as of 3.5.0. Use CG_ForceLHR instead.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.8 ST_Orientation

ST_Orientation — Deprecated alias for `CG_Orientation`.

Synopsis

integer **ST_Orientation**(geometry geom);

Description



AVERTISSEMENT
`ST_Orientation` is deprecated as of 3.5.0. Use CG_Orientation instead.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.9 ST_MinkowskiSum

ST_MinkowskiSum — Deprecated alias for `CG_MinkowskiSum`.

Synopsis

geometry **ST_MinkowskiSum**(geometry geom1, geometry geom2);

Description



AVERTISSEMENT
`ST_MinkowskiSum` is deprecated as of 3.5.0. Use CG_MinkowskiSum instead.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.10 ST_StraightSkeleton

ST_StraightSkeleton — Deprecated alias for `CG_StraightSkeleton`.

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

Description



AVERTISSEMENT
`ST_StraightSkeleton` is deprecated as of 3.1.0. Use CG_StraightSkeleton instead.



Note
Like `CG_StraightSkeleton`, this function ignores the Z dimension and always gives a 2D result, even when used on a 3D geometry.

Availability: 2.1.0


Cette méthode nécessite le backend SFCGAL.

8.4.11 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — Deprecated alias for `CG_ApproximateMedialAxis`.

Synopsis

geometry **ST_ApproximateMedialAxis**(geometry geom);

Description



AVERTISSEMENT
`ST_ApproximateMedialAxis` is deprecated as of 3.5.0. Use CG_ApproximateMedialAxis instead.



Note
Like `CG_ApproximateMedialAxis`, this function ignores the Z dimension and always gives a 2D result, even when used on a 3D geometry.

Availability: 2.2.0


Cette méthode nécessite le backend SFCGAL.

8.4.12 ST_IsPlanar

ST_IsPlanar — Deprecated alias for `CG_IsPlanar`.

Synopsis

boolean **ST_IsPlanar**(geometry geom);

Description



AVERTISSEMENT
`ST_IsPlanar` is deprecated as of 3.5.0. Use CG_IsPlanar instead.

Availability: 2.2.0: This was documented in 2.1.0 but got accidentally left out in 2.1 release.


Cette méthode nécessite le backend SFCGAL.

8.4.13 ST_Volume

ST_Volume — Deprecated alias for `CG_Volume`.

Synopsis

float **ST_Volume**(geometry geom1);

Description



AVERTISSEMENT
`ST_Volume` is deprecated as of 3.5.0. Use CG_Volume instead.

Availability: 2.2.0


Cette méthode nécessite le backend SFCGAL.

8.4.14 ST_MakeSolid

ST_MakeSolid — Deprecated alias for `CG_MakeSolid`.

Synopsis

geometry **ST_MakeSolid**(geometry geom1);

Description



AVERTISSEMENT
`ST_MakeSolid` is deprecated as of 3.5.0. Use CG_MakeSolid instead.

Availability: 2.2.0


Cette méthode nécessite le backend SFCGAL.

8.4.15 ST_IsSolid

ST_IsSolid — Deprecated alias for `CG_IsSolid`.

Synopsis

boolean **ST_IsSolid**(geometry geom1);

Description



AVERTISSEMENT
`ST_IsSolid` is deprecated as of 3.5.0. Use CG_IsSolid instead.

Availability: 2.2.0


Cette méthode nécessite le backend SFCGAL.

8.4.16 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Deprecated alias for `CG_ConstrainedDelaunayTriangles`.

Synopsis

geometry **ST_ConstrainedDelaunayTriangles**(geometry g1);

Description



AVERTISSEMENT
`ST_ConstrainedDelaunayTriangles` is deprecated as of 3.5.0. Use CG_ConstrainedDelaunayTriangles instead.

Availability: 3.0.0


Cette méthode nécessite le backend SFCGAL.

8.4.17 ST_3DConvexHull

ST_3DConvexHull — Deprecated alias for `CG_3DConvexHull`.

Synopsis

geometry **ST_3DConvexHull**(geometry geom1);

Description



AVERTISSEMENT
`ST_3DConvexHull` is deprecated as of 3.5.0. Use CG_3DConvexHull instead.

Availability: 3.3.0


Cette méthode nécessite le backend SFCGAL.

8.4.18 ST_AlphaShape

ST_AlphaShape — Deprecated alias for `CG_AlphaShape`.

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha , boolean allow_holes = false);

Description



AVERTISSEMENT
`ST_AlphaShape` is deprecated as of 3.5.0. Use CG_AlphaShape instead.


Cette méthode nécessite le backend SFCGAL.

8.4.19 ST_OptimalAlphaShape

ST_OptimalAlphaShape — Deprecated alias for `CG_OptimalAlphaShape`.

Synopsis

geometry **ST_OptimalAlphaShape**(geometry geom, boolean allow_holes = false , integer nb_components = 1);

Description



AVERTISSEMENT
`ST_OptimalAlphaShape` is deprecated as of 3.5.0. Use CG_OptimalAlphaShape instead.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.


Cette méthode nécessite le backend SFCGAL.

Chapitre 9

Topologie

Les types et fonctions topologiques de PostGIS sont utilisés pour gérer les objets topologiques tels que les faces, les arêtes et les des noeuds.

La présentation de Sandro Santilli à la conférence PostGIS Day Paris 2011 donne un bon aperçu de PostGIS Topology et de son évolution Topology with PostGIS 2.0 slide deck.

Vincent Picavet fournit un bon résumé et une vue d'ensemble de ce qu'est la topologie, comment elle est utilisée, et divers outils FOSS4G qui la supportent dans PostGIS Topology PGConf EU 2012.

An example of a topologically based GIS database is the US Census Topologically Integrated Geographic Encoding and Referencing System (TIGER) database. If you want to experiment with PostGIS topology and need some data.

The PostGIS topology module has existed for a long time but was not always part of the official documentation. Extensive cleanup removed deprecated functions, fixed known usability issues, documented the features and functions, added new functionality, and improved SQL-MM compliance.

This chapter is the maintained reference for the topology module, including its primitive tables, TopoGeometry type, topology management functions, editing functions, validation functions, and conversion helpers.

Toutes les fonctions et toutes les tables associées à ce module sont installées dans un schéma appelé `topology`.

Les fonctions qui sont définies dans le standard SQL/MM sont préfixées par ST_ et les fonctions spécifiques à PostGIS ne sont pas préfixées.

Topology support is built by default and can be disabled by specifying the --without-topology configure option at build time as described in Chapitre 2

9.1 Building Faces from Linework

This example builds topology faces from noded linework and associates point attributes with the resulting faces. It uses ST_CreateTopoGe to populate an empty topology, ValidateTopology to check the result, GetFaceByPoint to locate each attribute point, and ST_GetFaceGeor to materialize ordinary polygon geometries.

Create an empty topology and populate it with linework that encloses two faces:

Code

```
CREATE EXTENSION IF NOT EXISTS postgis_topology;

SELECT topology.CreateTopology('topo_example', 3857);

WITH lines(geom) AS (VALUES
    ('SRID=3857;LINESTRING(0 0, 2 0)'::geometry),
    ('SRID=3857;LINESTRING(0 1, 2 1)'::geometry),
    ('SRID=3857;LINESTRING(0 0, 0 1)'::geometry),
```

```
    ('SRID=3857;LINESTRING(1 0, 1 1)'::geometry),
    ('SRID=3857;LINESTRING(2 0, 2 1)'::geometry)
)
SELECT topology.ST_CreateTopoGeo(
    'topo_example',
    ST_Collect(geom)
)
FROM lines;

SELECT * FROM topology.ValidateTopology('topo_example');
```

An empty result from `ValidateTopology` means that no topology errors were found. Attribute points can now be matched to faces. A face identifier of `0` denotes the universal face, so the point is outside every bounded face.

Code

```
WITH labels(name, geom) AS (VALUES
    ('west', 'SRID=3857;POINT(0.5 0.5)'::geometry),
    ('east', 'SRID=3857;POINT(1.5 0.5)'::geometry),
    ('outside', 'SRID=3857;POINT(3 0.5)'::geometry)
),
located AS (
    SELECT name,
           topology.GetFaceByPoint('topo_example', geom, 0) AS face_id
    FROM labels
)
SELECT name,
       face_id,
       CASE WHEN face_id = 0 THEN NULL
            ELSE topology.ST_GetFaceGeometry('topo_example', face_id)
       END AS geom
FROM located
ORDER BY name;
```

Before copying attributes into a face table, check both for points outside bounded faces and for multiple source points assigned to the same face:

Code

```
WITH labels(name, geom) AS (VALUES
    ('west', 'SRID=3857;POINT(0.5 0.5)'::geometry),
    ('east', 'SRID=3857;POINT(1.5 0.5)'::geometry),
    ('outside', 'SRID=3857;POINT(3 0.5)'::geometry)
),
located AS (
    SELECT name,
           topology.GetFaceByPoint('topo_example', geom, 0) AS face_id
    FROM labels
)
SELECT 'outside face' AS problem,
       array_agg(name ORDER BY name) AS labels
FROM located
WHERE face_id = 0
HAVING count(*) > 0
UNION ALL
SELECT 'duplicate face ' || face_id,
       array_agg(name ORDER BY name)
FROM located
WHERE face_id != 0
GROUP BY face_id
HAVING count(*) > 1;
```

Use DropTopology to remove the example topology when it is no longer needed:

Code

```
SELECT topology.DropTopology('topo_example');
```

9.2 Topology Primitive Tables

The core primitives of any topology are stored in the `edge_data`, `node`, and `face` tables that live in the schema created by CreateTopology. Each row of `edge_data` represents an oriented edge: it records a directed curve from `start_node` to `end_node` together with the identifier of the face encountered on the left of that direction (`left_face`) and the face encountered on the right (`right_face`). The same geometric segment may therefore appear twice—once for each orientation—when it belongs to two faces.

The `next_left_edge` and `next_right_edge` columns complete this orientation information by encoding how to keep walking around a face. They store signed integers whose absolute value is the identifier of the next oriented edge and whose sign determines whether the stored orientation has to be followed as-is or reversed when traversing. Formally, the following rules hold for every edge `e`:

— `abs(next_left_edge)` is the identifier of the edge reached by continuing around the face that lies to the left of `e`. If the value is positive the walk continues from the end node of `e` along the stored orientation of the referenced edge; if it is negative the referenced edge must be followed backwards so that the shared face remains on the walker's left.

— `abs(next_right_edge)` analogously follows the boundary of the face located on the right of `e`. A positive value means that the next edge is taken with its recorded orientation starting from the end node of `e`, whereas a negative value instructs to traverse the referenced edge in reverse, starting from its end node, so that the right-hand face is preserved.

— A zero value indicates that the edge is dangling on the corresponding side (for example an isolated edge whose incident face is the universal face `0`). The `abs_next_left_edge` and `abs_next_right_edge` columns exposed by the `edge` view are convenience projections of these absolute values.

This representation is a variant of a doubly connected edge list and is exploited by many topology routines. Functions such as GetRingEdges and ValidateTopology rely on it to reconstruct face boundaries and to diagnose inconsistencies—hence the "invalid next_left_edge" and "invalid next_right_edge" diagnostics reported during validation. Constructors like AddEdge initialise the `next_*` attributes with trivial self references, while editing routines including ST_AddEdgeModFace and ST_RemEdgeModFace update the links as edges are inserted or removed. Other bulk operations (for example Polygonize) may intentionally leave the fields unset, which is why the documentation flags their behaviour explicitly.

9.3 Les types associés à "Topology"

9.3.1 getfaceedges_returntype

getfaceedges_returntype — Type composite composé d'un numéro de séquence et d'un numéro d'arête.

Description

Type composite composé d'un numéro de séquence et d'un numéro d'arête. Il s'agit du type de retour pour les fonctions `ST_GetFaceEdges` et `GetNodeEdges`.

1. `sequence` est un entier. Il fait référence à un élément topologique défini dans la table topology.topology qui définit le schéma de la topologie et le srid.
2. `edge` est un entier : c'est l'identifiant d'une arête.

9.3.2 TopoGeometry

TopoGeometry — Un type composite représentant une géométrie topologiquement définie.

Description

Un type composite qui fait référence à une géométrie topologique dans une couche topologique spécifique, ayant un type et un identifiant spécifiques. Les éléments d'une TopoGeometry ont les propriétés suivantes : topology_id, layer_id, id integer, type integer.

1. `topology_id` est un entier : il fait référence à un élément topologique défini dans la table topology.topology qui définit le schéma de la topologie et le srid.
2. `layer_id` est un entier : L'identifiant de la couche dans la table des couches à laquelle la TopoGeometry appartient. La combinaison de topology_id, layer_id fournit une référence unique dans la table topology.layers.
3. `id` est un nombre entier : L'id est le numéro de séquence autogénéré qui définit de manière unique la topogeometry dans la couche topologique correspondante.
4. `type` un entier entre 1 et 4 qui définit le type de la géométrie : 1 : [multi]point, 2 : [multi]line, 3 : [multi]poly, 4 : collection

Transtypages

Cette section liste les transtypages autorisés pour ce type de donnée, qu'ils soient automatiques ou bien explicites

Transtypage vers	Comportement
geometry	automatique

Voir aussi

CreateTopoGeom

9.3.3 validatetopology_returntype

validatetopology_returntype — Un type composite composé d'un message d'erreur et de id1 et id2 pour indiquer l'emplacement de l'erreur. Il s'agit du type de retour pour `ValidateTopology`.

Description

Un type composite composé d'un message d'erreur et de deux nombres entiers. La fonction ValidateTopology renvoie un ensemble d'entiers pour indiquer les erreurs de validation et les id1 et id2 pour indiquer les id des objets topologiques concernés par l'erreur.

Field	Type	Description
`error`	varchar	Denotes the type of error. Current error descriptors are: coincident nodes, edge crosses node, edge not simple, edge end node geometry mismatch, edge start node geometry mismatch, face overlaps face, face within face.
`id1`	integer	Denotes the identifier of the edge, face, or node in error.
`id2`	integer	For errors that involve two objects, denotes the secondary edge or node.

Voir aussi

ValidateTopology

9.4 Domaines de topologie

9.4.1 TopoElement

TopoElement — Un tableau de 2 entiers généralement utilisé pour identifier un composant TopoGeometry.

Description

Un tableau de 2 entiers utilisé pour représenter une composante d'une TopoGeometry simple ou hiérarchique.

Dans le cas d'une TopoGeometry simple, le premier élément du tableau représente l'identifiant d'une primitive topologique et le deuxième élément représente son type (1:nœud, 2:arête, 3:face). Dans le cas d'une TopoGeometry hiérarchique, le premier élément du tableau représente l'identifiant d'une TopoGeometry enfant et le deuxième élément représente son identifiant de couche.



Note
Pour une TopoGeometry hiérarchique donnée, tous les éléments TopoGeometry enfants proviendront de la même couche enfant, comme spécifié dans l'enregistrement topology.layer pour la couche de la TopoGeometry en cours de définition.

Exemples

Code

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1, 2]::topology.topoelement AS te ) f;
```

Output

```
id | type
----+------
  1 |    2
```

Code

```
SELECT ARRAY[1, 2]::topology.topoelement;
```

Output

```
te
-------
 {1,2}
```

This example shows what happens when casting a three-element array to topology.topoelement. The domain requires a two-element array, so the dimension check fails.

Code

```
SELECT ARRAY[1, 2, 3]::topology.topoelement;
```

Output

```
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

Voir aussi

GetTopoGeomElements, TopoElementArray, TopoGeometry, TopoGeom_addElement, TopoGeom_remElement

9.4.2 TopoElementArray

TopoElementArray — Un tableau d'objets TopoElement.

Description

Un tableau de 1 ou plusieurs objets TopoElement, généralement utilisé pour transmettre des composants d'objets TopoGeometry.

Exemples

Code

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray AS tea;
```

Output

```
{{1,2},{4,3}}
```

This is the more verbose equivalent.

Code

```
SELECT ARRAY[ARRAY[1, 2], ARRAY[4, 3]]::topology.topoelementarray AS tea;
```

Output

```
{{1,2},{4,3}}
```

This example uses the array aggregate function packaged with topology.

Code

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 4) As e CROSS JOIN generate_series(1, 3) As t;
```

Output

```
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

Code

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray AS tea;
```

Output

```
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

Voir aussi

TopoElement, GetTopoGeomElementArray, TopoElementArray_Agg

9.5 Gestion de la topologie et de TopoGeometry

9.5.1 AddTopoGeometryColumn

AddTopoGeometryColumn — Ajoute une colonne topogeometry à une table existante, enregistre cette nouvelle colonne en tant que couche dans topology.layer et renvoie le nouveau numéro d'identification de la couche.

Synopsis

integer **AddTopoGeometryColumn**(name topology_name, name schema_name, name table_name, name column_name, varchar feature_type, integer child_layer);
integer **AddTopoGeometryColumn**(name topology_name, regclass tab, name column_name, integer layer_id, varchar feature_type, integer child_layer);

Description

Chaque objet TopoGeometry appartient à une couche spécifique d'une topologie spécifique. Avant de créer un objet TopoGeometry, il faut créer sa couche topologique. Une couche topologique est une association d'une table de caractéristiques avec la topologie. Elle contient également des informations sur le type et la hiérarchie. Nous créons une couche à l'aide de la fonction AddTopoGeometryColumn() :

Cette fonction ajoutera la colonne demandée à la table et ajoutera un enregistrement à la table topology.layer avec toutes les informations données.

Si vous ne spécifiez pas [child_layer] (ou si vous lui attribuez la valeur NULL), cette couche contiendra des TopoGeometries basiques (composées d'éléments topologiques primitifs). Sinon, cette couche contiendra des TopoGeometries hiérarchiques (composées de TopoGéométries de la couche_enfant).

Une fois la couche créée (son identifiant est renvoyé par la fonction AddTopoGeometryColumn), vous êtes prêt à y construire des objets TopoGeometry

Les `feature_type` valides sont : POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

DIsponibilité : 1.1

Exemples

This example creates the new table in the `ma_topo` schema. The table could be created in a different schema, in which case `topology_name` and `schema_name` would be different.

Code

```
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

Code

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Voir aussi

DropTopoGeometryColumn, toTopoGeom, CreateTopology, CreateTopoGeom

9.5.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Renomme une colonne topogeometry

Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer_table, name feature_column, name new_name);

Description

Cette fonction modifie le nom d'une colonne TopoGeometry existante en veillant à ce que les informations de métadonnées la concernant soient mises à jour en conséquence.

Disponibilité : 3.4.0

Exemples

Code

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

Voir aussi

AddTopoGeometryColumn, RenameTopology

9.5.3 DropTopology

DropTopology — À utiliser avec précaution : Abandonne un schéma de topologie et supprime sa référence dans la table topology.topology et les références aux tables de ce schéma dans la table geometry_columns.

Synopsis

integer **DropTopology**(varchar topology_schema_name);

Description

Abandonne un schéma topologique et supprime sa référence dans la table topology.topology et les références aux tables de ce schéma dans la table geometry_columns. Cette fonction doit être utilisée avec précaution, car elle peut détruire des données qui vous sont chères. Si le schéma n'existe pas, elle supprime simplement les entrées de référence du schéma nommé.

DIsponibilité : 1.1

Exemples

Cascade abandonne le schéma ma_topo et supprime toutes les références à ce schéma dans topology.topology et geometry_columns.

Code

```
SELECT topology.DropTopology('ma_topo');
```

Voir aussi

DropTopoGeometryColumn

9.5.4 RenameTopology

RenameTopology — Renomme une topologie

Synopsis

varchar **RenameTopology**(varchar old_name, varchar new_name);

Description

Renomme un schéma topologique, en mettant à jour son enregistrement de métadonnées dans la table `topology.topology`.

Disponibilité : 3.4.0

Exemples

Renommer une topologie de `topo_stage` à `topo_prod`.

Code

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

Voir aussi

CopyTopology, RenameTopoGeometryColumn

9.5.5 DropTopoGeometryColumn

DropTopoGeometryColumn — Supprime la colonne topogeometry de la table nommée `table_name` dans le schéma `schema_name` et désenregistre les colonnes de la table topology.layer.

Synopsis

text **DropTopoGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);

Description

Supprime la colonne topogeometry de la table nommée `table_name` dans le schéma `schema_name` et désenregistre les colonnes de la table topology.layer. Renvoie un résumé de l'état d'abandon. NOTE : il définit d'abord toutes les valeurs à NULL avant de procéder à l'abandon afin de contourner les contrôles d'intégrité référentielle.

DIsponibilité : 1.1

Exemples

Code

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

Voir aussi

AddTopoGeometryColumn

9.5.6 FixCorruptTopoGeometryColumn

FixCorruptTopoGeometryColumn — Fixes topogeometry corruption caused by upgrade to postgis_topology 3.6.0 and higher

Synopsis

text **FixCorruptTopoGeometryColumn**(name layerSchema, name layerTable, name layerColumn);

Description

When upgrading from PostGIS topology <3.6.0 to version >3.6.0+, the topogeometry column definition was changed. This caused corruption in topogeometries created before the upgrade. This function fixes this corruption in affected tables.

The function refuses to process tables with expression or partial indexes; drop or rebuild those indexes manually around the repair.

Availability: 3.6.1

Exemples

Fix all topology columns

Code

```
SELECT topology.FixCorruptTopoGeometryColumn(schema_name, table_name, feature_column)
    FROM topology.layer;
```

Voir aussi

UpgradeTopology

9.5.7 Populate_Topology_Layer

Populate_Topology_Layer — Ajoute les entrées manquantes à la table topology.layer en lisant les métadonnées des tables topo.

Synopsis

setof record **Populate_Topology_Layer**();

Description

Ajoute les entrées manquantes à la table `topology.layer` en inspectant les contraintes topologiques sur les tables. Cette fonction est utile pour corriger les entrées dans le catalogue topologique après la restauration de schémas contenant des données topographiques.

Elle renvoie la liste des entrées créées. Les colonnes retournées sont `schema_name`, `table_name`, `feature_column`.

Disponibilité : 2.3.0

Exemples

Code

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
```

Output

```
createtopology
----------------
             1
(1 row)

CREATE SCHEMA
CREATE TABLE
 addtopogeometrycolumn
-----------------------
                     1
(1 row)
```

This query returns no records because the feature is already registered.

Code

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

Output

```
(0 rows)
```

Rebuild the topology layer catalog.

Code

```
TRUNCATE TABLE topology.layer;
```

Output

```
TRUNCATE TABLE
```

Code

```
SELECT *
FROM topology.Populate_Topology_Layer();
```

Output

```
schema_name | table_name | feature_column
-------------+------------+----------------
 strk        | parcels    | topo
(1 row)
```

Code

```
SELECT topology_id, layer_id,
       schema_name AS sn, table_name AS tn, feature_column AS fc
FROM topology.layer;
```

Output

```
topology_id | layer_id |  sn  |   tn    |  fc
------------+----------+------+---------+------
          1 |        1 | strk | parcels | topo
(1 row)
```

Voir aussi

AddTopoGeometryColumn

9.5.8 TopologySummary

TopologySummary — Prend un nom de topologie et fournit des totaux récapitulatifs des types d'objets dans la topologie.

Synopsis

text **TopologySummary**(varchar topology_schema_name);

Description

Prend un nom de topologie et fournit des totaux récapitulatifs des types d'objets dans la topologie.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT topology.TopologySummary('city_data');
```

Output

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
 Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
 Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
 Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
 Hierarchy level 1, child layer 1
```

```
 Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
 Hierarchy level 1, child layer 2
 Deploy: features.big_signs.feature
```

9.5.9 ValidateTopology

ValidateTopology — Renvoie un ensemble d'objets validatetopology_returntype détaillant les problèmes liés à la topologie.

Synopsis

setof validatetopology_returntype **ValidateTopology**(varchar toponame, geometry bbox);

Description

Renvoie un ensemble d'objets validatetopology_returntype détaillant les problèmes de topologie, en limitant éventuellement la vérification à la zone spécifiée par le paramètre `bbox`.

La liste des erreurs possibles, ce qu'elles signifient et ce que représentent les identifiants renvoyés est affichée ci-dessous :

Erreur	id1	id2	Signification
nœuds coïncidents	Identifiant du premier nœud.	Identifiant du second nœud.	Deux nœuds ont la même géométrie.
l'arête croise le nœud	Identifiant de l'arête.	Identifiant du nœud.	Une arête a un nœud à l'intérieur. Voir ST_Relate.
arête invalide	Identifiant de l'arête.		La géométrie d'une arête n'est pas valide. Voir ST_IsValid.
l'arête n'est pas simple	Identifiant de l'arête.		Une géométrie d'arête possède des auto-intersections. Voir ST_IsSimple.
une arête croise une arête	Identifiant de la première arête.	Identifiant de la deuxième arête.	Deux arêtes ont une intersection intérieure. Voir ST_Relate.
arête début nœud géométrie inadéquate	Identifiant de l'arête.	Identifiant du nœud de départ indiqué.	La géométrie du nœud indiqué comme nœud de départ d'une arête ne correspond pas au premier point de la géométrie de l'arête. Voir ST_StartPoint.
arête extrémité nœud géométrie non conforme	Identifiant de l'arête.	Identifiant du nœud final indiqué.	La géométrie du nœud indiqué comme nœud final d'une arête ne correspond pas au dernier point de la géométrie de l'arête. Voir ST_EndPoint.
face sans arêtes	Identifiant de la face orpheline.		Aucune arête ne signale une face existante sur l'un ou l'autre de ses côtés (face_gauche, face_droite).
la face n'a pas d'anneaux	Identifiant de la face partiellement définie.		Les arêtes rapportant une face sur leurs côtés ne forment pas un anneau.

Erreur	id1	id2	Signification
la face n'a pas le bon mbr (minimum bounding rectangle)	Identifiant de la face avec un cache mbr (minimum bounding rectangle) erroné.		Le rectangle minimal de délimitation d'une face ne correspond pas au rectangle minimal de délimitation de la collection d'arêtes rapportant la face sur leurs côtés.
le trou n'est pas dans la face annoncée	Identifiant signé d'une arête, identifiant l'anneau. Voir GetRingEdges.		Un anneau d'arêtes rapportant une face sur son extérieur est contenu dans une face différente.
le nœud non isolé a une face non-contenante	Identifiant du nœud mal défini.		Un nœud signalé comme étant à la limite d'une ou plusieurs arêtes indique une face contenante.
le nœud isolé a une face contenante	Identifiant du nœud mal défini.		Un nœud qui n'est pas signalé comme étant à la limite d'une arête n'a pas l'indication d'une face contenante.
Le nœud isolé n'a pas la bonne face contenante	Identifiant du nœud mal représenté.		Un nœud qui n'est pas signalé comme étant à la limite d'une arête indique une face contenante qui n'est pas la face réelle qui le contient. Voir GetFaceContainingPoint.
next_right_edge invalide	Identifiant de l'arête mal représentée.	Identifiant signé de l'arête qui doit être indiqué comme la prochaine arête droite.	L'arête indiquée comme le prochaine arête rencontrée en évoluant sur le côté droit d'une arête est faux.
next_left_edge invalide	Identifiant de l'arête mal représentée.	Identifiant signé de l'arête qui doit être indiquée comme l'arête gauche suivante.	L'arête indiquée comme la prochaine arête rencontrée en évoluant sur le côté gauche d'une arête est faux.
étiquetage mixte des faces dans l'anneau	Identifiant signé d'une arête, identifiant l'anneau. Voir GetRingEdges.		Les arêtes d'un anneau indiquent des faces conflictuelles dans le sens de la progression. C'est ce que l'on appelle un "Side Location Conflict" (Conflit d'emplacement latéral).
anneau non fermé	Identifiant signé d'une arête, identifiant l'anneau. Voir GetRingEdges.		Un anneau d'arêtes formé par les attributs next_left_edge/next_right_edge commence et se termine sur différents nœuds.
la face a plusieurs enveloppes	Identifiant de la face contestée.	Identifiant signé d'une arête, identifiant l'anneau. Voir GetRingEdges.	Plus d'un anneau d'arêtes indique la même face à l'intérieur.

Disponibilité : 1.0.0

Amélioration : 2.0.0 détection plus efficace des croisements d'arêtes et correction des faux positifs qui existaient dans les versions précédentes.

Modifié : 2.2.0 les valeurs pour id1 et id2 ont été échangées pour "edge crosses node" pour être cohérent avec la description de l'erreur.

Modifié : 3.2.0 a ajouté le paramètre optionnel bbox, effectuant des vérifications de l'étiquetage des faces et de la liaison des arêtes.

Exemples

Code

```
SELECT * FROM topology.ValidateTopology('ma_topo');
```

Output

```
error         | id1 | id2
------------------+-----+-----
face without edges |   1 |
```

Voir aussi

validatetopology_returntype

9.5.10 ValidateTopologyRelation

ValidateTopologyRelation — Renvoie des informations sur les enregistrements de relations topologiques non valides

Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

Description

Renvoie un ensemble d'enregistrements donnant des informations sur les invalidités dans la table des relations de la topologie.

Disponibilité : 3.2.0

Voir aussi

ValidateTopology

9.5.11 ValidateTopologyPrecision

ValidateTopologyPrecision — Returns non-precise vertices in the topology.

Synopsis

geometry **ValidateTopologyPrecision**(name toponame, geometry bbox, float8 gridSize);

Description

Returns all vertices that are not rounded to the topology or given `gridSize` as a puntal geometry, optionally limiting the check to the area specified by the `bbox` parameter.

Disponibilité : 3.6.0

Exemples

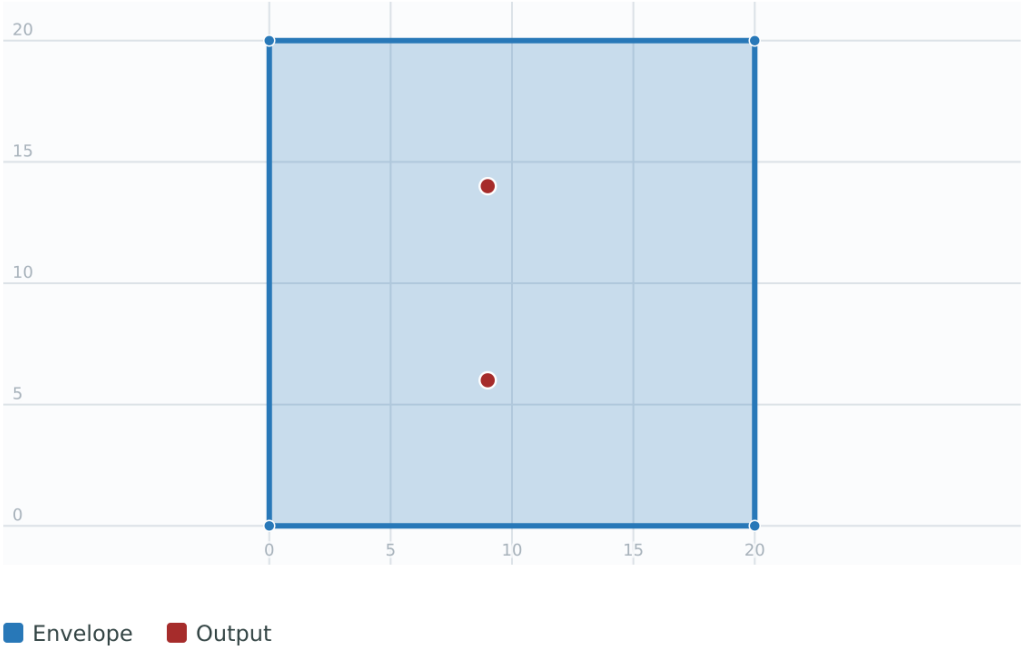
Code

```
SELECT g
FROM topology.ValidateTopologyPrecision(
    'city_data',
    gridSize =
> 2,
    bbox =
> ST_MakeEnvelope(0, 0, 20, 20)
) g;
```

Output

```
MULTIPOINT(9 6,9 14)
```

Figure



Voir aussi

MakeTopologyPrecise

9.5.12 MakeTopologyPrecise

MakeTopologyPrecise — Snap topology vertices to precision grid.

Synopsis

void **MakeTopologyPrecise**(name toponame, geometry bbox, float8 gridSize);

Description

Snaps all vertices of a topology to the topology precision grid or to the grid whose size is specified with the `gridSize` parameter, optionally limiting the operation to the objects intersecting the area specified by the `bbox` parameter.

Using a `gridSize` larger than the smallest edge extent raises an exception rather than collapsing the edge into an empty geometry. This safeguards adjacent topological relationships by preventing precision snapping from removing primitives altogether.



Note
Snapping could make the topology invalid, so it is recommended to check the outcome of operation with ValidateTopology.

Disponibilité : 3.6.0

Exemples

Code

```
SELECT topology.MakeTopologyPrecise(
    'city_data',
    gridSize =
> 2
);
```

Voir aussi

ValidateTopologyPrecision, ValidateTopology

9.5.13 FindTopology

FindTopology — Renvoie un enregistrement topologique par différents moyens.

Synopsis

topology **FindTopology**(topogeometry topogeom);
topology **FindTopology**(regclass layerTable, name layerColumn);
topology **FindTopology**(name layerSchema, name layerTable, name layerColumn);
topology **FindTopology**(text topoName);
topology **FindTopology**(int id);

Description

Prend un identifiant de topologie ou l'identifiant d'un objet lié à la topologie et renvoie un enregistrement topology.topology.

Disponibilité : 3.2.0

Exemples

Code

```
SELECT name(FindTopology('features.land_parcels', 'feature'));
```

Output

```
city_data
```

Voir aussi

FindLayer

9.5.14 FindLayer

FindLayer — Renvoie un enregistrement topology.layer par différents moyens.

Synopsis

topology.layer **FindLayer**(topogeometry tg);
topology.layer **FindLayer**(regclass layer_table, name feature_column);
topology.layer **FindLayer**(name schema_name, name table_name, name feature_column);
topology.layer **FindLayer**(integer topology_id, integer layer_id);

Description

Prend un identifiant de couche ou l'identifiant d'un objet lié à la topologie et renvoie un enregistrement topology.layer.

Disponibilité : 3.2.0

Exemples

Code

```
SELECT layer_id(FindLayer('features.land_parcels', 'feature'));
```

Output

```
1
```

Voir aussi

FindTopology

9.5.15 TotalTopologySize

TotalTopologySize — Total disk space used by the specified topology, including all indexes and TOAST data.

Synopsis

int8 **TotalTopologySize**(name toponame);

Description

Takes a topology name and provides the total disk space used by all its tables, including indexes and TOAST data.

Disponibilité : 3.6.0

Exemples

Code

```
SELECT topology.TopologySummary('city_data');
```

Output

```
Topology city_data (329), SRID 4326, precision: 0
22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
Layer 1, type Polygonal (3), 9 topogeoms
 Deploy: features.land_parcels.feature
Layer 2, type Puntal (1), 8 topogeoms
 Deploy: features.traffic_signs.feature
Layer 3, type Lineal (2), 8 topogeoms
 Deploy: features.city_streets.feature
Layer 4, type Polygonal (3), 3 topogeoms
 Hierarchy level 1, child layer 1
 Deploy: features.big_parcels.feature
Layer 5, type Puntal (1), 1 topogeoms
 Hierarchy level 1, child layer 2
 Deploy: features.big_signs.feature
```

9.5.16 UpgradeTopology

UpgradeTopology — Upgrades the specified topology to support large ids (int8) for topology and primitive ids.

Synopsis

void **UpgradeTopology**(name toponame);

Description

Takes a topology name and upgrades it to support large ids (int8) for topology and primitive ids. The function upgrades the following: - face (face_id column from int4 to int8, face_id_seq from int4 to int8) - node (node_id column from int4 to int8, containing_face column from int4 to int8, node_id_seq from int4 to int8) - edge_data (edge_id column from int4 to int8, edge_data_edge_id_seq from int4 to int8, left_face and right_face columns from int4 to int8, start_node and end_node columns from int4 to int8, next_left_edge and next_right_edge columns from int4 to int8) - relation (topogeo_id column from int4 to int8, element_id from int4 to int8) - topology (useslargeids column set to true)

Disponibilité : 3.6.0

Exemples

Code

```
SELECT topology.UpgradeTopology('city_data');
```

9.6 Gestion des statistiques de topologie

L'ajout d'éléments à une topologie déclenche de nombreuses requêtes dans la base de données pour trouver les arêtes existantes qui seront divisées, ajouter des nœuds et mettre à jour les arêtes qui formeront un nœud avec le nouveau réseau de lignes. C'est pourquoi il est utile que les statistiques relatives aux données contenues dans les tables de topologie soient à jour.

Les fonctions d'insertion et d'édition de topologie de PostGIS ne mettent pas automatiquement à jour les statistiques, car une mise à jour des statistiques après chaque changement dans une topologie serait exagérée, et c'est donc à l'appelant de s'en charger.



Note
Les statistiques mises à jour par autovacuum ne seront PAS visibles pour les transactions qui ont démarré avant la fin du processus d'autovacuum, de sorte que les transactions de longue durée devront exécuter ANALYZE elles-mêmes, pour utiliser les statistiques mises à jour.

9.7 Constructeurs de topologie

9.7.1 CreateTopology

CreateTopology — Crée un nouveau schéma topologique et l'enregistre dans la table topology.topology.

Synopsis

integer **CreateTopology**(name topology_schema_name, integer srid, double precision prec=0, boolean hasz=false, integer topoid=0, boolean useslargeids=false);

Description

Crée un nouveau schéma de topologie portant le nom `topology_name` et l'enregistre dans la table `topology.topology`. Les topologies doivent porter un nom unique. Les tables de topologie (`edge_data`, `face`, `node` et `relation`) sont créées dans le schéma. Elle renvoie l'identifiant de la topologie.

The `srid` is the spatial reference system SRID for the topology. The SRID defaults to -1 (unknown) if not specified.

La tolérance `prec` est mesurée dans les unités du système de référence spatiale. La tolérance est fixée par défaut à 0.

`hasz` La valeur par défaut est false si elle n'est pas spécifiée.

`topoid` optional explicit identifier (allows deterministic topology id assignment, needs to be unique)

`useslargeids` optional, defaults to false. If true, the topology will be created to support large ids (int8) for topology and primitive ids.

Cette fonction est similaire à la fonction SQL/MM ST_InitTopoGeo mais possède plus de fonctionnalités.

DIsponibilité : 1.1

Enhanced: 2.0 added support for `hasZ`.

Enhanced: 3.6.0 added `topoid` and `usesLargeIDs` parameters.

Exemples

Créez un schéma topologique appelé `ma_topo` qui stocke les arêtes et les nœuds en mètres de Massachusetts State Plane (SRID = 26986). La tolérance est de 0,5 mètre puisque le système de référence spatiale est basé sur le mètre.

Code

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Créer une topologie pour Rhode Island appelée `ri_topo` dans le système de référence spatiale State Plane-feet (SRID = 3438)

Code

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
```

Output

```
2
```

Voir aussi

Section 4.5, ST_InitTopoGeo

9.7.2 CopyTopology

CopyTopology — Copie une topologie (nœuds, arêtes, faces, couches et TopoGeometries) dans un nouveau schéma

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Description

Crée une nouvelle topologie avec le nom `new_name`, avec le SRID et la précision copiés à partir de `existing_topology_name` Les nœuds, les arêtes et les faces dans `existing_topology_name` sont copiés dans la nouvelle topologie, ainsi que les couches et leurs TopoGeometries associées.



Note
Les nouvelles lignes de la table `topology.layer` contiennent des valeurs synthétiques pour schema_name, table_name et feature_column. Ceci est dû au fait que les objets TopoGeometry n'existent qu'en tant que définition et ne sont pas encore disponibles dans une table définie par l'utilisateur.

Disponibilité : 2.0.0

Exemples

Faire une sauvegarde d'une topologie appelée `ma_topo`.

Code

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

Voir aussi

Section 4.5, CreateTopology, RenameTopology

9.7.3 ST_InitTopoGeo

ST_InitTopoGeo — Crée un nouveau schéma topologique et l'enregistre dans la table topology.topology.

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

Description

C'est l'équivalent SQL-MM de CreateTopology. Il manque des options pour le système de référence spatial et la tolérance. Il renvoie une description textuelle de la création de la topologie, au lieu de l'identifiant de la topologie.

DIsponibilité : 1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 Topo-Geo et Topo-Net 3 : Routine Details : X.3.17

Exemples

Code

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
```

Output

```
Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

Voir aussi

CreateTopology

9.7.4 ST_CreateTopoGeo

ST_CreateTopoGeo — Ajoute une collection de géométries à une topologie vide donnée et renvoie un message détaillant le succès.

Synopsis

text **ST_CreateTopoGeo**(varchar atopology, geometry acollection);

Description

Ajoute une collection de géométries à une topologie vide donnée et renvoie un message détaillant le succès.

Utile pour remplir une topologie vide.

Disponibilité : 2.0

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details -- X.3.18

Exemples

Populate the topology.

Code

```
SELECT topology.ST_CreateTopoGeo(
'ri_topo',
ST_GeomFromText(
 'MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799 236895,384811 ↩
    236890,384833 236884,
 384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
 385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
 385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
 385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
 385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
 385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
 385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
 385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
 385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291 237596,385284 ↩
    237630))',
 3438)
 );
```

Output

```
Topology ri_topo populated
```

Create the tables and topogeometries.

Code

```
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Voir aussi

TopoGeo_LoadGeometry, AddTopoGeometryColumn, CreateTopology, DropTopology

9.7.5 TopoGeo_AddPoint

TopoGeo_AddPoint — Ajoute un point à une topologie existante en utilisant une tolérance et en divisant éventuellement une arête existante.

Synopsis

bigint **TopoGeo_AddPoint**(varchar atopology, geometry apoint, float8 tolerance);

Description

Ajoute un point à une topologie existante et renvoie son identifiant. Le point donné s'accroche aux nœuds ou aux arêtes existants dans la limite d'une tolérance donnée. Une arête existante peut être divisée par le point accroché.

Disponibilité : 2.0.0

Voir aussi

TopoGeo_AddLineString, TopoGeo_AddPolygon, TopoGeo_LoadGeometry, AddNode, CreateTopology

9.7.6 TopoGeo_AddLineString

TopoGeo_AddLineString — Ajoute une ligne à une topologie existante en utilisant une tolérance et en divisant éventuellement les arêtes/faces existantes.

Synopsis

SETOF bigint **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance, int max_edges);

Description

Adds a linestring to an existing topology and returns a set of signed edge identifiers forming it up (negative identifies mean the edge goes in the opposite direction of the input linestring).

The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line. New nodes and faces may be added, in addition to new edges.

The returned edge identifiers may be either existing edges or newly created edges as needed to fully represent the input line as closely as possible.

The number of newly created edges (either covering space previously uncovered or resulting from split of existing edges) may be limited by the `max_edges` parameter.



Note
When adding many lines, the load order can have a large impact on performance because each call checks the line against topology elements already loaded. In bulk workflows, adding large closed or enclosing lines before smaller contained lines can avoid expensive later splits. For loading a collection in one call, see TopoGeo_LoadGeometry.



Note
La mise à jour des statistiques sur les topologies chargées via cette fonction est du ressort de l'appelant, voir maintaining statistics during topology editing and population.

Disponibilité : 2.0.0

Amélioration : la version 3.2.0 a ajouté la possibilité de renvoyer un identifiant signé.

Enhanced: 3.7.0 added support for limiting the number of new edges created in the topology.

Voir aussi

TopoGeo_AddPoint, TopoGeo_AddPolygon, TopoGeo_LoadGeometry, AddEdge, CreateTopology

9.7.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Ajoute un polygone à une topologie existante en utilisant une tolérance et en divisant éventuellement les arêtes/faces existantes. Renvoie les identifiants des faces.

Synopsis

SETOF bigint **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

Description

Ajoute un polygone à une topologie existante et renvoie un ensemble d'identifiants de face le constituant. La limite du polygone donné s'accrochera aux nœuds ou aux arêtes existants dans la limite d'une tolérance donnée. Les arêtes et les faces existantes peuvent être divisées par la limite du nouveau polygone.



Note
La mise à jour des statistiques sur les topologies chargées via cette fonction est du ressort de l'appelant, voir maintaining statistics during topology editing and population.

Disponibilité : 2.0.0

Voir aussi

TopoGeo_AddPoint, TopoGeo_AddLineString, TopoGeo_LoadGeometry, AddFace, CreateTopology

9.7.8 TopoGeo_LoadGeometry

TopoGeo_LoadGeometry — Charge une géométrie dans une topologie existante, en effectuant de l'accrochage et des divisions si nécessaire.

Synopsis

void **TopoGeo_LoadGeometry**(varchar atopology, geometry ageom, float8 tolerance);

Description

Charge une géométrie dans une topologie existante. La géométrie donnée s'accrochera aux nœuds ou aux arêtes existantes dans la limite d'une tolérance donnée. Les arêtes et les faces existantes peuvent être divisées suite au chargement.



Note
La mise à jour des statistiques sur les topologies chargées via cette fonction est du ressort de l'appelant, voir maintaining statistics during topology editing and population.

Disponibilité : 3.5.0

Voir aussi

TopoGeo_AddPoint, TopoGeo_AddLineString, TopoGeo_AddPolygon, CreateTopology

9.8 Éditeurs de topologie

9.8.1 ST_AddIsoNode

ST_AddIsoNode — Ajoute un noeud isolé à une face dans une topologie et renvoie le nodeid du nouveau noeud. Si la face est nulle, le noeud est quand même créé.

Synopsis

bigint **ST_AddIsoNode**(varchar atopology, bigint aface, geometry apoint);

Description

Ajoute un noeud isolé avec un point `apoint` à une face existante avec un faceid `aface` à une topologie `atopology` et renvoie le nodeid du nouveau noeud.

Si le système de référence spatiale (srid) de la géométrie du point n'est pas le même que la topologie, si le `apoint` n'est pas une géométrie de point, si le point est nul ou si le point croise une arête existante (même aux limites), une exception est levée. Si le point existe déjà en tant que nœud, une exception est levée.

Si `aface` n'est pas null et que le `apoint` n'est pas à l'intérieur de la face, une exception est levée.

DIsponibilité : 1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Net Routines : X+1.3.1

Exemples

Voir aussi

AddNode, CreateTopology, DropTopology, ST_Intersects

9.8.2 ST_AddIsoEdge

ST_AddIsoEdge — Ajoute une arête isolée définie par la géométrie `alinestring` à une topologie reliant deux nœuds isolés existants `anode` et `anothernode` et renvoie l'identifiant de l'arête de la nouvelle arête.

Synopsis

bigint **ST_AddIsoEdge**(varchar atopology, bigint anode, bigint anothernode, geometry alinestring);

Description

Ajoute une arête isolée définie par la géométrie `alinestring` à une topologie reliant deux nœuds isolés existants `anode` et `anothernode` et renvoie l'identifiant de l'arête de la nouvelle arête.

Si le système de référence spatiale (srid) de la géométrie `alinestring` n'est pas le même que la topologie, si l'un des arguments d'entrée est nul, si les noeuds sont contenus dans plus d'une face, ou si les noeuds sont des noeuds de départ ou d'arrivée d'une arête existante, une exception est levée.

Si le `alinestring` ne se trouve pas à l'intérieur de la face à laquelle appartiennent le `anode` et le `anothernode`, une exception est levée.

Si le `anode` et le `anothernode` ne sont pas les points de départ et d'arrivée du `alinestring`, une exception est levée.

DIsponibilité : 1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.4

Exemples

Voir aussi

ST_AddIsoNode, ST_IsSimple, ST_Within

9.8.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — Ajoutez une nouvelle arête et, si elle divise une face, supprimez la face d'origine et remplacez-la par deux nouvelles faces.

Synopsis

bigint **ST_AddEdgeNewFaces**(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

Description

Ajoutez une nouvelle arête et, si elle divise une face, supprimez la face d'origine et remplacez-la par deux nouvelles faces. Renvoie l'identifiant de l'arête nouvellement ajoutée.

Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Si l'un des arguments est nul, les nœuds donnés sont inconnus (doivent déjà exister dans la table `node` du schéma topologique), la `acurve` n'est pas une `LINESTRING`, le `anode` et `anothernode` ne sont pas les points de départ et d'arrivée de `acurve`, une erreur est générée.

Si le système de référence spatiale (srid) de la géométrie `acurve` n'est pas le même que la topologie, une exception est levée.

Disponibilité : 2.0

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.12

Exemples

Voir aussi

ST_RemEdgeNewFace

ST_AddEdgeModFace

9.8.4 ST_AddEdgeModFace

ST_AddEdgeModFace — Ajoutez une nouvelle arête et, si elle divise une face, modifiez la face d'origine et ajoutez une nouvelle face.

Synopsis

bigint **ST_AddEdgeModFace**(varchar atopology, bigint anode, bigint anothernode, geometry acurve);

Description

Ajouter une nouvelle arête et, si cela divise une face, modifier la face d'origine et en ajouter une nouvelle.



Note
Si possible, la nouvelle face sera créée sur le côté gauche de la nouvelle arête. Cela ne sera pas possible si la face du côté gauche doit être la face de l'univers (non borné).

Renvoie l'identifiant de l'arête nouvellement ajoutée.

Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Si l'un des arguments est nul, les nœuds donnés sont inconnus (doivent déjà exister dans la table `node` du schéma topologique), la `acurve` n'est pas une `LINESTRING`, le `anode` et `anothernode` ne sont pas les points de départ et d'arrivée de `acurve`, une erreur est générée.

Si le système de référence spatiale (srid) de la géométrie `acurve` n'est pas le même que la topologie, une exception est levée.

Disponibilité : 2.0

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.13

Exemples

Voir aussi

ST_RemEdgeModFace

ST_AddEdgeNewFaces

9.8.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — Enlève une arête et, si l'arête enlevée séparait deux faces, supprime les faces originales et les remplace par une nouvelle face.

Synopsis

bigint **ST_RemEdgeNewFace**(varchar atopology, bigint anedge);

Description

Supprime une arête et, si l'arête supprimée séparait deux faces, supprime les faces originales et les remplace par une nouvelle face.

Renvoie l'identifiant d'une face nouvellement créée ou NULL, si aucune nouvelle face n'est créée. Aucune nouvelle face n'est créée lorsque l'arête supprimée est pendante, isolée ou confinée avec la face de l'univers (ce qui peut entraîner "l'inondation de la face" de l'autre côté de l'univers).

Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Refuse de supprimer une arête participant à la définition d'une TopoGeometry existante. Refuse de recoller deux faces si une TopoGeometry n'est définie que par l'une d'entre elles (et non par l'autre).

Si l'un des arguments est nul, si l'arête donnée est inconnue (elle doit déjà exister dans la table `edge` du schéma topologique), si le nom de la topologie n'est pas valide, une erreur est générée.

Disponibilité : 2.0

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.14

Exemples

Voir aussi

ST_RemEdgeModFace

ST_AddEdgeNewFaces

9.8.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Supprime une arête et, si l'arête sépare deux faces, supprime une face et modifie l'autre face pour couvrir l'espace des deux.

Synopsis

bigint **ST_RemEdgeModFace**(varchar atopology, bigint anedge);

Description

Supprime une arête et, si l'arête supprimée sépare deux faces, supprime une face et modifie l'autre face pour couvrir l'espace des deux. La face de droite est conservée de préférence, par souci de cohérence avec ST_AddEdgeModFace. Renvoie l'identifiant de la face conservée.

Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Refuse de supprimer une arête participant à la définition d'une TopoGeometry existante. Refuse de recoller deux faces si une TopoGeometry n'est définie que par l'une d'entre elles (et non par l'autre).

Si l'un des arguments est nul, si l'arête donnée est inconnue (elle doit déjà exister dans la table `edge` du schéma topologique), si le nom de la topologie n'est pas valide, une erreur est générée.

Disponibilité : 2.0

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.15

Exemples

Voir aussi

ST_AddEdgeModFace

ST_RemEdgeNewFace

9.8.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — Modifie la forme d'une arête sans affecter la structure de la topologie.

Synopsis

text **ST_ChangeEdgeGeom**(varchar atopology, bigint anedge, geometry acurve);

Description

Modifie la forme d'une arête sans affecter la structure de la topologie.

Si l'un des arguments est nul, si l'arête donnée n'existe pas dans la table `edge` du schéma topologique, si la `acurve` n'est pas une `LINESTRING`, ou si la modification changerait la topologie sous-jacente, une erreur est déclenchée.

Si le système de référence spatiale (srid) de la géométrie `acurve` n'est pas le même que la topologie, une exception est levée.

Si la nouvelle `acurve` n'est pas simple, une erreur est générée.

Si le déplacement de l'arête de l'ancienne à la nouvelle position heurte un obstacle, une erreur est générée.

Disponibilité : 1.1.0

Amélioration : 2.0.0 ajoute l'application de la cohérence topologique

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details X.3.6

Exemples

Code

```
SELECT topology.ST_ChangeEdgeGeom(
    'ma_topo',
    1,
        ST_GeomFromText(
            'LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6 893816.6,227704.5 ↩
    893778.5)',
            26986)
);
```

Output

```
Edge 1 changed
```

Voir aussi

ST_AddEdgeModFace

ST_RemEdgeModFace

ST_ModEdgeSplit

9.8.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Fractionner une arête en créant un nouveau nœud le long d'une arête existante, en modifiant l'arête d'origine et en ajoutant une nouvelle arête.

Synopsis

bigint **ST_ModEdgeSplit**(varchar atopology, bigint anedge, geometry apoint);

Description

Fractionner une arête en créant un nouveau nœud le long d'une arête existante, en modifiant l'arête d'origine et en ajoutant une nouvelle arête. Cette opération met à jour toutes les arêtes jointes existantes et les relations en conséquence. Renvoie l'identifiant du nœud nouvellement ajouté.

DIsponibilité : 1.1

Modifié : 2.0 - Dans les versions antérieures, cette fonction était mal nommée ST_ModEdgesSplit

 Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.9

Exemples

Add an edge to split.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910,227600 893910) ↩
    ', 26986)) AS edgeid;
```

Output

```
3
```

Split the edge with a new node.

Code

```
SELECT topology.ST_ModEdgeSplit('ma_topo', 3, ST_SetSRID(ST_Point(227594, 893910), 26986)) ↩
    AS node_id;
```

Output

```
7
```

Voir aussi

ST_NewEdgesSplit, ST_ModEdgeHeal, ST_NewEdgeHeal, AddEdge

9.8.9 ST_ModEdgeHeal

ST_ModEdgeHeal — Répare deux arêtes en supprimant le nœud qui les relie, en modifiant la première arête et en supprimant la seconde. Renvoie l'identifiant du nœud supprimé.

Synopsis

bigint **ST_ModEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

Description

Répare deux arêtes en supprimant le nœud qui les relie, en modifiant la première arête et en supprimant la seconde. Renvoie l'identifiant du nœud supprimé. Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Disponibilité : 2.0

 Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.9

Voir aussi

ST_ModEdgeSplit ST_NewEdgesSplit

9.8.10 ST_NewEdgeHeal

ST_NewEdgeHeal — Répare deux arêtes en supprimant le nœud qui les relie, en supprimant les deux arêtes et en les remplaçant par une arête dont la direction est la même que la première arête fournie.

Synopsis

bigint **ST_NewEdgeHeal**(varchar atopology, bigint anedge, bigint anotheredge);

Description

Répare deux arêtes en supprimant le nœud qui les relie, en supprimant les deux arêtes et en les remplaçant par une arête dont la direction est la même que la première arête fournie. Renvoie l'identifiant de la nouvelle arête remplaçant les arêtes réparées. Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Disponibilité : 2.0

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X.3.9

Voir aussi

ST_ModEdgeHeal ST_ModEdgeSplit ST_NewEdgesSplit

9.8.11 ST_MoveIsoNode

ST_MoveIsoNode — Déplace un nœud isolé dans une topologie d'un point à un autre. Si la nouvelle géométrie `apoint` existe en tant que noeud, une erreur est générée. Retourne la description du déplacement.

Synopsis

text **ST_MoveIsoNode**(varchar atopology, bigint anode, geometry apoint);

Description

Déplace un nœud isolé dans une topologie d'un point à un autre. Si la nouvelle géométrie `apoint` existe en tant que noeud, une erreur est générée.

Si l'un des arguments est nul, si le `apoint` n'est pas un point, si le noeud existant n'est pas isolé (c'est un point de départ ou d'arrivée d'une arête existante), si l'emplacement du nouveau noeud coupe une arête existante (même aux points d'arrivée) ou si le nouvel emplacement se trouve dans une face différente (depuis la version 3.2.0), une exception est levée.

Si le système de référence spatiale (srid) de la géométrie du point n'est pas le même que la topologie, une exception est levée.

Disponibilité : 2.0.0

Amélioration : la version 3.2.0 garantit que le nœud ne peut pas être déplacé vers une autre face

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Net Routines: X.3.2

Exemples

Add an isolated node with no face.

Code

```
SELECT topology.ST_AddIsoNode('ma_topo', NULL, ST_GeomFromText('POINT(227579 893916)', ↩
    26986)) AS nodeid;
```

Output

```
7
```

Move the new node.

Code

```
SELECT topology.ST_MoveIsoNode('ma_topo', 7, ST_GeomFromText('POINT(227579.5 893916.5)', ↩
    26986)) AS descrip;
```

Output

```
Isolated Node 7 moved to location 227579.5,893916.5
```

Voir aussi

ST_AddIsoNode

9.8.12 ST_NewEdgesSplit

ST_NewEdgesSplit — Fractionne une arête en créant un nouveau nœud le long d'une arête existante, en supprimant l'arête d'origine et en la remplaçant par deux nouvelles arêtes. Renvoie l'identifiant du nouveau nœud créé qui relie les nouvelles arêtes.

Synopsis

bigint **ST_NewEdgesSplit**(varchar atopology, bigint anedge, geometry apoint);

Description

Divise une arête avec l'identifiant `anedge` en créant un nouveau noeud avec l'emplacement du point `apoint` le long de l'arête actuelle, en supprimant l'arête d'origine et en la remplaçant par deux nouvelles arêtes. Renvoie l'identifiant du nouveau nœud créé qui relie les nouvelles arêtes. Met à jour toutes les arêtes jointes existantes et les relations en conséquence.

Si le système de référence spatiale (srid) de la géométrie du point n'est pas le même que la topologie, si le `apoint` n'est pas une géométrie de point, si le point est nul, si le point existe déjà en tant que nœud, si l'arête ne correspond pas à une arête existante ou si le point n'est pas à l'intérieur de l'arête, une exception est levée.

DIsponibilité : 1.1

 Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Net Routines: X.3.8

Exemples

Add an edge to split.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) ↩
    ', 26986)) AS edgeid;
```

Output

```
2
```

Split the new edge.

Code

```
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', ↩
    26986)) AS newnodeid;
```

Output

```
6
```

Voir aussi

ST_ModEdgeSplit ST_ModEdgeHeal ST_NewEdgeHeal AddEdge

9.8.13 ST_RemoveIsoNode

ST_RemoveIsoNode — Supprime un noeud isolé et renvoie la description de l'action. Si le noeud n'est pas isolé (début ou fin d'une arête), une exception est levée.

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, bigint anode);

Description

Supprime un noeud isolé et renvoie la description de l'action. Si le nœud n'est pas isolé (est le début ou la fin d'une arête), une exception est levée.

DIsponibilité : 1.1

 Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X+1.3.3

Exemples

Remove an isolated node with no face.

Code

```
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) AS result;
```

Output

```
Isolated node 7 removed
```

Voir aussi

ST_AddIsoNode

9.8.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — Supprime une arête isolée et renvoie la description de l'action. Si l'arête n'est pas isolée, une exception est levée.

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, bigint anedge);

Description

Supprime une arête isolée et renvoie la description de l'action. Si l'arête n'est pas isolée, une exception est levée.

DIsponibilité : 1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM: Topo-Geo et Topo-Net 3: Routine Details: X+1.3.3

Exemples

Remove an isolated edge with no face.

Code

```
SELECT topology.ST_RemoveIsoEdge('ma_topo', 7) AS result;
```

Output

```
Isolated edge 7 removed
```

Voir aussi

ST_AddIsoNode

9.9 Accès à la topologie

9.9.1 GetEdgeByPoint

GetEdgeByPoint — Trouve l'identifiant d'une arête qui coupe un point donné.

Synopsis

bigint **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Récupère l'identifiant d'une arête qui coupe un point.

La fonction renvoie un entier (id-edge) à partir d'une topologie, d'un POINT et d'une tolérance. Si la tolérance = 0, le point doit couper l'arête.

Si `apoint` n'intersecte pas une arête, renvoie 0 (zéro).

Si la tolérance d'utilisation > 0 et qu'il y a plus d'une arête près du point, une exception est levée.

Note
Si la tolérance = 0, la fonction utilise ST_Intersects, sinon elle utilise ST_DWithin.

Effectué par le module GEOS.

Disponibilité : 2.0.0

Exemples

Ces exemples utilisent les arêtes que nous avons créées dans AddEdge

Code

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As with1mtol, topology.GetEdgeByPoint(' ↩
    ma_topo', geom, 0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
```

Output

```
with1mtol | withnotol
-----------+-----------
         2 |          0
```

This query errors because two or more edges are found.

Code

```
SELECT topology.GetEdgeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR:  Two or more edges found
```

Voir aussi

AddEdge, GetNodeByPoint, GetFaceByPoint

9.9.2 GetFaceByPoint

GetFaceByPoint — Recherche la face intersectant un point donné.

Synopsis

bigint **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Recherche une face référencée par un point, avec une tolérance donnée.

La fonction recherchera effectivement une face coupant un cercle dont le point est le centre et la tolérance le rayon.

Si aucune face n'intersecte l'emplacement de la requête, la valeur 0 est renvoyée (face universelle).

Si plus d'une face croise l'emplacement de la requête, une exception est levée.

Disponibilité : 2.0.0

Amélioration : 3.2.0 mise en œuvre plus efficace et contrat plus clair, arrêt du fonctionnement avec des topologies non valides.

Exemples

Code

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 10) As with1mtol, topology.GetFaceByPoint(' ↩
    ma_topo', geom, 0) As withnotol
FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;
```

Output

```
with1mtol | withnotol
-----------+-----------
         1 |         0
```

This query errors because two or more faces are found.

Code

```
SELECT topology.GetFaceByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR:  Two or more faces found
```

Voir aussi

GetFaceContainingPoint, AddFace, GetNodeByPoint, GetEdgeByPoint

9.9.3 GetFaceContainingPoint

GetFaceContainingPoint — Recherche la face contenant un point.

Synopsis

bigint **GetFaceContainingPoint**(text atopology, geometry apoint);

Description

Renvoie l'identifiant de la face contenant un point.

Une exception est levée si le point tombe sur la limite d'une face.



Note
La fonction repose sur une topologie valide, utilisant la liaison des arêtes et l'étiquetage des faces.

Disponibilité : 3.2.0

Voir aussi

ST_GetFaceGeometry

9.9.4 GetNodeByPoint

GetNodeByPoint — Recherche l'identifiant d'un nœud à un point donné.

Synopsis

bigint **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Description

Récupère l'identifiant d'un nœud à un point donné.

La fonction renvoie un entier (id-node) à partir d'une topologie, d'un POINT et d'une tolérance. Si tolérance = 0, il s'agit d'une intersection exacte, sinon le nœud est extrait d'un intervalle.

Si `apoint` n'intersecte pas un noeud, renvoie 0 (zéro).

Si la tolérance d'utilisation > 0 et qu'il y a plus d'un nœud près du point, une exception est levée.



Note
Si la tolérance = 0, la fonction utilise ST_Intersects, sinon elle utilise ST_DWithin.

Effectué par le module GEOS.

Disponibilité : 2.0.0

Exemples

Ces exemples utilisent les arêtes que nous avons créées dans AddEdge

Code

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
2
```

This query errors because the tolerance matches two or more nodes.

Code

```
SELECT topology.GetNodeByPoint('ma_topo', geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
```

Output

```
ERROR:  Two or more nodes found
```

Voir aussi

AddEdge, GetEdgeByPoint, GetFaceByPoint

9.9.5 GetTopologyID

GetTopologyID — Retourne l'identifiant d'une topologie dans la table topology.topology étant donné le nom de la topologie.

Synopsis

integer **GetTopologyID**(varchar toponame);

Description

Retourne l'identifiant d'une topologie dans la table topology.topology en fonction du nom de la topologie.

DIsponibilité : 1.1

Exemples

Code

```
SELECT topology.GetTopologyID('ma_topo') AS topo_id;
```

Output

```
1
```

Voir aussi

CreateTopology, DropTopology, GetTopologyName, GetTopologySRID

9.9.6 GetTopologySRID

GetTopologySRID — Renvoie le SRID d'une topologie dans la table topology.topology en fonction du nom de la topologie.

Synopsis

integer **GetTopologyID**(varchar toponame);

Description

Renvoie l'identifiant de référence spatiale d'une topologie dans la table topology.topology en fonction du nom de la topologie.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT topology.GetTopologySRID('ma_topo') AS srid;
```

Output

```
4326
```

Voir aussi

CreateTopology, DropTopology, GetTopologyName, GetTopologyID

9.9.7 GetTopologyName

GetTopologyName — Renvoie le nom d'une topologie (schéma) en fonction de l'identifiant de la topologie.

Synopsis

varchar **GetTopologyName**(integer topology_id);

Description

Renvoie le nom de la topologie (schéma) d'une topologie de la table topology.topology en fonction de l'identifiant de la topologie.

DIsponibilité : 1.1

Exemples

Code

```
SELECT topology.GetTopologyName(1) AS topo_name;
```

Output

```
ma_topo
```

Voir aussi

CreateTopology, DropTopology, GetTopologyID, GetTopologySRID

9.9.8 ST_GetFaceEdges

ST_GetFaceEdges — Renvoie un ensemble d'arêtes ordonnées qui délimitent `aface`.

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, bigint aface);

Description

Renvoie un ensemble d'arêtes ordonnées qui délimitent `aface`. Chaque sortie se compose d'une séquence et d'un numéro d'arête. Les numéros de séquence commencent par la valeur 1.

L'énumération des arêtes de chaque anneau commence par l'arête dont l'identifiant est le plus petit. L'ordre des arêtes suit la règle de la main gauche (la face liée se trouve à gauche de chaque arête dirigée).

Disponibilité : 2.0

 Cette méthode implémente la spécification SQL/MM. SQL-MM 3 Topo-Geo et Topo-Net 3: Routine Details: X.3.5

Exemples

This query returns the edges that bound face 1.

Code

```
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
```

Output

```
sequence | edge
----------+------
         1 |   -4
         2 |    5
         3 |    7
         4 |   -6
         5 |    1
         6 |    2
         7 |    3
```

This query returns the sequence, edge identifier, and geometry of the edges that bound face 1. Use ST_GetFaceGeometry if only the geometry and sequence are needed.

Code

```
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt', 1) As t(seq, edge)
    INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

Voir aussi

GetRingEdges, AddFace, ST_GetFaceGeometry

9.9.9 ST_GetFaceGeometry

ST_GetFaceGeometry — Renvoie le polygone dans la topologie donnée avec l'identifiant de face spécifié.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, bigint aface);

Description

Renvoie le polygone dans la topologie donnée avec l'identifiant de face spécifié. Construit le polygone à partir des arêtes composant la face.

DIsponibilité : 1.1

Cette méthode implémente la spécification SQL/MM. SQL-MM 3 Topo-Geo et Topo-Net 3: Routine Details: X.3.16

Exemples

This query returns the WKT of the polygon added with AddFace.

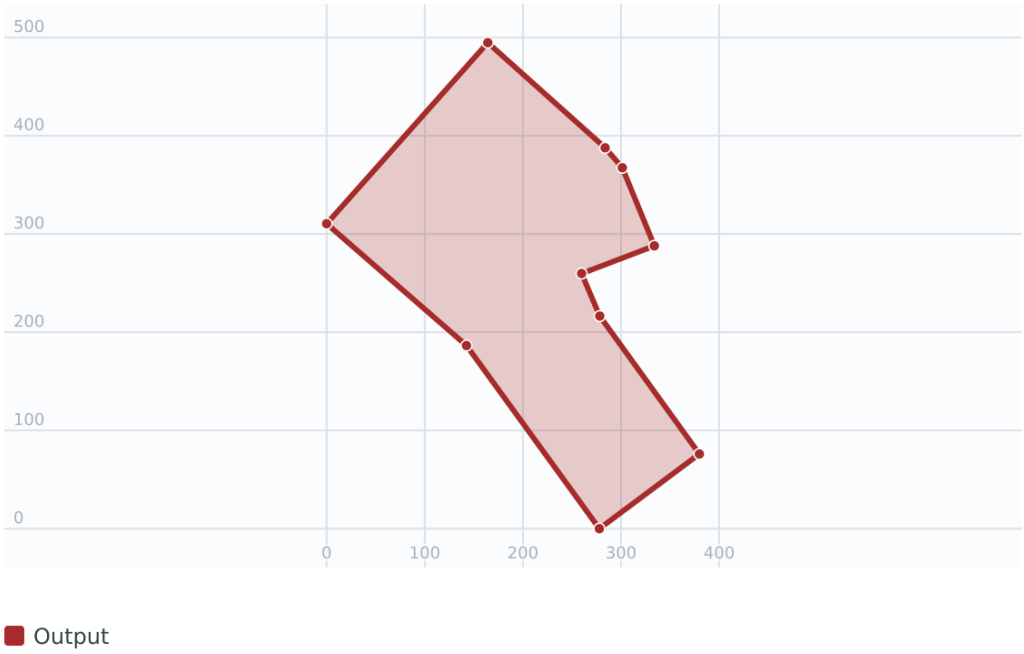
Code

```
SELECT topology.ST_GetFaceGeometry('ma_topo', 1) AS facegeomwkt;
```

Output

```
POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

Figure



Voir aussi

AddFace

9.9.10 GetRingEdges

GetRingEdges — Renvoie l'ensemble ordonné des identifiants d'arêtes signés rencontrés en évoluant sur un côté d'arête donné.

Synopsis

getfaceedges_returntype **GetRingEdges**(varchar atopology, bigint aring, integer max_edges=null);

Description

Renvoie l'ensemble ordonné des identifiants d'arêtes signés rencontrés en évoluant sur un côté d'arête donné. Chaque sortie se compose d'une séquence et d'un identifiant d'arête signé. Les numéros de séquence commencent par la valeur 1.

Si vous passez un identifiant d'arête positif, la marche commence du côté gauche de l'arête correspondante et suit la direction de l'arête. Si vous passez un identifiant d'arête négatif, la marche commence du côté droit de l'arête correspondante et va vers l'arrière.

Si `max_edges` n'est pas nul, cette fonction ne renvoie pas plus que ces enregistrements. Il s'agit d'un paramètre de sécurité dans le cas de topologies éventuellement non valides.



Note
Cette fonction utilise les métadonnées de liaison des anneaux des arêtes.

Disponibilité : 2.0.0

Voir aussi

ST_GetFaceEdges, GetNodeEdges

9.9.11 GetNodeEdges

GetNodeEdges — Renvoie un ensemble ordonné d'arêtes incidentes au nœud donné.

Synopsis

getfaceedges_returntype **GetNodeEdges**(varchar atopology, bigint anode);

Description

Renvoie un ensemble ordonné d'arêtes incidentes au nœud donné. Chaque sortie se compose d'une séquence et d'un numéro d'arête signé. Les numéros de séquence commencent par la valeur 1. Une arête positive commence au nœud donné. Une arête négative se termine au nœud donné. Les arêtes fermées apparaissent deux fois (avec les deux signes). L'ordre est celui des aiguilles d'une montre en partant du nord.



Note
Cette fonction calcule l'ordre plutôt que de dériver des métadonnées et peut donc être utilisée pour établir des liens entre les anneaux des arêtes.

Disponibilité : 2.0

Voir aussi

getfaceedges_returntype, GetRingEdges, ST_Azimuth

9.9.12 FindVertexSegmentPairsBelowDistance

FindVertexSegmentPairsBelowDistance — Find pairs of topology vertex/segment that are closer than tolerated distance

Synopsis

setof record **FindVertexSegmentPairsBelowDistance**(varchar atopology, float8 tolerance);

Description

Returns a relation in which each tuple is a pair of vertex/segment where the vertex is NOT an endpoint of the segment but the distance between them is below the given tolerance.

Availability: 3.7

Voir aussi

ValidateTopologyPrecision

9.10 Traitement de la topologie

9.10.1 Polygonize

Polygonize — Recherche et enregistre toutes les faces définies par les arêtes de la topologie.

Synopsis

text **Polygonize**(varchar toponame);

Description

Enregistre toutes les faces qui peuvent être construites à partir d'une primitive d'arête topologique.

La topologie cible est supposée ne pas contenir d'arêtes s'auto-intersectant.



Note
Les faces déjà connues sont reconnues, de sorte qu'il est possible d'appeler Polygonize plusieurs fois sur la même topologie.



Note
Cette fonction n'utilise ni ne définit les champs next_left_edge et next_right_edge de la table des arêtes.

Disponibilité : 2.0.0

Voir aussi

AddFace, ST_Polygonize

9.10.2 AddNode

AddNode — Ajoute un nœud ponctuel à la table des nœuds dans le schéma topologique spécifié et renvoie le nodeid du nouveau nœud. Si le point existe déjà en tant que nœud, l'identifiant du nœud existant est renvoyé.

Synopsis

bigint **AddNode**(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

Description

Ajoute un nœud ponctuel à la table des nœuds dans le schéma topologique spécifié. La fonction AddEdge ajoute automatiquement les points de départ et d'arrivée d'une arête lorsqu'elle est appelée, de sorte qu'il n'est pas nécessaire d'ajouter explicitement les nœuds d'une arête.

Si une arête traversant le nœud est trouvée, une exception est levée ou l'arête est scindée, en fonction de la valeur du paramètre `allowEdgeSplitting`.

Si `computeContainingFace` est true, un nœud nouvellement ajouté verra sa face contenante calculée correctement.



Note
Si la géométrie `apoint` existe déjà en tant que noeud, le noeud n'est pas ajouté mais le nodeid existant est renvoyé.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986)) AS ↩
    nodeid;
```

Output

```
4
```

Voir aussi

AddEdge, CreateTopology

9.10.3 AddEdge

AddEdge — Ajoute une arête linéaire à la table des arêtes et les points de départ et d'arrivée associés à la table des nœuds de points du schéma topologique spécifié en utilisant la géométrie linéaire spécifiée et renvoie l'identifiant de l'arête nouvelle (ou existante).

Synopsis

bigint **AddEdge**(varchar toponame, geometry aline);

Description

Ajoute une arête à la table des arêtes et les nœuds associés à la table des nœuds du schéma `toponame` spécifié en utilisant la géométrie de ligne spécifiée et renvoie l'identifiant de l'arête du nouvel enregistrement ou de l'enregistrement existant. L'arête nouvellement ajoutée a une face "univers" des deux côtés et est liée à elle-même.



Note
Si la géométrie `aline` croise, chevauche, contient ou est contenue par un bord de ligne existant, une erreur est générée et l'arête n'est pas ajoutée.



Note
La géométrie de `aline` doit avoir la même `srid` que celle définie pour la topologie, sinon une erreur sys de référence spatiale non valide sera levée.

Effectué par le module GEOS.



AVERTISSEMENT
AddEdge est obsolète depuis la version 3.5.0. Utilisez plutôt TopoGeo_AddLineString.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9 ↩
    893900.4)', 26986)) AS edgeid;
```

Output

```
1
```

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 ↩
    893844.2,227641.6 893816.5,227704.5 893778.5)', 26986)) AS edgeid;
```

Output

```
2
```

This query errors because the edge intersects an existing edge away from the endpoints.

Code

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900,227591.9 ↩
    893900.4,227704.5 893778.5)', 26986)) AS edgeid;
```

Output

```
ERROR:  Edge intersects (not on endpoints) with existing edge 1
```

Voir aussi

TopoGeo_AddLineString, CreateTopology, Section 4.5

9.10.4 AddFace

AddFace — Enregistre une primitive de face dans une topologie et obtient son identifiant.

Synopsis

bigint **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

Description

Enregistre une primitive de face dans une topologie et obtient son identifiant.

Pour une face nouvellement ajoutée, les arêtes formant ses limites et celles contenues dans la face seront mises à jour pour que les valeurs des champs left_face et right_face soient correctes. Les nœuds isolés contenus dans la face seront également mis à jour pour que la valeur du champ containing_face soit correcte.



Note
Cette fonction n'utilise ni ne définit les champs next_left_edge et next_right_edge de la table des arêtes.

La topologie cible est supposée valide (ne contenant pas d'arêtes se recoupant). Une exception est levée si : La limite du polygone n'est pas entièrement définie par les arêtes existantes ou le polygone chevauche une face existante.

Si la géométrie `apolygon` existe déjà en tant que face, alors : si `force_new` est false (par défaut), l'identifiant de la face existante est renvoyé ; si `force_new` est true, un nouvel identifiant sera attribué à la face nouvellement enregistrée.



Note
Lorsqu'un nouvel enregistrement d'une face existante est effectué (force_new=true), aucune action ne sera entreprise pour résoudre les références pendantes à la face existante dans les tables d'arêtes, de nœuds et de relations, et le champ MBR (Minimum Bounding Rectangle) de l'enregistrement de la face existante ne sera pas mis à jour. C'est à l'appelant de s'en occuper.



Note
La géométrie `apolygon` doit avoir la même `srid` que celle définie pour la topologie, sinon une erreur sys de référence spatiale non valide sera déclenchée.

Disponibilité : 2.0.0

Exemples

First add the edges using `generate_series` as an iterator. This query only works for polygons with fewer than 10000 points because of the maximum value in `generate_series`.

Code

```
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) ))  ↩
    As edgeid
FROM (
  SELECT ST_NPoints(geom) AS npt, geom
  FROM (
    SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6  ↩
    899356.9,234872.5 899328.7,
                234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
                234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) )  As geom
  ) AS geoms
) AS facen
CROSS JOIN generate_series(1,10000) AS i
WHERE i < npt;
```

Output

```
3
4
5
6
7
8
9
10
11
12
```

Then add the face.

Code

```
SELECT topology.AddFace(
    'ma_topo',
    ST_GeomFromText(
    'POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5 899328.7,
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))',
    26986) ) As faceid;
```

Output

```
1
```

Voir aussi

AddEdge, CreateTopology, Section 4.5

9.10.5 ST_Simplify

ST_Simplify — Renvoie une version géométrique "simplifiée" de la TopoGeometry donnée en utilisant l'algorithme de Douglas-Peucker.

Synopsis

geometry **ST_Simplify**(topogeometry tg, float8 tolerance);

Description

Renvoie une version géométrique "simplifiée" de la TopoGeometry donnée en utilisant l'algorithme de Douglas-Peucker sur chaque arête de la composante.



Note
La géométrie renvoyée peut être non simple ou non valide.
La séparation des arêtes des composants peut contribuer à préserver la simplicité et la validité.

Effectué par le module GEOS.

Disponibilité : 2.1.0

Voir aussi

Geometry ST_Simplify, ST_IsSimple, ST_IsValid, ST_ModEdgeSplit

9.10.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Supprime les primitives topologiques qui ne sont pas nécessaires pour définir les objets TopoGeometry existants.

Synopsis

bigint **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

Description

Recherche toutes les primitives (nœuds, arêtes, faces) qui ne sont pas strictement nécessaires pour représenter les objets TopoGeometry existants et les supprime, en maintenant la validité de la topologie (liaison des arêtes, étiquetage des faces) et l'occupation de l'espace TopoGeometry.

Aucun nouvel identifiant de primitive n'est créé, mais les primitives existantes sont étendues pour inclure les faces fusionnées (en supprimant les arêtes) ou les arêtes réparées (en supprimant les nœuds).

Disponibilité : 3.3.0

Voir aussi

ST_ModEdgeHeal, ST_RemEdgeModFace

9.11 Constructeurs de TopoGeometry

9.11.1 CreateTopoGeom

CreateTopoGeom — Crée un nouvel objet géométrique topo à partir d'un tableau d'éléments topo - tg_type : 1 :[multi]point, 2 :[multi]ligne, 3 :[multi]poly, 4:collection

Synopsis

topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs, bigint tg_id);
topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id);

Description

Crée un objet topogéométrique pour la couche désignée par `layer_id` et l'enregistre dans la table des relations du schéma `toponame`.

`tg_type` est un nombre entier : 1 : [multi]point (ponctuel), 2 : [multi]ligne (linéaire), 3 : [multi]poly (aréolaire), 4:collection. `layer_id` est l'identifiant de la couche dans la table topology.layer.

Les couches ponctuelles sont formées à partir d'un ensemble de nœuds, les couches linéaires sont formées à partir d'un ensemble d'arêtes, les couches aréales sont formées à partir d'un ensemble de faces, et les collections peuvent être formées à partir d'un mélange de nœuds, d'arêtes et de faces.

L'omission du tableau de composants génère un objet TopoGeometry vide.

DIsponibilité : 1.1

Exemples

Form from existing edges.

Créer un topogeom dans le schéma ri_topo pour la couche 2 (nos ri_roads), de type (2) LINE, pour la première arête (que nous avons chargée dans `ST_CreateTopoGeo`).

Code

```
INSERT INTO ri.ri_roads(road_name, topo)
VALUES (
    'Unknown',
    topology.CreateTopoGeom(
        'ri_topo',
        2,
        2,
        '{{1,2}}'::topology.topoelementarray
    )
);
```

Convert an areal geometry to best guess topogeometry.

Supposons que nous ayons des géométries qui doivent être formées à partir d'une collection de faces. Nous disposons par exemple d'un tableau de groupes de blocs et nous voulons connaître la géométrie topographique de chaque groupe de blocs. Si nos données étaient parfaitement alignées, nous pourrions le faire :

Create the topogeometry column.

Code

```
SELECT topology.AddTopoGeometryColumn(
'topo_boston',
'boston', 'blockgroups', 'topo', 'POLYGON');
```

Output

```
1
```

Update the column assuming everything is perfectly aligned with the edges.

Code

```
UPDATE boston.blockgroups AS bg
SET topo = topology.CreateTopoGeom(
    'topo_boston',
    3,
    1,
    foo.bfaces
)
FROM (
    SELECT
        b.gid,
        topology.TopoElementArray_Agg(ARRAY[f.face_id, 3]) AS bfaces
    FROM boston.blockgroups AS b
    INNER JOIN topo_boston.face AS f
        ON b.geom && f.mbr
    WHERE ST_Covers(
        b.geom,
        topology.ST_GetFaceGeometry('topo_boston', f.face_id)
    )
    GROUP BY b.gid
) AS foo
WHERE foo.gid = bg.gid;
```

The world is rarely perfect, so this version allows some error. It counts a face if 50 percent of it falls within the expected block group boundary.

Code

```
WITH candidate_faces AS (
    SELECT
        b.gid,
        b.geom AS block_geom,
        f.face_id,
        topology.ST_GetFaceGeometry('topo_boston', f.face_id) AS face_geom
    FROM boston.blockgroups AS b
    INNER JOIN topo_boston.face AS f
        ON b.geom && f.mbr
),
block_faces AS (
    SELECT
        gid,
        topology.TopoElementArray_Agg(ARRAY[face_id, 3]) AS bfaces
    FROM candidate_faces
    WHERE ST_Covers(block_geom, face_geom)
        OR (
            ST_Intersects(block_geom, face_geom)
            AND ST_Area(ST_Intersection(block_geom, face_geom))

> ST_Area(face_geom) * 0.5
        )
    GROUP BY gid
)
UPDATE boston.blockgroups AS bg
SET topo = topology.CreateTopoGeom(
    'topo_boston',
    3,
    1,
    block_faces.bfaces
)
FROM block_faces
WHERE block_faces.gid = bg.gid;
```

To convert the topogeometry back to a denormalized geometry aligned with faces and edges, cast the topogeometry to a geometry. The resulting geometries are aligned with the TIGER street centerlines.

Code

```
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

Voir aussi

AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray_Agg

9.11.2 toTopoGeom

toTopoGeom — Convertit une géométrie simple en une géométrie topographique.

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance=-1);
topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance=-1);

Description

Convertit une géométrie simple en une TopoGeometry.

Les primitives topologiques requises pour représenter la géométrie d'entrée seront ajoutées à la topologie sous-jacente, éventuellement en divisant les primitives existantes, et elles seront associées à la TopoGeometry de sortie dans la table `relation`.

Les objets TopoGeometry existants (à l'exception éventuelle de `topogeom`, s'il en existe) conserveront leur forme.

When `tolerance` is given it will be used to snap the input geometry to existing primitives. The special value -1 (default) means use the topology precision as tolerance.

Dans le premier formulaire, une nouvelle TopoGéométrie sera créée pour la couche donnée (`layer_id`) de la topologie donnée (`toponame`).

Dans la seconde forme, les primitives résultant de la conversion seront ajoutées à la TopoGeometry préexistante (`topogeom`), ajoutant éventuellement de l'espace à sa forme finale. Pour que la nouvelle forme remplace complètement l'ancienne, voir clearTopoGeom.

Disponibilité : 2.0

Amélioré : 2.1.0 ajoute la version d'une TopoGeometry existante.

Changed: 3.7.0 changes meaning of tolerance 0 to really mean 0 and -1 (new default) to request using the topology precision.

Exemples

Il s'agit d'un flux de travail complet et autonome

Create the topology without allowing any tolerance.

Code

```
SELECT topology.CreateTopology('topo_boston_test', 2249);
```

Create a new table.

Code

```
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
```

Add a topogeometry column to the new table.

Code

```
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ↩
    MULTIPOLYGON') AS new_layer_id;
```

Output

```
1
```

Use the new layer identifier to populate the topogeometry column.

Code

```
INSERT INTO nei_topo(nei, topo)
SELECT nei, topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;
```

Verify what happened.

Code

```
SELECT * FROM
    topology.TopologySummary('topo_boston_test');
```

Output

```
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
 Deploy: public.nei_topo.topo
```

Shrink all TopoGeometry polygons by 10 meters.

Code

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

Find faces that are left without any TopoGeometry reference after the shrink operation.

Code

```
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
  FROM topo_boston_test.face f
  WHERE f.face_id
> 0
  AND NOT EXISTS (
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
  );
```

Voir aussi

CreateTopology, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

9.11.3 TopoElementArray_Agg

TopoElementArray_Agg — Renvoie un `topoelementarray` pour un ensemble de tableaux de type, element_id (topoelements).

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

Description

Utilisé pour créer un TopoElementArray à partir d'un ensemble de TopoElement.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT topology.TopoElementArray_Agg(ARRAY[e, t] ORDER BY e, t) As tea
FROM generate_series(1, 3) As e CROSS JOIN generate_series(1, 4) As t;
```

Output

```
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

Voir aussi

TopoElement, TopoElementArray

9.11.4 TopoElement

TopoElement — Convertit une topogeometry en un topoelement.

Synopsis

topoelement **TopoElement**(topogeometry topo);

Description

Convertit une TopoGeometry en TopoElement.

Disponibilité : 3.4.0

Exemples

Il s'agit d'un flux de travail complet et autonome

Return the TopoElement representation of each topogeometry.

Code

```
SELECT TopoElement(topo)
FROM neighborhoods;
```

Use a cast to aggregate topogeometries into a TopoElementArray.

Code

```
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

Voir aussi

TopoElementArray_Agg, TopoGeometry, TopoElement

9.12 Editeurs de TopoGeometry

9.12.1 clearTopoGeom

clearTopoGeom — Efface le contenu d'une géométrie topo.

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Description

Efface le contenu d'une TopoGeometry en la transformant en une vide. Principalement utile en conjonction avec toTopoGeom pour remplacer la forme des objets existants et de tout objet dépendant dans les niveaux hiérarchiques supérieurs.

Disponibilité : 2.1

Exemples

Shrink all TopoGeometry polygons by 10 meters.

Code

```
UPDATE nei_topo SET topo = toTopoGeom(ST_Buffer(topo, -10), clearTopoGeom(topo), 0);
```

Voir aussi

toTopoGeom

9.12.2 TopoGeom_addElement

TopoGeom_addElement — Ajoute un élément à la définition d'une TopoGeometry.

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Description

Ajoute un TopoElement à la définition d'un objet TopoGeometry. Il n'y a pas d'erreur si l'élément fait déjà partie de la définition.

Disponibilité : 2.3

Exemples

Add edge 5 to TopoGeometry `tg`.

Code

```
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

Voir aussi

TopoGeom_remElement, CreateTopoGeom

9.12.3 TopoGeom_remElement

TopoGeom_remElement — Supprime un élément de la définition d'une TopoGeometry.

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Description

Supprime une TopoElement de la définition d'un objet TopoGeometry.

Disponibilité : 2.3

Exemples

Remove face 43 from TopoGeometry `tg`.

Code

```
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

Voir aussi

TopoGeom_addElement, CreateTopoGeom

9.12.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — Ajoute un élément d'une TopoGeometry à la définition d'une autre TopoGeometry.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

Description

Ajoute les éléments d'une TopoGeometry à la définition d'une autre TopoGeometry, en changeant éventuellement son type mis en cache (attribut type) en une collection, si nécessaire pour contenir tous les éléments de l'objet source.

Les deux objets TopoGeometry doivent être définis par rapport à la même topologie et, s'ils sont définis hiérarchiquement, ils doivent être composés d'éléments de la même couche enfant.

Disponibilité : 3.2

Exemples

Set an overall TopoGeometry value to be composed of all elements from specific TopoGeometry values.

Code

```
UPDATE mylayer SET tg_overall = TopoGeom_addTopoGeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

Voir aussi

TopoGeom_addElement, clearTopoGeom, CreateTopoGeom

9.12.5 toTopoGeom

toTopoGeom — Ajoute une forme géométrique à une géométrie topographique existante.

Description

Se référer à toTopoGeom.

9.13 Accès aux TopoGeometry

9.13.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Renvoie un `topoelementarray` (un tableau de topoelements) contenant les éléments topologiques et le type de la TopoGeometry donnée (éléments primitifs).

Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer_id, bigint tg_id);

topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

Description

Renvoie un TopoElementArray contenant les éléments topologiques et le type de la TopoGeometry donnée (éléments primitifs). Cette fonction est similaire à GetTopoGeomElements, sauf qu'elle renvoie les éléments sous la forme d'un tableau plutôt que sous la forme d'un ensemble de données.

tg_id est l'identifiant topogeometry de l'objet topogeometry de la topologie dans la couche désignée par `layer_id` dans la table topology.layer.

DIsponibilité : 1.1

Exemples

Voir aussi

GetTopoGeomElements, TopoElementArray

9.13.2 GetTopoGeomElements

GetTopoGeomElements — Renvoie un ensemble d'objets `topoelement` contenant les éléments topologiques element_id,element_typ de la TopoGeometry donnée (éléments primitifs).

Synopsis

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer_id, bigint tg_id);

setof topoelement **GetTopoGeomElements**(topogeometry tg);

Description

Renvoie un ensemble de element_id,element_type (topoelements) correspondant aux éléments topologiques primitifs TopoElement (1 : noeuds, 2 : arêtes, 3 : faces) dont est composé un objet topogeometry donné dans le schéma `toponame`.

tg_id est l'identifiant topogeometry de l'objet topogeometry de la topologie dans la couche désignée par `layer_id` dans la table topology.layer.

Disponibilité : 2.0.0

Exemples

Voir aussi

GetTopoGeomElementArray, TopoElement, TopoGeom_addElement, TopoGeom_remElement

9.13.3 ST_SRID

ST_SRID — Renvoie l'identifiant de référence spatiale d'une topogeometry.

Synopsis

integer **ST_SRID**(topogeometry tg);

Description

Returns the spatial reference identifier of the topology containing the TopoGeometry.

The identifier is recorded for the topology when it is created and refers to an entry in Section 4.5.

Disponibilité : 3.2.0

Exemples

Return the SRID of a TopoGeometry stored in the `ri_topo` topology.

Code

```
SELECT topology.ST_SRID(topo) AS srid
FROM ri.roads
WHERE road_name = 'Unknown';
```

Output

```
srid
------
 3438
```

Voir aussi

Section 4.5, ST_SetSRID, ST_Transform, ST_SRID

9.14 Sorties TopoGeometry

9.14.1 AsGML

AsGML — Renvoie la représentation GML d'une topogeometry.

Synopsis

text **AsGML**(topogeometry tg);
text **AsGML**(topogeometry tg, text nsprefix_in);
text **AsGML**(topogeometry tg, regclass visitedTable);
text **AsGML**(topogeometry tg, regclass visitedTable, text nsprefix);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix, int gml-version);

Description

Renvoie la représentation GML d'une topogeometry au format GML3. Si aucun `nsprefix_in` n'est spécifié, `gml` est utilisé. Fournir une chaîne de caractères vide pour nsprefix afin d'obtenir un espace de noms non qualifié. Les paramètres precision (par défaut : 15) et options (par défaut 1), s'ils sont donnés, sont transmis tels quels à l'appel sous-jacent à ST_AsGML.

Le paramètre `visitedTable`, s'il est donné, est utilisé pour garder une trace des éléments de nœud et d'arêtes visités afin d'utiliser des références croisées (xlink:xref) plutôt que de dupliquer les définitions. La table doit comporter (au moins) deux champs entiers : "element_type" et "element_id". L'utilisateur appelant doit avoir les privilèges de lecture et d'écriture sur la table donnée. Pour de meilleures performances, un index doit être défini sur `element_type` et `element_id`, dans cet ordre. Cet index sera créé automatiquement par l'ajout d'une contrainte d'unicité aux champs. Exemple :

Code

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

Le paramètre `idprefix`, s'il est indiqué, sera ajouté aux identifiants des balises Edge et Node.

Le paramètre `gmlver`, s'il est donné, sera transmis au ST_AsGML sous-jacent. La valeur par défaut est 3.

Disponibilité : 2.0.0

Exemples

Ceci utilise la géométrie topo que nous avons créée dans CreateTopoGeom

Code

```
SELECT topology.AsGML(topo) As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';
```

Output

```
<gml:TopoCurve>
    <gml:directedEdge>
        <gml:Edge gml:id="E1">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml:directedNode
></gml:directedNode>
            <gml:curveProperty>
                <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <gml:segments>
                        <gml:LineStringSegment>
                            <gml:posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954 236898 385087 236932 385117 ↩
    236938
                            385167 236938 385203 236941 385224 236946 385233 236950 385241 236956 385254 236971
                            385260 236979 385268 236999 385273 237018 385273 237037 385271 237047 385267 237057 385225 ↩
    237125
                            385210 237144 385192 237161 385167 237192 385162 237202 385159 237214 385159 237227 385162 ↩
    237241
                            385166 237256 385196 237324 385209 237345 385234 237375 385237 237383 385238 237399 385236 ↩
    237407
                            385227 237419 385213 237430 385193 237439 385174 237451 385170 237455 385169 237460 385171 ↩
    237475
                            385181 237503 385190 237521 385200 237533 385206 237538 385213 237541 385221 237542 385235 ↩
    237540 385242 237541
                            385249 237544 385260 237555 385270 237570 385289 237584 385292 237589 385291 237596 385284 ↩
    237630</gml:posList>
                        </gml:LineStringSegment>
                    </gml:segments>
                </gml:Curve>
            </gml:curveProperty>
        </gml:Edge>
    </gml:directedEdge>
</gml:TopoCurve>
```

Même exercice que le précédent sans namespace

Code

```
SELECT topology.AsGML(topo,'') As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';
```

Output

```
<TopoCurve>
    <directedEdge>
        <Edge id="E1">
            <directedNode orientation="-">
                <Node id="N1"/>
            </directedNode>
            <directedNode
></directedNode>
            <curveProperty>
                <Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <segments>
                        <LineStringSegment>
                            <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954 236898 385087 236932 385117 ↩
    236938
                            385167 236938 385203 236941 385224 236946 385233 236950 385241 236956 385254 236971
                            385260 236979 385268 236999 385273 237018 385273 237037 385271 237047 385267 237057 385225 ↩
    237125
                            385210 237144 385192 237161 385167 237192 385162 237202 385159 237214 385159 237227 385162 ↩
    237241
                            385166 237256 385196 237324 385209 237345 385234 237375 385237 237383 385238 237399 385236 ↩
    237407
                            385227 237419 385213 237430 385193 237439 385174 237451 385170 237455 385169 237460 385171 ↩
    237475
                            385181 237503 385190 237521 385200 237533 385206 237538 385213 237541 385221 237542 385235 ↩
    237540 385242 237541
                            385249 237544 385260 237555 385270 237570 385289 237584 385292 237589 385291 237596 385284 ↩
    237630</posList>
                        </LineStringSegment>
                    </segments>
                </Curve>
```

```
            </curveProperty>
        </Edge>
    </directedEdge>
</TopoCurve>
```

Voir aussi

CreateTopoGeom, ST_CreateTopoGeo

9.14.2 AsTopoJSON

AsTopoJSON — Renvoie la représentation TopoJSON d'une topogeometry.

Synopsis

text **AsTopoJSON**(topogeometry tg, regclass edgeMapTable);

Description

Renvoie la représentation TopoJSON d'une topogeometry. Si `edgeMapTable` n'est pas nul, il sera utilisé comme correspondance de consultation/stockage des identifiants d'arêtes aux indices d'arcs. Cela permet d'obtenir un tableau compact des "arcs" dans le document final.

La table, si elle est donnée, doit avoir un champ "arc_id" de type "serial" et un champ "edge_id" de type integer; le code interrogera la table pour "edge_id" et il est donc recommandé d'ajouter un index sur ce champ.



Note
Les indices d'arc dans la sortie TopoJSON sont basés sur 0 mais ils sont basés sur 1 dans la table "edgeMapTable".

Un document TopoJSON complet devra contenir, en plus des extraits renvoyés par cette fonction, les arcs proprement dits et quelques en-têtes. Voir le TopoJSON specification.

Disponibilité : 2.1.0

Amélioration : 2.2.1 ajout de la prise en charge des entrées "puntal"

Voir aussi

ST_AsGeoJSON

Exemples

Create a temporary edge map for the TopoJSON output.

Code

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);
```

Build the TopoJSON header, objects, arcs, and footer.

Code

```
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ↩
    ": {'

UNION ALL SELECT '"' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcels WHERE feature_name = 'P3P4';

WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (ST_DumpPoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
         CASE WHEN p1.path IS NULL THEN p2.geom
              ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
         END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcsdump AS (
  SELECT arc_id, (regexp_matches(ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcsdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}, "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

UNION ALL SELECT ']}'::text as t;
```

Output

```
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]],[[6,5,-5,-4,-3,1]]]}
}, "arcs": [
 [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
 [[35,6],[0,8]],
 [[35,6],[12,0]],
 [[47,6],[0,8]],
 [[47,14],[0,8]],
 [[35,22],[12,0]],
 [[35,14],[0,8]]
 ]}
```

9.15 Relations spatiales de topologie

9.15.1 Equals

Equals — Retourne vrai si deux topogeometries sont composées des mêmes primitives topologiques.

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

Description

Retourne vrai si deux topogeometries sont composées des mêmes primitives topologiques : faces, arêtes, nœuds.



Note
Cette fonction n'est pas prise en charge pour les topogeometries qui sont des collections de géométries. Elle ne permet pas non plus de comparer des topogeometries issues de topologies différentes.

Disponibilité : 1.1.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

The following example uses the `nei_topo` table created in the toTopoGeom examples.

Code

```
SELECT topology.Equals(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

Output

```
t
```

Voir aussi

GetTopoGeomElements, ST_Equals

9.15.2 Intersects

Intersects — Retourne true si une paire de primitives des deux topogeometries s'intersectent.

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

Description

Retourne true si une paire de primitives des deux topogeometries s'intersectent.



Note
Cette fonction n'est pas prise en charge pour les topogeometries qui sont des collections de géométries. Elle ne permet pas non plus de comparer des topogeometries issues de topologies différentes. Elle n'est pas non plus compatible avec les topogeometries hiérarchiques (topogeometries composées d'autres topogeometries).

Disponibilité : 1.1.0


Cette fonction prend en charge la 3D et ne supprime pas l'indice z.

Exemples

The following example uses the `nei_topo` table created in the toTopoGeom examples.

Code

```
SELECT topology.Intersects(a.topo, b.topo)
FROM nei_topo AS a
JOIN nei_topo AS b ON b.gid = a.gid
WHERE a.gid = 1;
```

Output

```
t
```

Voir aussi

ST_Intersects

9.16 Importer et exporter des topologies

Une fois que vous avez créé des topologies, et éventuellement des couches topologiques associées, vous pouvez les exporter dans un format de fichier pour les sauvegarder ou les transférer dans une autre base de données.

L'utilisation des outils standards de dump/restauration de PostgreSQL est problématique car les topologies sont composées d'un ensemble de tables (4 pour les primitives, un nombre arbitraire pour les couches) et d'enregistrements dans des tables de métadonnées (topology.topology et topology.layer). De plus, les identifiants de topologie ne sont pas univoques d'une base de données à l'autre, de sorte que les paramètres de votre topologie devront être modifiés lors de sa restauration.

Afin de simplifier l'exportation/la restauration des topologies, une paire d'exécutables est fournie : `pgtopo_export` et `pgtopo_import`. Exemple d'utilisation :

Code

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

9.16.1 Utiliser l'exportateur de topologie

Le script `pgtopo_export` prend le nom d'une base de données et d'une topologie et produit un fichier dump qui peut être utilisé pour importer la topologie (et les couches associées) dans une nouvelle base de données.

Par défaut, `pgtopo_export` écrit le fichier dump sur la sortie standard afin qu'il puisse être acheminé vers `pgtopo_import` ou redirigé vers un fichier (refusant d'écrire dans le terminal). Vous pouvez optionnellement spécifier un nom de fichier de sortie en utilisant l'argument `-f` dans la ligne de commandes.

Par défaut `pgtopo_export` inclut un dump de toutes les couches définies par rapport à la topologie donnée. Cela peut être plus de données que vous n'en avez besoin, ou peut ne pas fonctionner (dans le cas où vos tables de couches ont des dépendances complexes), auquel cas vous pouvez demander à ignorer les couches avec l'argument `--skip-layers` et les traiter séparément.

En utilisant `pgtopo_export` avec l'argument `--help` (ou `-h` en abrégé) affichera toujours une courte chaîne de caractères sur l'utilisation.

Le format du fichier dump est une archive tar compressée d'un répertoire `pgtopo_export` contenant au moins un fichier `pgtopo_dump_version` avec des informations sur la version du format. A partir de la version `1`, le répertoire contient des fichiers CSV délimités par des tabulations avec les données des tables primitives de topologie (node, edge_data, face, relation), les enregistrements de topologie et de couche associés et (sauf si `--skip-layers` est donné) un dump PostgreSQL au format personnalisé des tables signalées comme étant des couches de la topologie donnée.

9.16.2 Utiliser l'importateur de topologie

Le script `pgtopo_import` prend un dump topologique au format `pgtopo_export` et un nom à donner à la topologie à créer et produit un script SQL reconstruisant la topologie et les couches associées.

Le fichier SQL généré contiendra des instructions qui créent une topologie avec le nom donné, chargent les données primitives, restaurent et enregistrent toutes les couches de la topologie en liant correctement toutes les valeurs TopoGeometry à leur topologie correcte.

Par défaut, `pgtopo_import` lit le dump depuis l'entrée standard afin qu'il puisse être utilisé en conjonction avec `pgtopo_export` dans une chaîne de traitement. Vous pouvez optionnellement spécifier un nom de fichier d'entrée avec l'argument `-f` en ligne de commande.

Par défaut, `pgtopo_import` inclut dans le fichier SQL de sortie le code permettant de restaurer toutes les couches trouvées dans le dump.

Ceci peut être indésirable ou ne pas fonctionner si votre base de données cible a déjà des tables avec le même nom que celles dans le dump. Dans ce cas, vous pouvez demander à ignorer les couches avec l'argument `--skip-layers` et les traiter séparément (ou plus tard).

SQL pour charger et lier uniquement les couches à une topologie nommée peut être généré en utilisant l'argument `--only-layers`. Cela peut être utile pour charger des couches APRÈS avoir résolu les conflits de noms ou pour lier des couches à une topologie différente (par exemple une version spatialement simplifiée de la topologie de départ).

If the target topology already exists and you want it dropped upfront you can pass the `--drop-topology` switch (since PostGIS-3.6.0).

Chapitre 10

Gestion des données raster, requêtes et applications

10.1 Chargement et création de rasters

Dans la plupart des cas, vous créerez des rasters PostGIS en chargeant des fichiers raster existants à l'aide du chargeur de rasters `raster2pgsql`.

10.1.1 Utilisation de raster2pgsql pour charger des rasters

Le `raster2pgsql` est un exécutable de chargement de données raster qui charge les formats de données matricielles pris en charge par GDAL en SQL adapté au chargement dans une table raster PostGIS. Il est capable de charger des dossiers de fichiers raster ainsi que de créer des aperçus de rasters.

Comme raster2pgsql est le plus souvent compilé dans le cadre de PostGIS (à moins que vous ne compiliez votre propre bibliothèque GDAL), les types de données raster pris en charge par l'exécutable seront les mêmes que ceux compilés dans la bibliothèque de dépendance GDAL. Pour obtenir une liste des types de données raster que votre `raster2pgsql` supporte, utilisez l'argument `-G`.



Note
Lors de la création de vues d'ensemble d'un facteur spécifique à partir d'un ensemble de rasters alignés, il est possible que les vues d'ensemble ne soient pas alignées. Consultez http://trac.osgeo.org/postgis/ticket/1764 pour un exemple où les vues d'ensemble ne sont pas alignées.

10.1.1.1 Exemple d'utilisation

Une session d'exemple utilisant le chargeur pour créer un fichier d'entrée et le charger en morceaux de 100 x 100 tuiles peut ressembler à ceci :

The loader assigns SRID 4326, creates an index and standard constraints, analyzes the loaded table, records source filenames, and stores the input rasters in 100 by 100 pixel tiles.

Code

```
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql
psql -d gisdb -f elev.sql
```



Note
Si vous ne spécifiez pas le schéma dans le nom de la table cible, la table sera créée dans le schéma par défaut de la base de données ou de l'utilisateur avec lequel vous vous connectez.

Une conversion et un téléchargement peuvent être effectués en une seule étape à l'aide de commandes UNIX :

Code

```
raster2pgsql -s 4326 -I -C -M -F -t 100x100 \
  *.tif public.demelevation |
  psql -d gisdb
```

Charger les rasters Massachusetts state plane meters aerial tiles dans un schéma appelé `aerial` et créer une vue complète, des tables d'ensemble à 2 et 4 niveaux, utiliser le mode copy pour l'insertion (pas de fichier intermédiaire juste directement dans la base de données), et -e ne pas forcer tout dans une transaction (bien si vous voulez voir les données dans les tables tout de suite sans attendre). Décomposer les rasters en tuiles de 128x128 pixels et appliquer les contraintes des rasters. Utilisez le mode copy au lieu de l'insertion de table. (-F) Inclure un champ appelé filename pour contenir le nom du fichier à partir duquel les tuiles ont été découpées.

Code

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128 -l 2,4 \
  bostonaerials2008/*.jpg aerials.boston |
  psql -U postgres -d gisdb -h localhost -p 5432
```

List the raster formats supported by this build:

Code

```
raster2pgsql -G
```

The `-G` command outputs a list something like:

Output

```
Available GDAL raster formats:
  Virtual Raster
  GeoTIFF
  National Imagery Transmission Format
  Raster Product Format TOC format
  ECRG TOC format
  Erdas Imagine Images (.img)
  CEOS SAR Image
  CEOS Image
  ...
  Arc/Info Export E00 GRID
  ZMap Plus Grid
  NOAA NGS Geoid Height Grids
```

10.1.1.2 options raster2pgsql

Long options that take values also accept the `--option=value` spelling.

-?, --help Affiche l'écran d'aide. L'aide s'affichera également si vous ne fournissez aucun argument.

-G, --gdal-formats Affiche les formats raster pris en charge.

(c|a|d|p) Ces options s'excluent mutuellement :

- **-c** Créer une nouvelle table et la remplir de données raster, *c'est le mode par défaut*
- **-a** Ajouter des données raster à une table existante.
- **-d** Supprimer la table, en créer une nouvelle et l'alimenter avec des données raster
- **-p** En mode préparation, ne créez que la table.

--if-not-exists Use `IF NOT EXISTS` for table creation in `-c` and `-p` modes. When `-I` or `--create-index` is also specified, use `IF NOT EXISTS` for index creation too. This option cannot be used with `-d`. Append mode requires an explicit creation action.

Atomic loader actions The short modes are presets over explicit actions. The equivalent long options are `--drop-table`, `--create-table`, `--load-data`, `--create-index`, `--add-constraints`, `--vacuum`, `--analyze`, and `--no-transaction`. Add `--if-not-exists` to make active creation actions use `IF NOT EXISTS`. `--drop-table` emits `DROP TABLE IF EXISTS` before the selected actions. With no mode specified, the default create/load actions still apply.

Traitement des données raster : Application de contraintes pour un enregistrement correct dans les catalogues de données

-C, --add-constraints Appliquer les contraintes du raster -- srid, pixelsize etc. pour s'assurer que le raster est correctement enregistré dans la vue `raster_columns`.

-x, --no-extent Disable setting the max extent constraint. Only applied if -C or --add-constraints is also used.

-r, --regular-blocking Set the constraints (spatially unique and coverage tile) for regular blocking. Only applied if -C or --add-constraints is also used.

Traitement des données raster : Paramètres facultatifs utilisés pour manipuler le jeu de données raster d'entrée

-s, --srid [<FROM_SRID>:]<SRID> Assign the output raster to the specified SRID. When FROM_SRID is also provided, raster2pgsql marks the input raster with FROM_SRID and emits SQL that reprojects the raster to the target SRID. If the source SRID is omitted or is zero, the raster metadata will be checked to determine an appropriate source SRID. Reprojection cannot be used with copy mode.

-b, --band BAND Indice (basé sur 1) de la bande à extraire du raster. Pour plus d'un indice de bande, séparez-les par une virgule (,). Si rien n'est spécifié, toutes les bandes de l'image seront extraites.

-t, --tile-size TILE_SIZE Découpe le raster en tuiles qui seront insérées une par une dans la table. `TILE_SIZE` est exprimée en LARGEURxHAUTEUR ou fixée à la valeur "auto" pour permettre au chargeur de calculer une taille de tuile appropriée en utilisant le premier raster et en l'appliquant à tous les rasters.

-P, --pad Les tuiles les plus à droite et les plus en bas sont remplacées pour garantir que toutes les tuiles ont la même largeur et la même hauteur.

-R, --register Enregistrer le raster en tant que raster de système de fichiers (out-db).

Seules les métadonnées du raster et le chemin d'accès au raster sont stockés dans la base de données (pas les pixels).

-l, --overview-factor OVERVIEW_FACTOR Create overview of the raster. For more than one factor, separate with comma(,). Overview table name follows the pattern o_`overview factor_table`, where `overview factor` is a placeholder for numerical overview factor and `table` is replaced with the base table name. If a source raster already has an overview with matching dimensions, that overview is reused. Created overview is stored in the database and is not affected by -R. Note that your generated sql file will contain both the main table and overview tables.

-N, --nodata NODATA Valeur NODATA à utiliser pour les bandes sans valeur NODATA.

Paramètres facultatifs utilisés pour manipuler les objets de la base de données

-f, --raster-column COLUMN Spécifier le nom de la colonne raster de destination, la valeur par défaut étant "rast"

-F, --filename Ajouter une colonne avec le nom du fichier

-n, --filename-column COLUMN Spécifier le nom de la colonne "nom de fichier". Implique -F.

-q, --quote Mettre les identifiants PostgreSQL entre guillemets.

-I Alias for `--create-index`.

--create-index Create a GiST index on the raster column at the end of this raster2pgsql run. With repeated `-a` append runs, create the index on the final run or after loading; add `--if-not-exists` to make reruns tolerate an existing index.

-M Vacuum analyze la table raster.

-k, --skip-nodata-check Conserve les tuiles vides et saute les vérifications des valeurs NODATA pour chaque bande de données matricielles. Vous gagnez du temps en vérifiant, mais vous risquez de vous retrouver avec beaucoup plus de lignes inutiles dans votre base de données et ces lignes inutiles ne sont pas marquées comme des tuiles vides.

-T, --tablespace tablespace Spécifier le tablespace pour la nouvelle table. Notez que les index (y compris la clé primaire) utiliseront toujours le tablespace par défaut à moins que l'option -X ne soit également utilisée.

-X, --index-tablespace tablespace Spécifier le tablespace pour le nouvel index de la table. Ceci s'applique à la clé primaire et à l'index spatial si l'option -I est utilisée.

-Y, --copy [MAX_ROWS_PER_COPY] Utiliser les instructions de copie au lieu des instructions d'insertion. Spécifier optionnellement `max_rows_per_copy` ; par défaut 50 si non spécifié.

-e, --no-transaction Exécutez chaque déclaration individuellement, n'utilisez pas de transaction.

-E, --endian ENDIAN Contrôle l'ordre des octets de la sortie binaire générée du raster ; spécifier 0 pour XDR et 1 pour NDR (par défaut) ; seule la sortie NDR est prise en charge actuellement

-V, --wkb-version version Indique la version du format de sortie. La valeur par défaut est 0. Seule la valeur 0 est prise en charge pour l'instant.

10.1.2 Création de rasters à l'aide des fonctions raster de PostGIS

Il arrive souvent que vous souhaitiez créer des rasters et des tables de rasters directement dans la base de données. Il existe une multitude de fonctions pour ce faire. Les étapes générales à suivre.

1. Créer une table avec une colonne raster pour contenir les nouveaux enregistrements raster, ce qui peut être fait avec :

   Code

   ```
   CREATE TABLE myrasters(rid serial primary key, rast raster);
   ```

2. Il existe de nombreuses fonctions pour vous aider à atteindre cet objectif. Si vous créez des rasters qui ne sont pas des dérivés d'autres rasters, vous devrez commencer par ST_MakeEmptyRaster, suivi de ST_AddBand

   Vous pouvez également créer des rasters à partir de géométries. Pour ce faire, vous devrez utiliser ST_AsRaster éventuellement accompagné d'autres fonctions telles que ST_Union ou ST_MapAlgebraFct ou n'importe quelle autre fonction d'algèbre de carte.

   Il existe encore bien d'autres options pour créer de nouvelles tables raster à partir de tables existantes. Par exemple, vous pouvez créer une table raster dans une projection différente de celle d'une table existante en utilisant ST_Transform

3. Une fois que vous avez rempli votre table initialement, vous devez créer un index spatial sur la colonne des données raster avec quelque chose comme :

   Code

   ```
   CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull( ↩
       rast));
   ```

   Notez l'utilisation de ST_ConvexHull puisque la plupart des opérateurs de données raster sont basés sur l'enveloppe convexe des données raster.

   Note
   Les versions antérieures à la version 2.0 de PostGIS raster étaient basées sur l'enveloppe plutôt que sur l'enveloppe convexe. Pour que les index spatiaux fonctionnent correctement, vous devez les supprimer et les remplacer par des index basés sur l'enveloppe convexe.

4. Appliquer des contraintes raster en utilisant AddRasterConstraints

10.1.3 Utilisation de rasters "out db" stockés sur le cloud

L'outil `raster2pgsql` utilise GDAL pour accéder aux données raster et peut tirer parti d'une fonctionnalité clé de GDAL : la possibilité de lire des données raster qui sont stockées à distance dans des "magasins d'objets" en nuage (par exemple AWS S3, Google Cloud Storage).

L'utilisation efficace des rasters stockés dans le cloud nécessite l'utilisation d'un format "optimisé pour le cloud". Le plus connu et le plus largement utilisé est le format "cloud optimized GeoTIFF". L'utilisation d'un format non optimisé pour le cloud, comme un JPEG ou un TIFF non structuré, se traduira par des performances très médiocres, car le système devra télécharger l'intégralité du raster chaque fois qu'il devra accéder à un sous-ensemble.

Tout d'abord, chargez votre raster dans le service de stockage cloud de votre choix. Une fois qu'il est chargé, vous aurez un URI pour y accéder, soit un URI "http", soit parfois un URI spécifique au service (par exemple, "s3://bucket/object") Pour accéder aux buckets non publics, vous devrez fournir des options de configuration GDAL pour authentifier votre connexion. Notez que cette commande *lit* à partir du raster (stocké dans le cloud) et *écrit* dans la base de données.

Code

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
  -s 990000 \
  -t 256x256 \
  -I \
  -R \
  /vsis3/your.bucket.com/your_file.tif \
  your_table \
  | psql your_db
```

Une fois la table chargée, vous devez donner à la base de données la permission de lire les rasters distants, en définissant deux permissions, postgis.enable_outdb_rasters et postgis.gdal_enabled_drivers.

Code

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Pour que les changements soient durables, définissez-les directement dans votre base de données. Vous devrez vous reconnecter pour bénéficier des nouveaux paramètres.

Code

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

Pour les rasters non publics, il se peut que vous deviez fournir des clés d'accès pour lire les rasters stockés dans le cloud. Les mêmes clés que vous avez utilisées pour écrire l'appel `raster2pgsql` peuvent être définies pour être utilisées à l'intérieur de la base de données, avec la configuration postgis.gdal_vsi_options. Notez que plusieurs options peuvent être définies en séparant les paires `key=value` par des espaces.

Code

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Une fois les données chargées et les autorisations définies, vous pouvez interagir avec la table raster comme avec n'importe quelle autre table raster, en utilisant les mêmes fonctions. La base de données se chargera de tous les mécanismes de connexion aux données stockées dans le cloud lorsqu'elle aura besoin de lire les données des pixels.

10.2 Raster Coverages and Raster/Vector Analysis

A coverage is a way to interpret spatial data as a mapping from a *domain* of locations to a *range* of values. Evaluating a coverage means asking for the value at a location; an inverse query asks for the locations whose values match a condition. PostGIS does not expose a generic ISO 19123 coverage SQL type, but geometry tables and raster values provide the building blocks for both directions.

A coverage can be continuous, with values calculated or interpolated for arbitrary locations, or discrete, with values supplied by stored spatial objects or grid cells. The coverage model describes how locations map to values, independently of whether those

values come from an equation, a table, or one or more raster values. A time column or function argument can add a temporal dimension without changing that distinction.

In a vector coverage, a geometry column supplies the domain and other columns supply attributes in the range. A vector range can contain text, numbers, or another spatial value, provided its meaning and type are consistent wherever the coverage is defined. In a raster, each cell location is part of the domain and the band values at that cell form the range. Other columns in the same table row describe the raster value or tile as a whole; they do not attach separate values to individual cells.

A logical raster coverage may be stored as one raster value or as many tiles in a table. Tiles that belong to one coverage should have compatible SRIDs, scales, alignment, band counts, pixel types, and NODATA conventions. The constraints exposed by Section 10.3.1 record and validate those properties.

Raster table arrangements. A raster column can represent an image warehouse, a tiled coverage, or a collection of rasterized objects. These arrangements have different assumptions about tile size, alignment, gaps, and overlaps. PostGIS does not infer the intended arrangement from the column type alone.

Arrangement	Row relationship	Typical use
Image warehouse	Independent rasters; sizes, alignment, and extents may differ or overlap	Cataloging unrelated source images
Irregular tiled coverage	One logical coverage with gaps allowed, variable tile sizes, and normally no overlap	Incremental or source-driven mosaics
Regular tiled coverage	One alignment and tile size with no overlap; gaps may remain at the boundary or inside the coverage	Indexed analysis over a large coverage
Rectangular regular coverage	Equal aligned tiles fill a rectangular extent without gaps or overlaps	Predictable block processing and overview generation
Separate tiled images	Each image is internally regular but stored in its own table or partition	Keeping source-image identity and lifecycle separate
Raster-object coverage	Each row is a rasterized feature; extents and sizes may differ and overlap	Object models where cells replace or supplement vector geometry

Use AddRasterConstraints to record common SRID, scale, alignment, band, extent, and regular-blocking properties, and use spatial indexes and application constraints for relationships that the catalog cannot express.

Queries against a tiled coverage normally have two stages:

1. Use the raster spatial index and ST_Intersects to select candidate tiles.
2. Read, clip, aggregate, or vectorize values inside those tiles with functions such as ST_Value, ST_Clip, ST_SummaryStatsAgg, and ST_DumpAsPolygons.

ST_DumpAsPolygons returns geomval rows that pair geometries with cell values. This allows ordinary geometry overlay and filtering to answer inverse coverage queries, but polygonizing many cells is usually more expensive than keeping the operation in raster space.

Choose boundary semantics deliberately. Clipping and aggregating raster pixels is usually the shorter and faster path. If a partial boundary cell must contribute in proportion to its covered area, intersect the geometry with the geomval rows and calculate an area-weighted result in an appropriate projected coordinate system. That exact partial-cell method costs more because it vectorizes the selected cells.

Tile size affects both sides of the indexed query: smaller tiles reduce the amount of raster data inspected for each candidate row, while increasing row and index overhead. Benchmark representative queries instead of treating one tile size as a universal default.

For a current end-to-end raster/vector pattern, load and tile data with Section 10.1.1, then see the building example in ST_SummaryStats. It selects intersecting tiles, clips them to each polygon, and combines per-tile statistics into one result per feature. For point observations such as animal locations, first buffer the points in an appropriate projected coordinate system, then apply the same tile selection, clipping, and aggregation pattern to an elevation or other continuous raster coverage.

10.3 Catalogues Raster

Deux vues de catalogue raster sont fournies avec PostGIS. Ces deux vues utilisent des informations intégrées dans les contraintes des tables raster. Par conséquent, les vues de catalogue sont toujours cohérentes avec les données raster des tables puisque les contraintes sont appliquées.

1. `raster_columns` cette vue répertorie toutes les colonnes des tables raster de votre base de données.
2. `raster_overviews` cette vue répertorie toutes les colonnes des tables raster de votre base de données qui servent d'aperçu pour une table plus finement structurée. Les tables de ce type sont générées lorsque vous utilisez le commutateur `-l` pendant le chargement.

10.3.1 Catalogue des colonnes raster

Le `raster_columns` est un catalogue de toutes les colonnes des tables raster de votre base de données qui sont de type raster. Il s'agit d'une vue qui utilise les contraintes des tables afin que les informations soient toujours cohérentes, même si vous restaurez une table raster à partir d'une sauvegarde d'une autre base de données. Les colonnes suivantes existent dans le catalogue `raster_columns`.

Si vous n'avez pas créé vos tables avec le chargeur ou si vous avez oublié de spécifier l'option `-C` pendant le chargement, vous pouvez appliquer les contraintes après coup en utilisant AddRasterConstraints pour que le catalogue `raster_columns` enregistre les informations communes sur vos tuiles raster.

— `r_table_catalog` La base de données dans laquelle se trouve la table. La lecture se fera toujours dans la base de données actuelle.

— `r_table_schema` Schéma de la base de données auquel appartient la table raster.

— `r_table_name` table raster

— `r_raster_column` la colonne de la table `r_table_name` qui est de type raster. Rien dans PostGIS ne vous empêche d'avoir plusieurs colonnes raster par table. Il est donc possible d'avoir une table raster listée plusieurs fois avec une colonne raster différente pour chacune d'entre elles.

— `srid` L'identifiant de référence spatiale du raster. Il doit s'agir d'une entrée dans le fichier Section 4.5.

— `scale_x` L'échelle entre les coordonnées spatiales géométriques et le pixel. Elle n'est disponible que si toutes les tuiles de la colonne raster ont la même `scale_x` et que cette contrainte est appliquée. Se référer à ST_ScaleX pour plus de détails.

— `scale_y` L'échelle entre les coordonnées spatiales géométriques et le pixel. Elle n'est disponible que si toutes les tuiles de la colonne matricielle ont la même `scale_y` et que la contrainte `scale_y` est appliquée. Voir ST_ScaleY pour plus de détails.

— `blocksize_x` La largeur (nombre de pixels) de chaque tuile raster. Pour plus de détails, voir ST_Width.

— `blocksize_y` La largeur (nombre de pixels vers le bas) de chaque tuile raster. Voir ST_Height pour plus de détails.

— `same_alignment` Un booléen qui est vrai si toutes les tuiles raster ont le même alignement . Pour plus de détails, voir ST_SameAlignment.

— `regular_blocking` Si la colonne de données raster est soumise aux contraintes d'unicité spatiale et de couverture des tuiles, la valeur sera TRUE. Dans le cas contraire, la valeur sera FALSE.

— `num_bands` Le nombre de bandes dans chaque tuile de votre jeu de données raster. Il s'agit de la même information que celle fournie par ST_NumBands

— `pixel_types` Un tableau définissant le type de pixel pour chaque bande. Le nombre d'éléments de ce tableau est identique au nombre de bandes. Les types de pixels sont l'un des suivants, définis dans ST_BandPixelType.

— `nodata_values` Un tableau de nombres en double précision indiquant la `nodata_value` pour chaque bande. Le nombre d'éléments de ce tableau est égal au nombre de bandes. Ces nombres définissent la valeur du pixel pour chaque bande qui doit être ignorée pour la plupart des opérations. Il s'agit d'informations similaires à celles fournies par ST_BandNoDataValue.

— `out_db` Un tableau d'indicateurs booléens indiquant si les données des bandes raster sont conservées en dehors de la base de données. Le nombre d'éléments de ce tableau est égal au nombre de bandes.

— `extent` Il s'agit de l'étendue de toutes les lignes de votre jeu de données raster. Si vous prévoyez de charger d'autres données qui modifieront l'étendue de l'ensemble, vous devrez exécuter la fonction DropRasterConstraints avant le chargement, puis réappliquer les contraintes avec AddRasterConstraints après le chargement.

— `spatial_index` Un booléen qui est vrai si la colonne raster a un index spatial.

10.3.2 Aperçu des données raster

`raster_overviews` catalogue des informations sur les colonnes des tables raster utilisées pour les aperçus et des informations supplémentaires à leur sujet qu'il est utile de connaître lors de l'utilisation des aperçus. Les tableaux de synthèse sont catalogués à la fois dans `raster_columns` et `raster_overviews` parce qu'il s'agit d'images raster à part entière, mais aussi parce qu'elles servent de caricature à plus faible résolution d'un tableau à plus haute résolution. Elles sont générées en même temps que la table raster principale lorsque vous utilisez le commutateur `-l` dans le chargement des données raster ou peuvent être générées manuellement en utilisant AddOverviewConstraints.

Les tableaux d'aperçus contiennent les mêmes contraintes que les autres tableaux raster, ainsi que des contraintes supplémentaires spécifiques aux aperçus.



Note
Les informations contenues dans `raster_overviews` ne font pas double emploi avec les informations contenues dans `raster_columns`. Si vous avez besoin des informations sur un tableau de synthèse présentes dans `raster_columns`, vous pouvez joindre les `raster_overviews` et `raster_columns` pour obtenir l'ensemble des informations dont vous avez besoin.

Les deux principales raisons pour lesquelles les aperçus sont nécessaires sont les suivantes :

1. Représentation à basse résolution des tables centrales, couramment utilisée pour l'agrandissement rapide des cartes.
2. Les calculs y sont généralement plus rapides que sur leurs homologues à plus haute résolution, car il y a moins d'enregistrements et chaque pixel couvre un plus grand territoire. Bien que les calculs ne soient pas aussi précis que les tableaux à haute résolution qu'ils supportent, ils peuvent être suffisants pour de nombreux calculs empiriques.

Le catalogue `raster_overviews` contient les colonnes d'information suivantes.

— `o_table_catalog` La base de données dans laquelle se trouve l'aperçu. Il s'agit toujours de la base de données actuelle.
— `o_table_schema` Schéma de la base de données auquel appartient l'aperçu de la table raster.
— `o_table_name` nom de la table d'aperçu du raster
— `o_raster_column` la colonne raster dans le tableau de synthèse.
— `r_table_catalog` La base de données dans laquelle se trouve la table raster pour laquelle cette vue d'ensemble est utilisée. Il s'agit toujours de la base de données actuelle.
— `r_table_schema` Schéma de la base de données auquel appartient la table raster à laquelle ce service de vue d'ensemble est associé.
— `r_table_name` table raster que cet aperçu dessert.
— `r_raster_column` la colonne raster que cette colonne d'aperçu dessert.
— `overview_factor` - il s'agit du niveau pyramidal de l'aperçu. Plus le nombre est élevé, plus la résolution du tableau est faible. raster2pgsql, s'il reçoit un dossier d'images, calculera la vue d'ensemble de chaque fichier image et le chargera séparément. Le niveau 1 est supposé être le fichier original. Au niveau 2, chaque tuile représente 4 de l'original. Par exemple, si vous avez un dossier de fichiers images de 5000x5000 pixels que vous avez choisi de diviser en 125x125, pour chaque fichier image votre table de base aura (5000*5000)/(125*125) enregistrements = 1600, votre table (l=2) `o_2` aura un plafond(1600/Power(2,2)) = 400 lignes, votre (l=3) `o_3` aura un plafond(1600/Power(2,3)) = 200 lignes. Si vos pixels ne sont pas divisibles par la taille de vos tuiles, vous obtiendrez des bouts de tuiles (tuiles non complètement remplies). Notez que chaque tuile d'aperçu générée par raster2pgsql a le même nombre de pixels que son parent, mais a une résolution inférieure, chaque pixel représentant (Puissance(2,facteur_d'aperçu) pixels de l'original).

10.4 Créer des applications personnalisées avec PostGIS Raster

Le fait que PostGIS raster vous fournisse des fonctions SQL pour rendre les rasters dans les formats d'image connus vous donne beaucoup d'options pour les rendre. Par exemple, vous pouvez utiliser OpenOffice / LibreOffice pour le rendu comme démontré dans Rendering PostGIS Raster graphics with LibreOffice Base Reports. En outre, vous pouvez utiliser une grande variété de langages, comme le montre cette section.

10.4.1 Exemple de sortie PHP utilisant ST_AsPNG avec d'autres fonctions raster

Dans cette section, nous allons montrer comment utiliser le driver PHP PostgreSQL et la famille de fonctions ST_AsGDALRaster pour produire les bandes 1,2,3 d'un raster dans un flux de requête PHP qui peut ensuite être incorporé dans une balise html img src.

L'exemple de requête montre comment combiner tout un ensemble de fonctions raster pour récupérer toutes les tuiles qui intersectent une boîte de délimitation wgs 84 particulière, puis unit avec ST_Union les tuiles intersectées en retournant toutes les bandes, transforme en projection spécifiée par l'utilisateur en utilisant ST_Transform, et sort les résultats au format png en utilisant ST_AsPNG.

You would call the script with `http://mywebserver/test_raster.php?srid=2249` to get the raster image in Massachusetts state plane feet.

Code

```
<?php
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');

if (!empty($_REQUEST['srid']) &amp;amp;&amp;amp; is_numeric($_REQUEST['srid'])) {
    $input_srid = intval($_REQUEST['srid']);
} else {
    $input_srid = 26986;
}

$sql = "SET bytea_output = 'escape';
SELECT ST_AsPNG(
    ST_Transform(
        ST_AddBand(
            ST_Union(rast, 1),
            ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
        ),
        $input_srid
    )
) AS new_rast
FROM aerials.boston
WHERE ST_Intersects(
    rast,
    ST_Transform(
        ST_MakeEnvelope(-71.1217, 42.227, -71.1210, 42.218, 4326),
        26986
    )
)";

$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

10.4.2 Exemple ASP.NET C# Sortie utilisant ST_AsPNG en conjonction avec d'autres fonctions raster

In this section, we'll demonstrate how to use Npgsql PostgreSQL .NET driver and the ST_AsGDALRaster family of functions to output band 1,2,3 of a raster to the HTTP response stream so it can be embedded in an `img` tag.

Pour cet exercice, vous aurez besoin du driver PostgreSQL npgsql .NET dont vous pouvez obtenir la dernière version sur http://npgsql.projects.postgresql.org/. Il vous suffit de télécharger la dernière version et de la placer dans votre dossier ASP.NET bin.

L'exemple de requête montre comment combiner tout un ensemble de fonctions raster pour récupérer toutes les tuiles qui intersectent une boîte de délimitation wgs 84 particulière, puis unit avec ST_Union les tuiles intersectées en retournant toutes les bandes, transforme en projection spécifiée par l'utilisateur en utilisant ST_Transform, et sort les résultats au format png en utilisant ST_AsPNG.

Il s'agit du même exemple que Section 10.4.1 sauf qu'il est implémenté en C#.

You would call the handler with `http://mywebserver/TestRaster.ashx?srid=2249` to get the raster image in Massachusetts state plane feet.

web.config connection string section:

Code

```
<connectionStrings>
    <add name="DSN"
        connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ↩
    mypwd"/>
</connectionStrings>
```

Code for `TestRaster.ashx`:

Code

```
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "image/png";
        context.Response.BinaryWrite(GetResults(context));
    }

    public bool IsReusable {
        get { return false; }
    }

    public byte[] GetResults(HttpContext context)
    {
        byte[] result = null;
        NpgsqlCommand command;
        string sql = null;
        int input_srid = 26986;
        try {
            string connectionString = System.Configuration.ConfigurationManager
                .ConnectionStrings["DSN"].ConnectionString;
            using (NpgsqlConnection conn = new NpgsqlConnection(connectionString)) {
                conn.Open();

                if (context.Request["srid"] != null)
                {
                    input_srid = Convert.ToInt32(context.Request["srid"]);
                }
                sql = @"SELECT ST_AsPNG(
                            ST_Transform(
                                ST_AddBand(
                                    ST_Union(rast, 1),
                                    ARRAY[ST_Union(rast, 2), ST_Union(rast, 3)]
```

```
                                ),
                                :input_srid
                            )
                        ) As new_rast
                        FROM aerials.boston
                        WHERE ST_Intersects(
                            rast,
                            ST_Transform(
                                ST_MakeEnvelope(
                                    -71.1217, 42.227,
                                    -71.1210, 42.218,
                                    4326
                                ),
                                26986
                            )
                        )";
                command = new NpgsqlCommand(sql, conn);
                command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));


                result = (byte[]) command.ExecuteScalar();
                conn.Close();
            }

        }
        catch (Exception ex)
        {
            result = null;
            context.Response.Write(ex.Message.Trim());
        }
        return result;
    }
}
```

10.4.3 Application console Java qui produit une requête raster sous forme de fichier image

Il s'agit d'une simple application console Java qui prend une requête renvoyant une image et l'envoie dans le fichier spécifié.

Vous pouvez télécharger les derniers drivers JDBC de PostgreSQL à partir de http://jdbc.postgresql.org/download.html

You can compile the following code with commands like:

Code

```
CLASSPATH=".:../postgresql.jar" javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

And call it from the command line with something like:

Code

```
java -jar SaveQueryImage.jar \
  "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1, 5), 10, 'quad_segs=2'), 150, 150, '8BUI ↩
    ', 100));" \
  "test.png"
```

`Manifest.txt`:

Code

```
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage
```

Code for `SaveQueryImage.java`:

Code

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
  public static void main(String[] argv) {
      System.out.println("Checking if Driver is registered with DriverManager.");

      try {
        //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
        Class.forName("org.postgresql.Driver");
      }
      catch (ClassNotFoundException cnfe) {
        System.out.println("Couldn't find the driver!");
        cnfe.printStackTrace();
        System.exit(1);
      }

      Connection conn = null;

      try {
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb", "myuser" ↩
    , "mypwd");
        conn.setAutoCommit(false);

        PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

        ResultSet rs = sGetImg.executeQuery();

        FileOutputStream fout;
        try
        {
            rs.next();
            /** Output to file name requested by user **/
            fout = new FileOutputStream(new File(argv[1]) );
            fout.write(rs.getBytes(1));
            fout.close();
        }
        catch(Exception e)
        {
            System.out.println("Can't create file");
            e.printStackTrace();
        }

        rs.close();
        sGetImg.close();
        conn.close();
      }
      catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
```

```
        }
    }
}
```

10.4.4 Utiliser PLPython pour extraire des images via SQL

Il s'agit d'une fonction stockée en plpython qui crée un fichier dans le répertoire du serveur pour chaque enregistrement. Nécessite l'installation de plpython. Elle devrait fonctionner correctement avec plpythonu et plpython3u.

Code

```
CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;
```

Write out 5 images to the PostgreSQL server in varying sizes. The PostgreSQL daemon account needs write access to the folder; this echoes back the file names created.

Code

```
SELECT write_file(
    ST_AsPNG(
        ST_AsRaster(
            ST_Buffer(ST_Point(1, 5), j * 5, 'quad_segs=2'),
            150 * j,
            150 * j,
            '8BUI',
            100
        )
    ),
    'C:/temp/slices' || j || '.png'
)
FROM generate_series(1, 5) AS j;
```

Output

```
write_file
--------------------
 C:/temp/slices1.png
 C:/temp/slices2.png
 C:/temp/slices3.png
 C:/temp/slices4.png
 C:/temp/slices5.png
```

10.4.5 Sortie de données raster avec PSQL

Sadly PSQL doesn't have easy to use built-in functionality for outputting binaries. This is a bit of a hack that piggy backs on PostgreSQL somewhat legacy large object support. To use it, first launch your **psql** command line connected to your database.

Contrairement à l'approche python, cette approche crée le fichier sur votre ordinateur local.

Code

```
SELECT
    oid,
    lowrite(lo_open(oid, 131072), png) AS num_bytes
FROM (VALUES (
```

```
    lo_create(0),
    ST_AsPNG((SELECT rast FROM aerials.boston WHERE rid = 1))
)) AS v(oid, png);
```

Output

```
oid    | num_bytes
---------+-----------
 2630819 |     74860
```

Note the large object OID and export it to a file on your local computer, replacing the example path with the desired file path.

Code

```
\lo_export 2630819 'C:/temp/aerial_samp.png'
```

This deletes the file from large object storage in the database.

Code

```
SELECT lo_unlink(2630819);
```

Chapitre 11

Référence Raster

`raster` is a PostGIS type for storing and analyzing raster data.

Pour charger des rasters depuis des fichiers, voir Section 10.1

Some examples in this reference use a raster table of dummy data, created with the following code:

Code

```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'0000000000000040' -- scaleX (float64 2)
||
'0000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::raster
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2, ('01000003009A9999999999A93F9A9999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000 ↩
    FFFFFFFF050005000400FDFEFDFEFEFDFEFEFDF9FAFEF' ||
' ↩
    EFCF9FBFDFEFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F5861708704004656648 7A1506
    ')::raster);
```

The functions below are ones which a user of PostGIS Raster is likely to need. There are other raster support functions which are not of interest to a general user.

11.1 Types de données pour la prise en charge raster

11.1.1 geomval

geomval — Un type spatial comportant deux champs : geom (stockant un objet géométrique) et val (stockant la valeur du pixel en double précision depuis une bande du raster).

Description

geomval est un type composite, comprenant un objet géométrique dans le champ .geom et val, une valeur en double précision qui représente la valeur du pixel à un emplacement spécifique d'une bande raster. Ce type est utilisé par la fonction ST_DumpAsPolygon et la famille de fonctions d'intersection Raster, en tant que type de sortie pour éclater une bande raster en polygones.

Voir aussi

Section 13.6

11.1.2 addbandarg

addbandarg — Un type composite utilisé comme entrée pour la fonction ST_AddBand pour définir les attributs et la valeur initiale d'une nouvelle bande.

Description

Un type composite utilisé comme entrée pour la fonction ST_AddBand pour définir les attributs et la valeur initiale d'une nouvelle bande.

Field	Type	Description
`index`	integer	Indice (basé sur 1) où insérer la nouvelle bande dans la liste des bandes du raster. Si NULL, la nouvelle bande sera ajoutée en dernière position.
`pixeltype`	text	Pixel type of the new band. One of the pixel types described in ST_BandPixelType.
`initialvalue`	double precision	Initial value that all pixels of the new band will be set to.
`nodataval`	double precision	Valeur NODATA pour la nouvelle bande. Si NULL, la nouvelle bande n'aura pas de valeur NODATA attribuée.

Voir aussi

ST_AddBand

11.1.3 rastbandarg

rastbandarg — Un type composite pour représenter à la fois un raster et un indice d'une bande de ce raster.

Description

Un type composite pour représenter à la fois un raster et un indice d'une bande de ce raster.

Field	Type	Description
rast	raster	The raster in question.
nband	integer	1-based value indicating the raster band.

Voir aussi

ST_MapAlgebra (callback function version)

11.1.4 raster

raster — Type de données raster spatial.

Description

raster est le type de données spatial pour représenter les données rasters, telles qu'importées depuis des fichiers JPEGs, TIFFs, PNGs, modèles d'élévation numérique. Chaque raster a une ou plusieurs bandes, chacune correspondant à un ensemble de valeurs de pixels. Les rasters peuvent être géo-référencés.



Note
PostGIS doit être compilé avec le support GDAL. Actuellement, les rasters peuvent être implicitement convertis en type geometry, mais la conversion retourne un ST_ConvexHull du raster. Ce transtypage implicite peut être supprimé dans le futur proche et ne devrait pas être utilisé.

Transtypages

Cette section liste les transtypages autorisés pour ce type de donnée, qu'ils soient automatiques ou bien explicites

Transtypage vers	Comportement
geometry	automatique

Voir aussi

Chapitre 11

11.1.5 reclassarg

reclassarg — A composite type used as input into the ST_Reclass function defining the behavior of reclassification.

Field	Type	Description

Description

A composite type used as input into the ST_Reclass function defining the behavior of reclassification.

Field	Type	Description
`nband`	integer	L'indice de bande à reclassifier.
`reclassexpr`	text	Comma-delimited `range:map_range` mappings that define how old band values map to new band values.
`pixeltype`	text	One of defined pixel types as described in ST_BandPixelType.
`nodataval`	double precision	Valeur à utiliser comme NODATA. Pour les formats d'images supportant la transparence, ces pixels seront transparents.

The `reclassexpr` range delimiters mean: `(` greater than, `)` less than, `]` less than or equal, and `[` greater than or equal.

— `[a-b]` means `a <= x <= b`.
— `(a-b]` means `a < x <= b`.
— `[a-b)` means `a <= x < b`.
— `(a-b)` means `a < x < b`.

Opening-parenthesis notation is optional, so `a-b` means the same as `(a-b)`.

Exemples

Reclassify band 2 as an 8BUI where 255 is nodata value.

Code

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

Reclassify band 1 as an 1BB and no nodata value defined.

Code

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

Voir aussi

ST_Reclass

11.1.6 summarystats

summarystats — Un type composite retournée par les fonctions ST_SummaryStats et ST_SummaryStatsAgg.

Description

Un type composite retournée par les fonctions ST_SummaryStats et ST_SummaryStatsAgg.
count integer Nombre de pixels comptabilisés pour les résumés statistiques.
sum double precision Somme des valeurs de tous les pixels comptabilisés.
mean double precision Moyenne arithmétique des valeurs de tous les pixels comptabilisés.
stddev double precision Écart type des valeurs de tous les pixels comptabilisés.
min double precision Minimum des valeurs de tous les pixels comptabilisés.
max double precision Maximum des valeurs de tous les pixels comptabilisés.

Voir aussi

ST_SummaryStats, ST_SummaryStatsAgg

11.1.7 unionarg

unionarg — Un type composite utilisé comme entrée de la fonction ST_Union pour définir les bandes à traiter et le comportement de l'opération d'union.

Description

Un type composite utilisé comme entrée de la fonction ST_Union pour définir les bandes à traiter et le comportement de l'opération d'union.

Field	Type	Description
`nband`	integer	Indice (basé sur 1) indiquant la bande de chaque raster d'entrée à traiter.
`uniontype`	text	Type of union operation. One of the types described in ST_Union.

Voir aussi

ST_Union

11.2 Gestion raster

11.2.1 AddRasterConstraints

AddRasterConstraints — Ajoute des contraintes raster à une table pour une colonne spécifique pour contraindre le système de référence spatial, l'échelle, la taille des blocs, l'alignement, le nombre de bandes, le type de pixels, la contrainte d'unicité spatiale. La table doit être chargée avec des données pour que les contraintes puissent être inférées. Retourne true si les contraintes ont été ajoutées, ou émet une notice sinon.

Synopsis

boolean **AddRasterConstraints**(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true , boolean pixel_types=true , boolean nodata_values=true , boolean out_db=true , boolean extent=true);
boolean **AddRasterConstraints**(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=fa
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true , boolean out_db=true , boolean extent=true);

Description

Génère des contraintes pour une colonne raster, qui peuvent être utilisées pour afficher des informations depuis le catalogue raster `raster_columns`. `rastschema` est le nom du schéma de la table. `srid` doit être un entier défini dans la table SPATIAL_REF_SYS.

Le chargeur de rasters `raster2pgsql` utilise cette fonction pour enregistrer les tables raster

Noms de contraintes à passer en arguments : se référer à Section 10.3.1 pour plus de détails.

— `blocksize` défini la largeur en X et en Y
— `blocksize_x` défini la largeur (nombre de pixels) de chaque tuile raster
— `blocksize_y` défini la hauteur (nombre de pixels vers le bas) de chaque tuile raster
— `extent` calcule l'étendue de l'ensemble de la table et contraint tous les rasters à être inclus dans cette étendue
— `num_bands` nombre de bandes
— `pixel_types` tableau de types de pixels pour chaque bande - contraint chaque bande n au même type de pixels
— `regular_blocking` défini les contraintes d'unité spatiale (deux rasters ne peuvent pas être spatialement identiques) et de couverture de tuiles (raster est aligné sur une couverture)
— `same_alignment` garantie que toutes les tuiles ont le même alignement. Comparer n'importe quel couple de tuiles retournera true. Voir ST_SameAlignment pour plus de détails.
— `srid` garantie que tous les rasters ont le même srid
— Plus -- n'importe quelle contrainte listée parmi les arguments des fonctions décrites ci-dessus



Note
Cette fonction infère les contraintes depuis les données déjà présentes dans la table. Ainsi, vous devez d'abord créer la colonne raster et charger des données dedans avant de pouvoir utiliser cette fonction.



Note
Si vous avez besoin de charger plus de données dans la table après avoir défini les contraintes, vous pourrez avoir besoin d'exécuter la fonction DropRasterConstraints avant le chargement, en particulier si l'étendue des données est modifiée.

Disponibilité : 2.0.0

Exemples

Apply all possible constraints on column based on data.

Verify the raster table is registered correctly in the raster_columns view.

Code

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0, 4326), 1, '8BSI':: ↩
    text, -129, NULL);

SELECT AddRasterConstraints('myrasters'::name, 'rast'::name);
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';
```

Output

```
srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types| nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+---------------
 4326 |       2 |       2 |        1000 |        1000 |         1 | {8BSI}      | {0}
```

Apply single constraint.

Code

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0, 4326), 1, '8BSI':: ↩
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name, ' ↩
    regular_blocking', 'blocksize');
```

Output

```
NOTICE:  Adding regular blocking constraint
NOTICE:  Adding blocksize-X constraint
NOTICE:  Adding blocksize-Y constraint
```

Voir aussi

Section 10.3.1, ST_AddBand, ST_MakeEmptyRaster, DropRasterConstraints, ST_BandPixelType, ST_SRID

11.2.2 DropRasterConstraints

DropRasterConstraints — Supprime des contraintes raster PostGIS raster sur une colonne raster d'une table. Utile pour recharger des données ou mettre à jour les données d'une colonne raster.

Synopsis

boolean **DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale_x, boolean scale_y, boolean blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking, boolean num_bands=true, boolean pixel_types=t
boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=fa
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

Description

Supprime des contraintes raster PostGIS ajoutées via AddRasterConstraints sur une colonne raster d'une table. Utile pour recharger des données ou mettre à jour les données d'une colonne raster. Vous n'avez pas besoin d'appeler cette fonction si vous voulez supprimer une table raster ou une colonne raster.

To drop a raster table use standard SQL: `DROP TABLE mytable`.

To drop just a raster column and leave the rest of the table, use standard SQL: `ALTER TABLE mytable DROP COLUMN rast`.

La table disparaîtra du catalogue `raster_columns` si la colonne ou la table est supprimée. Si seulement les contraintes ont été supprimées, la colonne raster sera toujours listée dans le catalogue `raster_columns`, mais il n'y aura pas d'autre information que les noms de la colonne et de la table.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT DropRasterConstraints ('myrasters', 'rast');
```

Output

```
droprasterconstraints
----------------------
 t
(1 row)
```

Verify change in raster_columns.

Code

```
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';
```

Output

```
srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types| nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+---------------
    0 |         |         |             |             |           |             |
```

Voir aussi

AddRasterConstraints

11.2.3 AddOverviewConstraints

AddOverviewConstraints — Marque une colonne raster comme étant un aperçu d'une autre colonne.

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);
boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

Description

Ajoute des contraintes sur une colonne raster, qui peuvent être utilisées pour afficher des informations depuis le catalogue raster `raster_overviews`.

La paramètre `ovfactor` représente le facture de multiplication d'échelle pour la colonne d'aperçu : plus le facteur d'aperçu est élevé, plus résolution est basse.

Si les paramètres `ovschema` et `refschema` sont omis, la première table trouvée lors du scan de `search_path` sera utilisée.

Disponibilité : 2.0.0

Exemples

Verify the raster table is registered correctly in the raster_overviews view.

Code

```
CREATE TABLE res1 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
  1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(ST_MakeEmptyRaster(500, 500, 0, 0, 4),
  1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
```

Output

```
ot  | oc |  rt  | rc | f
------+----+------+----+---
 res2 | r2 | res1 | r1 | 2
(1 row)
```

Voir aussi

Section 10.3.2, DropOverviewConstraints, ST_CreateOverview, AddRasterConstraints

11.2.4 DropOverviewConstraints

DropOverviewConstraints — Supprime les contraintes d'aperçu de la colonne.

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

Description

Supprime dans le catalogue raster `raster_overviews` les contraintes raster de la colonne indiquant que c'est un aperçu une autre colonne.

Si le paramètre `ovschema` est omis, la première table trouvée lors du scan de `search_path` sera utilisée.

Disponibilité : 2.0.0

Voir aussi

Section 10.3.2, AddOverviewConstraints, DropRasterConstraints

11.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — Retourne la version de la bibliothèque GDAL utilisée par PostGIS.

Synopsis

text **PostGIS_GDAL_Version**();

Description

Retourne la version de la bibliothèque GDAL utilisée par PostGIS. Vérifie également si GDAL peut accéder à ses fichiers de données et reportera une erreur si ce n'est pas le cas.

Exemples

Code

```
SELECT PostGIS_GDAL_Version();
```

Output

```
GDAL 1.11dev, released 2013/04/13
```

Voir aussi

postgis.gdal_datapath

11.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — Retourne la date de compilation de la bibliothèque raster.

Synopsis

text **PostGIS_Raster_Lib_Build_Date**();

Description

Retourne la date de compilation de la bibliothèque raster

Exemples

Code

```
SELECT PostGIS_Raster_Lib_Build_Date();
```

Output

```
2010-04-28 21:15:10
```

Voir aussi

PostGIS_Raster_Lib_Version

11.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — Retourne la version complète de la bibliothèque raster et les informations sur la configuration de la compilation.

Synopsis

text **PostGIS_Raster_Lib_Version**();

Description

Retourne la version complète de la bibliothèque raster et les informations sur la configuration de la compilation.

Exemples

Code

```
SELECT PostGIS_Raster_Lib_Version();
```

Output

```
2.0.0
```

Voir aussi

PostGIS_Lib_Version

11.2.8 ST_GDALDrivers

ST_GDALDrivers — Retourne la liste des formats raster supportés par PostGIS via GDAL. Seuls les formats avec can_write=True peuvent être utilisés par ST_AsGDALRaster

Synopsis

setof record **ST_GDALDrivers**(integer OUT idx, text OUT short_name, text OUT long_name, text OUT can_read, text OUT can_write, text OUT create_options);

Description

Retourne la liste des formats raster supportés par GDAL, sous forme short_name, long_name et creator. Vous pouvez utiliser short_name comme paramètre `format` pour ST_AsGDALRaster. Les options peuvent varier selon les drivers compilés dans votre bibliothèque libgdal. `create_options` retourne un XML contenant un ensemble de CreationOptionList/Option avec comme attribut name et comme attributs optionnels `type` et `description`, ainsi qu'un ensemble de `VALUE` pour chaque option de création pour ce driver.

Changement : 2.5.0 - ajout des colonnes can_read et can_write.

Changement : 2.0.6, 2.1.3 - aucun driver n'est activé par défaut, sauf si la variable GUC ou d'environnement gdal_enabled_drivers est définie.

Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0.

Exemples

List of Drivers.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM ST_GDALDrivers()
ORDER BY short_name;
```

Output

```
short_name    |                        long_name                         | can_write
--------------+----------------------------------------------------------+-----------
 AAIGrid      | Arc/Info ASCII Grid                                      | t
 ACE2         | ACE2                                                     | f
 ADRG         | ARC Digitized Raster Graphics                            | f
 AIG          | Arc/Info Binary Grid                                     | f
 AirSAR       | AirSAR Polarimetric Image                                | f
 ARG          | Azavea Raster Grid format                                | t
 BAG          | Bathymetry Attributed Grid                               | f
 BIGGIF       | Graphics Interchange Format (.gif)                       | f
 BLX          | Magellan topo (.blx)                                     | t
 BMP          | MS Windows Device Independent Bitmap                     | f
 BSB          | Maptech BSB Nautical Charts                              | f
 PAux         | PCI .aux Labelled                                        | f
 PCIDSK       | PCIDSK Database File                                     | f
 PCRaster     | PCRaster Raster File                                     | f
 PDF          | Geospatial PDF                                           | f
 PDS          | NASA Planetary Data System                               | f
 PDS4         | NASA Planetary Data System 4                             | t
 PLMOSAIC     | Planet Labs Mosaics API                                  | f
 PLSCENES     | Planet Labs Scenes API                                   | f
 PNG          | Portable Network Graphics                                | t
 PNM          | Portable Pixmap Format (netpbm)                          | f
 PRF          | Racurs PHOTOMOD PRF                                      | f
 R            | R Object Data Store                                      | t
 Rasterlite   | Rasterlite                                               | t
 RDA          | DigitalGlobe Raster Data Access driver                   | f
 RIK          | Swedish Grid RIK (.rik)                                  | f
 RMF          | Raster Matrix Format                                     | f
 ROI_PAC      | ROI_PAC raster                                           | f
 RPFTOC       | Raster Product Format TOC format                         | f
 RRASTER      | R Raster                                                 | f
 RS2          | RadarSat 2 XML Product                                   | f
 RST          | Idrisi Raster A.1                                        | t
 SAFE         | Sentinel-1 SAR SAFE Product                              | f
 SAGA         | SAGA GIS Binary Grid (.sdat, .sg-grd-z)                  | t
 SAR_CEOS     | CEOS SAR Image                                           | f
 SDTS         | SDTS Raster                                              | f
 SENTINEL2    | Sentinel 2                                               | f
 SGI          | SGI Image File Format 1.0                                | f
 SNODAS       | Snow Data Assimilation System                            | f
 SRP          | Standard Raster Product (ASRP/USRP)                      | f
 SRTMHGT      | SRTMHGT File Format                                      | t
 Terragen     | Terragen heightfield                                     | f
 TIL          | EarthWatch .TIL                                          | f
 TSX          | TerraSAR-X Product                                       | f
 USGSDEM      | USGS Optional ASCII DEM (and CDED)                       | t
 VICAR        | MIPL VICAR file                                          | f
 VRT          | Virtual Raster                                           | t
 WCS          | OGC Web Coverage Service                                 | f
 WMS          | OGC Web Map Service                                      | t
 WMTS         | OGC Web Map Tile Service                                 | t
 XPM          | X11 PixMap Format                                        | t
 XYZ          | ASCII Gridded XYZ                                        | t
 ZMap         | ZMap Plus Grid                                           | t
```

List of options for each driver.

Output the create options XML column of JPEG as a table. These creator options can be used in the ST_AsGDALRaster options argument.

Code

```
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  ST_GDALDrivers()
WHERE short_name = 'JPEG') As g;
```

Output

```
oname        |  otype  |      descrip
-------------------+---------+-------------------------------------------------
 PROGRESSIVE       | boolean | whether to generate a progressive JPEG
 QUALITY           | int     | good=100, bad=0, default=75
 WORLDFILE         | boolean | whether to generate a worldfile
 INTERNAL_MASK     | boolean | whether to generate a validity mask
 COMMENT           | string  | Comment
 SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64
 EXIF_THUMBNAIL    | boolean | whether to generate an EXIF thumbnail(overview).
                               By default its max dimension will be 128
 THUMBNAIL_WIDTH   | int     | Forced thumbnail width
 THUMBNAIL_HEIGHT  | int     | Forced thumbnail height
(9 rows)
```

This example turns a few common GeoTIFF creator options into a compact table.

Code

```
WITH gtiff_options AS (
    SELECT
        option_name,
        option_type,
        default_value
    FROM ST_GDALDrivers() AS driver
    CROSS JOIN LATERAL XMLTABLE(
        '/CreationOptionList/Option'
        PASSING (driver.create_options::xml)
        COLUMNS
            option_name text PATH '@name',
            option_type text PATH '@type',
            default_value text PATH '@default'
    ) AS options
    WHERE driver.short_name = 'GTiff'
)
SELECT option_name AS name, option_type AS type, coalesce(default_value, '') AS ↩
    default_value
FROM gtiff_options
WHERE option_name = ANY (ARRAY[
    'COMPRESS',
    'INTERLEAVE',
    'TILED',
    'BLOCKXSIZE',
    'BLOCKYSIZE',
    'JPEG_QUALITY'
])
ORDER BY 1;
```

Output

```
name      |     type      | default_value
---------------+---------------+---------------
 BLOCKXSIZE    | int           |
 BLOCKYSIZE    | int           |
 COMPRESS      | string-select |
 INTERLEAVE    | string-select | PIXEL
 JPEG_QUALITY  | int           | 75
 TILED         | boolean       |
(6 rows)
```

Use `XMLTABLE` to expose selected GTiff creation options as rows. The same pattern can be used without the name filter to inspect the full XML document.

Code

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';

SELECT option_name, option_type
FROM ST_GDALDrivers() AS driver
CROSS JOIN LATERAL XMLTABLE(
    '/CreationOptionList/Option'
    PASSING (driver.create_options::xml)
    COLUMNS
        option_name text PATH '@name',
        option_type text PATH '@type'
) AS options
WHERE driver.short_name = 'GTiff'
  AND option_name IN ('COMPRESS', 'TILED', 'BIGTIFF')
ORDER BY option_name;
```

Output

```
option_name |  option_type
------------+--------------
 BIGTIFF    | string-select
 COMPRESS   | string-select
 TILED      | boolean
(3 rows)
```

Voir aussi

ST_AsGDALRaster, ST_SRID, postgis.gdal_enabled_drivers

11.2.9 UpdateRasterSRID

UpdateRasterSRID — Change le SRID de tous les rasters dans la table et colonne en paramètres.

Synopsis

raster **UpdateRasterSRID**(name schema_name, name table_name, name column_name, integer new_srid);
raster **UpdateRasterSRID**(name table_name, name column_name, integer new_srid);

Description

Change le SRID de tous les rasters dans la table et colonne en paramètres. Cette fonction supprimera toutes les contraintes nécessaires (étendue, alignement et SRID) avant de changer le SRID de la colonne raster.



Note
Les données (les valeurs des pixels) des rasters ne sont pas changées par cette fonction. Seules les métadonnées du rasters sont modifiées.

Disponibilité : 2.1.0

Voir aussi

UpdateGeometrySRID

11.2.10 ST_CreateOverview

ST_CreateOverview — Créé une version à plus faible résolution d'une colonne raster.

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

Description

Créé une table d'aperçu avec les tuiles rééchantillonnées depuis la table source. Les tuiles de sortie auront la même taille que les tuiles d'entrées et couvriront la même étendue, mais avec une résolution plus faible (la taille d'un pixel sera 1/`factor` dans les deux directions).

La table d'aperçu sera disponible dans le catalogue `raster_overviews` avec les contraintes rasters ajoutées.

Les algorithmes disponibles sont : 'NearestNeighbor' (plus proche voisin), 'Bilinear' (Bilinéaire), 'Cubic' (Cubique), 'CubicSpline' (Cubique Spline) et 'Lanczos'. Voir GDAL Warp resampling methods pour plus de détails.

Disponibilité : 2.2.0

Exemple

Sortie de meilleure qualité en général, mais plus lent à générer

Code

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Sortie plus rapide, via la méthode de plus proche voisin

Code

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

Voir aussi

ST_Retile, AddOverviewConstraints, AddRasterConstraints, Section 10.3.2

11.3 Constructeurs de raster

11.3.1 ST_AddBand

ST_AddBand — Retourne un raster avec la/les nouvelle(s) bande(s) ajouté(s) à un index donné, de type et valeur initiale donnés. Si aucun index n'est spécifié, la bande est ajoutée à la fin.

Synopsis

(1) raster **ST_AddBand**(raster rast, addbandarg[] addbandargset);
(2) raster **ST_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
(3) raster **ST_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
(4) raster **ST_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at_end);
(5) raster **ST_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at_end);
(6) raster **ST_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
(7) raster **ST_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at_end, double precision nodataval=NULL);

Description

Renvoie un raster avec une nouvelle bande ajoutée à la position donnée (index), du type donné, de la valeur initiale donnée et de la valeur de nodata donnée. Si aucun index n'est spécifié, la bande est ajoutée à la fin. Si aucun `fromband` n'est spécifié, la bande 1 est supposée. Le type de pixel est une représentation sous forme de chaîne de l'un des types de pixels spécifiés dans ST_BandPixelType. Si un index existant est spécifié, toutes les bandes suivantes >= cet index sont incrémentées de 1. Si une valeur initiale supérieure à la valeur maximale du type de pixel est spécifiée, la valeur initiale est fixée à la valeur la plus élevée autorisée par le type de pixel.

Pour la variante acceptant un tableau de addbandarg en paramètre (Variante 1), un index spécifique au paramètre addbandarg est relative au raster au moment où la bande décrite par ce paramètre addbandarg est ajoutée au raster. Voir l'exemple Multiples nouvelles bandes ci-dessous.

Pour la variante acceptant un tableau de rasters (Variante 5), si `torast` est NULL, alors la bande `fromband` de chaque raster est accumulée dans le nouveau raster.

Pour les variantes acceptant un paramètre `outdbfile` (Variantes 6 et 7), la valeur doit correspondre au chemin complet vers le fichier raster. Ce fichier doit être accessible par le processus du serveur postgres.

Amélioration : 2.1.0 ajout du paramètre addbandarg.

Amélioration : 2.1.0 introduction du support des bandes out-db.

Exemples

Single New Band.

This example adds another 8-bit unsigned integer band with pixels initialized to 200.

Code

```
UPDATE dummy_rast
    SET rast = ST_AddBand(rast, '8BUI'::text, 200)
WHERE rid = 1;
```

Create an empty raster 100x100 units, with upper left right at 0, add 2 bands (band 1 is 0/1 boolean bit switch, band2 allows values 0-15) Uses addbandargs.

Output metadata for the raster bands to verify they are correct.

Code

```
INSERT INTO dummy_rast(rid, rast)
    VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
        ROW(1, '1BB'::text, 0, NULL),
        ROW(2, '4BUI'::text, 0, NULL)
            ]::addbandarg[]
     )
    );

SELECT   (bmd).*
FROM (SELECT ST_BandMetaData(rast, generate_series(1, 2)) As bmd
    FROM dummy_rast WHERE rid = 10) AS foo;
```

Output

```
pixeltype | nodatavalue | isoutdb | path
-----------+--------------+-------------+---------+------
 1BB       |             | f       |
 4BUI      |             | f       |
```

This example outputs metadata for the raster.

Code

```
SELECT  (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
    FROM dummy_rast WHERE rid = 10) AS foo;
```

Output

```
width | height | numbands
-------+--------+----------
   100 |    100 |        2
```

Multiple New Bands.

Code

```
SELECT
    *
FROM ST_BandMetaData(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
);
```

Output

```
bandnum | pixeltype | nodatavalue | isoutdb | path | outdbbandnum | filesize | filetimestamp
---------+-----------+-------------+---------+------+--------------+----------+---------------
       1 | 8BUI      |           0 | f       | null |         null |     null |          null
       2 | 32BF      |          -1 | f       | null |         null |     null |          null
       3 | 32BUI     |          12 | f       | null |         null |     null |          null
       4 | 16BUI     |           2 | f       | null |         null |     null |          null
```

Aggregate the 1st band of a table of like rasters into a single raster. With as many bands as there are test_types and as many rows (new rasters) as there are mice. Note that The ORDER BY test_type clause relies on aggregate ORDER BY support. The resulting raster will have a band for each test_type alphabetical by test_type. For mouse lovers: No mice were harmed in this exercise.

Code

```
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

New Out-db band.

Code

```
SELECT
    *
FROM ST_BandMetaData(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        '/home/raster/mytestraster.tif'::text, NULL::int[]
    ),
    ARRAY[]::integer[]
);
```

Output

```
bandnum | pixeltype | nodatavalue | isoutdb | path
---------+-----------+-------------+---------+------
       1 | 8BUI      |             | t       | /home/raster/mytestraster.tif
       2 | 8BUI      |             | t       | /home/raster/mytestraster.tif
       3 | 8BUI      |             | t       | /home/raster/mytestraster.tif
```

Voir aussi

ST_BandMetaData, ST_BandPixelType, ST_MakeEmptyRaster, ST_MetaData, ST_NumBands, ST_Reclass

11.3.2 ST_AsRaster

ST_AsRaster — Convertit une géométrie PostGIS en un raster PostGIS.

Synopsis

raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

Description

Convertit une géométrie PostGIS en un raster PostGIS. Les nombreuses variantes permettent trois groupes de possibilités pour définir l'alignement et la taille des pixels du raster en sortie.

Le premier groupe, constitué des deux premières variantes, produit un raster avec le même alignement (`scalex`, `scaley`, `gridx` et `gridy`), le même type de pixels et la même valeur nodata que le raster donné en référence. En général, vous passerez cette référence en faisant une jointure entre la table contenant la géométrie avec la table contenant le raster de référence.

Le second groupe, composé de quatre variantes, vous permettent de spécifier les dimensions du raster via les paramètres de taille de pixel (`scalex` & `scaley` et `skewx` & `skewy`). Les dimensions `width` & `height` du raster résultant seront ajustés pour s'adapter à l'étendu de la géométrie. Dans la plupart des cas, vous devrez transtyper les paramètres `scalex` & `scaley` de integer à double precision pour que PostgreSQL choisisse la variante correcte.

Le troisième groupe, composé de quatre variantes, vous permettent de spécifier les dimensions du raster directement (`width` & `height`). Les paramètres de taille de pixel (`scalex` & `scaley` et `skewx` & `skewy`) du raster résultant seront ajustés pour s'adapter à l'étendue de la géométrie.

Les deux premières variantes des deux derniers groupes vous permettent de spécifier l'alignement avec un coin arbitraire de la grille d'alignement (`gridx` & `gridy`) et les deux dernières variantes via le coin haut-gauche (`upperleftx` & `upperlefty`).

Chaque groupe de variantes permet de produire un raster à une ou plusieurs bandes. Pour plusieurs bandes, vous devez fournir un tableau de types de pixels (`pixeltype[]`), un tableau de valeurs initiales (`value`) et un tableau de valeurs nodata (`nodataval`). Si non fournis, pixeltype est par défaut à 8BUI, value à 1 et nodataval à 0.

Le raster en sortie aura le même système de référence spatial que la géométrie source, sauf pour les variantes utilisant un raster de référence. Dans ce cas, le raster en sortie aura le même SRID que le raster de référence.

Le paramètre optionnel `touched` est par défaut à false et correspond à l'option GDAL ALL_TOUCHED, qui détermine si les pixels touchés par des lignes ou polygones seront mis à jour, et pas seulement ceux sur le chemin de la ligne ou dont le point central est dans le polygone.

Ceci est particulièrement utile pour le rendu de JPEGs et de PNGs directement depuis la base de données en utilisant une combinaison de ST_AsPNG et des fonctions de la famille de ST_AsGDALRaster.

Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0.

Curved geometry inputs are linearized before rasterization, including NURBSCurve inputs handled by PostGIS. TIN and PolyhedralSurface inputs are not supported.

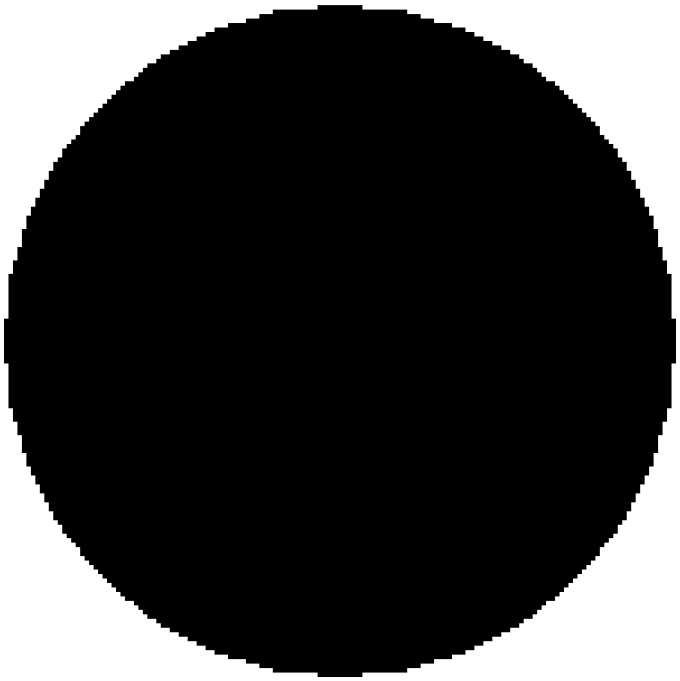
Exemples : Rendu de géométries en fichiers PNG

This example outputs a black circle on a white background, occupying 150 by 150 pixels.

Code

```
SELECT ST_AsPNG(
    ST_AsRaster(
        ST_Buffer(ST_Point(1, 5), 10),
        150, 150,
        ARRAY['8BUI', '8BUI', '8BUI'],
        ARRAY[0, 0, 0],
        ARRAY[255, 255, 255]
    )
) AS image;
```

Figure



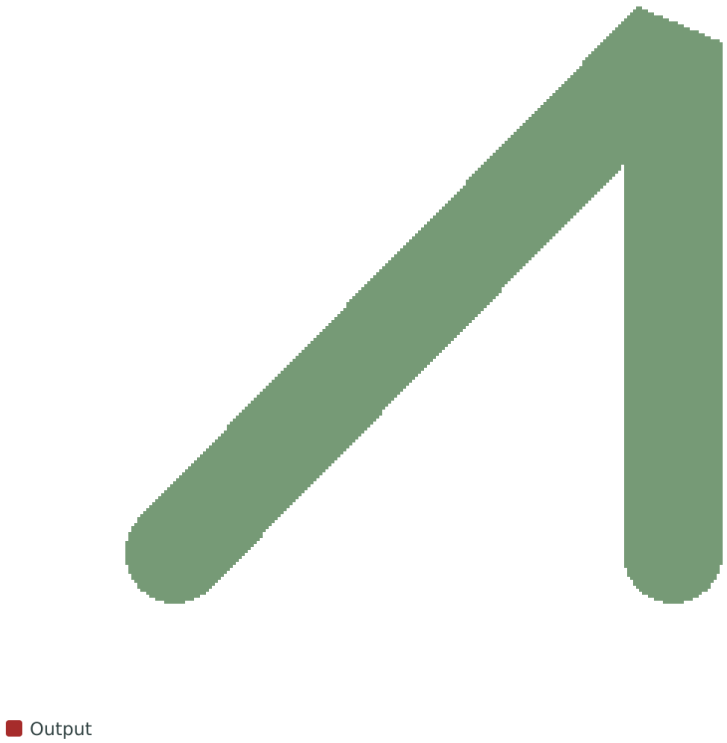
Output

This example maps bands to RGB channels, producing the teal value (118,154,118).

Code

```
SELECT ST_AsPNG(
    ST_AsRaster(
        ST_Buffer('LINESTRING(50 50,150 150,150 50)'::geometry, 10, 'join=bevel'),
        200, 200,
        ARRAY['8BUI', '8BUI', '8BUI'],
        ARRAY[118, 154, 118],
        ARRAY[255, 255, 255]
    )
) AS image;
```

Figure



Voir aussi

ST_BandPixelType, ST_Buffer, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsJPEG, ST_SRID

11.3.3 ST_AsRasterAgg

ST_AsRasterAgg — Aggregate. Rend les géométries PostGIS dans un nouveau raster.

Synopsis

raster **ST_AsRasterAgg**(geometry geom, double precision val, raster ref, text pixeltype, double precision nodataval, text uniontype, boolean touched);

Description

Renvoie un raster à bande unique contenant la version rendue de toutes les géométries entrantes, chacune avec sa valeur associée.

Disponibilité : 3.6.0

Exemples

Code

```
WITH inp(g, v) AS (
    VALUES
        ( ST_Buffer(ST_MakePoint(10, 0), 10), 1 ),
        ( ST_Buffer(ST_MakePoint(20, 0), 10), 2 )
),
agg AS (
    SELECT ST_AsRasterAgg(
        g,
        v,
        ST_MakeEmptyRaster(0, 0, 0, 0, 1.0),
        '8BUI',
        99,
        'SUM',
        true
    ) r
    FROM inp
)
SELECT
    ST_Width(r) w,
    ST_Height(r) h,
    ST_Value(r, 'POINT(5 0)') v5_0,
    ST_Value(r, 'POINT(15 0)') v15_0,
    ST_Value(r, 'POINT(25 0)') v25_0
FROM agg;
```

Output

```
w  | h  | v5_0 | v15_0 | v25_0
----+----+------+-------+-------
 30 | 20 |    1 |     3 |     2
(1 row)
```

Voir aussi

ST_AsRaster, ST_DumpAsPolygons, ST_Union

11.3.4 ST_Band

ST_Band — Retourne une ou plusieurs bandes d'un raster existant en tant que nouveau raster. Utile pour construire de nouveaux rasters à partir de rasters existants.

Synopsis

raster **ST_Band**(raster rast, integer[] nbands = ARRAY[1]);
raster **ST_Band**(raster rast, integer nband);
raster **ST_Band**(raster rast, text nbands, character delimiter=,);

Description

Retourne une ou plusieurs bandes d'un raster existant en tant que nouveau raster. Utile pour construire de nouveaux rasters à partir de rasters existants ou pour exporter certaines bandes d'un raster, ou pour réarranger l'ordre des bandes d'un raster. Si aucune bande n'est spécifiée, ou si aucune des bandes spécifiées n'existent dans le raster, alors toutes les bandes sont retournées. Cette fonction est utilisée en interne par diverses fonctions, comme pour supprimer une bande.



AVERTISSEMENT

Pour les variantes avec le paramètre `nbands` en texte, le délimiteur par défaut est `,`, il faut donc passer par exemple `'1,2,3'`. Pour utiliser un autre délimiteur, utilisez le paramètre delimiter : `ST_Band(rast, '1@2@3', '@')`. Néanmoins, pour utiliser plusieurs bandes, nous encourageons fortement d'utiliser la variante avec le paramètre de type tableau (e.g. `ST_Band(rast, '{1,2,3}' ::int[]) ;`) ; la variante avec la liste de bandes en `text` peut être supprimée des prochaines versions de PostGIS.

Disponibilité : 2.0.0

Exemples

This example creates two rasters: one containing band 1 of dummy_rast, and one containing band 2 reclassified as 2BUI.

Code

```
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
 ST_NumBands(rast2) As numb2, ST_BandPixelType(rast2) As pix2
FROM (
    SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast, 3), '100-200):1, [200-254:2', '2 ↩
    BUI') As rast2
        FROM dummy_rast
        WHERE rid = 2) As foo;
```

Output

```
numb1 | pix1 | numb2 | pix2
-------+------+-------+------
     1 | 8BUI |     1 | 2BUI
```

Return bands 2 and 3 using array cast syntax.

Code

```
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
FROM dummy_rast WHERE rid=2;
```

Output

```
2
```

Return bands 2 and 3 using array syntax.

Code

```
SELECT ST_NumBands(ST_Band(rast, ARRAY[2, 3])) As num_bands
    FROM dummy_rast
WHERE rid=2;
```

Output

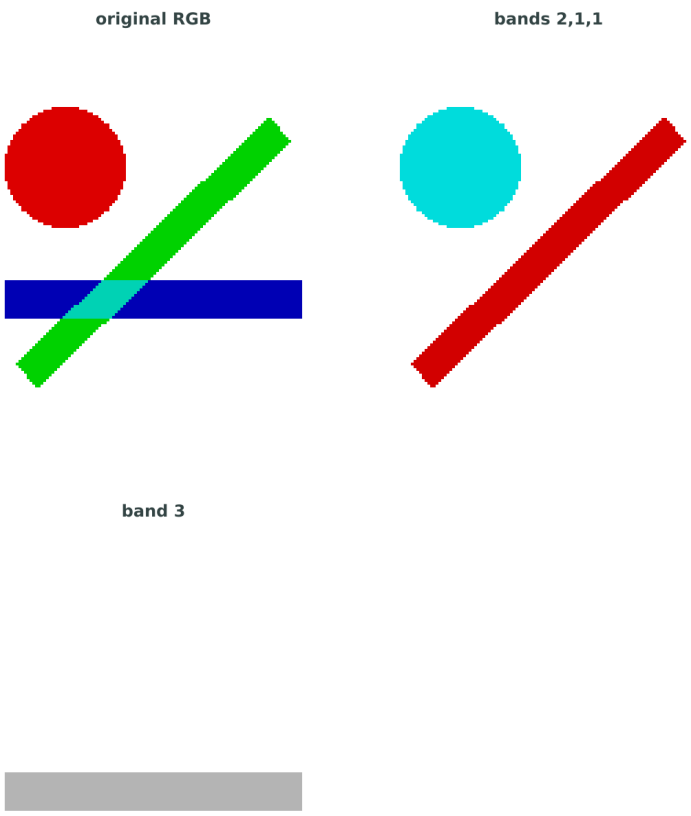
```
2
```

This example uses three distinct shapes as the red, green, and blue bands of a synthetic raster. It compares the original RGB raster with band 2 followed by band 1 twice, and with band 3 alone.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(128, 128, 0, 128, 1, -1, 0, 0, 0),
                '8BUI'::text, 0::double precision, 0::double precision
            ),
            '8BUI'::text, 0::double precision, 0::double precision
        ),
        '8BUI'::text, 0::double precision, 0::double precision
    ) AS rast
), bands AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_SetValue(rast, 1, ST_Buffer('POINT(32 96)'::geometry, 22), 220),
            2, ST_Buffer('LINESTRING(18 20,110 110)'::geometry, 6,
                         'endcap=flat join=bevel'), 210
        ),
        3, ST_Buffer('LINESTRING(10 48,118 48)'::geometry, 7,
                     'endcap=flat'), 180
    ) AS rast
    FROM canvas
)
SELECT ST_AsPNG(rast) AS "original RGB",
    ST_AsPNG(ST_Band(rast, ARRAY[2, 1, 1])) AS "bands 2,1,1",
    ST_AsPNG(ST_Band(rast, 3)) AS "band 3"
FROM bands;
```

Figure



Band selection from a multi-band raster.



originale (colonne rast)



dupe_band



sing_band

Voir aussi

ST_AddBand, ST_NumBands, ST_Reclass, Chapitre 11

11.3.5 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — Couvre une zone géo-référencée avec une grille raster de tuiles vides.

Synopsis

raster **ST_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

Description

Créé un ensemble de tuiles rasters via ST_MakeEmptyRaster. La dimension de la grille est définie par `width` & `height`. La dimension des tuiles est définie par `tilewidth` & `tileheight`. La zone géo-référencée couverte démarre au coin supérieur gauche (`upperleftx`, `upperlefty`) jusqu'au coin inférieur droit (`upperleftx` + `width` * `scalex`, `upperlefty` + `height` * `scaley`).



Note
A noter que scaley est généralement négatif pour les rasters, alors que scalex est généralement positif. Le coin inférieur droit aura une valeur Y inférieure et une valeur X supérieure au coin supérieur gauche.

Disponibilité : 2.4.0

Exemples de base

Créer 16 tuiles de 1x1px dans une grille 4x4 pour couvrir une zone WGS84 du coin supérieur gauche (22, 77) jusqu'au coin inférieur droit (55, 33).

Code

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4):: ↩
    float, (33 - 77)/(4)::float, 0., 0., 4326) tile;
```

Output

```
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----------------------------------------------------------------------------------------
         22 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
         22 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
         22 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
         22 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      30.25 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
       38.5 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
      46.75 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |        0
```

Voir aussi

ST_MakeEmptyRaster

11.3.6 ST_MakeEmptyRaster

ST_MakeEmptyRaster — Retourne un raster vide (sans aucune bande) de dimension donnée (width & height), de coin supérieur gauche à X et Y, de paramètres de taille de pixel données (scalex, scaley, skewx & skewy) et de système de référence spatial (srid) donné. Si un raster est spécifié, retourne un nouveau raster de même taille, alignement et SRID. Si srid n'est pas spécifié, le système de référence spatial est défini à inconnu (0).

Synopsis

raster **ST_MakeEmptyRaster**(raster rast);
raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley, float8 skewx, float8 skewy, integer srid=unknown);
raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 pixelsize);

Description

Retourne un raster vide (sans aucune bande) de dimension donnée (width & height), géo-référencé via le coin supérieur gauche (upperleftx & upperlefty), via les paramètres de taille de pixel données (scalex, scaley, skewx & skewy) et via le système de référence spatial (srid) donné.

La dernière variante utilise un seul paramètre pour spécifier la taille des pixels (pixelsize). scalex est défini à la valeur de ce paramètre et scaley est défini à la valeur négative de ce paramètre. skewx et skewy sont définis à 0.

Si un raster existant est spécifié, retourne un nouveau raster avec les mêmes métadonnées (sans aucune bande).

Si aucun srid n'est spécifié, la valeur 0 est utilisée. Après avoir créé un raster vide, vous voudrez probablement ajouter des bandes ou l'éditer. Voir ST_AddBand pour définir des bandes et ST_SetValue pour initialiser des valeurs.

Exemples

Use an existing raster as a template and output metadata of the rasters we added.

Code

```
INSERT INTO dummy_rast(rid, rast)
VALUES(3, ST_MakeEmptyRaster(100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );
INSERT INTO dummy_rast(rid, rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;
SELECT
    rid,
    (md).width,
    (md).height,
    (md).scalex,
    (md).scaley,
    (md).srid
FROM (
    SELECT rid, ST_MetaData(rast) AS md
    FROM dummy_rast
    WHERE rid IN (3, 4)
) AS foo
ORDER BY rid;
```

Output

```
rid | width | height | scalex | scaley | srid
----+-------+--------+--------+--------+------
  3 |   100 |    100 |      1 |      1 | 4326
  4 |   100 |    100 |      1 |      1 | 4326
```

Voir aussi

ST_AddBand, ST_MetaData, ST_ScaleX, ST_ScaleY, ST_SetValue, ST_SkewX, , ST_SkewY

11.3.7 ST_Tile

ST_Tile — Retourne un ensemble de rasters issus de la division d'un raster d'entrée selon les dimensions spécifiées pour les rasters de sortie.

Synopsis

setof raster **ST_Tile**(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

Description

Retourne un ensemble de rasters issus de la division d'un raster d'entrée selon les dimensions spécifiées pour les rasters de sortie.

Si `padwithnodata` est FALSE, les tuiles des bords droits et bas peuvent avoir des dimensions différentes du reste des tuiles. Si `padwithnodata` est TRUE, toutes les tuiles auront les mêmes dimensions, potentiellement avec les tuiles des bords complétées par la valeur NODATA. Si la bande raster n'a pas de valeur NODATA définie, cette valeur peut être spécifiée via le paramètre `nodataval`.



Note
Si une bande spécifiée du raster d'entrée est out-of-db, la bande correspondant dans les rasters de sortie sera aussi out-of-db.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
```

```
    SELECT
        row_number() OVER (
            ORDER BY ST_UpperLeftY(rast) DESC, ST_UpperLeftX(rast)
        ) AS tile_id,
        rast
    FROM (
        SELECT ST_Tile(rast, 3, 3, TRUE) AS rast
        FROM bar
    ) AS tiled
)
SELECT
    tile_id,
    ST_UpperLeftX(rast) AS ulx,
    ST_UpperLeftY(rast) AS uly,
    ST_Value(rast, 1, 1, 1) AS band1,
    ST_Value(rast, 2, 1, 1) AS band2
FROM baz;
```

Output

```
tile_id | ulx | uly | band1 | band2
---------+-----+-----+-------+-------
       1 |   0 |   0 |     1 |    10
       2 |   3 |   0 |     2 |    20
       3 |   6 |   0 |     3 |    30
       4 |   0 |  -3 |     4 |    40
       5 |   3 |  -3 |     5 |    50
       6 |   6 |  -3 |     6 |    60
       7 |   0 |  -6 |     7 |    70
       8 |   3 |  -6 |     8 |    80
       9 |   6 |  -6 |     9 |    90
(9 rows)
```

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT
        row_number() OVER (
            ORDER BY ST_UpperLeftY(rast) DESC, ST_UpperLeftX(rast)
        ) AS tile_id,
        rast
    FROM (
```

```
        SELECT ST_Tile(rast, ARRAY[2], 3, 3) AS rast
        FROM bar
    ) AS tiled
)
SELECT
    tile_id,
    ST_UpperLeftX(rast) AS ulx,
    ST_UpperLeftY(rast) AS uly,
    ST_Value(rast, 1, 1, 1) AS band_value
FROM baz;
```

Output

```
tile_id | ulx | uly | band_value
---------+-----+-----+------------
       1 |   0 |   0 |         10
       2 |   3 |   0 |         20
       3 |   6 |   0 |         30
       4 |   0 |  -3 |         40
       5 |   3 |  -3 |         50
       6 |   6 |  -3 |         60
       7 |   0 |  -6 |         70
       8 |   3 |  -6 |         80
       9 |   6 |  -6 |         90
(9 rows)
```

Voir aussi

ST_Union, ST_Retile

11.3.8 ST_Retile

ST_Retile — Retourne un ensemble de tuiles configurées à partir d'une couverture raster composée de tuiles arbitraires.

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Description

Retourne un ensemble de tuiles ayant l'échelle spécifiée (`sfx`, `sfy`), de taille maximale (`tw`, `th`) et couvrant l'étendue spécifiée (`ext`), et avec les données provenant de la couverture raster spécifiée (`tab`, `col`).

Les algorithmes disponibles sont : 'NearestNeighbor' (plus proche voisin), 'Bilinear' (Bilinéaire), 'Cubic' (Cubique), 'CubicSpline' (Cubique Spline) et 'Lanczos'. Voir GDAL Warp resampling methods pour plus de détails.

Disponibilité : 2.2.0

Voir aussi

ST_CreateOverview

11.3.9 ST_FromGDALRaster

ST_FromGDALRaster — Retourne un raster depuis un fichier raster supporté par GDAL.

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Description

Retourne un raster depuis un fichier raster supporté par GDAL. Le paramètre `gdaldata` de type bytea est le contenu du fichier raster GDAL.

Si `srid` est NULL, la fonction essaye de déterminer le SRID depuis le raster GDAL. Si `srid` est spécifié, la valeur spécifiée écrase tout SRID déterminé automatiquement.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1,  ↩
    -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png
),
bar AS (
    SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
    UNION ALL
    SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
)
SELECT
    rid,
    ST_MetaData(rast) AS metadata,
    ST_SummaryStats(rast, 1) AS stats1,
    ST_SummaryStats(rast, 2) AS stats2,
    ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;
```

Output

```
rid |         metadata          |    stats1     |    stats2     |     stats3
-----+---------------------------+---------------+---------------+----------------
   1 | (0,0,2,2,1,-1,0,0,0,3)    | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
   2 | (0,0,2,2,1,-1,0,0,3310,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
(2 rows)
```

Voir aussi

ST_AsGDALRaster

11.4 Fonctions d'accès aux rasters

11.4.1 ST_GeoReference

ST_GeoReference — Retourne les méta-données de géo-référencement au format GDAL (par défaut) ou ESRI, tel qu'utilisé généralement dans les fichiers world file.

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

Description

Retourne les méta-données de géo-référencement au format GDAL ou ESRI, tel qu'utilisé généralement dans les fichiers world file. Si aucun type n'est spécifié, utilise le format GDAL par défaut. type est une chaîne 'GDAL' ou 'ESRI'.

La différence entre les représentations de format est la suivante :

GDAL :

Output

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

ESRI :

Output

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Exemples

Code

```
SELECT
    format,
    replace(ST_GeoReference(rast, format), E'\n', ' | ') AS georef
FROM dummy_rast
CROSS JOIN (VALUES ('ESRI'), ('GDAL')) AS formats(format)
WHERE rid = 1;
```

Output

```
format | georef
--------+-------------------------------------------------------------------------
 ESRI   | 2.0000000000 | 0.0000000000 | 0.0000000000 | 3.0000000000 | 1.5000000000 | 2.0000000000
 GDAL   | 2.0000000000 | 0.0000000000 | 0.0000000000 | 3.0000000000 | 0.5000000000 | 0.5000000000
(2 rows)
```

Voir aussi

ST_SetGeoReference, ST_ScaleX, ST_ScaleY

11.4.2 ST_Height

ST_Height — Retourne la hauteur du raster en pixels.

Synopsis

integer **ST_Height**(raster rast);

Description

Retourne la hauteur du raster en pixels.

Exemples

Code

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

Output

```
rid | rastheight
-----+------------
   1 |         20
   2 |          5
```

Voir aussi

ST_Width

11.4.3 ST_IsEmpty

ST_IsEmpty — Retourne true si le raster est vide (largeur = 0 et hauteur = 0). Sinon, retourne false.

Synopsis

boolean **ST_IsEmpty**(raster rast);

Description

Retourne true si le raster est vide (largeur = 0 et hauteur = 0). Sinon, retourne false.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
```

Output

```
f
```

Code

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
```

Output

```
t
```

Voir aussi

ST_HasNoBand

11.4.4 ST_MemSize

ST_MemSize — Retourne l'espace utilisé par le raster (en octets).

Synopsis

integer **ST_MemSize**(raster rast);

Description

Retourne l'espace utilisé par le raster (en octets).

Cette fonction est un bon complément aux fonctions PostgreSQL natives pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relation_size.



Note
pg_relation_size qui donne la taille en octets d'une table peut renvoyer une taille en octets inférieure à ST_MemSize. Cela est dû au fait que pg_relation_size n'ajoute pas la contribution des tables toasted et que les grandes géométries sont stockées dans les tables TOAST. pg_column_size peut retourner une valeur inférieure parce qu'elle retourne la taille compressée.
pg_total_relation_size - comprend la table, les tables toasted et les index.

Disponibilité : 2.2.0

Exemples

Code

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1, 5), 10, 1000), 150, 150, '8BUI')) AS ↩
    rast_mem;
```

Output

```
22568
```

Voir aussi

11.4.5 ST_MetaData

ST_MetaData — Retourne les méta-données de base de l'objet raster : taille des pixels, rotation, coin haut/bas gauche, etc.

Synopsis

record **ST_MetaData**(raster rast);

Description

Retourne les méta-données de base de l'objet raster : taille des pixels, rotation, coin haut/bas gauche, etc. Les colonnes retournées sont : upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

Exemples

Code

```
SELECT
    rid,
    format('(%s, %s)', (md).upperleftx, (md).upperlefty) AS upper_left,
    format('%sx%s', (md).width, (md).height) AS size,
    format('(%s, %s)', (md).scalex, (md).scaley) AS scale,
    format('(%s, %s)', (md).skewx, (md).skewy) AS skew,
    (md).srid AS srid,
    (md).numbands AS numbands
FROM (
    SELECT rid, ST_MetaData(rast) AS md
    FROM dummy_rast
) AS foo
ORDER BY rid;
```

Output

```
rid |        upper_left         | size  |      scale      |  skew  | srid | numbands
-----+---------------------------+-------+-----------------+--------+------+----------
   1 | (0.5, 0.5)                | 10x20 | (2, 3)          | (0, 0) |    0 |        0
   2 | (3427927.75, 5793244)     | 5x5   | (0.05, -0.05)   | (0, 0) |    0 |        3
(2 rows)
```

Voir aussi

ST_BandMetaData, ST_NumBands

11.4.6 ST_NumBands

ST_NumBands — Retourne le nombre de bandes de l'objet raster.

Synopsis

integer **ST_NumBands**(raster rast);

Description

Retourne le nombre de bandes de l'objet raster.

Exemples

Code

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

Output

```
rid | numbands
----+----------
  1 |        0
  2 |        3
```

Voir aussi

ST_Value

11.4.7 ST_PixelHeight

ST_PixelHeight — Retourne la hauteur d'un pixel, dans l'unité du système de référence spatial.

Synopsis

double precision **ST_PixelHeight**(raster rast);

Description

Retourne la hauteur d'un pixel, dans l'unité du système de référence spatial. Dans le cas général où il n'y a pas d'obliquité (skew), la hauteur d'un pixel est simplement l'échelle entre les coordonnées géométriques et les pixels rasters.

Voir ST_PixelWidth pour une représentation schématique de la relation.

Exemples

Rasters with no skew.

Code

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
        ST_SkewY(rast) As skewy
FROM dummy_rast;
```

Output

```
rastheight | pixheight | scalex | scaley | skewx | skewy
-----------+-----------+--------+--------+-------+----------
        20 |         3 |      2 |      3 |     0 |        0
         5 |      0.05 |   0.05 |  -0.05 |     0 |        0
```

Rasters with skew different than 0.

Code

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
        ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast, 0.5, 0.5) As rast
        FROM dummy_rast) As skewed;
```

Output

```
rastheight |     pixheight     | scalex | scaley | skewx | skewy
-----------+-------------------+--------+--------+-------+-------
        20 |  3.04138126514911 |      2 |      3 |   0.5 |   0.5
         5 | 0.502493781056044 |   0.05 |  -0.05 |   0.5 |   0.5
```

Voir aussi

ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.4.8 ST_PixelWidth

ST_PixelWidth — Retourne la largeur d'un pixel, dans l'unité du système de référence spatial.

Synopsis

double precision **ST_PixelWidth**(raster rast);

Description

Retourne la largeur d'un pixel, dans l'unité du système de référence spatial. Dans le cas général où il n'y a pas d'obliquité (skew), la largeur d'un pixel est simplement l'échelle entre les coordonnées géométriques et les pixels rasters.

The following executable example draws the two pixel basis vectors on the same coordinate frame as a skewed raster footprint. The `i direction` vector is `(scaleX, skewY)`; the `j direction` vector is `(skewX, scaleY)`.

Exemples

Pixel basis vectors and footprint for a raster with non-zero skew.

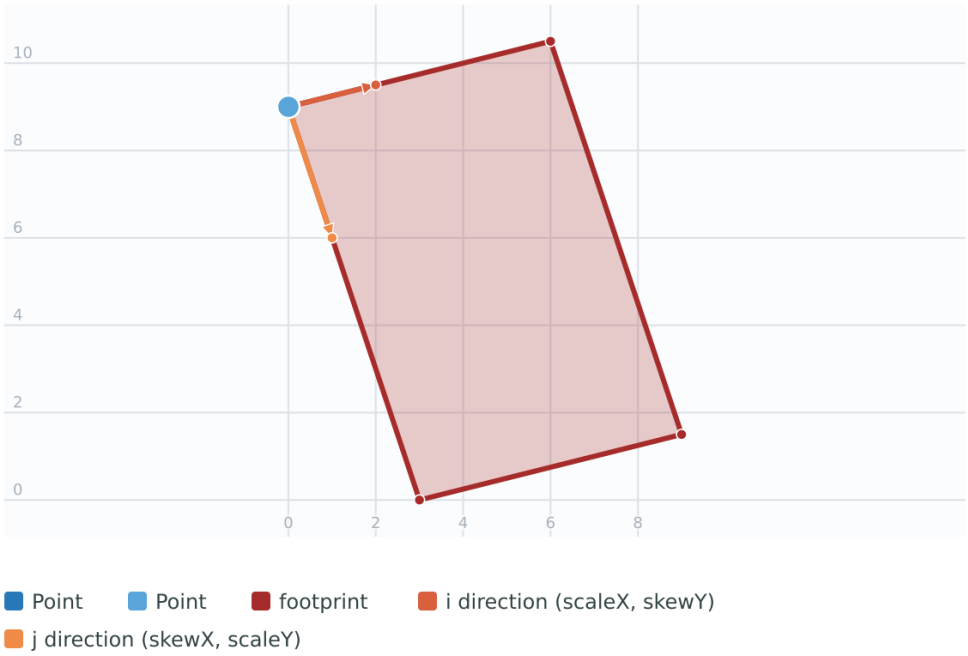
Code

```
WITH raster AS (
    SELECT ST_MakeEmptyRaster(
        3, 3, 0, 0,
        2, -3, 1, 0.5,
        0
    ) AS rast
)
SELECT
    ST_ConvexHull(rast) AS footprint,
    ST_MakeLine(
        ST_Point(0, 0),
        ST_Point(ST_ScaleX(rast), ST_SkewY(rast))
    ) AS "i direction (scaleX, skewY)",
    ST_MakeLine(
        ST_Point(0, 0),
        ST_Point(ST_SkewX(rast), ST_ScaleY(rast))
    ) AS "j direction (skewX, scaleY)"
FROM raster;
```

Output

```
POLYGON((0 0,6 1.5,9 -7.5,3 -9,0 0)) | LINESTRING(0 0,2 0.5) | LINESTRING(0 0,1 -3)
```

Figure



Rasters with no skew.

Code

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
ST_SkewY(rast) As skewy
FROM dummy_rast;
```

Output

```
rastwidth | pixwidth | scalex | scaley | skewx | skewy
-----------+----------+--------+--------+-------+-----------
10 |        2 |      2 |      3 |     0 |        0
 5 |     0.05 |   0.05 |  -0.05 |     0 |        0
```

Rasters with skew different than 0.

Code

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSkew(rast, 0.5, 0.5) As rast
FROM dummy_rast) As skewed;
```

Output

```
rastwidth |     pixwidth      | scalex | scaley | skewx | skewy
-----------+-------------------+--------+--------+-------+-----------
10 |  2.06155281280883 |      2 |      3 |   0.5 |       0.5
 5 | 0.502493781056044 |   0.05 |  -0.05 |   0.5 |       0.5
```

Voir aussi

ST_PixelHeight, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.4.9 ST_ScaleX

ST_ScaleX — Renvoie la composante X de la largeur du pixel dans l'unité du système de référence spatial.

Synopsis

float8 **ST_ScaleX**(raster rast);

Description

Renvoie la composante X de la largeur du pixel dans l'unité du système de référence spatial. Voir World File pour plus de détails.

Changement : 2.0.0. Dans les versions WKTRaster, cette fonction était appelée ST_PixelSizeX.

Exemples

Code

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;
```

Output

```
rid | rastpixwidth
-----+--------------
   1 |            2
   2 |         0.05
```

Voir aussi

ST_Width

11.4.10 ST_ScaleY

ST_ScaleY — Renvoie la composante Y de la hauteur du pixel dans l'unité du système de référence spatial.

Synopsis

float8 **ST_ScaleY**(raster rast);

Description

Renvoie la composante Y de la hauteur du pixel dans l'unité du système de référence spatial. Peut être négative. Voir World File pour plus de détails.

Changement : 2.0.0. Dans les versions WKTRaster, cette fonction était appelée ST_PixelSizeY.

Exemples

Code

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;
```

Output

```
rid | rastpixheight
-----+--------------
   1 |             3
   2 |         -0.05
```

Voir aussi

ST_Height

11.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — Retourne le coin supérieur gauche du raster, sous forme de coordonnées X et Y (longitude et latitude) d'un colonne et d'une ligne. Les numéros de colonne et de ligne commencent à 1.

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

Description

Retourne le coin supérieur gauche du raster, sous forme de coordonnées X et Y (longitude et latitude) d'un colonne et d'une ligne. Les valeurs X et Y retournées sont dans l'unité du système de référence spatial du raster géo-référencé. Les numéros de colonne et de ligne commencent à 1, mais si l'un des deux paramètres est 0, un nombre négatif ou supérieur à la dimension du raster, la fonction retourne des coordonnées en dehors du raster, en supposant que la grille du raster est applicable en dehors de son étendue.

Disponibilité : 2.1.0

Exemples

This example uses a non-skewed raster.

Code

```
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 2)).*
FROM dummy_rast
```

Output

```
rid | longitude  | latitude | longitude |  latitude
-----+------------+----------+-----------+------------
   1 |        0.5 |      0.5 |       2.5 |        3.5
   2 | 3427927.75 |  5793244 | 3427927.8 | 5793243.95
```

This example uses a skewed raster.

Code

```
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo
```

Output

```
rid | longitude  | latitude | longitude | latitude
-----+------------+----------+-----------+----------
   1 |        0.5 |      0.5 |     203.5 |       6.5
   2 | 3427927.75 |  5793244 | 3428128.8 | 5793243.9
```

Voir aussi

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SetSkew

11.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — Retourne la coordonnée X du coin supérieur gauche du raster à column et row. Les numéros de colonne et de ligne commencent à 1.

Synopsis

float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn);
float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

Description

Retourne la coordonnée X du coin supérieur gauche du raster à column et row dans l'unité du système de référence spatial du raster géo-référencé. Les numéros de colonne et de ligne commencent à 1, mais si l'un des deux paramètres est 0, un nombre négatif ou supérieur à la dimension du raster, la fonction retourne des coordonnées en dehors du raster, en supposant que la grille du raster est applicable en dehors de son étendue (même taille de pixel et skew).



Note
Pour des rasters sans obliquité (skew), fournir le numéro de colonne X est suffisant. Pour des rasters avec obliquité, les coordonnées géo-référencées sont fonction de ST_ScaleX et ST_SkewX et de la ligne et colonne. Une erreur sera émise si seulement la colonne X est fournie pour une raster avec obliquité.

Changement : 2.1.0 Jusqu'à la version 2.0.x, cette fonction était appelée ST_Raster2WorldCoordX

Exemples

This example shows that providing a column is sufficient for a non-skewed raster.

Code

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1) As x1coord,
    ST_RasterToWorldCoordX(rast, 2) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

Output

```
rid |  x1coord   |  x2coord  | pixelx
----+------------+-----------+--------
  1 |        0.5 |       2.5 |      2
  2 | 3427927.75 | 3427927.8 |   0.05
```

For fun let's skew it.

Code

```
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
    ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;
```

Output

```
rid |  x1coord   |  x2coord  | pixelx
----+------------+-----------+--------
  1 |        0.5 |     203.5 |      2
  2 | 3427927.75 | 3428128.8 |   0.05
```

Voir aussi

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

11.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Retourne la coordonnée Y du coin supérieur gauche du raster à column et row. Les numéros de colonne et de ligne commencent à 1.

Synopsis

float8 **ST_RasterToWorldCoordY**(raster rast, integer yrow);
float8 **ST_RasterToWorldCoordY**(raster rast, integer xcolumn, integer yrow);

Description

Retourne la coordonnée Y du coin supérieur gauche du raster à column et row dans l'unité du système de référence spatial du raster géo-référencé. Les numéros de colonne et de ligne commencent à 1, mais si l'un des deux paramètres est 0, un nombre négatif ou supérieur à la dimension du raster, la fonction retourne des coordonnées en dehors du raster, en supposant que la grille du raster est applicable en dehors de son étendue (même taille de pixel et skew).



Note
Pour des rasters sans obliquité (skew), fournir le numéro de ligne Y est suffisant. Pour des rasters avec obliquité, les coordonnées géo-référencées sont fonction de ST_ScaleY et ST_SkewY et de la ligne et colonne. Une erreur sera émise si seulement la ligne Y est fournie pour une raster avec obliquité.

Changement : 2.1.0 Jusqu'à la version 2.0.x, cette fonction était appelée ST_Raster2WorldCoordY

Exemples

Non-skewed raster providing row is sufficient.

Code

```
SELECT rid, ST_RasterToWorldCoordY(rast, 1) As y1coord,
    ST_RasterToWorldCoordY(rast, 3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

Output

```
rid | y1coord |  y2coord  | pixely
-----+---------+-----------+--------
   1 |     0.5 |       6.5 |      3
   2 | 5793244 | 5793243.9 |  -0.05
```

For fun let's skew it.

Code

```
SELECT rid, ST_RasterToWorldCoordY(rast, 1, 1) As y1coord,
    ST_RasterToWorldCoordY(rast, 2, 3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast, 0, 100.5) As rast FROM dummy_rast) As foo;
```

Output

```
rid | y1coord |  y2coord  | pixely
-----+---------+-----------+--------
   1 |     0.5 |       107 |      3
   2 | 5793244 | 5793344.4 |  -0.05
```

Voir aussi

ST_ScaleY, ST_RasterToWorldCoordX, ST_SetSkew, ST_SkewY

11.4.14 ST_Rotation

ST_Rotation — Retourne l'angle de rotation du raster en radians.

Synopsis

float8 **ST_Rotation**(raster rast);

Description

Retourne la rotation uniforme du raster en radians. Si le raster n'a pas de rotation uniforme, retourne NaN. Voir World File pour plus de détails.

Exemples

Code

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) AS rot FROM  ↩
    dummy_rast;
```

Output

```
rid |        rot
-----+-------------------
   1 | 0.785398163397448
   2 | 0.785398163397448
```

Voir aussi

ST_SetRotation, ST_SetScale, ST_SetSkew

11.4.15 ST_SkewX

ST_SkewX — Retourne l'obliquité géo-référencée X (paramètre de rotation).

Synopsis

float8 **ST_SkewX**(raster rast);

Description

Retourne l'obliquité géo-référencée X (paramètre de rotation). Voir World File pour plus de détails.

Exemples

Code

```
WITH refs AS (
    SELECT
        rid,
        format('(%s, %s)', ST_SkewX(rast), ST_SkewY(rast)) AS skew,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
)
SELECT
    rid,
    skew,
    term,
    georef[idx] AS value
FROM refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY rid, idx;
```

Output

```
rid |  skew  |  term   |       value
----+--------+---------+--------------------
   1 | (0, 0) | scaleX  | 2.0000000000
   1 | (0, 0) | skewY   | 0.0000000000
   1 | (0, 0) | skewX   | 0.0000000000
   1 | (0, 0) | scaleY  | 3.0000000000
   1 | (0, 0) | originX | 0.5000000000
   1 | (0, 0) | originY | 0.5000000000
   2 | (0, 0) | scaleX  | 0.0500000000
   2 | (0, 0) | skewY   | 0.0000000000
   2 | (0, 0) | skewX   | 0.0000000000
   2 | (0, 0) | scaleY  | -0.0500000000
   2 | (0, 0) | originX | 3427927.7500000000
   2 | (0, 0) | originY | 5793244.0000000000
(12 rows)
```

Voir aussi

ST_GeoReference, ST_SkewY, ST_SetSkew

11.4.16 ST_SkewY

ST_SkewY — Retourne l'obliquité géo-référencée Y (paramètre de rotation).

Synopsis

float8 **ST_SkewY**(raster rast);

Description

Retourne l'obliquité géo-référencée Y (paramètre de rotation). Voir World File pour plus de détails.

Exemples

Code

```
WITH refs AS (
    SELECT
        rid,
        format('(%s, %s)', ST_SkewX(rast), ST_SkewY(rast)) AS skew,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
)
SELECT
    rid,
    skew,
    term,
    georef[idx] AS value
FROM refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY rid, idx;
```

Output

```
rid |  skew  |  term   |       value
-----+--------+---------+--------------------
   1 | (0, 0) | scaleX  | 2.0000000000
   1 | (0, 0) | skewY   | 0.0000000000
   1 | (0, 0) | skewX   | 0.0000000000
   1 | (0, 0) | scaleY  | 3.0000000000
   1 | (0, 0) | originX | 0.5000000000
   1 | (0, 0) | originY | 0.5000000000
   2 | (0, 0) | scaleX  | 0.0500000000
   2 | (0, 0) | skewY   | 0.0000000000
   2 | (0, 0) | skewX   | 0.0000000000
   2 | (0, 0) | scaleY  | -0.0500000000
   2 | (0, 0) | originX | 3427927.7500000000
   2 | (0, 0) | originY | 5793244.0000000000
(12 rows)
```

Voir aussi

ST_GeoReference, ST_SkewX, ST_SetSkew

11.4.17 ST_SRID

ST_SRID — Retourne l'identifiant du système de référence spatial du raster, tel que défini dans la table spatial_ref_sys.

Synopsis

integer **ST_SRID**(raster rast);

Description

Retourne l'identifiant du système de référence spatial du raster, tel que défini dans la table spatial_ref_sys.



Note
Non-georeferenced rasters and geometries use SRID 0.

Exemples

Code

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;
```

Output

```
0
```

Voir aussi

Section 4.5, ST_SRID

11.4.18 ST_Summary

ST_Summary — Retourne un résumé du contenu du raster sous forme de texte.

Synopsis

text **ST_Summary**(raster rast);

Description

Retourne un résumé du contenu du raster sous forme de texte.

Disponibilité : 2.1.0

Exemples

Code

```
SELECT ST_Summary(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1,  ↩
    0, 0, 0),
                1, '8BUI', 1, 0
            ),
            2, '32BF', 0, -9999
        ),
        3, '16BSI', 0, NULL
    )
);
```

Output

```
Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
     band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
     band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
     band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

Voir aussi

ST_MetaData, ST_BandMetaData, ST_Summary ST_Extent

11.4.19 ST_UpperLeftX

ST_UpperLeftX — Retourne la coordonnée X du coin supérieur gauche du raster projeté dans le système de référence spatial.

Synopsis

float8 **ST_UpperLeftX**(raster rast);

Description

Retourne la coordonnée X du coin supérieur gauche du raster projeté dans le système de référence spatial.

Exemples

Code

```
SELECt rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;
```

Output

```
rid |    ulx
-----+-----------
   1 |        0.5
   2 | 3427927.75
```

Voir aussi

ST_UpperLeftY, ST_GeoReference, Box3D

11.4.20 ST_UpperLeftY

ST_UpperLeftY — Retourne la coordonnée Y du coin supérieur gauche du raster projeté dans le système de référence spatial.

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Description

Retourne la coordonnée Y du coin supérieur gauche du raster projeté dans le système de référence spatial.

Exemples

Code

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;
```

Output

```
rid |   uly
-----+---------
   1 |     0.5
   2 | 5793244
```

Voir aussi

ST_UpperLeftX, ST_GeoReference, Box3D

11.4.21 ST_Width

ST_Width — Retourne la largeur du raster en pixels.

Synopsis

integer **ST_Width**(raster rast);

Description

Retourne la largeur du raster en pixels.

Exemples

Code

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;
```

Output

```
10
```

Voir aussi

ST_Height

11.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — Retourne le coin supérieur gauche comme colonne et ligne, en fonction de coordonnées géométriques X et Y (longitude et latitude) ou un point géométrique dans le système de référence spatial du raster.

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Description

Retourne le coin supérieur gauche comme colonne et ligne, en fonction de coordonnées géométriques X et Y (longitude et latitude) ou un point géométrique. Cette fonction fonctionne aussi bien si X et Y ou le point sont dans l'étendue du raster ou non. Les coordonnées géométriques X et Y ou les coordonnées du point doivent être dans le système de référence spatial du raster.

Disponibilité : 2.1.0

Exemples

Code

```
SELECT
    rid,
    (ST_WorldToRasterCoord(rast, 3427927.8, 20.5)).*,
    (ST_WorldToRasterCoord(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)))).*
FROM dummy_rast;
```

Output

```
rid | columnx |   rowy    | columnx |   rowy
-----+---------+-----------+---------+-----------
   1 | 1713964 |         7 | 1713964 |         7
   2 |       2 | 115864471 |       2 | 115864471
```

Voir aussi

ST_WorldToRasterCoordX, ST_WorldToRasterCoordY, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — Retourne la colonne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw) exprimés dans le système de référence spatial du raster.

Synopsis

integer **ST_WorldToRasterCoordX**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw, double precision yw);

Description

Retourne la colonne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw). Un point, ou le couple (xw, yw) sont requis pour un raster avec obliquité (skew). Sinon, xw est suffisant. Les coordonnées sont exprimées dans le système de référence spatial du raster.

Changement : 2.1.0 Dans les versions précédentes, cette fonctionne était appelée ST_World2RasterCoordX

Exemples

Code

```
SELECT rid, ST_WorldToRasterCoordX(rast, 3427927.8) As xcoord,
        ST_WorldToRasterCoordX(rast, 3427927.8, 20.5) As xcoord_xwyw,
        ST_WorldToRasterCoordX(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)) ↩
    ) As ptxcoord
FROM dummy_rast;
```

Output

```
rid | xcoord  |  xcoord_xwyw   | ptxcoord
-----+---------+---------+----------
   1 | 1713964 | 1713964 |  1713964
   2 |       1 |       1 |        1
```

Voir aussi

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — Retourne la ligne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw) exprimés dans le système de référence spatial du raster.

Synopsis

integer **ST_WorldToRasterCoordY**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw, double precision yw);

Description

Retourne la ligne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw). Un point, ou le couple (xw, yw) sont requis pour un raster avec obliquité (skew). Sinon, xw est suffisant. Les coordonnées sont exprimées dans le système de référence spatial du raster.

Changement : 2.1.0 Dans les versions précédentes, cette fonctionne était appelée ST_World2RasterCoordY

Exemples

Code

```
SELECT rid, ST_WorldToRasterCoordY(rast, 20.5) As ycoord,
        ST_WorldToRasterCoordY(rast, 3427927.8, 20.5) As ycoord_xwyw,
        ST_WorldToRasterCoordY(rast, ST_GeomFromText('POINT(3427927.8 20.5)', ST_SRID(rast)) ↩
    ) As ptycoord
FROM dummy_rast;
```

Output

```
rid |  ycoord   | ycoord_xwyw | ptycoord
-----+-----------+-------------+-----------
   1 |         7 |           7 |         7
   2 | 115864471 |   115864471 | 115864471
```

Voir aussi

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

11.5 Fonctions d'accès aux bandes raster

11.5.1 ST_BandMetaData

ST_BandMetaData — Retourne les méta-données de base d'une bande raster spécifique. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

(1) record **ST_BandMetaData**(raster rast, integer band=1);
(2) record **ST_BandMetaData**(raster rast, integer[] band);

Description

Retourne les méta-données de base d'une bande raster spécifique. Les colonnes retournées sont : pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.



Note
Si le raster ne contient aucune bande, une erreur est émise.



Note
Si la bande n'a pas de valeur NODATA, nodatavalue est NULL.



Note
Si isoutdb est False, alors path, outdbbandnum, filesize et filetimestamp sont NULL. Si l'accès outdb est désactivé, filesize et filetimestamp sont également NULL.

Amélioration : 2.5.0 ajout de *outdbbandnum*, *filesize* et *filetimestamp* pour les rasters outdb.

Exemples

Variant 1.

Code

```
SELECT
    rid,
    (foo.md).*
FROM (
    SELECT
        rid,
        ST_BandMetaData(rast, 1) AS md
    FROM dummy_rast
    WHERE rid=2
) As foo;
```

Output

```
rid | pixeltype | nodatavalue | isoutdb | path | outdbbandnum
-----+-----------+-------------+---------+------+--------------
   2 | 8BUI      |           0 | f       |      |
(1 row)
```

Variant 2.

Code

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
    loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    bandnum,
    pixeltype,
    isoutdb,
    regexp_replace(path, '^.*/', '') AS file,
    outdbbandnum
FROM ST_BandMetaData((SELECT rast FROM foo),
    ARRAY[1, 3, 2]::int[]
);
```

Output

```
bandnum | pixeltype | isoutdb |     file      | outdbbandnum
---------+-----------+---------+---------------+--------------
       1 | 8BUI      | t       | Projected.tif |            1
       3 | 8BUI      | t       | Projected.tif |            3
       2 | 8BUI      | t       | Projected.tif |            2
```

Voir aussi

ST_MetaData, ST_BandPixelType

11.5.2 ST_BandNoDataValue

ST_BandNoDataValue — Retourne la valeur qui représente l'absence de valeur (nodata) pour cette bande. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

Description

Retourne la valeur qui représente l'absence de valeur (nodata) pour cette bande

Exemples

Code

```
SELECT ST_BandNoDataValue(rast, 1) As bnval1,
    ST_BandNoDataValue(rast, 2) As bnval2, ST_BandNoDataValue(rast, 3) As bnval3
FROM dummy_rast
WHERE rid = 2;
```

Output

```
bnval1 | bnval2 | bnval3
--------+--------+--------
      0 |      0 |      0
```

Voir aussi

ST_NumBands

11.5.3 ST_BandIsNoData

ST_BandIsNoData — Retourne true si la bande ne contient que des valeurs nodata.

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=false);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=false);

Description

Retourne true si la bande ne contient que des valeurs nodata. Si aucune bande n'est spécifiée, la bande 1 est utilisée. Si le dernier paramètre est TRUE, la bande entière est vérifiée pixel par pixel. Sinon, la fonction retourne simplement la valeur du flag isnodata pour cette bande. La valeur par défaut pour ce paramètre est FALSE.

Disponibilité : 2.0.0



Note
If the flag is dirty (this is, the result is different using TRUE as last parameter and not using it) you should update the raster to set this flag to true, by using ST_SetBandIsNoData(), or ST_SetBandNoDataValue() with TRUE as last argument. See ST_SetBandIsNoData.

Exemples

This example creates a dummy table with one raster column. The raster has two bands and one pixel per band. In the first band, the NODATA value and pixel value are both 3. In the second band, the NODATA value is 13 and the pixel value is 4.

Code

```
create table dummy_rast (rid integer, rast raster);

insert into dummy_rast values(
1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
```

```
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected true
select ST_BandIsNoData(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

Voir aussi

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_SetBandIsNoData

11.5.4 ST_BandPath

ST_BandPath — Retourne le chemin système du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Description

Retourne le chemin système du fichier d'une bande. Renvoie une erreur si la bande n'est pas outofdb.

Exemples

Code

```
WITH foo AS (
    SELECT ST_AddBand(
        NULL::raster,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif',
        NULL::int[]
    ) AS rast
)
SELECT bandnum, ST_BandPath(rast, bandnum) AS path
FROM foo
CROSS JOIN generate_series(1, 3) AS bandnum;
```

Output

```
bandnum |                                    path
---------+-------------------------------------------------------------------------
       1 | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
       2 | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
       3 | /home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
```

Voir aussi

ST_BandMetaData, ST_SetBandPath, ST_SetBandIndex

11.5.5 ST_BandFileSize

ST_BandFileSize — Retourne la taille du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

Description

Retourne la taille du fichier d'une bande stockée sur le système de fichier. Renvoie une erreur si la bande n'est pas outofdb, ou si l'accès outdb est désactivé.

Cette fonction est généralement utilisée conjointement à ST_BandPath() et ST_BandFileTimestamp(), pour qu'un client puisse déterminer le nom du fichier d'un raster outdb, tel que vu par le serveur.

Disponibilité : 2.5.0

Exemples

Code

```
SELECT ST_BandFileSize(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
240574
```

11.5.6 ST_BandFileTimestamp

ST_BandFileTimestamp — Retourne le timestamp du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

Description

Retourne le timestamp (nombre de secondes depuis Jan 1st 1970 00:00:00 UTC) du fichier d'une bande stockée sur le système de fichier. Renvoie une erreur si la bande n'est pas outofdb, ou si l'accès outdb est désactivé.

Cette fonction est généralement utilisée conjointement à ST_BandPath() et ST_BandFileSize(), pour qu'un client puisse déterminer le nom du fichier d'un raster outdb, tel que vu par le serveur.

Disponibilité : 2.5.0

Exemples

Code

```
SELECT ST_BandFileTimestamp(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
1521807257
```

11.5.7 ST_BandPixelType

ST_BandPixelType — Retourne le type de pixel d'une bande. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

Description

Retourne le nom décrivant le type de données et la taille des valeurs stockées pour chaque pixel de la bande spécifiée.

Il existe 11 types de pixels. Les types de pixels supportés sont :

— 1BB - booléen 1-bit
— 2BUI - entier non signé 2-bits
— 4BUI - entier non signé 4-bits
— 8BSI - entier signé 8-bits
— 8BUI - entier non signé 8-bits
— 16BSI - entier signé 16-bits
— 16BUI - entier non signé 16-bits
— 32BSI - entier signé 32-bits
— 32BUI - entier non signé 32-bits
— 32BF - flottant 32-bits
— 64BF - flottant 64-bits

Exemples

Code

```
SELECT ST_BandPixelType(rast, 1) As btype1,
    ST_BandPixelType(rast, 2) As btype2, ST_BandPixelType(rast, 3) As btype3
FROM dummy_rast
WHERE rid = 2;
```

Output

```
btype1 | btype2 | btype3
--------+--------+--------
 8BUI   | 8BUI   | 8BUI
```

Voir aussi

ST_NumBands

11.5.8 ST_MinPossibleValue

ST_MinPossibleValue — Retourne la valeur minimale supportée par le type de pixel pixeltype.

Synopsis

integer **ST_MinPossibleValue**(text pixeltype);

Description

Retourne la valeur minimale supportée par le type de pixel pixeltype.

Exemples

Code

```
SELECT ST_MinPossibleValue('16BSI');
```

Output

```
-32768
```

Code

```
SELECT ST_MinPossibleValue('8BUI');
```

Output

```
0
```

Voir aussi

ST_BandPixelType

11.5.9 ST_HasNoBand

ST_HasNoBand — Retourne true si le raster n'a pas la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

Description

Retourne true si le raster n'a pas la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast, 2) as hb2,
ST_HasNoBand(rast, 4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;
```

Output

```
rid | hb1 | hb2 | hb4 | numbands
-----+-----+-----+-----+----------
1 | t   | t   | t   |        0
2 | f   | f   | t   |        3
```

Voir aussi

ST_NumBands

11.6 Fonctions d'accès et de modifications des pixels raster

11.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — Retourne la géométrie polygonale qui délimite le pixel pour une ligne et colonne spécifiées.

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Description

Retourne la géométrie polygonale qui délimite le pixel pour une ligne et colonne spécifiées.

Disponibilité : 2.0.0

Exemples

This example returns a raster pixel polygon.

Code

```
SELECT i, j, ST_PixelAsPolygon(foo.rast, i, j) As b1pgeom
FROM dummy_rast As foo
    CROSS JOIN generate_series(1, 2) As i
    CROSS JOIN generate_series(1, 1) As j
WHERE rid=2;
```

Output

```
i | j |                                                    b1pgeom
---+---+-------------------------------------------------------------------------------
 1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
 2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..
```

Voir aussi

ST_DumpAsPolygons, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Intersection, ST_AsText

11.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — Retourne la géométrie polygonale qui délimite chaque pixel de la bande raster, avec la valeur et les coordonnées raster X et Y de chaque pixel.

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Retourne la géométrie polygonale qui délimite chaque pixel de la bande raster, avec la valeur (double précision) et les coordonnées raster X et Y (entiers) de chaque pixel.

Format de l'enregistrement de retour : *geom* geometry, *val* double precision, *x* entier, *y* entier.



Note
When *exclude_nodata_value* is TRUE, only pixels whose values are not NODATA are returned. Unlike ST_DumpAsPolygons, this function returns one polygon per pixel rather than merging adjacent pixels with the same value.

Disponibilité : 2.0.0

Amélioration : 2.1.0 ajout du paramètre optionnel exclude_nodata_value.

Changement : 2.1.1 Changement du comportement de behavior of exclude_nodata_value.

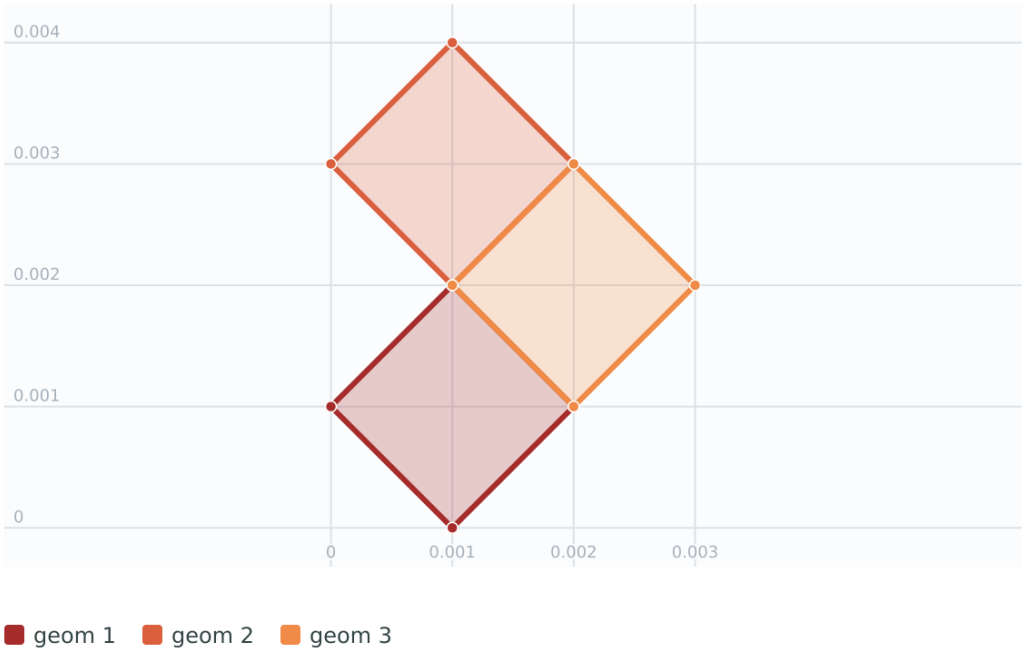
Exemples

This example returns a raster pixel polygon.

Code

```
WITH raster AS (
    SELECT ST_SetValue(
        ST_SetValue(
            ST_AddBand(
                ST_MakeEmptyRaster(
                    2, 2, 0, 0, 0.001, -0.001, 0.001, 0.001, 4269
                ),
                1, '8BUI', 1, 0
            ),
            2, 2, 10
        ),
        1, 1, NULL
    ) AS rast
), pixels AS (
    SELECT (ST_PixelAsPolygons(rast)).*
    FROM raster
)
SELECT x, y, val, geom AS geom
FROM pixels
ORDER BY x, y;
```

Figure



Voir aussi

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_AsText

11.6.3 ST_PixelAsPoint

ST_PixelAsPoint — Retourne le point géométrique du coin supérieur gauche du pixel.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Description

Retourne le point géométrique du coin supérieur gauche du pixel.

Disponibilité : 2.1.0

Exemples

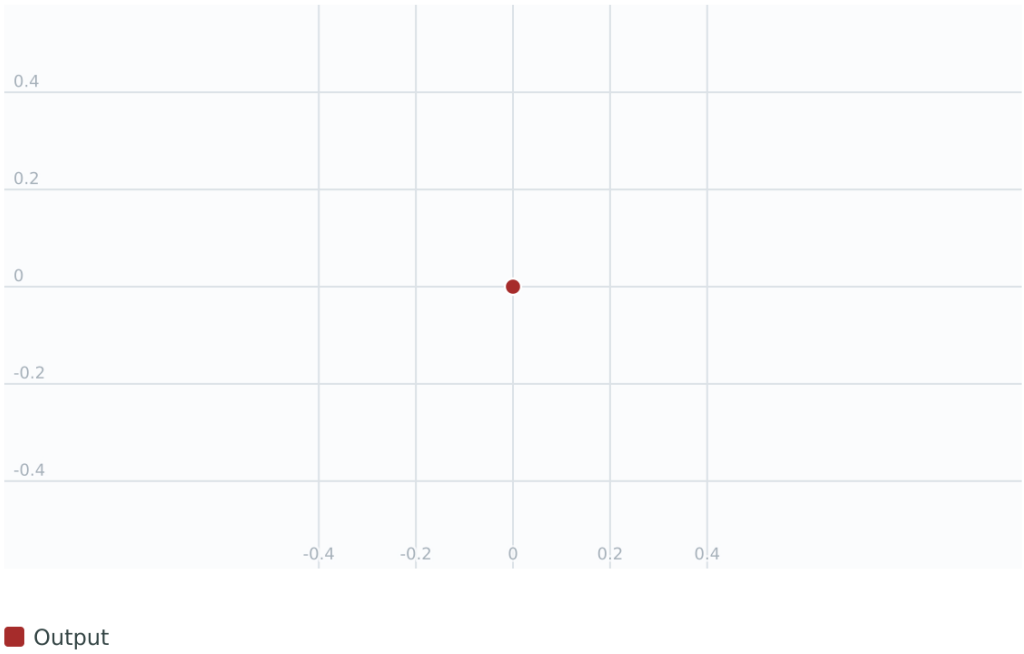
Code

```
SELECT ST_PixelAsPoint(rast, 1, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
POINT(0.5 0.5)
```

Figure



Voir aussi

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids

11.6.4 ST_PixelAsPoints

ST_PixelAsPoints — Retourne un point géométrique pour chaque pixel de la bande raster, avec sa valeur et ses coordonnées raster X et Y. Les coordonnées du points sont ceux du coin supérieur gauche du pixel.

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Retourne un point géométrique pour chaque pixel de la bande raster, avec sa valeur et ses coordonnées raster X et Y. Les coordonnées du points sont ceux du coin supérieur gauche du pixel.

Format de l'enregistrement de retour : *geom* geometry, *val* double precision, *x* entier, *y* entier.



Note

Si *exclude_nodata_value* = TRUE, seuls les pixels qui ne sont pas NODATA sont retournés.

Disponibilité : 2.1.0

Changement : 2.1.1 Changement du comportement de behavior of exclude_nodata_value.

Exemples

This example returns point features together with a derived marker size that a client can use for value-scaled point rendering.

Code

```
SELECT
    x,
    y,
    val,
    round((val / 32.0)::numeric, 2) AS marker_scale,geom AS geom
FROM (
    SELECT (ST_PixelAsPoints(rast, 1)).*
    FROM dummy_rast
    WHERE rid = 2
) AS foo
ORDER BY y, x;
```

Output

```
x | y | val | marker_scale |          geom
---+---+-----+--------------+------------------------------
 1 | 1 | 253 |         7.91 | POINT(3427927.75 5793244)
 2 | 1 | 254 |         7.94 | POINT(3427927.8 5793244)
 3 | 1 | 253 |         7.91 | POINT(3427927.85 5793244)
 4 | 1 | 254 |         7.94 | POINT(3427927.9 5793244)
 5 | 1 | 254 |         7.94 | POINT(3427927.95 5793244)
 1 | 2 | 253 |         7.91 | POINT(3427927.75 5793243.95)
 2 | 2 | 254 |         7.94 | POINT(3427927.8 5793243.95)
 3 | 2 | 254 |         7.94 | POINT(3427927.85 5793243.95)
 4 | 2 | 253 |         7.91 | POINT(3427927.9 5793243.95)
 5 | 2 | 249 |         7.78 | POINT(3427927.95 5793243.95)
 1 | 3 | 250 |         7.81 | POINT(3427927.75 5793243.9)
 2 | 3 | 254 |         7.94 | POINT(3427927.8 5793243.9)
 3 | 3 | 254 |         7.94 | POINT(3427927.85 5793243.9)
 4 | 3 | 252 |         7.88 | POINT(3427927.9 5793243.9)
 5 | 3 | 249 |         7.78 | POINT(3427927.95 5793243.9)
 1 | 4 | 251 |         7.84 | POINT(3427927.75 5793243.85)
 2 | 4 | 253 |         7.91 | POINT(3427927.8 5793243.85)
 3 | 4 | 254 |         7.94 | POINT(3427927.85 5793243.85)
 4 | 4 | 254 |         7.94 | POINT(3427927.9 5793243.85)
 5 | 4 | 253 |         7.91 | POINT(3427927.95 5793243.85)
 1 | 5 | 252 |         7.88 | POINT(3427927.75 5793243.8)
 2 | 5 | 250 |         7.81 | POINT(3427927.8 5793243.8)
 3 | 5 | 254 |         7.94 | POINT(3427927.85 5793243.8)
 4 | 5 | 254 |         7.94 | POINT(3427927.9 5793243.8)
 5 | 5 | 254 |         7.94 | POINT(3427927.95 5793243.8)
```

Figure



Voir aussi

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsCentroid, ST_PixelAsCentroids

11.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — Retourne le centroïde (point géométrique) de la zone représentée par un pixel.

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

Description

Retourne le centroïde (point géométrique) de la zone représentée par un pixel.

Amélioration : 3.2.0 Plus rapide, désormais implémenté en C.

Disponibilité : 2.1.0

Exemples

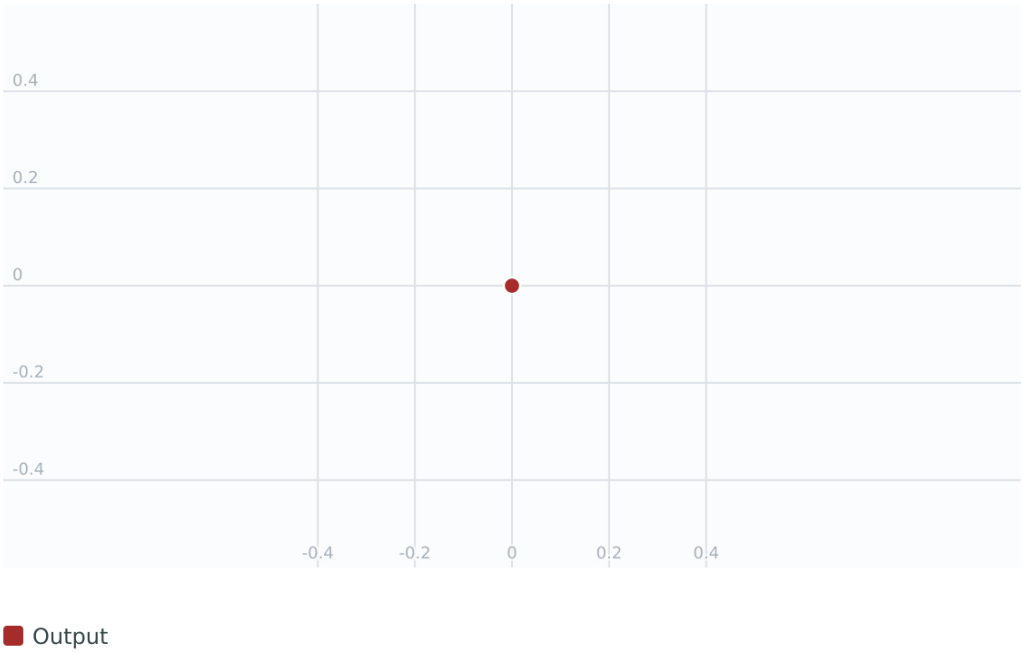
Code

```
SELECT ST_PixelAsCentroid(rast, 1, 1) FROM dummy_rast WHERE rid = 1;
```

Output

```
POINT(1.5 2)
```

Figure



Voir aussi

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroids

11.6.6 ST_PixelAsCentroids

ST_PixelAsCentroids — Retourne le centroïde (point géométrique) pour chaque pixel de la bande raster, avec sa valeur et les coordonnées raster X et Y. Le point géométrique est le centroïde de la zone représentée par un pixel.

Synopsis

setof record **ST_PixelAsCentroids**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Description

Retourne le centroïde (point géométrique) pour chaque pixel de la bande raster, avec sa valeur et les coordonnées raster X et Y. Le point géométrique est le centroïde de la zone représentée par un pixel.

Format de l'enregistrement de retour : *geom* geometry, *val* double precision, *x* entier, *y* entier.



Note

Si *exclude_nodata_value* = TRUE, seuls les pixels qui ne sont pas NODATA sont retournés.

Amélioration : 3.2.0 Plus rapide, désormais implémenté en C.

Changement : 2.1.1 Changement du comportement de behavior of exclude_nodata_value.

Disponibilité : 2.1.0

Exemples

This example uses LATERAL syntax.

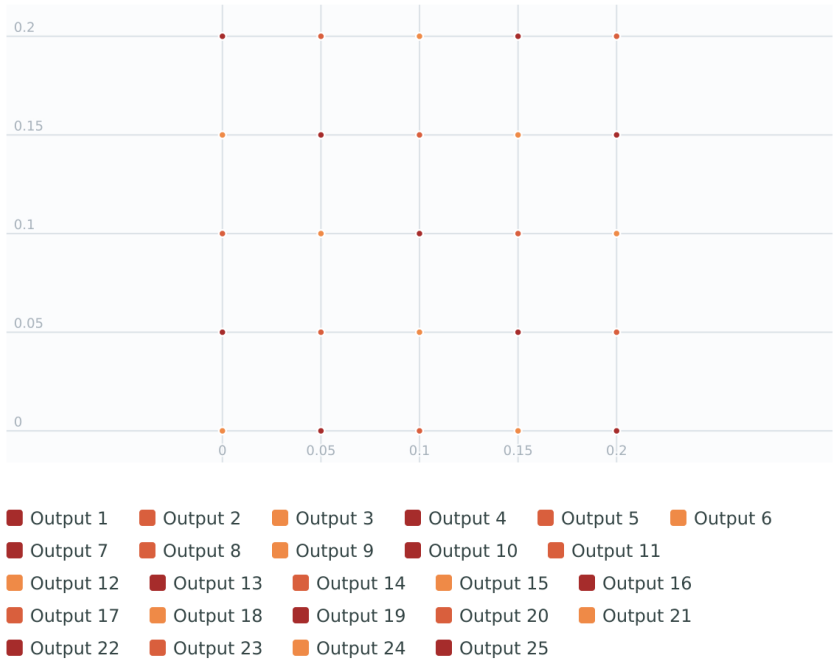
Code

```
SELECT x, y, val,geom
    FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid
    = 2) foo;
```

Output

```
x | y | val |           st_astext
---+---+-----+-------------------------------
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
 1 | 2 | 253 | POINT(3427927.775 5793243.925)
 2 | 2 | 254 | POINT(3427927.825 5793243.925)
 3 | 2 | 254 | POINT(3427927.875 5793243.925)
 4 | 2 | 253 | POINT(3427927.925 5793243.925)
 5 | 2 | 249 | POINT(3427927.975 5793243.925)
 1 | 3 | 250 | POINT(3427927.775 5793243.875)
 2 | 3 | 254 | POINT(3427927.825 5793243.875)
 3 | 3 | 254 | POINT(3427927.875 5793243.875)
 4 | 3 | 252 | POINT(3427927.925 5793243.875)
 5 | 3 | 249 | POINT(3427927.975 5793243.875)
 1 | 4 | 251 | POINT(3427927.775 5793243.825)
 2 | 4 | 253 | POINT(3427927.825 5793243.825)
 3 | 4 | 254 | POINT(3427927.875 5793243.825)
 4 | 4 | 254 | POINT(3427927.925 5793243.825)
 5 | 4 | 253 | POINT(3427927.975 5793243.825)
 1 | 5 | 252 | POINT(3427927.775 5793243.775)
 2 | 5 | 250 | POINT(3427927.825 5793243.775)
 3 | 5 | 254 | POINT(3427927.875 5793243.775)
 4 | 5 | 254 | POINT(3427927.925 5793243.775)
 5 | 5 | 254 | POINT(3427927.975 5793243.775)
```

Figure



Voir aussi

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid

11.6.7 ST_Value

ST_Value — Retourne la valeur d'une bande raster spécifiée au pixel donné par columnx, rowy, ou à un point géométrique spécifié. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié. Si `exclude_nodata_value` vaut false, tous les pixels y compris ceux ayant la valeur `nodata` sont considérés comme intersectés et leur valeur sera retournée. Si `exclude_nodata_value` n'est pas spécifié, la valeur est lue depuis les méta-données du raster.

Synopsis

double precision **ST_Value**(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_Value**(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true, text resample='nearest');
double precision **ST_Value**(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision **ST_Value**(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);

Description

Retourne la valeur d'une bande raster spécifiée au pixel donné par columnx, rowy, ou à un point géométrique spécifié. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié.

Si `exclude_nodata_value` vaut true, seulement les pixels différents de la valeur `nodata` seront utilisés. Si `exclude_nodata_v` vaut false, tous les pixels seront utilisés.

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, "bilinear" which performs a bilinear interpolation to estimate the value between pixel centers, and the nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". Boundary options choose which pixel is returned when the point lies on a horizontal or vertical pixel boundary, including corner intersections.

Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR".

Amélioration : 3.2.0 ajout du paramètre optionnel resample.

Amélioration : 2.0.0 ajout du paramètre optionnel exclude_nodata_value.

Exemples

Get raster values at particular postgis geometry points. The srid of your geometry should be same as for your raster.

Code

```
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As ↩
    pt_geom) As foo
WHERE rid=2;
```

Output

```
rid | b1pval | b2pval
-----+--------+--------
   2 |    252 |     79
```

General fictitious example using a real table.

Code

```
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast, sometable.geom);
```

Code

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;
```

Output

```
rid | b1pval | b2pval | b3pval
-----+--------+--------+--------
   2 |    253 |     78 |     70
```

Get all values in bands 1,2,3 of each pixel.

Code

```
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

Output

```
 x | y | b1val | b2val | b3val
---+---+-------+-------+-------
 1 | 1 |   253 |    78 |    70
 1 | 2 |   253 |    96 |    80
 1 | 3 |   250 |    99 |    90
 1 | 4 |   251 |    89 |    77
 1 | 5 |   252 |    79 |    62
 2 | 1 |   254 |    98 |    86
 2 | 2 |   254 |   118 |   108
 :
 :
```

Get all values in bands 1,2,3 of each pixel same as above but returning the upper left point point of each pixel.

Code

```
SELECT ST_SetSRID(ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
        ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
        ST_SRID(rast)) As uplpt,
    ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);
```

Output

```
            uplpt            | b1val | b2val | b3val
-----------------------------+-------+-------+-------
 POINT(3427929.25 5793245.5) |   253 |    78 |    70
 POINT(3427929.25 5793247)   |   253 |    96 |    80
 POINT(3427929.25 5793248.5) |   250 |    99 |    90
:
```

Get a polygon formed by union of all pixels that fall in a particular value range and intersect a particular polygon.

Code

```
SELECT ST_Union(pixpolyg) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast),
            ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
    5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;
```

Output

```
        shadow
-------------------------------------------------------------------------------
 MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95 5793243.9,
 3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05 5793243.9,3427928 5793243.9)),((3427927.95  ↩
    5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9 5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85 5793243.7,3427927.8 5793243.7,3427927.8 ↩
     5793243.75,
3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95 5793243.7,3427927.95 5793243.75,3427927.95  ↩
    5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9 5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))
```

Checking all the pixels of a large raster tile can take a long time. You can dramatically improve speed at some lose of precision by orders of magnitude. By sampling pixels using the step optional parameter of generate_series. This next example does the same as previous but by checking 1 for every 4 (2x2) pixels and putting in the last checked. Putting in the checked pixel as the value for subsequent 4.

Code

```
SELECT ST_Union(pixpolyg) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
            ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
WHERE rid =  2
    AND x <= ST_Width(rast)  AND y <= ST_Height(rast)  ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
    5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;
```

Output

```
        shadow
-----------------------------------------------------------------------------------
 MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
 3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928 ↩
    5793243.85,3427927.9 5793243.85)),
 ((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8 ↩
    5793243.85,3427927.9 5793243.85,
 3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928 ↩
    5793243.65,3427927.9 5793243.65)))
```

Voir aussi

ST_DumpValues, ST_SetValue, ST_DumpAsPolygons, ST_NumBands, ST_PixelAsPolygon, ST_ScaleX, ST_ScaleY, ST_UpperLeftX ST_UpperLeftY, ST_SRID, ST_AsText, ST_Point, ST_MakeEnvelope, ST_Intersects, ST_Intersection

11.6.8 ST_NearestValue

ST_NearestValue — Retourne la valeur la plus proche différent de NODATA pour une bande raster spécifiée au pixel donné par columnx et rowy, ou à un point géométrique spécifié dans le même système de référence spatial que le raster.

Synopsis

double precision **ST_NearestValue**(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);

Description

Retourne la valeur la plus proche différent de NODATA pour une bande raster spécifiée au pixel donné par columnx et rowy, ou à un point géométrique spécifié. Si le pixel donné par columnx, rowy ou par le point géométrique est de valeur NODATA, la fonction trouve le plus proche pixel dont la valeur est différente de NODATA.

Le numéro de bande démarre à 1, et la bande 1 est utilisée si bandnum non spécifié. Si exclude_nodata_value vaut false, tous les pixels y compris ceux ayant la valeur nodata sont considérés comme intersectés et leur valeur sera retournée. Si exclude_nodata_value n'est pas spécifié, la valeur est lue depuis les méta-données du raster.

Disponibilité : 2.1.0



Note
ST_NearestValue remplace directement ST_Value.

Exemples

This example reads a value from pixel 2x2.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 0],
            [1, 1, 0, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_Value(rast, 2, 2) AS value,
    ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM source
```

Output

```
value | nearestvalue
-------+--------------
     1 |            1
```

This example reads a NODATA value from pixel 2x3.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 0],
            [1, 1, 0, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_Value(rast, 2, 3) AS value,
    ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM source
```

Output

```
value | nearestvalue
-------+--------------
       |            1
```

Voir aussi

ST_Neighborhood, ST_Value

11.6.9 ST_SetZ

ST_SetZ — Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée Z copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.

Synopsis

geometry **ST_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée Z copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.

Le paramètre `resample` peut être "nearest" (plus proche) pour copier les valeurs de chaque pixel auquel le vertex correspond, ou "bilinear" pour utiliser une interpolation bilinéaire pour calculer une valeur qui prend aussi en compte les pixels voisins.

Disponibilité : 3.2.0

Exemples

2x2 test raster with values. 10 50. 40 20.

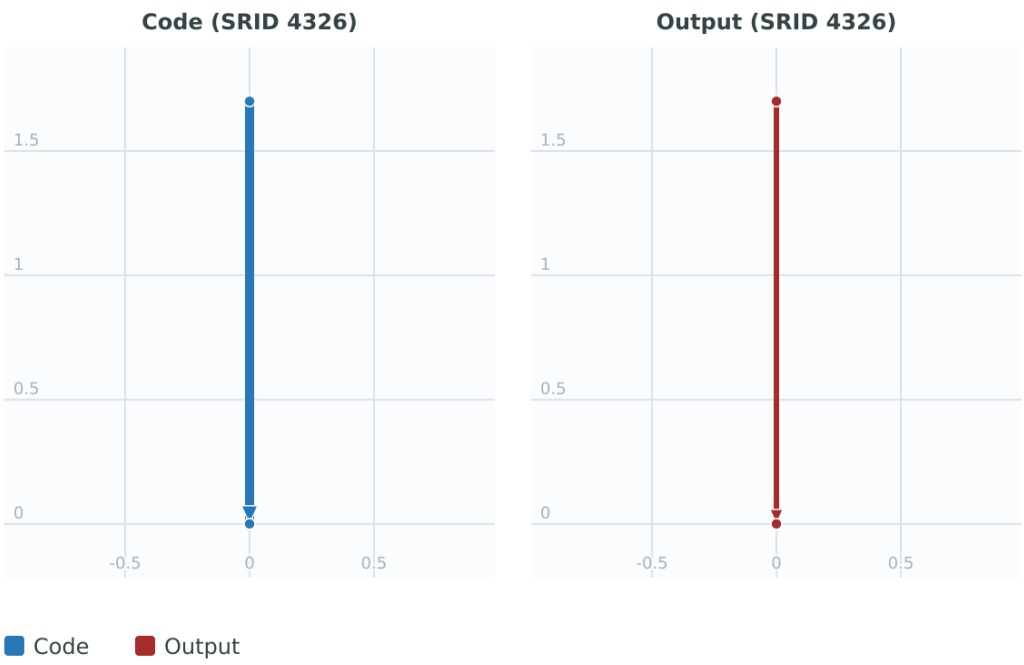
Code

```
WITH test_raster AS (
SELECT
ST_SetValues(
ST_AddBand(
ST_MakeEmptyRaster(
      width => 2,
      height => 2,
      upperleftx => 0,
      upperlefty => 2,
      scalex => 1.0,
      scaley => -1.0,
      skewx => 0,
      skewy => 0,
      srid => 4326),
    index => 1,
    pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999
),
  1,
  1,
  1,
  newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ↩
    ]]) AS rast
)
SELECT
ST_SetZ(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9,1.0 0.2)'::geometry,
    resample => 'bilinear'
)
FROM test_raster
```

Output

```
LINESTRING Z (1 1.9 38,1 0.2 27)
```

Figure



Voir aussi

ST_Value, ST_SetM

11.6.10 ST_SetM

ST_SetM — Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée M copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.

Synopsis

geometry **ST_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

Description

Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée M copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.

Le paramètre `resample` peut être "nearest" (plus proche) pour copier les valeurs de chaque pixel auquel le vertex correspond, ou "bilinear" pour utiliser une interpolation bilinéaire pour calculer une valeur qui prend aussi en compte les pixels voisins.

Disponibilité : 3.2.0

Exemples

2x2 test raster with values. 10 50. 40 20.

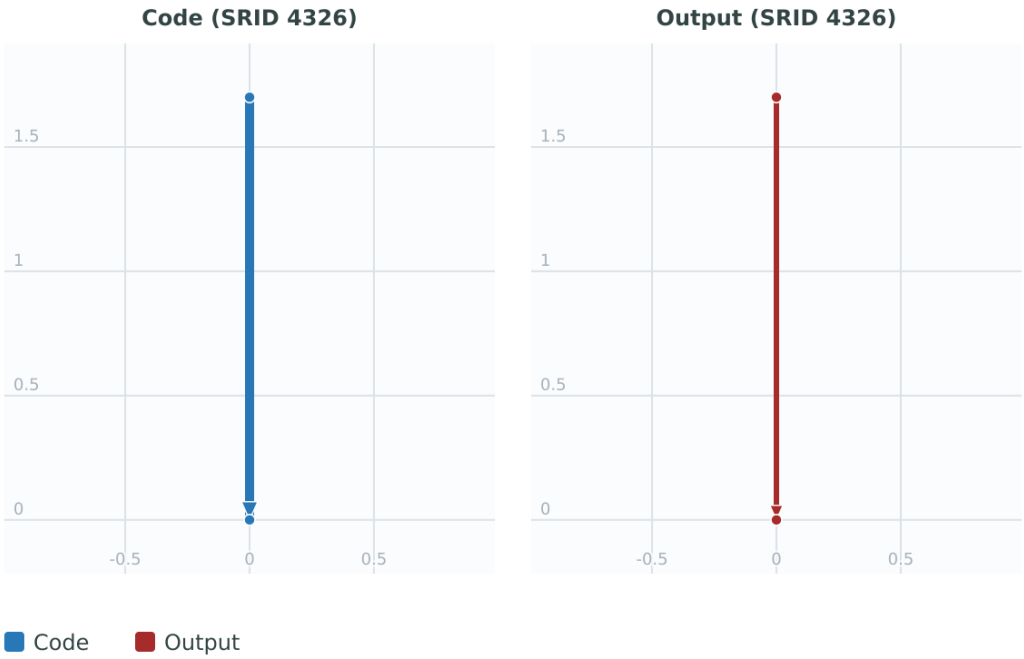
Code

```
WITH test_raster AS (
SELECT
ST_SetValues(
ST_AddBand(
ST_MakeEmptyRaster(
      width => 2,
      height => 2,
      upperleftx => 0,
      upperlefty => 2,
      scalex => 1.0,
      scaley => -1.0,
      skewx => 0,
      skewy => 0,
      srid => 4326),
    index => 1,
    pixeltype => '16BSI',
    initialvalue => 0,
    nodataval => -999
),
  1,
  1,
  1,
  newvalueset => ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8 ↩
    ]]) AS rast
)
SELECT
ST_SetM(
    rast,
    band => 1,
    geom => 'SRID=4326;LINESTRING(1.0 1.9,1.0 0.2)'::geometry,
    resample => 'bilinear'
)
FROM test_raster
```

Output

```
LINESTRING M (1 1.9 38,1 0.2 27)
```

Figure



Voir aussi

ST_Value, ST_SetZ

11.6.11 ST_Neighborhood

ST_Neighborhood — Retourne un tableau 2-D de double avec les valeurs non NODATA autour du pixel de la bande spécifiée, aux coordonnées spécifiées par columnX & rowY ou par un point géométrique dans le même système de référence spatial que le raster.

Synopsis

double precision[][] **ST_Neighborhood**(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=tru

Description

Retourne un tableau 2-D de double avec les valeurs non NODATA autour du pixel de la bande spécifiée, aux coordonnées spécifiées par columnX & rowY ou par un point géométrique dans le même système de référence spatial que le raster. Les paramètres distanceX et distanceY définissent le nombre de pixels autour du pixel spécifié sur les axes X et Y. Exemple : vous pouvez récupérer toutes les valeurs autour du pixel souhaité, à une distance de 3 pixels selon l'axe X et à une distance de 2 pixels selon l'axe Y. La valeur centrale du tableau 2-D sera la valeur du pixel spécifié par columnX et rowY ou par le point géométrique.

Le numéro de bande démarre à 1, et la bande 1 est utilisée si bandnum non spécifié. Si exclude_nodata_value vaut false, tous les pixels y compris ceux ayant la valeur nodata sont considérés comme intersectés et leur valeur sera retournée. Si exclude_nodata_value n'est pas spécifié, la valeur est lue depuis les méta-données du raster.



Note
Le nombre d'éléments retournés dans le tableau 2-D sur chaque axe sera 2 * (`distanceX|distanceY`) + 1. Pour `distanceX` et `distanceY` de 1, le tableau sera de taille 3x3.



Note
Le tableau 2-D en sortie peut être passé à n'importe quelle fonction de traitement raster, comme ST_Min4ma, ST_Sum4ma, ST_Mean4ma.

Disponibilité : 2.1.0

Exemples

This example reads a value from pixel 2x2.

Code

```
SELECT
    ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
    SELECT
        ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo
```

Output

```
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

This example reads a NODATA value from pixel 2x3.

Code

```
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
```

```
            1
        ) AS rast
) AS foo
```

Output

```
{{1,1,1},{1,NULL,1},{1,1,1}}
```

Pixel 3x3 has value. Exclude_nodata_value = FALSE.

Code

```
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
```

Output

```
{{1,1,0},{0,1,1},{1,1,1}}
```

Voir aussi

ST_NearestValue, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.6.12 ST_SetValue

ST_SetValue — Retourne un nouveau raster en modifiant la valeur du pixel pour la bande spécifiée et aux coordonnées columnx, rowy, ou pour tous les pixels qui intersectent une géométrie spécifiée. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié.

Synopsis

raster **ST_SetValue**(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster **ST_SetValue**(raster rast, integer columnx, integer rowy, double precision newvalue);

Description

Retourne un nouveau raster en modifiant la valeur du pixel pour la bande spécifiée et aux coordonnées columnx, rowy, ou pour tous les pixels qui intersectent une géométrie spécifiée. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié.

Amélioration : 2.1.0 La variante de ST_SetValue() avec la géométrie supporte n'importe quelle type de géométrie, pas uniquement des points. La variante avec la géométrie est une enveloppe autour de la variante de ST_SetValues() avec geomval[]

Exemples

This example uses geometry input.

Code

```
                SELECT (foo.geomval).val, ST_Union((foo.geomval).geom)
FROM (SELECT ST_DumpAsPolygons(ST_SetValue(
            rast,
            1,
            ST_Point(3427927.75, 5793243.95),
            50)
            ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;
```

Output

```
 val |                                                        st_astext
-----+---------------------------------------------------------------------------
  50 | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...
```

This example stores the changed raster.

Code

```
UPDATE dummy_rast SET rast = ST_SetValue(rast, 1, ST_Point(3427927.75, 5793243.95), 100)
    WHERE rid = 2   ;
```

Voir aussi

ST_Value, ST_DumpAsPolygons

11.6.13 ST_SetValues

ST_SetValues — Retourne un nouveau raster en modifiant les valeurs de certains pixels d'une bande spécifiée.

Synopsis

raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);

Description

Retourne un nouveau raster en modifiant les valeurs de certains pixels d'une bande spécifiée. Les paramètres `columnx` et `rowy` démarrent à 1.

Si `keepnodata` est TRUE, les pixels qui sont NODATA ne seront pas mis à jour avec la valeur donnée par `newvalueset`.

Pour la variante 1, les pixels sont déterminés par les coordonnées du pixel `columnx` et `rowy`, ainsi que la dimension du tableau `newvalueset`. `noset` peut être utilisé pour s'assurer qu'aucun pixel présent dans `newvalueset` et ayant déjà une valeur soient modifiés (PostgreSQL ne supportant pas les tableaux irréguliers/agglomérés). Voir l'exemple Variante 1.

La variante 2 est similaire à la variante 1, mais avec une valeur unique en double precision `nosetvalue` au lieu du tableau de booléens `noset`. Les éléments dans `newvalueset` avec la valeur `nosetvalue` seront ignorés. Voir l'exemple Variante 2.

Pour la variante 3, les pixels à modifier doivent être explicitement spécifiés par les coordonnées `columnx`, `rowy`, ainsi que les tailles `width` (largeur) et `height` (hauteur). Voir l'exemple Variante 3.

La variante 4 est identique à la variante 3, en modifiant la bande raster 1 du raster `rast`.

Pour la variante 5, un tableau de geomval est utilisé pour déterminer les pixels à modifier. Si toutes les géométries du tableau sont de type POINT ou MULTIPOINT, la fonction utilise un raccourci où la longitude et la latitude du chaque point sont utilisés pour modifier directement le pixel. Sinon, les géométries sont converties en raster, puis itérés en une passe. Voir l'exemple Variante 5.

Disponibilité : 2.1.0

Exemples

Variant 1.

These examples pass `FALSE` for *`exclude_nodata_value`* so that every raster cell remains visible. NODATA cells display the band's NODATA value, which is `0` here.

Set the lower-right 2 by 2 block of a 3 by 3 raster to `9`.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Replace the complete raster from an array; the central `NULL` becomes NODATA.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   9
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

Use the *noset* mask to retain the left-middle source pixel while replacing the other cells.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][]
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

With *keepnodata* enabled, an existing NODATA pixel is not overwritten.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, ↩
     -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, NULL
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][],
                TRUE
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   0
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

Variant 2.

Use `-1` as the value that should not be written; only the lower-right 2 by 2 block changes.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -1
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

The same update can use `NULL` as the value that should not be written.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double precision ↩
    [][], NULL::double precision
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Variant 3.

Set a 2 by 2 rectangle starting at column 2, row 2.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0,  ↩
    0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        ),
        1, FALSE
```

```
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

When *keepnodata* is true, the NODATA cell inside the rectangle remains unchanged.

Code

```
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(ST_SetValues(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, ↩
     -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, NULL
            ),
            1, 2, 2, 2, 2, 9, TRUE
        ),
        1, FALSE
    ) AS poly
) foo
ORDER BY 1, 2;
```

Output

```
x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   0
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Variant 5.

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
    0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1,4 -1,4 -4,1 -4,1 -1))'::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0,5 0,5 -1,1 -1,1 -4,0 -4,0 0))'::geometry geom  ↩
    UNION ALL
```

```
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0,4 4,4 -4)'::geometry
), changed AS (
    SELECT rid, gid, ST_SetValue(rast, 1, geom, gid) AS rast
    FROM foo
    CROSS JOIN bar
)
SELECT
    rid,
    gid,
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY rid, gid, y
ORDER BY rid, gid, y;
```

Output

```
rid | gid | y |           values
-----+-----+---+---------------------------
   1 |   1 | 1 | {NULL,NULL,NULL,NULL,NULL}
   1 |   1 | 2 | {NULL,NULL,NULL,NULL,NULL}
   1 |   1 | 3 | {NULL,NULL,1,NULL,NULL}
   1 |   1 | 4 | {NULL,NULL,NULL,NULL,NULL}
   1 |   1 | 5 | {NULL,NULL,NULL,NULL,NULL}
   1 |   2 | 1 | {NULL,NULL,NULL,NULL,NULL}
   1 |   2 | 2 | {NULL,2,2,2,NULL}
   1 |   2 | 3 | {NULL,2,2,2,NULL}
   1 |   2 | 4 | {NULL,2,2,2,NULL}
   1 |   2 | 5 | {NULL,NULL,NULL,NULL,NULL}
   1 |   3 | 1 | {3,3,3,3,3}
   1 |   3 | 2 | {3,NULL,NULL,NULL,NULL}
   1 |   3 | 3 | {3,NULL,NULL,NULL,NULL}
   1 |   3 | 4 | {3,NULL,NULL,NULL,NULL}
   1 |   3 | 5 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 1 | {4,NULL,NULL,NULL,NULL}
   1 |   4 | 2 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 3 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 4 | {NULL,NULL,NULL,NULL,NULL}
   1 |   4 | 5 | {NULL,NULL,NULL,NULL,4}
(20 rows)
```

L'exemple suivant montre qu'une géométrie située plus loin dans le tableau peut écraser une valeur issue d'une géométrie précédente

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
    0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1,4 -1,4 -4,1 -4,1 -1))'::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0,5 0,5 -1,1 -1,1 -4,0 -4,0 0))'::geometry geom ↩
    UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0,4 4,4 -4)'::geometry
), changed AS (
    SELECT ST_SetValues(
        rast,
        1,
        ARRAY[
            ROW((SELECT geom FROM bar WHERE gid = 1), 1),
            ROW((SELECT geom FROM bar WHERE gid = 2), 2)
        ]::geomval[]
    ) AS rast
    FROM foo
)
```

```
SELECT
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY y
ORDER BY y;
```

Output

```
y |           values
---+---------------------------
 1 | {NULL,NULL,NULL,NULL,NULL}
 2 | {NULL,2,2,2,NULL}
 3 | {NULL,2,2,2,NULL}
 4 | {NULL,2,2,2,NULL}
 5 | {NULL,NULL,NULL,NULL,NULL}
(5 rows)
```

Cet exemple est le contraire de l'exemple précédent

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
    0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1,4 -1,4 -4,1 -4,1 -1))'::geometry geom UNION ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0,5 0,5 -1,1 -1,1 -4,0 -4,0 0))'::geometry geom ↩
    UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0,4 4,4 -4)'::geometry
), changed AS (
    SELECT ST_SetValues(
        rast,
        1,
        ARRAY[
            ROW((SELECT geom FROM bar WHERE gid = 2), 2),
            ROW((SELECT geom FROM bar WHERE gid = 1), 1)
        ]::geomval[]
    ) AS rast
    FROM foo
)
SELECT
    y,
    array_agg(ST_Value(rast, 1, x, y) ORDER BY x) AS values
FROM changed
CROSS JOIN generate_series(1, 5) AS x
CROSS JOIN generate_series(1, 5) AS y
GROUP BY y
ORDER BY y;
```

Output

```
y |           values
---+---------------------------
 1 | {NULL,NULL,NULL,NULL,NULL}
 2 | {NULL,2,2,2,NULL}
 3 | {NULL,2,1,2,NULL}
 4 | {NULL,2,2,2,NULL}
 5 | {NULL,NULL,NULL,NULL,NULL}
(5 rows)
```

Voir aussi

ST_Value, ST_SetValue, ST_PixelAsPolygons

11.6.14 ST_DumpValues

ST_DumpValues — Retourne les valeurs d'une bande raster spécifiée, sous forme d'un tableau à deux dimensions.

Synopsis

setof record **ST_DumpValues**(raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true);
double precision[][] **ST_DumpValues**(raster rast , integer nband , boolean exclude_nodata_value=true);

Description

Retour les valeurs d'une bande, sous forme d'un tableau à 2 dimensions (le premier index étant la ligne, le second étant la colonne). Si `nband` est NULL ou non spécifié, toutes les bandes raster sont traitées.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),  ↩
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast)).*
FROM foo;
```

Output

```
nband |                      valarray
------+-----------------------------------------------------
     1 | {{1,1,1},{1,1,1},{1,1,1}}
     2 | {{3,3,3},{3,3,3},{3,3,3}}
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
(3 rows)
```

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),  ↩
    1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;
```

Output

```
nband |                      valarray
------+-----------------------------------------------------
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
     1 | {{1,1,1},{1,1,1},{1,1,1}}
(2 rows)
```

Code

```
WITH foo AS (
    SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
     1, 0), 1, 2, 5) AS rast
)
SELECT
    (ST_DumpValues(rast, 1))[2][1]
FROM foo;
```

Output

```
5
```

Voir aussi

ST_Value, ST_SetValue, ST_SetValues

11.6.15 ST_PixelOfValue

ST_PixelOfValue — Retourne les coordonnées columnx, rowy du pixel dont la valeur est égale à la valeur recherchée.

Synopsis

setof record **ST_PixelOfValue**(raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , double precision[] search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , integer nband , double precision search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , double precision search , boolean exclude_nodata_value=true);

Description

Retourne les coordonnées columnx, rowy du pixel dont la valeur est égale à la valeur recherchée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Disponibilité : 2.1.0

Exemples

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
            1, '8BUI', 1, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 1, 1, 1],
            [1, 1, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 1, 1, 1, 255],
            [1, 1, 0, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT (ST_PixelOfValue(rast, 1, ARRAY[1, 255])).*
FROM source
```

Output

```
val | x | y
-----+---+---
   1 | 1 | 2
   1 | 1 | 3
   1 | 1 | 4
   1 | 1 | 5
   1 | 2 | 1
   1 | 2 | 2
   1 | 2 | 4
   1 | 2 | 5
   1 | 3 | 1
   1 | 3 | 2
   1 | 3 | 3
   1 | 3 | 4
   1 | 4 | 1
   1 | 4 | 3
   1 | 4 | 4
   1 | 4 | 5
   1 | 5 | 1
   1 | 5 | 2
   1 | 5 | 3
 255 | 5 | 4
   1 | 5 | 5
```

11.7 Éditeurs de raster

11.7.1 ST_SetGeoReference

ST_SetGeoReference — Définit les 6 paramètres de géo-référencement en un seul appel. Les nombres doivent être séparés par un espace. Accepte les formats GDAL (par défaut) ou ESRI.

Synopsis

raster **ST_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);
raster **ST_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

Description

Définit les 6 paramètres de géo-référencement en un seul appel. Accepte les formats 'GDAL' (par défaut) ou 'ESRI'. Si les 6 coordonnées ne sont pas spécifiées, retourne null.

La différence entre les représentations de format est la suivante :

`GDAL` uses the value order shown below.

Code

```
scalex skewy skewx scaley upperleftx upperlefty
```

`ESRI` uses the value order shown below.

Code

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



Note
Si le raster a des bandes out-db, modifier le géoréférencement peut entraîner un accès incorrect aux données de la bande out-db.

Amélioration : 2.1.0 Ajout de la variante ST_SetGeoReference(raster, double precision, ...)

Exemples

Code

```
WITH source AS (
    SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
), variants AS (
    SELECT 0 AS rid, rast FROM source
    UNION ALL
    SELECT 1, ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL')
    FROM source
    UNION ALL
    SELECT 2, ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI')
    FROM source
    UNION ALL
    SELECT 3, ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001)
    FROM source
)
SELECT
    rid,
    round(ST_UpperLeftX(rast)::numeric, 3) AS upperleftx,
    round(ST_UpperLeftY(rast)::numeric, 3) AS upperlefty,
    ST_ScaleX(rast) AS scalex,
    ST_ScaleY(rast) AS scaley,
    ST_SkewX(rast) AS skewx,
    ST_SkewY(rast) AS skewy
FROM variants
ORDER BY rid;
```

Output

```
rid | upperleftx | upperlefty | scalex | scaley | skewx | skewy
-----+------------+------------+--------+--------+-------+-------
   0 |      0.000 |      0.000 |      1 |     -1 |     0 |     0
   1 |      0.100 |      0.100 |     10 |    -10 |     0 |     0
   2 |      0.100 |      0.100 |     10 |    -10 |     0 |     0
   3 |      1.000 |      1.000 |     10 |    -10 | 0.001 | 0.001
(4 rows)
```

Voir aussi

ST_GeoReference, ST_ScaleX, ST_ScaleY, ST_UpperLeftX, ST_UpperLeftY

11.7.2 ST_SetRotation

ST_SetRotation — Définit la rotation du raster en radians.

Synopsis

raster **ST_SetRotation**(raster rast, float8 rotation);

Description

Applique une rotation uniforme du raster. L'angle de rotation est en radians. Voir World File pour plus de détails.

Exemples

Code

```
WITH variants AS (
    SELECT rid, 'rotated' AS kind, ST_SetRotation(rast, 15) AS rast
    FROM dummy_rast
    UNION ALL
    SELECT rid, 'original', rast
    FROM dummy_rast
)
SELECT
    rid,
    kind,
    round(ST_ScaleX(rast)::numeric, 4) AS scale_x,
    round(ST_ScaleY(rast)::numeric, 4) AS scale_y,
    round(ST_SkewX(rast)::numeric, 4) AS skew_x,
    round(ST_SkewY(rast)::numeric, 4) AS skew_y
FROM variants
ORDER BY rid, kind;
```

Output

```
rid |   kind   | scale_x | scale_y | skew_x | skew_y
-----+----------+---------+---------+--------+--------
   1 | original |  2.0000 |  3.0000 | 0.0000 | 0.0000
   1 | rotated  | -1.5194 | -2.2791 | 1.9509 | 1.3006
   2 | original |  0.0500 | -0.0500 | 0.0000 | 0.0000
   2 | rotated  | -0.0380 | -0.0380 | 0.0325 | 0.0325
(4 rows)
```

Voir aussi

ST_Rotation, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

11.7.3 ST_SetScale

ST_SetScale — Définit la résolution des pixels en X et Y en unité du système de référence spatial : nombre d'unités/pixel en largeur/hauteur.

Synopsis

raster **ST_SetScale**(raster rast, float8 xy);
raster **ST_SetScale**(raster rast, float8 x, float8 y);

Description

Définit la résolution des pixels en X et Y en unité du système de référence spatial : nombre d'unités/pixel en largeur/hauteur. Si une seule unité est spécifiée, la même résolution est utilisée pour X et pour Y.



Note
ST_SetScale est différent de ST_Rescale : ST_SetScale ne rééchantillonne pas le raster pour correspondre à l'étendue du raster. La fonction modifie seulement les méta-données (ou la géo-référence) d'un raster pour corriger une échelle incorrecte. ST_Rescale produit un raster de taille différente pour correspondre à l'étendue spatiale du raster d'entrée. ST_SetScale ne modifie pas la largeur ou hauteur du raster.

Changement : 2.0.0 Dans les versions WKTRaster, cette fonction était appelée ST_SetPixelSize.

Exemples

Code

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

Output

```
pixx | pixy |                    newbox
------+------+-------------------------------------------------
  1.5 |  1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)
```

Code

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;
```

Output

```
pixx | pixy |                   newbox
------+------+-----------------------------------------------
  1.5 | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0)
```

Voir aussi

ST_ScaleX, ST_ScaleY, Box3D

11.7.4 ST_SetSkew

ST_SetSkew — Définit l'obliquité X et Y (skew, ou paramètre de rotation). Si une seule valeur est spécifiée, la même valeur est utilisée pour X et pour Y.

Synopsis

raster **ST_SetSkew**(raster rast, float8 skewxy);
raster **ST_SetSkew**(raster rast, float8 skewx, float8 skewy);

Description

Définit l'obliquité X et Y (skew, ou paramètre de rotation). Si une seule valeur est spécifiée, la même valeur est utilisée pour X et pour Y. Voir World File pour plus de détails.

Exemples

This is the first example.

Code

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast, 1, 2) WHERE rid = 1;
SELECT
    rid,
    format('(%s, %s)', skewx, skewy) AS skew,
    term,
    georef[idx] AS value
FROM (
    SELECT
        rid,
        ST_SkewX(rast) AS skewx,
        ST_SkewY(rast) AS skewy,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
    WHERE rid = 1
) AS refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
) AS parts(idx, term)
ORDER BY idx;
```

Output

```
rid |  skew  |  term   |    value
-----+--------+---------+--------------
   1 | (1, 2) | scaleX  | 2.0000000000
   1 | (1, 2) | skewY   | 2.0000000000
   1 | (1, 2) | skewX   | 1.0000000000
   1 | (1, 2) | scaleY  | 3.0000000000
   1 | (1, 2) | originX | 0.5000000000
   1 | (1, 2) | originY | 0.5000000000
(6 rows)
```

This second example sets both values to the same number.

Code

```
UPDATE dummy_rast SET rast = ST_SetSkew(rast, 0) WHERE rid = 1;
SELECT
    rid,
    format('(%s, %s)', skewx, skewy) AS skew,
    term,
    georef[idx] AS value
FROM (
    SELECT
        rid,
        ST_SkewX(rast) AS skewx,
        ST_SkewY(rast) AS skewy,
        string_to_array(ST_GeoReference(rast), E'\n') AS georef
    FROM dummy_rast
    WHERE rid = 1
) AS refs
CROSS JOIN (VALUES
    (1, 'scaleX'),
    (2, 'skewY'),
    (3, 'skewX'),
    (4, 'scaleY'),
    (5, 'originX'),
    (6, 'originY')
```

```
) AS parts(idx, term)
ORDER BY idx;
```

Output

```
rid |  skew  |  term   |    value
-----+--------+---------+-------------
   1 | (0, 0) | scaleX  | 2.0000000000
   1 | (0, 0) | skewY   | 0.0000000000
   1 | (0, 0) | skewX   | 0.0000000000
   1 | (0, 0) | scaleY  | 3.0000000000
   1 | (0, 0) | originX | 0.5000000000
   1 | (0, 0) | originY | 0.5000000000
(6 rows)
```

Voir aussi

ST_GeoReference, ST_SetGeoReference, ST_SkewX, ST_SkewY

11.7.5 ST_SetSRID

ST_SetSRID — Modifie le SRID d'un raster à une valeur définie dans la table spatial_ref_sys.

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Description

Modifie le SRID d'un raster à une valeur spécifiée.



Note
Cette fonction ne modifie pas le raster en lui-même : elle définit juste la méta-données définissant le système de référence spatial. Ceci est utile pour ensuite effectuer des transformations.

Voir aussi

Section 4.5, ST_SRID

11.7.6 ST_SetUpperLeft

ST_SetUpperLeft — Modifie les coordonnées du pixel du coin supérieur gauche du raster, selon les coordonnées X et Y projetées.

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Description

Modifie les coordonnées du pixel du coin supérieur gauche du raster aux coordonnées X et Y projetées

Exemples

Code

```
SELECT ST_SetUpperLeft(rast, -71.01, 42.37)
FROM dummy_rast
WHERE rid = 2;
```

Voir aussi

ST_UpperLeftX, ST_UpperLeftY

11.7.7 ST_Resample

ST_Resample — Rééchantillonne un raster, en utilisant l'algorithme spécifié, les nouvelles dimensions, un coin arbitraire de la grille et un ensemble de paramètres de géo-référencement définis ou empruntés à un autre raster.

Synopsis

raster **ST_Resample**(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, raster ref, text algorithm=NearestNeighbor, double precision maxerr=0.125, boolean usescale=true);
raster **ST_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Rééchantillonne un raster, en utilisant l'algorithme spécifié, les nouvelles dimensions (width & height), un coin de la grille (gridx & gridy) et un ensemble de paramètres de géo-référencement (scalex, scaley, skewx & skewy) définis ou empruntés à un autre raster. Si un raster de référence est spécifié, les deux rasters doivent avoir le même SRID.

Les nouvelles valeurs des pixels sont calculées par un des algorithmes de rééchantillonnage suivants :

— NearestNeighbor (plus proche voisin)
— Bilinear (Bilinéaire)
— Cubic (Cubique)
— CubicSpline (Cubique Spline)
— Lanczos
— Max
— Min

L'algorithme par défaut est NearestNeighbor, qui est le plus rapide mais donne le moins bon résultat.

Une erreur maximale de 0.125 est utilisée si le paramètre `maxerr` n'est pas spécifié.



Note
Voir GDAL Warp resampling methods pour plus de détails.

Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+

Amélioration : 3.4.0 ajout des options de rééchantillonnage max et min

Exemples

Code

```
SELECT
    ST_Width(orig) AS orig_width,
    ST_Width(reduce_100) AS new_width
FROM (
    SELECT
        rast AS orig,
        ST_Resample(rast, 100, 100) AS reduce_100
    FROM aerials.boston
    WHERE ST_Intersects(
        rast,
        ST_Transform(ST_MakeEnvelope(-71.128, 42.2392, -71.1277, 42.2397, 4326),
            26986)
    )
    LIMIT 1
) AS foo;
```

Output

```
orig_width | new_width
-----------+-------------
        200 |         100
```

Voir aussi

ST_Rescale, ST_Resize, ST_Transform

11.7.8 ST_Rescale

ST_Rescale — Rééchantillonne un raster en ajustant juste son échelle (ou la taille des pixels). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline), Lanczos, Max ou Min. La valeur par défaut est NearestNeighbor.

Synopsis

raster **ST_Rescale**(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Rescale**(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Rééchantillonne un raster en ajustant juste son échelle (ou la taille des pixels). Les nouvelles valeurs des pixels sont calculées en utilisant un des algorithmes de rééchantillonnage suivants :

— NearestNeighbor (plus proche voisin)
— Bilinear (Bilinéaire)
— Cubic (Cubique)
— CubicSpline (Cubique Spline)
— Lanczos
— Max
— Min

L'algorithme par défaut est NearestNeighbor, qui est le plus rapide mais donne le moins bon résultat.

`scalex` et `scaley` définissent la nouvelle taille des pixels. scaley est souvent négatif pour obtenir un raster correctement orienté.

Quand la nouvelle valeur de scalex (respectivement scaley) n'est pas un diviseur de la hauteur (respectivement largeur) du raster, l'étendue du raster résultant est agrandie pour englober l'étendue du raster d'entrée. Si vous voulez conserver l'étendue exacte du raster d'entrée, utilisez ST_Resize

`maxerr` est le seuil pour l'approximation de l'algorithme de rééchantillonnage (en pixels). La valeur de 0.125 est utilisée si le paramètre `maxerr` n'est pas spécifié, qui est la même valeur que celle utilisée par l'utilitaire GDAL gdalwarp. Si définie à 0, aucune approximation n'est effectuée.



Note
Voir GDAL Warp resampling methods pour plus de détails.



Note
ST_Rescale est différent de ST_SetScale : ST_SetScale ne rééchantillonne pas le raster pour correspondre à l'étendue du raster. ST_SetScale modifie seulement les méta-données (ou lagéo-référence) d'un raster pour corriger une échelle incorrecte. ST_Rescale produit un raster de taille différente pour correspondre à l'étendue spatiale du raster d'entrée. ST_SetScale ne modifie pas la largeur ou hauteur du raster.

Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+

Amélioration : 3.4.0 ajout des options de rééchantillonnage max et min

Changement : 2.1.0 Fonctionne avec les rasters sans SRID

Exemples

Exemple simple de rééchantillonnage d'un raster d'une taille de pixel de 0.001 degrés vers une taille de pixel de 0.0015 degrés.

This example returns the original raster pixel size.

Code

```
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0,  ↩
    4269), '8BUI'::text, 1, 0)) width
```

Output

```
0.001
```

This example returns the pixel size of the rescaled raster.

Code

```
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, ↩
     0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width
```

Output

```
0.0015
```

Voir aussi

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

11.7.9 ST_Reskew

ST_Reskew — Rééchantillonne un raster en ajustant simplement son obliquité (skew, ou paramètre de rotation). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.

Synopsis

raster **ST_Reskew**(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Reskew**(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Rééchantillonne un raster en ajustant simplement son obliquité (skew, ou paramètre de rotation). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor, qui est le plus rapide mais donne le moins bon résultat.

`skewx` et `skewy` définissent la nouvelle obliquité.

L'étendue du nouveau raster englobe l'étendu du raster en entrée.

Un pourcentage maximal d'erreur de 0.125 % est utilisé si le paramètre `maxerr` n'est pas spécifié.



Note
Voir GDAL Warp resampling methods pour plus de détails.



Note
ST_Reskew est différent de ST_SetSkew : ST_SetSkew ne rééchantillonne pas le raster pour correspondre à l'étendue du raster. ST_SetSkew modifie seulement les méta-données (ou la géo-référence) du raster pour corriger une obliquité incorrecte. ST_Reskew produit un raster de taille différente pour correspondre à l'étendue spatiale du raster d'entrée. ST_SetSkew ne modifie pas la largeur ou hauteur du raster.

Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+

Changement : 2.1.0 Fonctionne avec les rasters sans SRID

Exemples

Exemple simple de rééchantillonnage d'un raster d'une obliquité de 0.0 à 0.0015.

This example uses the original non-rotated raster.

Code

```
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), ↩
    '8BUI'::text, 1, 0));
```

Output

```
st_rotation
-------------
           0
(1 row)
```

The reskewed raster rotation.

Code

```
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, ↩
    0, 4269), '8BUI'::text, 1, 0), 0.0015));
```

Output

```
st_rotation
------------------
 -0.982793723247329
(1 row)
```

Voir aussi

ST_Resample, ST_Rescale, ST_SetSkew, ST_SetRotation, ST_SkewX, ST_SkewY, ST_Transform

11.7.10 ST_SnapToGrid

ST_SnapToGrid — Rééchantillonne un raster en l'accrochant sur une grille. Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.

Synopsis

raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeigl
double precision maxerr=0.125);

Description

Rééchantillonne un raster en l'accrochant sur une grille définie par un coin arbitraire (gridx & gridy) et une taille de pixel optionnelle (scalex & scaley). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor, qui est le plus rapide mais donne le moins bon résultat.

`gridx` et `gridy` définissent le pixel d'un coin arbitraire de la nouvelle grille. Ce n'est pas nécessairement le coin supérieur gauche du nouveau raster, et n'a pas besoin d'être à l'intérieur ou sur les bords de l'emprise du nouveau raster.

Il est possible de définir une taille de pixel pour la nouvelle grille avec les paramètres optionnels `scalex` et `scaley`.

L'étendue du nouveau raster englobe l'étendu du raster en entrée.

Un pourcentage maximal d'erreur de 0.125 % est utilisé si le paramètre `maxerr` n'est pas spécifié.



Note
Voir GDAL Warp resampling methods pour plus de détails.



Note
Voir ST_Resample pour plus de contrôle sur les paramètres de la grille.

Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+

Changement : 2.1.0 Fonctionne avec les rasters sans SRID

Exemples

Exemple simple d'accrochage d'un raster sur une grille légèrement différente.

This example returns the upper-left X coordinate of the original raster.

Code

```
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269), ↩
    '8BUI'::text, 1, 0));
```

Output

```
st_upperleftx
---------------
              0
(1 row)
```

The upper left of raster after snapping.

Code

```
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001,  ↩
   -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));
```

Output

```
st_upperleftx
---------------
       -0.0008
(1 row)
```

Voir aussi

ST_Resample, ST_Rescale, ST_UpperLeftX, ST_UpperLeftY

11.7.11 ST_Resize

ST_Resize — Redimensionne un raster à une nouvelle largeur/hauteur

Synopsis

raster **ST_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Redimensionne un raster à une nouvelle largeur/hauteur. La nouvelle largeur/hauteur peut être spécifiée comme un nombre exact de pixels ou un pourcentage de la taille du raster. L'étendue du nouveau raster est la même que le raster d'entrée.

Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor, qui est le plus rapide mais donne le moins bon résultat.

La variante 1 attend la largeur/hauteur réelle du raster de sortie.

La variante 2 attend des valeurs décimales entre zéro (0) et un (1) indiquant le pourcentage de la largeur/hauteur du raster d'entrée.

La variante 3 prend en compte soit la largeur/hauteur réelle du raster de sortie ou un pourcentage sous forme de texte ("20%") indiquant le pourcentage de la largeur/hauteur du raster d'entrée.

Disponibilité : 2.1.0 Nécessite GDAL 1.6.1+

Exemples

Code

```
WITH foo AS(
SELECT
    1 AS rid,
    ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 255, 0
        ),
    '50%', '500') AS rast
UNION ALL
SELECT
    2 AS rid,
    ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 255, 0
        ),
    500, 100) AS rast
UNION ALL
SELECT
    3 AS rid,
    ST_Resize(ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0),
            1, '8BUI', 255, 0
        ),
    0.25, 0.9) AS rast
)
SELECT
    rid,
    ST_Width(rast) AS width,
    ST_Height(rast) AS height,
    ST_ScaleX(rast) AS scale_x,
    round(ST_ScaleY(rast)::numeric, 4) AS scale_y
FROM foo
ORDER BY rid
```

Output

```
rid | width | height | scale_x | scale_y
-----+-------+--------+---------+---------
   1 |   500 |    500 |       2 | -2.0000
   2 |   500 |    100 |       2 | -10.0000
   3 |   250 |    900 |       4 | -1.1111
(3 rows)
```

Voir aussi

ST_Resample, ST_Rescale, ST_Reskew, ST_SnapToGrid

11.7.12 ST_Transform

ST_Transform — Reprojette un raster depuis un système de référence spatial vers un autre, en utilisant l'algorithme de rééchantillonnage spécifié. Les algorithmes possibles sont NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.

Synopsis

raster **ST_Transform**(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);
raster **ST_Transform**(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Transform**(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Description

Reprojette un raster depuis un système de référence spatial vers un autre, en utilisant l'algorithme de rééchantillonnage spécifié. L'algorithme utilisé par défaut est NearestNeighbor si aucun n'est spécifié. Le pourcentage d'erreur maximal est 0.125 si maxerr n'est pas spécifié.

Les algorithmes disponibles sont : 'NearestNeighbor' (plus proche voisin), 'Bilinear' (Bilinéaire), 'Cubic' (Cubique), 'CubicSpline' (Cubique Spline) et 'Lanczos'. Voir GDAL Warp resampling methods pour plus de détails.

ST_Transform est souvent confondu avec ST_SetSRID(). ST_Transform modifie réellement les coordonnées du raster (et réechantillonne les valeurs des pixels) d'un système de référence spatial vers un autre, alors que ST_SetSRID() change uniquement l'identifiant SRID du raster.

Contrairement aux autres variantes, la variante 3 nécessite un raster de référence `alignto`. Le raster de sortie sera transformé dans le système de référence spatial (SRID) du raster de référence et sera aligné (ST_SameAlignment = TRUE) avec le raster de référence.



Note
If you find your transformation support is not working right, you may need to set the environment variable PROJSO to the projection library your PostGIS build is using. This just needs to have the name of the file. So for example on windows, you would in Control Panel -> System -> Environment Variables add a system variable called `PROJSO` and set it to `libproj.dll`. You'll have to restart your PostgreSQL service/daemon after this change.



AVERTISSEMENT
Lors de la transformation d'une couverture tuilée, vous voudrez en général utiliser un raster de référence pour garantir le même alignement et sans lacunes dans les tuiles, comme le montre l'exemple Variante 3.

Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+

Amélioration : 2.1.0 Ajout de la variante ST_Transform(rast, alignto)

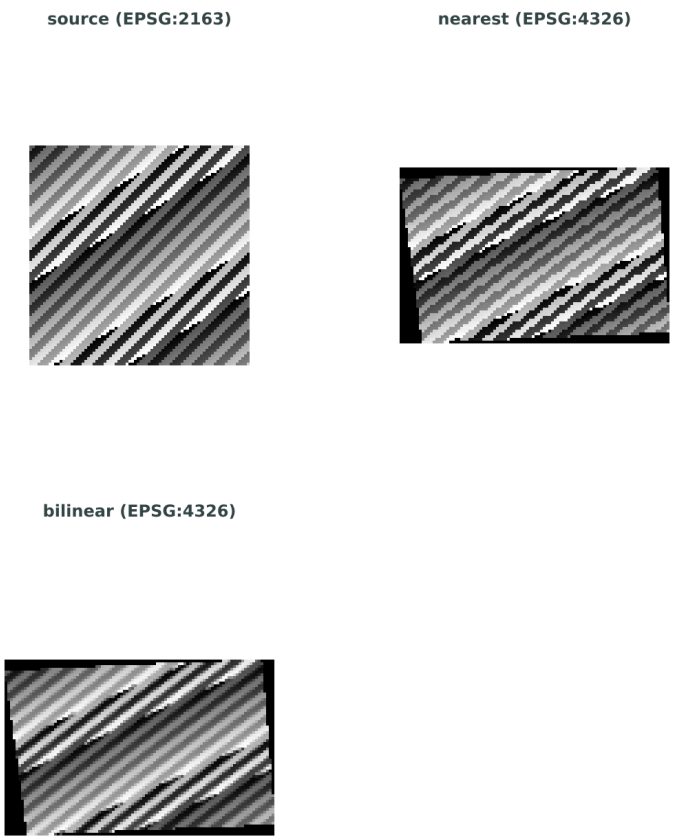
Exemples

Compare a source raster with nearest-neighbor and bilinear reprojections. The diagonal bands make the resampling behavior visible: nearest-neighbor keeps hard pixel steps, while bilinear smooths values between pixels. All three figures are generated from this query at manual build time.

Code

```
WITH rows AS (
    SELECT
        y,
        array_agg(
            ((
                    x * 3 + y * 5
                    + CASE WHEN (x + y) % 8 < 4 THEN 80 ELSE 0 END
                ) % 255)::double precision
            ORDER BY x
        ) AS row_values
    FROM generate_series(1, 80) AS y
    CROSS JOIN generate_series(1, 80) AS x
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(
                80, 80,
                -500000, 600000,
                2500, -2500,
                0, 0,
                2163
            ),
            1, '8BUI', 0, NULL
        ),
        1, 1, 1,
        array_agg(row_values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), transformed AS (
    SELECT
        rast,
        ST_Transform(rast, 4326, 'NearestNeighbor') AS nearest,
        ST_Transform(rast, 4326, 'Bilinear') AS bilinear
    FROM source
)
SELECT
    ST_AsPNG(rast) AS "source (EPSG:2163)",
    ST_AsPNG(nearest) AS "nearest (EPSG:4326)",
    ST_AsPNG(bilinear) AS "bilinear (EPSG:4326)"
FROM transformed;
```

Figure



Raster reprojection and resampling.



Massachusetts State Plane source



WGS 84 nearest-neighbor



WGS 84 bilinear

Variant 3.

L'exemple suivant montre la différence entre ST_Transform(raster, srid) et ST_Transform(raster, alignto)

Code

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0,
    2163), 1, '16BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163),
    1, '16BUI', 2, 0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163),
    1, '16BUI', 3, 0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163),
    1, '16BUI', 10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163),
    1, '16BUI', 20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163),
    1, '16BUI', 30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163),
    1, '16BUI', 100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163),
    1, '16BUI', 200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163),
    1, '16BUI', 300, 0) AS rast
), bar AS (
    SELECT
        ST_Transform(rast, 4269) AS alignto
    FROM foo
    LIMIT 1
), baz AS (
    SELECT
        rid,
        rast,
        ST_Transform(rast, 4269) AS not_aligned,
        ST_Transform(rast, alignto) AS aligned
    FROM foo
    CROSS JOIN bar
)
SELECT
    ST_Collect(ST_ConvexHull(not_aligned) ORDER BY rid) AS not_aligned,
    ST_Collect(ST_ConvexHull(aligned) ORDER BY rid) AS aligned
FROM baz
```

Output

```
MULTIPOLYGON(((-107.007 50.2,-107.005 50.2,-107.005 50.198,-107.007 50.198,-107.007 50.2)),((-107.004 50.201,-107.002
    50.201,-107.002 50.198,-107.004 50.198,-107.004 50.201)),((-107.001 50.201,-106.999 50.201,-106.999
    50.198,-107.001 50.198,-107.001 50.201)),((-107.007 50.199,-107.004 50.199,-107.004 50.196,-107.007
    50.196,-107.007 50.199)),((-107.004 50.199,-107.001 50.199,-107.001 50.196,-107.004 50.196,-107.004 50.199))
    ,((-107.001 50.199,-106.999 50.199,-106.999 50.196,-107.001 50.196,-107.001 50.199)),((-107.006 50.197,-107.004
    50.197,-107.004 50.194,-107.006 50.194,-107.006 50.197)),((-107.004 50.197,-107.001 50.197,-107.001
    50.194,-107.004 50.194,-107.004 50.197)),((-107.001 50.197,-106.998 50.197,-106.998 50.195,-107.001
    50.195,-107.001 50.197))) | MULTIPOLYGON(((-107.007 50.2,-107.005 50.2,-107.005 50.198,-107.007 50.198,-107.007
    50.2)),((-107.005 50.202,-107.001 50.202,-107.001 50.198,-107.005 50.198,-107.005 50.202)),((-107.002
    50.202,-106.998 50.202,-106.998 50.198,-107.002 50.198,-107.002 50.202)),((-107.007 50.199,-107.003
    50.199,-107.003 50.195,-107.007 50.195,-107.007 50.199)),((-107.005 50.199,-107.001 50.199,-107.001
    50.195,-107.005 50.195,-107.005 50.199)),((-107.002 50.199,-106.998 50.199,-106.998 50.195,-107.002
    50.195,-107.002 50.199)),((-107.007 50.198,-107.003 50.198,-107.003 50.194,-107.007 50.194,-107.007 50.198))
    ,((-107.005 50.198,-107.001 50.198,-107.001 50.194,-107.005 50.194,-107.005 50.198)),((-107.002 50.198,-106.998
    50.198,-106.998 50.194,-107.002 50.194,-107.002 50.198)))
```

Figure



Voir aussi

ST_Transform, ST_SetSRID

11.8 Éditeurs de bandes raster

11.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — Définit la valeur pour l'absence de données (nodata) pour la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée. Pour indiquer qu'une bande n'a pas de valeur nodata, définir la valeur nodata = NULL.

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

Description

Définit la valeur pour l'absence de données (nodata) pour la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée. Cette valeur affecte les résultats de ST_Polygon, ST_DumpAsPolygons, ainsi que les fonctions ST_PixelAs...().

Exemples

This example changes only the first band NODATA value.

Code

```
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast, 1, 254)
WHERE rid = 2;
```

This example changes the NODATA value for bands 1, 2, and 3.

Code

```
UPDATE dummy_rast
    SET rast =
        ST_SetBandNoDataValue(ST_SetBandNoDataValue(ST_SetBandNoDataValue(rast, 1, 254),
                2, 99),
                3, 108)
        WHERE rid = 2;
```

This example removes the NODATA value so that all pixels are considered by processing functions.

Code

```
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast, 1, NULL)
WHERE rid = 2;
```

Voir aussi

ST_BandNoDataValue, ST_NumBands

11.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — Définit la valeur du flag isnodata de la bande à TRUE.

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

Description

Définit la valeur du flag isnodata de la bande à TRUE. Si aucune bande n'est spécifiée, la bande 1 est utilisée. Cette fonction ne devrait être appelée que si le flag est considéré comme obsolète, i.e. si le résultat de ST_BandIsNoData est différent selon qu'on utilise TRUE comme dernier argument ou non

Disponibilité : 2.0.0

Exemples

This example creates a dummy table with one raster column. The raster has two bands and one pixel per band. In the first band, the NODATA value and pixel value are both 3. In the second band, the NODATA value is 13 and the pixel value is 4.

Code

```
create table dummy_rast (rid integer, rast raster);

insert into dummy_rast values(
1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected false
select ST_BandIsNoData(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true
```

The `isnodata` flag is dirty, so set it to true.

Code

```
update dummy_rast set rast = ST_SetBandIsNoData(rast, 1) where rid = 1;


select ST_BandIsNoData(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

Voir aussi

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_BandIsNoData

11.8.3 ST_SetBandPath

ST_SetBandPath — Met à jour le chemin externe et le numéro de bande d'une bande out-db

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

Description

Met à jour le chemin externe et le numéro de bande d'une bande externe out-db.



Note
Si `force` est true, aucun test n'est effectué pour garantir la compatibilité (e.g. alignement, support des pixels) entre le fichier raster externe et le raster PostGIS. Ce mode est prévu pour les changements sur le système de fichiers où réside le raster externe.

Disponibilité : 2.5.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_AddBand(
        NULL::raster,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif',
        NULL::int[]
    ) AS rast
)
SELECT
    1 AS query,
    bandnum,
    regexp_replace(path, '^.*/', '') AS path,
    outdbbandnum
FROM ST_BandMetaData(
    (SELECT rast FROM foo),
    ARRAY[1, 3, 2]::int[]
) AS metadata
UNION ALL
SELECT
    2, bandnum,
    regexp_replace(path, '^.*/', ''),
    outdbbandnum
FROM ST_BandMetaData((
        SELECT
            ST_SetBandPath(rast,
                2,
                '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif' ↩
    ,
                1
```

```
            ) AS rast
        FROM foo
    ),
    ARRAY[1, 3, 2]::int[]
) AS metadata
ORDER BY 1, 2;
```

In the second result set, band 2 now points at `Projected2.tif` and uses out-db band 1.

Output

```
query | bandnum | path           | outdbbandnum
------+---------+----------------+--------------
    1 |       1 | Projected.tif  |            1
    1 |       2 | Projected.tif  |            2
    1 |       3 | Projected.tif  |            3
    2 |       1 | Projected.tif  |            1
    2 |       2 | Projected2.tif |            1
    2 |       3 | Projected.tif  |            3
```

Voir aussi

ST_BandMetaData, ST_SetBandIndex

11.8.4 ST_SetBandIndex

ST_SetBandIndex — Met à jour le numéro de bande externe d'une bande out-db

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Description

Met à jour le numéro de bande externe d'une bande out-db. Cette opération n'affecte pas le fichier raster externe associé à la bande out-db



Note
Si `force` est true, aucun test n'est effectué pour assurer la compatibilité (e.g., l'alignement, le support des pixels) entre le fichier raster externe et le raster PostGIS. Ce mode est destiné aux cas où les bandes sont déplacées dans le fichier raster externe.



Note
En interne, cette méthode remplacement la bande du raster PostGIS à l'index `band` avec la nouvelle bande, au lieu de mettre à jour les informations de chemin existantes.

Disponibilité : 2.5.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_AddBand(
        NULL::raster,
        '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif',
        NULL::int[]
    ) AS rast
)
SELECT
    1 AS query,
    bandnum,
    regexp_replace(path, '^.*/', '') AS path,
    outdbbandnum
FROM ST_BandMetaData(
    (SELECT rast FROM foo),
    ARRAY[1, 3, 2]::int[]
) AS metadata
UNION ALL
SELECT
    2, bandnum,
    regexp_replace(path, '^.*/', ''),
    outdbbandnum
FROM ST_BandMetaData((
        SELECT
            ST_SetBandIndex(rast,
                2,
                1
            ) AS rast
        FROM foo
    ),
    ARRAY[1, 3, 2]::int[]
) AS metadata
ORDER BY 1, 2;
```

In the second result set, band 2 still points at the same file but now reads external band 1.

Output

```
query | bandnum | path          | outdbbandnum
------+---------+---------------+-------------
    1 |       1 | Projected.tif |            1
    1 |       2 | Projected.tif |            2
    1 |       3 | Projected.tif |            3
    2 |       1 | Projected.tif |            1
    2 |       2 | Projected.tif |            1
    2 |       3 | Projected.tif |            3
```

Voir aussi

ST_BandMetaData, ST_SetBandPath

11.9 Statistiques et analyses des bandes raster

11.9.1 ST_Count

ST_Count — Renvoie le nombre de pixels dans une bande donnée d'un raster ou d'une couverture raster. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels dont la valeur est différente de la valeur nodata seront comptés.

Synopsis

bigint **ST_Count**(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint **ST_Count**(raster rast, boolean exclude_nodata_value);

Description

Renvoie le nombre de pixels dans une bande donnée d'un raster ou d'une couverture raster. Si aucune bande `nband` n'est spécifiée, la valeur par défaut est 1.



Note
Si `exclude_nodata_value` est true, seuls les pixels dont la valeur est différente de la valeur `nodata` du raster seront comptés. Définir `exclude_nodata_value` à false pour compter tous les pixels

Changement : 3.1.0 - Suppression des variantes ST_Count(rastertable, rastercolumn, ...). Utiliser ST_CountAgg à la place.

Disponibilité : 2.0.0

Exemples

Example will count all pixels not 249 and one will count all pixels.

Code

```
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast, 249)) As exclude_nodata,
        ST_Count(ST_SetBandNoDataValue(rast, 249), false) As include_nodata
    FROM dummy_rast WHERE rid=2;
```

Output

```
rid | exclude_nodata | include_nodata
-----+----------------+---------------
   2 |             23 |             25
```

Voir aussi

ST_CountAgg, ST_SummaryStats, ST_SetBandNoDataValue

11.9.2 ST_CountAgg

ST_CountAgg — Agrégat. Renvoie le nombre de pixels dans une bande donnée d'un ensemble de rasters. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels différents de la valeur NODATA seront comptés.

Synopsis

bigint **ST_CountAgg**(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
bigint **ST_CountAgg**(raster rast, integer nband, boolean exclude_nodata_value);
bigint **ST_CountAgg**(raster rast, boolean exclude_nodata_value);

Description

Renvoie le nombre de pixels dans une bande donnée d'un ensemble de rasters. Si aucune bande `nband` n'est spécifiée, la valeur par défaut est 1.

Si `exclude_nodata_value` est true, seuls les pixels dont la valeur est différente de la valeur `nodata` du raster seront comptés. Définir `exclude_nodata_value` à false pour compter tous les pixels

Par défaut, tous les pixels sont échantillonnés. Pour obtenir une réponse plus rapide, définir `sample_percent` à une valeur comprise entre zéro (0) et un (1)

Disponibilité : 2.2.0

Exemples

Code

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(10, 10, 10, ↩
     10, 2, 2, 0, 0, 0),
                        1, '64BF', 0, 0
                    ),
                    1, 1, 1, -10
                ),
                1, 5, 4, 0
            ),
            1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    ST_CountAgg(rast, 1, TRUE) AS non_nodata_pixels
FROM foo;
```

Output

```
non_nodata_pixels
------------------
                20
(1 row)
```

Voir aussi

ST_Count, ST_SummaryStats, ST_SetBandNoDataValue

11.9.3 ST_Histogram

ST_Histogram — Retourne un ensemble d'enregistrements résumant une distribution de données raster ou de couverture raster, dans des classes distinctes. Le nombre de classes est calculé automatiquement s'il n'est pas spécifié.

Synopsis

SETOF record **ST_Histogram**(raster rast, integer nband=1, boolean exclude_nodata_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, boolean right);

Description

Retourne un ensemble d'enregistrements composés de min, max, count, percent pour une bande raster donnée pour chaque classe. Si aucune bande `nband` n'est spécifiée, la bande 1 est utilisée.



Note
Par défaut, seules les valeurs de pixels différentes de la valeur `nodata` sont prises en compte. Définir `exclude_nodata_value` à false pour obtenir le comptage de tous les pixels.

`width` width : un tableau indiquant la largeur de chaque classe. Si le nombre de classes est supérieur au nombre de largeurs, les largeurs sont répétées.

Exemple : 9 classes, les largeurs étant [a, b, c], le résultat sera [a, b, c, a, b, c, a, b, c]

`bins` Nombre de classes -- c'est le nombre d'enregistrements retournés par la fonction si spécifié. Si non spécifié, le nombre de classes est calculé automatiquement.

`right` calcule l'histogramme à partir de la droite plutôt que de la gauche (par défaut). Cela modifie les critères d'évaluation d'une valeur x de [a, b) à (a, b]

Changement : 3.1.0 Suppression de la variante ST_Histogram(table_name, column_name).

Disponibilité : 2.0.0

Exemples

Single raster tile - compute histograms for bands 1, 2, 3 and autocompute bins.

Code

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1, 3) As band
     WHERE rid=2) As foo;
```

Output

```
band |  min  |  max  | count | percent
------+-------+-------+-------+---------
   1 |   249 |   250 |     2 |    0.08
   1 |   250 |   251 |     2 |    0.08
   1 |   251 |   252 |     1 |    0.04
   1 |   252 |   253 |     2 |    0.08
   1 |   253 |   254 |    18 |    0.72
   2 |    78 | 113.2 |    11 |    0.44
   2 | 113.2 | 148.4 |     4 |    0.16
   2 | 148.4 | 183.6 |     4 |    0.16
   2 | 183.6 | 218.8 |     1 |    0.04
   2 | 218.8 |   254 |     5 |     0.2
   3 |    62 | 100.4 |    11 |    0.44
   3 | 100.4 | 138.8 |     5 |     0.2
   3 | 138.8 | 177.2 |     4 |    0.16
   3 | 177.2 | 215.6 |     1 |    0.04
   3 | 215.6 |   254 |     4 |    0.16
```

Just band 2 but for 6 bins.

Code

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2, 6) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;
```

Output

```
min     |    max     | count | percent
------------+------------+-------+---------
         78 | 107.333333 |     9 |    0.36
 107.333333 | 136.666667 |     6 |    0.24
 136.666667 |        166 |     0 |       0
        166 | 195.333333 |     4 |    0.16
 195.333333 | 224.666667 |     1 |    0.04
 224.666667 |        254 |     5 |     0.2
(6 rows)
```

Same as previous but we explicitly control the pixel value range of each bin.

Code

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2, 6, ARRAY[0.5, 1, 4, 100, 5]) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;
```

Output

```
min  |  max  | count | percent
-------+-------+-------+----------
    78 |  78.5 |     1 |     0.08
  78.5 |  79.5 |     1 |     0.04
  79.5 |  83.5 |     0 |        0
  83.5 | 183.5 |    17 |   0.0068
 183.5 | 188.5 |     0 |        0
 188.5 |   254 |     6 | 0.003664
(6 rows)
```

Voir aussi

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg

11.9.4 ST_Quantile

ST_Quantile — Calcule les quantiles d'un raster ou d'une couverture raster, dans le contexte de l'échantillon ou de la population. Ainsi, une valeur peut être examinée pour se situer au percentile de 25%, 50% ou 75% du raster.

Synopsis

SETOF record **ST_Quantile**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record **ST_Quantile**(raster rast, double precision[] quantiles);
SETOF record **ST_Quantile**(raster rast, integer nband, double precision[] quantiles);
double precision **ST_Quantile**(raster rast, double precision quantile);
double precision **ST_Quantile**(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);

Description

Calcule les quantiles d'un raster ou d'une couverture raster, dans le contexte de l'échantillon ou de la population. Ainsi, une valeur peut être examinée pour se situer au percentile de 25%, 50% ou 75% du raster.



Note
Si `exclude_nodata_value` est false, les pixels de valeur nodata seront également pris en compte.

Changement : 3.1.0 Suppression de la variante ST_Quantile(table_name, column_name).

Disponibilité : 2.0.0

Exemples

Example will consider only pixels of band 1 that are not 249 and in named quantiles.

Code

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25, 0.75]) As pvq
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvq).quantile;
```

Output

```
quantile | value
----------+-------
    0.25 |   253
    0.75 |   254
```

Code

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;
```

Output

```
254
```

Real live example. Quantile of all pixels in band 2 intersecting a geometry.

Code

```
SELECT rid, (ST_Quantile(rast, 2)).* As pvc
    FROM o_4_boston
        WHERE ST_Intersects(
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↩
    892151,224486 892151))', 26986)
            )
ORDER BY value, quantile, rid
;
```

Output

```
rid | quantile | value
-----+----------+-------
   1 |        0 |     0
   2 |        0 |     0
  14 |        0 |     1
  15 |        0 |     2
  14 |     0.25 |    37
   1 |     0.25 |    42
  15 |     0.25 |    47
   2 |     0.25 |    50
  14 |      0.5 |    56
   1 |      0.5 |    64
  15 |      0.5 |    66
   2 |      0.5 |    77
  14 |     0.75 |    81
  15 |     0.75 |    87
   1 |     0.75 |    94
   2 |     0.75 |   106
  14 |        1 |   199
   1 |        1 |   244
   2 |        1 |   255
  15 |        1 |   255
```

Voir aussi

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg, ST_SetBandNoDataValue

11.9.5 ST_SummaryStats

ST_SummaryStats — Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.

Synopsis

summarystats **ST_SummaryStats**(raster rast, boolean exclude_nodata_value);
summarystats **ST_SummaryStats**(raster rast, integer nband, boolean exclude_nodata_value);

Description

Retourne un résumé statistique summarystats composé de count, sum, mean, stddev, min, max pour une bande raster ou une couverture raster spécifiée. Si aucune bande `nband` n'est spécifiée, la bande 1 est utilisée.



Note
Par défaut, seules les valeurs de pixels différentes de la valeur `nodata` sont prises en compte. Définir `exclude_nodata_value` à false pour obtenir le comptage de tous les pixels.



Note
Par défaut, tous les pixels sont échantillonnés. Pour obtenir une réponse plus rapide, définir `sample_percent` à une valeur inférieure à 1

Changement : 3.1.0 Suppression des variantes ST_SummaryStats(rastertable, rastercolumn, ...). Utiliser ST_SummaryStatsAgg à la place.

Disponibilité : 2.0.0

Exemples

Single raster tile.

Code

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1, 3) As band
     WHERE rid=2) As foo;
```

Output

```
rid | band | count | sum  |    mean    |  stddev   | min | max
----+------+-------+------+------------+-----------+-----+-----
  2 |    1 |    23 | 5821 | 253.086957 |  1.248061 | 250 | 254
  2 |    2 |    25 | 3682 |     147.28 | 59.862188 |  78 | 254
  2 |    3 |    25 | 3290 |      131.6 | 61.647384 |  62 | 254
```

Summarize pixels that intersect buildings of interest.

This example selects the target buildings, clips band 2 of the raster tiles to each building boundary, computes statistics for each clipped region, and then summarizes those statistics by building. It took 574ms on PostGIS Windows 64-bit with all of Boston Buildings and aerial Tiles (tiles each 150x150 pixels ~ 134,000 tiles), ~102,000 building records.

Code

```
WITH
-- our features of interest
feat AS (
    SELECT gid AS building_id, geom_26986 AS geom
    FROM buildings AS b
    WHERE gid IN (100, 103, 150)
),
-- clip band 2 of raster tiles to boundaries of builds
-- then get stats for these clipped regions
b_stats AS (
    SELECT building_id, (stats).*
    FROM (
        SELECT
            building_id,
            ST_SummaryStats(ST_Clip(rast, 2, geom)) AS stats
        FROM aerials.boston
        INNER JOIN feat
            ON ST_Intersects(feat.geom, rast)
    ) AS foo
)
-- finally summarize stats
SELECT
    building_id,
    SUM(count) AS num_pixels,
    MIN(min) AS min_pval,
    MAX(max) AS max_pval,
    SUM(mean * count) / SUM(count) AS avg_pval
FROM b_stats
WHERE count
> 0
GROUP BY building_id
ORDER BY building_id;
```

Output

```
building_id | num_pixels | min_pval | max_pval |     avg_pval
------------+------------+----------+----------+------------------
        100 |       1090 |        1 |      255 | 61.0697247706422
        103 |        655 |        7 |      182 | 70.5038167938931
        150 |        895 |        2 |      252 | 185.642458100559
```

Raster coverage.

This example returns statistics for each band.

Code

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston', 'rast', band) As stats
    FROM generate_series(1, 3) As band) As foo;
```

Output

```
band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 |   0 | 255
    2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 |   0 | 255
    3 | 8450000 | 575943 |  74.682739408284 | 44.2143885481407 |   0 | 255
```

For a table, sampling less than 100% can improve speed. Here the sampling is set to 25%.

Code

```
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston', 'rast', band, true, 0.25) As stats
    FROM generate_series(1, 3) As band) As foo;
```

Output

```
band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 |   0 | 255
    2 | 2112500 | 174571 |  81.448503668639 | 44.2252623171821 |   0 | 255
    3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 |   0 | 255
```

Voir aussi

summarystats, ST_SummaryStatsAgg, ST_Count, ST_Clip

11.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.

Synopsis

summarystats **ST_SummaryStatsAgg**(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats **ST_SummaryStatsAgg**(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats **ST_SummaryStatsAgg**(raster rast, integer nband, boolean exclude_nodata_value);

Description

Retourne un résumé statistique summarystats composé de count, sum, mean, stddev, min, max pour une bande raster ou une couverture raster spécifiée. Si aucune bande `nband` n'est spécifiée, la bande 1 est utilisée.



Note
Par défaut, seules les valeurs de pixels différentes de la valeur `NODATA` sont prises en compte. Définir `exclude_nodata_value` à false pour obtenir le comptage de tous les pixels.



Note
Par défaut, tous les pixels sont échantillonnés. Pour obtenir une réponse plus rapide, définir `sample_percent` à une valeur comprise entre zéro (0) et un (1)

Disponibilité : 2.2.0

Exemples

Code

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(10, 10, 10, ↩
     10, 2, 2, 0, 0, 0),
                        1, '64BF', 0, 0
                    ),
                    1, 1, 1, -10
                ),
                1, 5, 4, 0
            ),
            1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    (stats).count AS count,
    round((stats).sum::numeric, 3) AS sum,
    round((stats).mean::numeric, 3) AS mean,
    round((stats).stddev::numeric, 3) AS stddev,
    round((stats).min::numeric, 3) AS min,
    round((stats).max::numeric, 3) AS max
FROM (
    SELECT
        ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
    FROM foo
) bar;
```

Output

```
count |   sum    |  mean  | stddev |   min   |  max
------+----------+--------+--------+---------+-------
    20 | -68.584 | -3.429 |  6.571 | -10.000 | 3.142
(1 row)
```

Voir aussi

summarystats, ST_SummaryStats, ST_Count, ST_Clip

11.9.7 ST_ValueCount

ST_ValueCount — Retourne un ensemble d'enregistrements contenant une valeur de pixels et le nombre de pixels ayant cette valeur dans la bande du raster spécifié (ou de la couverture raster). Si aucune bande n'est spécifiée, la bande 1 est utilisée. Par défaut, les pixels de valeur nodata ne sont pas comptés, et toutes les autres valeurs sont retournées, avec leur valeur arrondies à l'entier le plus proche.

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint**ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

Description

Retourne un ensemble d'enregistrements contenant les colonnes `value` `count`, correspondant à la valeur de pixel dans la bande et le nombre de pixels ayant cette valeur dans la bande du raster spécifié (ou de la couverture raster).

Si aucune bande `nband` n'est spécifiée, la bande 1 est utilisée. Si aucune valeur `searchvalues` n'est spécifiée, retourne toutes les valeurs de pixel trouvées dans le raster ou dans la couverture raster. Si une valeur searchvalue est spécifiée, retourne un entier au lieu d'enregistrements, correspondant au nombre de pixels ayant la valeur recherchée



Note
Si `exclude_nodata_value` est false, les pixels de valeur nodata seront également pris en compte.

Disponibilité : 2.0.0

Exemples

Example will count only pixels of band 1 that are not 249.

Code

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast, 249) WHERE rid=2;

SELECT (pvc).*
```

```
FROM (SELECT ST_ValueCount(rast) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
```

Output

```
value | count
-------+-------
   250 |     2
   251 |     1
   252 |     2
   253 |     6
   254 |    12
```

Example will coount all pixels of band 1 including 249.

Code

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 1, false) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
```

Output

```
value | count
-------+-------
   249 |     2
   250 |     2
   251 |     1
   252 |     2
   253 |     6
   254 |    12
```

Example will count only non-nodata value pixels of band 2.

Code

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast, 2) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
```

Output

```
value | count
-------+-------
    78 |     1
    79 |     1
    88 |     1
    89 |     1
    96 |     1
    97 |     1
    98 |     1
    99 |     2
   112 |     2
:
```

Real live example. Count all the pixels in an aerial raster tile band 2 intersecting a geometry. And return only the pixel band values that have a count > 500.

Code

```
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast, 2) As pvc
    FROM o_4_boston
        WHERE ST_Intersects(
```

```
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↩
    892151,224486 892151))', 26986)
             )
        ) As foo
    GROUP BY (pvc).value
    HAVING SUM((pvc).count) > 500
    ORDER BY (pvc).value;
```

Output

```
value | total
------+-----
    51 | 502
    54 | 521
```

Just return count of pixels in each raster tile that have value of 100 of tiles that intersect a specific geometry.

Code

```
SELECT rid, ST_ValueCount(rast, 2, 100) As count
    FROM o_4_boston
        WHERE ST_Intersects(
            rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706 ↩
    892151,224486 892151))', 26986)
             ) ;
```

Output

```
rid | count
----+-------
  1 |    56
  2 |    95
 14 |    37
 15 |    64
```

Voir aussi

ST_Count, ST_SetBandNoDataValue

11.10 Import de raster

11.10.1 ST_RastFromWKB

ST_RastFromWKB — Retourne un raster à partir d'un raster Well-Known Binary (WKB).

Synopsis

raster **ST_RastFromWKB**(bytea wkb);

Description

Construit un raster PostGIS à partir d'une représentation raster Well-Known Binary (WKB).

Disponibilité : 2.5.0

Exemples

Code

```
SELECT (ST_MetaData(ST_RastFromWKB('\001\000\000\000\000\000\000\000\000\000\000\000@ ↩
    \000\000\000\000\000\000\010@ ↩
    \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\00
    '::bytea
    )
)).* AS metadata;
```

Output

```
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----------+------------+-------+--------+--------+--------+-------+-------+------+----------
       0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |        0
```

Voir aussi

ST_MetaData, ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

11.10.2 ST_RastFromHexWKB

ST_RastFromHexWKB — Retourne un raster à partir d'un raster Well-Known Binary (WKB) en hexadécimal.

Synopsis

raster **ST_RastFromHexWKB**(text wkb);

Description

Construit un raster PostGIS à partir d'une représentation hexadécimale raster Well-Known Binary (WKB).

Disponibilité : 2.5.0

Exemples

Code

```
SELECT (ST_MetaData(ST_RastFromHexWKB(' ↩
    0100000000000000000000004000000000000008400000000000000 ↩
    E03F000000000000E03F0000000000000000000000000000000A0000000A001400'
    )
)).* AS metadata;
```

Output

```
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands
-----------+------------+-------+--------+--------+--------+-------+-------+------+----------
       0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |        0
```

Voir aussi

ST_MetaData, ST_RastFromWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

11.11 Export de raster

11.11.1 ST_AsBinary/ST_AsWKB

ST_AsBinary/ST_AsWKB — Retourne la représentation Well-Known Binary (WKB) de ce raster.

Synopsis

bytea **ST_AsBinary**(raster rast, boolean outasin=FALSE);
bytea **ST_AsWKB**(raster rast, boolean outasin=FALSE);

Description

Retourne la représentation binaire du raster. Si `outasin` est TRUE, les bandes out-db sont traitées comme in-db. Voir raster/doc/RFC2-WellKnownBinaryFormat situé dans le dossier source de PostGIS pour plus de détails sur la représentation.

Cette fonction est utile dans les curseurs binaires pour extraire des données de la base de données sans les convertir en une représentation sous forme de chaîne de caractères.



Note
Par défaut, la sortie WKB contient le chemin d'accès au fichier externe pour les bandes out-db. Si le client n'a pas accès au fichier raster sous-jacent d'une bande out-db, définir `outasin` à TRUE.

Amélioration : 2.1.0 Ajout de `outasin`

Amélioration : 2.5.0 Ajout de `ST_AsWKB`

Exemples

Code

```
SELECT ST_AsBinary(rast) AS rastbin FROM dummy_rast WHERE rid=1;
```

Output

```
\001\000\000\000\000\000\000\000\000\000\000@\000\000\000\000\000\000\010@ ↩
     \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\0
```

Voir aussi

ST_RastFromWKB, ST_AsHexWKB

11.11.2 ST_AsHexWKB

ST_AsHexWKB — Retourne la représentation hexadécimale Well-Known Binary (WKB) de ce raster.

Synopsis

bytea **ST_AsHexWKB**(raster rast, boolean outasin=FALSE);

Description

Retourne la représentation binaire en hexadécimal du raster. Si `outasin` est TRUE, les bandes out-db sont traitées comme in-db. Voir raster/doc/RFC2-WellKnownBinaryFormat situé dans le dossier source de PostGIS pour les détails de la représentation.



Note
Par défaut, la sortie Hex WKB contient le chemin d'accès au fichier externe pour les bandes out-db. Si le client n'a pas accès au fichier raster sous-jacent de la bande out-db, définir `outasin` à TRUE.

Disponibilité : 2.5.0

Exemples

Code

```
SELECT ST_AsHexWKB(rast) AS rastbin FROM dummy_rast WHERE rid=1;
```

Output

```
01000000000000000000000040000000000000084000000000000000 ↩
    E03F000000000000E03F000000000000000000000000000000000A0000000A001400
```

Voir aussi

ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB

11.11.3 ST_AsGDALRaster

ST_AsGDALRaster — Retourne la tuile raster dans le format GDAL raster spécifié. Les formats raster disponibles sont ceux supportés par votre bibliothèque compilée. Utilisez ST_GDALDrivers() pour obtenir la liste des formats supportés par votre bibliothèque.

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Description

Retourne la tuile raster dans le format spécifié. Les paramètres sont détaillés ci-dessous :

— `format` format de sortie. Cela dépend des drivers compilés dans votre bibliothèque libgdal. Les formats généralement disponibles sont 'JPEG', 'GTIff', 'PNG'. Utilisez ST_GDALDrivers pour obtenir une liste des formats pris en charge par votre bibliothèque.

— `options` tableau textuel d'options GDAL. Les options valides dépendent du format. Pour plus de détails, voir GDAL Raster format options.

— `srid` the spatial reference identifier to embed in the output raster

Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0.

Exemples

Export multiple raster tiles as a single JPEG image with 50 percent quality.

Code

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

One way to export raster into another format is using PostgreSQL large object export functions. The following repeats the JPEG example and exports the result. Note that this requires superuser access because it uses server-side large object functions. It will also export to path on server network. If you need export locally, use the psql equivalent lo_ functions which export to the local file system instead of the server file system.

Code

```
DROP TABLE IF EXISTS tmp_out ;

CREATE TABLE tmp_out AS
SELECT lo_from_bytea(
       0,
       ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50'])
        ) AS loid
  FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
   FROM tmp_out;

SELECT lo_unlink(loid)
  FROM tmp_out;
```

Export a raster as a GeoTIFF using the default creation options.

Code

```
SELECT ST_AsGDALRaster(rast, 'GTiff') AS rasttiff
FROM dummy_rast WHERE rid = 2;
```

Export a GeoTIFF using JPEG compression at 90 percent quality and embed SRID 4269.

Code

```
SELECT ST_AsGDALRaster(
  rast,
  'GTiff',
  ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
  4269) AS rasttiff
FROM dummy_rast WHERE rid = 2;
```

Voir aussi

Section 10.4, ST_GDALDrivers, ST_SRID

11.11.4 ST_AsJPEG

ST_AsJPEG — Retourne les bandes sélectionnées du raster sous la forme d'une image JPEG (sous forme de tableau d'octets). Si aucune bande n'est spécifiée, et que le raster a 1 ou plus de 3 bandes, seule la première bande est utilisée. Si le raster a exactement 3 bandes, les 3 bandes sont utilisées et mappées en RGB.

Synopsis

bytea **ST_AsJPEG**(raster rast, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer nband, integer quality);
bytea **ST_AsJPEG**(raster rast, integer nband, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer[] nbands, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer[] nbands, integer quality);

Description

Retourne les bandes sélectionnées du raster sous la forme d'une seule image JPEG (Joint Photographic Exports Group Image). Utilisez ST_AsGDALRaster si vous avez besoin d'exporter vers des types de raster moins courants. Si aucune bande n'est spécifiée, et que le raster a 1 ou plus de 3 bandes, seule la première bande est utilisée. Si le raster a exactement 3 bandes, les 3 bandes sont utilisées. Il existe de nombreuses variantes de la fonction avec de nombreuses options. Celles-ci sont détaillées ci-dessous :

— `nband` pour exporter une seule bande.
— `nbands` spécifie un tableau des bandes à exporter (à noter que JPEG ne supporte que 3 bandes aux maximum). L'ordre des bandes est RGB. Par exemple, ARRAY[3,2,1] signifie que la bande 3 est la rouge, la bande 2 est la verte et la bande 1 est la bleue
— `quality` nombre de 0 à 100. Plus le chiffre est élevé, plus l'image est nette.
— `options` tableau textuel d'options GDAL, parmi les options définies pour JPEG (voir create_options pour JPEG de ST_GDALDrive
Pour JPEG, les options disponibles sont `PROGRESSIVE` ON ou OFF et `QUALITY` une valeur de 0 à 100 et par défaut 75. Voir GDAL Raster format options pour plus de détails.

Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0.

Exemples : Export

Output the first three bands at 75 percent quality.

Code

```
SELECT ST_AsJPEG(rast) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output only the first band at 90 percent quality.

Code

```
SELECT ST_AsJPEG(rast, 1, 90) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Output the first three bands, mapping band 2 to red, band 1 to green, and band 3 to blue, using progressive encoding at 90 percent quality.

Code

```
SELECT ST_AsJPEG(rast, ARRAY[2, 1, 3], ARRAY['QUALITY=90', 'PROGRESSIVE=ON']) As rastjpg
FROM dummy_rast WHERE rid=2;
```

Voir aussi

Section 10.4, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsTIFF

11.11.5 ST_AsPNG

ST_AsPNG — Retourne les bandes sélectionnées du raster sous la forme d'une image PNG (sous forme de tableau d'octets). Si aucune bande n'est spécifiée et que le raster a 1, 3 ou 4 bandes, toutes les bandes sont utilisées. Si aucune bande n'est spécifiée et que le raster a 2 ou plus de 4 bandes, seule la bande 1 est utilisée. Les bandes sont mappées en RGB ou RGBA.

Synopsis

bytea **ST_AsPNG**(raster rast, text[] options=NULL);
bytea **ST_AsPNG**(raster rast, integer nband, integer compression);
bytea **ST_AsPNG**(raster rast, integer nband, text[] options=NULL);
bytea **ST_AsPNG**(raster rast, integer[] nbands, integer compression);
bytea **ST_AsPNG**(raster rast, integer[] nbands, text[] options=NULL);

Description

Retourne les bandes sélectionnées du raster sous la forme d'une seule PNG (image graphique portable). Utilisez ST_AsGDALRaster si vous avez besoin d'exporter vers des types de raster moins courants. Si aucune bande n'est spécifiée, les 3 premières bandes sont exportées. Il existe de nombreuses variantes de cette fonction avec de nombreuses options. Si aucun `srid` n'est spécifié, c'est le srid du raster qui est utilisé. Ces options sont détaillées ci-dessous :

— `nband` pour exporter une seule bande.
— `nbands` spécifie un tableau des bandes à exporter (à noter que PNG ne supporte que 4 bandes aux maximum). L'ordre des bandes est RGBA. Par exemple, ARRAY[3,2,1] signifie que la bande 3 est la rouge, la bande 2 est la verte et la bande 1 est la bleue
— `compression` nombre de 1 à 9. Plus le chiffre est élevé, plus la compression est importante.
— `options` tableau textuel d'options GDAL, parmi les options définies pour PNG (voir create_options pour PNG de ST_GDALDrivers Pour PNG, seule l'option ZLEVEL est valide (temps à consacrer à la compression -- par défaut 6), par exemple ARRAY['ZLEVEL=9']. WORLDFILE n'est pas autorisé car la fonction devrait produire deux sorties. Voir GDAL Raster format options pour plus de détails.

Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0.

Exemples

Export the first 3 bands and map band 3 to red, band 1 to green, and band 2 to blue.

Code

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;
SELECT ST_AsPNG(rast, ARRAY[3, 1, 2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

Voir aussi

ST_AsGDALRaster, ST_ColorMap, ST_GDALDrivers, Section 10.4

11.11.6 ST_AsTIFF

ST_AsTIFF — Retourne les bandes sélectionnées du raster sous la forme d'une seule image TIFF (sous forme de tableau d'octets). Si aucune bande n'est spécifiée ou si l'une des bandes spécifiées n'existe pas dans le raster, toutes les bandes sont utilisées.

Synopsis

bytea **ST_AsTIFF**(raster rast, text[] options='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text[] options, integer srid=sameassource);

Description

Retourne les bandes sélectionnées du raster sous la forme d'un fichier TIFF (Tagged Image File Format). Si aucune bande n'est spécifiée, toutes les bandes seront utilisées. Cette fonction est une enveloppe autour de ST_AsGDALRaster. Utilisez ST_AsGDALRaster si vous avez besoin d'exporter vers des types de raster moins courants. Il existe de nombreuses variantes de la fonction avec de nombreuses options. Si aucun texte SRS de référence spatiale n'est présent, la référence spatiale du raster est utilisée. Ces options sont détaillées ci-dessous :

— `nbands` spécifie un tableau des bandes à exporter (à noter que PNG ne supporte que 3 bandes aux maximum). L'ordre des bandes est RGB. Par exemple, ARRAY[3,2,1] signifie que la bande 3 est la rouge, la bande 2 est la verte et la bande 1 est la bleue
— `compression` Expression de compression -- JPEG90 (ou un autre pourcentage), LZW, JPEG, DEFLATE9.
— `options` tableau textuel d'options GDAL, parmi les options définies pour GTiff (voir create_options pour GTiff de ST_GDALDrive Voir GDAL Raster format options pour plus de détails.
— `srid` srid du système de référence spatial du raster. Il est utilisé pour remplir les informations de géo-référence

Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0.

Exemple : Utiliser la compression jpeg à 90%

Code

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

Voir aussi

ST_GDALDrivers, ST_AsGDALRaster, ST_SRID

11.12 Traitement des données raster : algèbre cartographique

11.12.1 ST_Clip

ST_Clip — Renvoie le raster découpé par la géométrie d'entrée. Si le numéro de bande n'est pas spécifié, toutes les bandes sont traitées. Si `crop` n'est pas spécifié ou est TRUE, le raster de sortie est rogné. Si `touched` est défini sur TRUE, les pixels touchés sont inclus, sinon seulement si le centre du pixel est dans la géométrie, il est inclus.

Synopsis

raster **ST_Clip**(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, boolean crop, boolean touched=FALSE);
raster **ST_Clip**(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE, boolean touched=FALSE);
raster **ST_Clip**(raster rast, geometry geom, boolean crop, boolean touched=FALSE);

Description

Retourne un raster coupé par la géométrie d'entrée `geom`. Si l'index des bandes n'est pas spécifié, toutes les bandes sont traitées.

Les rasters résultant de ST_Clip doivent avoir une valeur de nodata assignée pour les zones découpées, une pour chaque bande. Si aucune valeur n'est fournie et que le raster d'entrée n'a pas de valeur de nodata définie, les valeurs de nodata du raster résultant sont fixées à ST_MinPossibleValue(ST_BandPixelType(rast, band)). Lorsque la taille du tableau nodata inférieure au nombre de bandes, la dernière valeur du tableau est utilisée pour les bandes restantes. Si la taille du tableau nodata est supérieure au nombre de bandes, les valeurs de nodata supplémentaires sont ignorées. Toutes les variantes acceptant un tableau de valeurs de nodata acceptent également une valeur unique qui sera assignée à chaque bande.

Si `crop` n'est pas spécifié ou si défini à true, le raster de sortie est recadré à l'intersection des extensions `geom`et `rast`. Si `crop` est défini à false, le nouveau raster a la même étendue que `rast`. Si `touched` est défini comme vrai, tous les pixels de `rast` qui intersectent la géométrie sont sélectionnés.



Note
Le comportement par défaut est touched=false, ce qui sélectionnera uniquement les pixels dont le centre est couvert par la géométrie.

Amélioration : 3.5.0 - ajout de l'argument touched.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Réécrit en C

Les exemples présentés ici utilisent les données aériennes du Massachusetts disponibles sur le site de MassGIS MassGIS Aerial Orthos.

Exemples

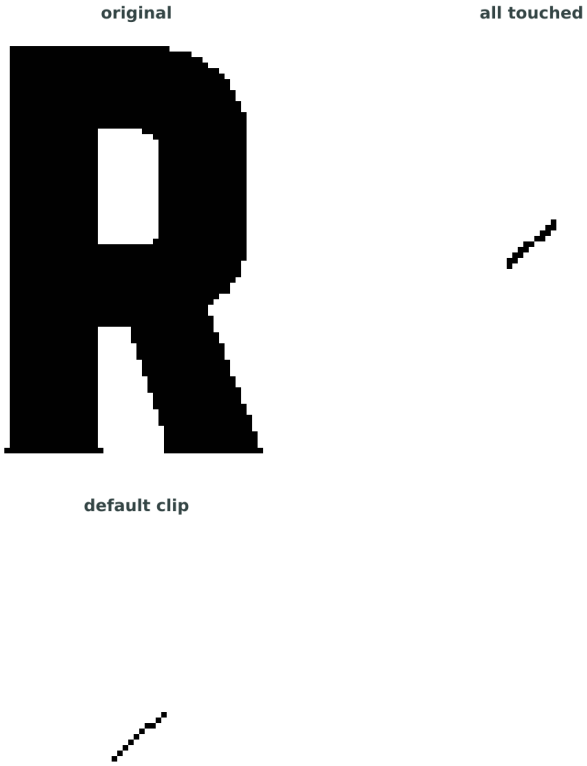
Comparing selecting all touched vs. not all touched.

Code

```
WITH source AS (
    SELECT rast, geom
    FROM ST_AsRaster(ST_Letters('R'), scalex =
> 1.0, scaley =
> -1.0) AS r(rast)
    INNER JOIN ST_GeomFromText('LINESTRING(0 1,5 6,10 10)') AS g(geom)
        ON ST_Intersects(r.rast, g.geom)
), clipped AS (
    SELECT rast,
        ST_Clip(rast, geom, touched =
> true) AS rast_touched,
        ST_Clip(rast, geom, touched =
> false) AS rast_not_touched
    FROM source
), variants AS (
    SELECT 'original' AS title, ST_Count(rast) AS pixel_count, rast AS rendered
    FROM clipped
    UNION ALL
    SELECT 'all touched', ST_Count(rast_touched), rast_touched
    FROM clipped
    UNION ALL
    SELECT 'default clip', ST_Count(rast_not_touched), rast_not_touched
    FROM clipped
)
SELECT title, pixel_count, ST_AsPNG(rendered) AS image
```

```
FROM variants
ORDER BY CASE title
    WHEN 'original' THEN 1
    WHEN 'all touched' THEN 2
    ELSE 3
END;
```

Figure



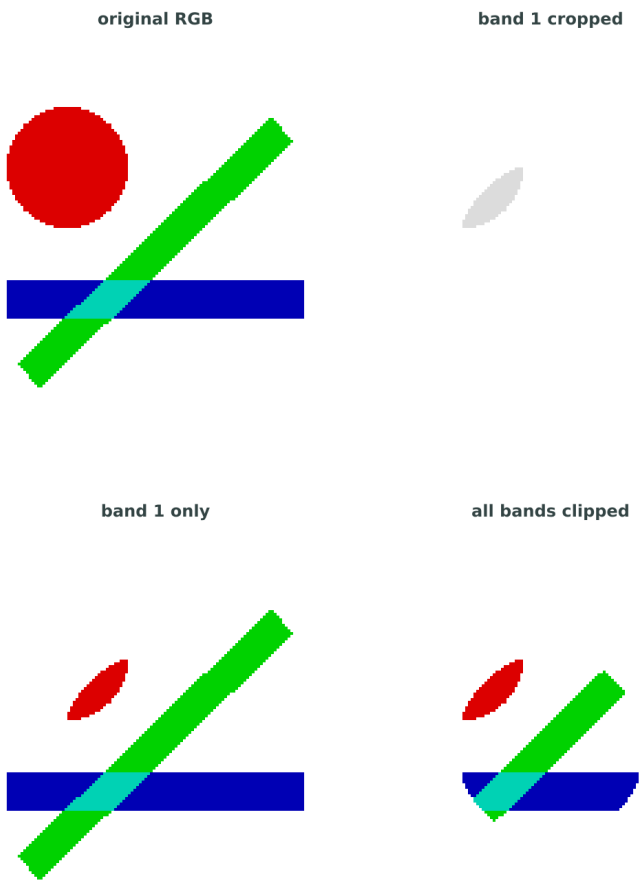
The following self-contained example uses three simple shapes as RGB bands. It compares cropping band 1, clipping band 1 without cropping and then restoring bands 2 and 3, and clipping every band without cropping.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(128, 128, 0, 128, 1, -1, 0, 0, 0),
                '8BUI'::text, 0::double precision, 0::double precision
            ),
            '8BUI'::text, 0::double precision, 0::double precision
        ),
        '8BUI'::text, 0::double precision, 0::double precision
    ) AS rast
), source AS (
```

```
    SELECT ST_SetValue(
        ST_SetValue(
            ST_SetValue(rast, 1, ST_Buffer('POINT(32 96)'::geometry, 22), 220),
            2, ST_Buffer('LINESTRING(18 20,110 110)'::geometry, 6,
                         'endcap=flat join=bevel'), 210
        ),
        3, ST_Buffer('LINESTRING(10 48,118 48)'::geometry, 7,
                     'endcap=flat'), 180
    ) AS rast,
    ST_Buffer('POINT(64 64)'::geometry, 34) AS clipper
    FROM canvas
), variants AS (
    SELECT 'original RGB' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'band 1 cropped', ST_Clip(rast, 1, clipper, true) FROM source
    UNION ALL
    SELECT
        'band 1 only',
        ST_AddBand(
            ST_Clip(rast, 1, clipper, false),
            ARRAY[ST_Band(rast, 2), ST_Band(rast, 3)]
        )
    FROM source
    UNION ALL
    SELECT 'all bands clipped', ST_Clip(rast, clipper, false) FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
    WHEN 'original RGB' THEN 1
    WHEN 'band 1 cropped' THEN 2
    WHEN 'band 1 only' THEN 3
    ELSE 4
END;
```

Figure



Crop, single-band clipping, and all-band clipping.



Source tile



Cropped



One-band source



One-band clip



All-band source



All-band clip

Voir aussi

ST_AddBand, ST_Count, ST_MapAlgebra (callback function version), ST_Intersection

11.12.2 ST_ColorMap

ST_ColorMap — Crée un nouveau raster comprenant jusqu'à quatre bandes 8BUI (niveaux de gris, RGB, RGBA) à partir du raster source et d'une bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

raster **ST_ColorMap**(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);

raster **ST_ColorMap**(raster rast, text colormap, text method=INTERPOLATE);

Description

Appliquer une palette de couleurs `colormap` à la bande `nband` du raster `rast`, pour obtenir un nouveau raster comprenant jusqu'à quatre bandes 8BUI. Le nombre de bandes 8BUI dans le nouveau raster est déterminé par le nombre de composantes de couleur définies dans `colormap`.

Si `nband` n'est pas spécifié, la bande 1 est utilisée.

`colormap` peut être un mot-clé d'une palette de couleurs prédéfinie, ou un ensemble de lignes définissant la valeur et les composants de couleur.

Valeurs prédéfinies disponibles pour `colormap` :

— `grayscale` ou `greyscale` pour un raster à 1 bande 8BUI en nuances de gris.
— `pseudocolor` pour un raster à 4 bandes 8BUI (RGBA) avec des couleurs allant du bleu au rouge en passant par le vert.
— `fire` pour un raster à 4 bandes 8BUI (RGBA) avec des couleurs allant du noir au rouge en passant par le jaune pâle.
— `bluered` pour un raster à 4 bandes 8BUI (RGBA) avec des couleurs allant du bleu au rouge en passant par le blanc pâle.

Vous pouvez passer un ensemble d'entrées (une par ligne) à `colormap` pour spécifier des palettes de couleurs personnalisées. Chaque entrée se compose généralement de cinq valeurs : la valeur du pixel et les composantes rouge, verte, bleue et alpha correspondantes (composantes de couleur comprises entre 0 et 255). Des valeurs en pourcentage peuvent être utilisées à la place des valeurs en pixels, où 0% et 100% sont les valeurs minimales et maximales trouvées dans la bande raster. Les valeurs peuvent être séparées par des virgules (','), des tabulations, des deux-points (':') et/ou des espaces. La valeur du pixel peut être fixée à *nv*, *null* ou *nodata* pour la valeur NODATA. Un exemple est fourni ci-dessous.

Code

```
5 0 0 0 255
4 100:50 55 255
1 150, 100 150 255
0% 255 255 255 255
nv 0 0 0 0
```

La syntaxe de `colormap` est similaire à celle du mode couleur du relief de GDAL gdaldem.

Valeurs disponibles pour `method` :

— `INTERPOLATE` pour utiliser l'interpolation linéaire afin de mélanger les couleurs entre les valeurs de pixels données
— `EXACT` pour correspondre strictement aux valeurs des pixels trouvés dans la palette de couleurs. Les pixels dont la valeur ne correspond pas à une entrée de la palette de couleurs seront mis à 0 0 0 0 (RGBA)
— `NEAREST` pour utiliser l'entrée de la palette de couleurs dont la valeur est la plus proche de la valeur du pixel



Note
Une excellente référence pour les palette de couleurs est ColorBrewer.



AVERTISSEMENT
Les bandes résultantes du nouveau raster n'auront pas de valeur NODATA définie. Utilisez ST_SetBandNoDataValue pour définir une valeur NODATA si nécessaire.

Disponibilité : 2.1.0

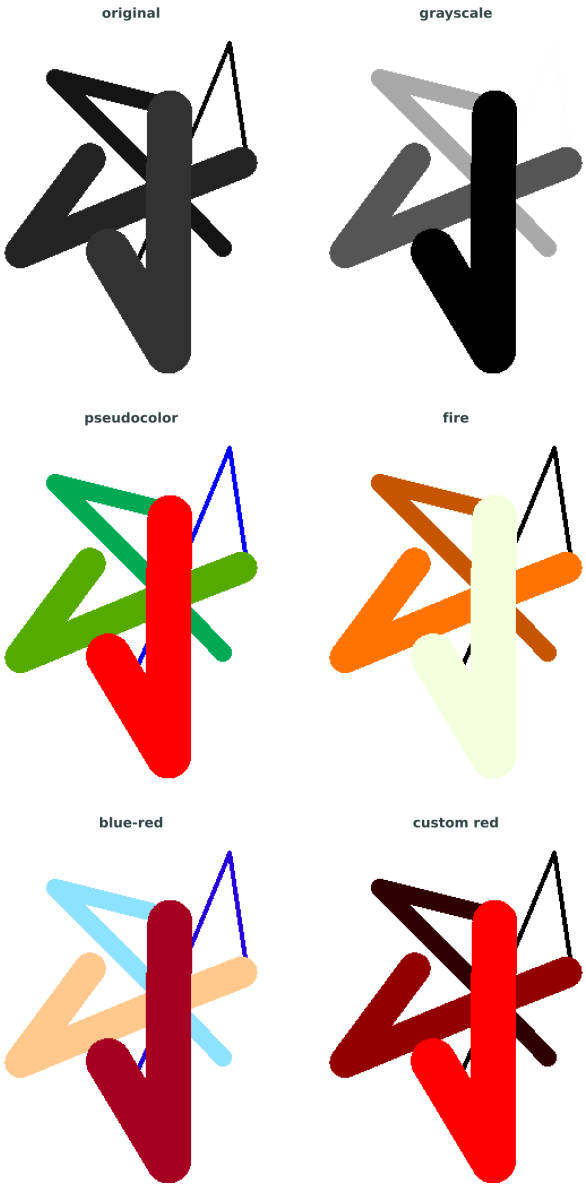
Exemples

This example builds a raster from rotated buffered lines and compares the original values with predefined and custom color maps.

Code

```
WITH ref AS (
    SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
), source AS (
    SELECT ST_Union(rast) AS rast
    FROM (
    SELECT
        ST_AsRaster(ST_Rotate(ST_Buffer('LINESTRING(0 2,50 50,150 150,125 50)'::geometry,
                    i*2
                ),
                pi() * i * 0.125, ST_Point(50, 50)
            ),
            ref.rast, '8BUI'::text, i * 5
        ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes
)
SELECT title, ST_AsPNG(rendered) AS image
FROM source
CROSS JOIN LATERAL (VALUES
    ('original', rast),
    ('grayscale', ST_ColorMap(rast, 1, 'greyscale')),
    ('pseudocolor', ST_ColorMap(rast, 1, 'pseudocolor')),
    ('fire', ST_ColorMap(rast, 1, 'fire')),
    ('blue-red', ST_ColorMap(rast, 1, 'bluered')),
    ('custom red', ST_ColorMap(
        rast,
        1,
        E'100% 255 0 0 255\n80% 160 0 0 255\n50% 130 0 0 255\n' ||
        E'30% 30 0 0 255\n20% 60 0 0 255\n0% 0 0 0 255\n' ||
        E'nv 255 255 255 0'
    ))
) AS variants(title, rendered);
```

Figure



Voir aussi

ST_AsPNG, ST_AsRaster ST_MapAlgebra (callback function version), ST_Grayscale ST_NumBands, ST_Reclass, ST_SetBandNoDat ST_Union

11.12.3 ST_Grayscale

ST_Grayscale — Crée un nouveau raster à 1 bande 8BUI à partir du raster source et des bandes spécifiées représentant les composantes rouge, vert et bleu

Synopsis

(1) raster **ST_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extenttype=INTERSECTION);

(2) raster **ST_Grayscale**(rastbandarg[] rastbandargset, text extenttype=INTERSECTION);

Description

Crée un raster à 1 bande 8BUI à partir de trois bandes d'entrée (provenant d'un ou plusieurs rasters). Toute bande d'entrée dont le type de pixel n'est pas 8BUI sera reclassée avec ST_Reclass.



Note
Cette fonction est différente de ST_ColorMap avec le mot-clé `grayscale`, car ST_ColorMap n'opère que sur une seule bande alors que cette fonction s'attend à trois bandes pour RGB. Cette fonction applique l'équation suivante pour convertir les composantes RGB en niveaux de gris : 0,2989 * ROUGE + 0,5870 * VERT + 0,1140 * BLEU

Disponibilité : 2.5.0

Exemples

This self-contained example paints an apple with gradients and highlights into three raster bands. It compares the original RGB raster with the equivalent grayscale results produced by both function variants.

Code

```
WITH pixels AS (
    SELECT
        col,
        row,
        x,
        y,
        (((x - 48.0)^2 / (25.0^2) + (y - 54.0)^2 / (29.0^2) < 1)
         OR ((x - 72.0)^2 / (25.0^2) + (y - 54.0)^2 / (29.0^2) < 1))
         AND NOT ((x - 60.0)^2 / (8.0^2) + (y - 26.0)^2 / (5.0^2) < 1) AS fruit,
        ((x - 82.0) * 0.84 + (y - 28.0) * 0.54)^2 / (22.0^2)
          + (-(x - 82.0) * 0.54 + (y - 28.0) * 0.84)^2 / (8.0^2) < 1 AS leaf,
        abs((x - 62.0) * 0.96 - (y - 29.0) * 0.28) < 3
          AND x BETWEEN 60 AND 69 AND y BETWEEN 21 AND 45 AS stem
    FROM (
        SELECT
            col,
            row,
            col / 2.0 AS x,
            row / 2.0 AS y
        FROM generate_series(1, 180) AS row
        CROSS JOIN generate_series(1, 240) AS col
    ) AS source_pixels
), shaded AS (
    SELECT
        col,
        row,
        x,
        y,
        CASE
            WHEN stem THEN 95 + y * 0.7 + (x % 3) * 8
            WHEN leaf THEN 28 + (90 - x) * 0.5
            WHEN fruit THEN LEAST(255, GREATEST(0,
                150
                + (85 - sqrt((x - 39.0)^2 + (y - 37.0)^2)) * 1.1
                + CASE
                    WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                    THEN 35
```

```
                    ELSE 0
                END
                - CASE
                    WHEN (x - 77.0)^2 / (20.0^2) + (y - 67.0)^2 / (13.0^2) < 1
                    THEN 35
                    ELSE 0
                END
                + ((x * 13 + y * 7) % 19 - 9)
            ))
            ELSE 245
        END AS red,
        CASE
            WHEN stem THEN 50 + y * 0.35
            WHEN leaf THEN LEAST(190, GREATEST(40,
                105 + (95 - x) * 1.2 + (y - 20) * 1.5
                + CASE
                    WHEN (x - 76.0)^2 / (12.0^2) + (y - 24.0)^2 / (4.0^2) < 1
                    THEN 45
                    ELSE 0
                END
            ))
            WHEN fruit THEN LEAST(95, GREATEST(15,
                24 + (68 - y) * 0.35
                + CASE
                    WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                    THEN 30
                    ELSE 0
                END
                + ((x * 5 + y * 11) % 13 - 6)
            ))
            ELSE 245
        END AS green,
        CASE
            WHEN stem THEN 22 + (x % 4) * 3
            WHEN leaf THEN 42 + (100 - x) * 0.4
            WHEN fruit THEN LEAST(75, GREATEST(10,
                18
                + CASE
                    WHEN (x - 43.0)^2 / (10.0^2) + (y - 43.0)^2 / (7.0^2) < 1
                    THEN 25
                    ELSE 0
                END
                + ((x * 3 + y * 5) % 11 - 5)
            ))
            ELSE 245
        END AS blue
    FROM pixels
), rows AS (
    SELECT
        row,
        array_agg(red::double precision ORDER BY col) AS red_values,
        array_agg(green::double precision ORDER BY col) AS green_values,
        array_agg(blue::double precision ORDER BY col) AS blue_values
    FROM shaded
    GROUP BY row
), source AS (
    SELECT ST_SetValues(ST_SetValues(ST_SetValues(
        ST_AddBand(ST_AddBand(ST_AddBand(
            ST_MakeEmptyRaster(240, 180, 0, 180, 1, -1, 0, 0, 0),
            '8BUI'::text, 245::double precision, 0::double precision),
            '8BUI'::text, 245::double precision, 0::double precision),
            '8BUI'::text, 245::double precision, 0::double precision),
```

```
        1, 1, 1, array_agg(red_values ORDER BY row)::double precision[][]),
        2, 1, 1, array_agg(green_values ORDER BY row)::double precision[][]),
        3, 1, 1, array_agg(blue_values ORDER BY row)::double precision[][]) AS rast
    FROM rows
)
SELECT ST_AsPNG(rast) AS "original RGB",
    ST_AsPNG(ST_Grayscale(rast)) AS "grayscale (variant 1)",
    ST_AsPNG(ST_Grayscale(ARRAY[
        ROW(rast, 1)::rastbandarg,
        ROW(rast, 2)::rastbandarg,
        ROW(rast, 3)::rastbandarg
    ]::rastbandarg[])) AS "grayscale (variant 2)"
FROM source;
```

Figure



Original photographic illustration:



original apple



grayscale apple

Voir aussi

ST_AsPNG, ST_Reclass, ST_ColorMap

11.12.4 ST_Intersection

ST_Intersection — Retourne un raster ou un ensemble de paires (géométrie, valeur de pixel) représentant la partie partagée de deux rasters ou l'intersection géométrique d'une vectorisation du raster et d'une géométrie.

Synopsis

setof geomval **ST_Intersection**(geometry geom, raster rast, integer band_num=1);
setof geomval **ST_Intersection**(raster rast, geometry geom);
setof geomval **ST_Intersection**(raster rast, integer band, geometry geomin);
raster **ST_Intersection**(raster rast1, raster rast2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);

Description

Retourne un raster ou un ensemble de paires (géométrie, valeur de pixel) représentant la partie partagée de deux rasters ou l'intersection géométrique d'une vectorisation du raster et d'une géométrie.

Les trois premières variantes, qui retournent un ensemble de géométries, fonctionnent dans l'espace vectoriel. Le raster est d'abord vectorisé (via ST_DumpAsPolygons) en un ensemble de lignes geomval, et ces lignes sont ensuite intersectées avec la géométrie via la fonction PostGIS ST_Intersection (geometry, geometry). Les géométries intersectant uniquement une zone de valeur nodata d'un raster renvoient une géométrie vide. Elles sont normalement exclues des résultats par l'utilisation correcte de ST_Intersects dans la clause WHERE.

Vous pouvez accéder aux parties géométrie et valeur de l'ensemble résultant de geomval en les entourant de parenthèses et en ajoutant '.geom' ou '.val' à la fin de l'expression. par exemple (ST_Intersection(rast, geom)).geom

Les autres variantes, qui retournent un raster, fonctionnent dans l'espace raster. Elles utilisent la variante de ST_MapAlgebraExpr utilisant deux rasters pour réaliser l'intersection.

L'étendue du raster résultant correspond à l'intersection géométrique des deux étendues des rasters. Le raster résultant inclut les bandes 'BAND1', 'BAND2' ou 'BOTH', en fonction du paramètre `returnband`. Les zones de valeurs nodata présentes dans

n'importe quelle bande se traduisent par des zones de valeurs nodata dans toutes les bandes du résultat. En d'autres termes, tout pixel croisant un pixel à valeur nodata devient un pixel à valeur nodata dans le résultat.

Les rasters résultant de ST_Intersection doivent avoir une valeur nodata assignée pour les zones non intersectées. Vous pouvez définir ou remplacer la valeur nodata pour toute bande résultante en fournissant un tableau `nodataval[]` d'une ou deux valeurs nodata selon que vous demandez les bandes 'BAND1', 'BAND2' ou 'BOTH'. La première valeur du tableau remplace la valeur nodata dans la première bande et la deuxième valeur remplace la valeur nodata dans la deuxième bande. Si une bande d'entrée n'a pas de valeur nodata définie et qu'aucune n'est fournie sous forme de tableau, une valeur est choisie via la fonction ST_MinPossibleValue. Toutes les variantes acceptant un tableau de valeurs nodata peuvent également accepter une valeur unique qui sera assignée à chaque bande demandée.

Dans toutes les variantes, si aucune bande n'est spécifiée, la bande 1 est utilisée. Si vous avez besoin d'une intersection entre un raster et une géométrie qui retourne un raster, utilisez ST_Clip.



Note
For more control over the result extent or NODATA handling, use the two-raster variant of ST_MapAlgebraExpr. To compute a raster-space intersection with a geometry, use ST_Clip.
Use this function together with ST_Intersects and an index on the raster or geometry column when filtering table data.

Amélioration : 2.0.0 - Ajout de l'intersection dans l'espace raster. Dans les versions antérieures à la version 2.0.0, seules les intersections réalisées dans l'espace vectoriel étaient prises en charge.

Exemples : Géométrie, Raster -- avec pour résultat des geomval

Code

```
SELECT
    foo.rid,
    foo.gid,
    GeometryType((foo.geomval).geom) AS geom_type,
    (foo.geomval).val AS value
FROM (
    SELECT
        A.rid,
        g.gid,
        ST_Intersection(A.rast, g.geom) As geomval
    FROM dummy_rast AS A
    CROSS JOIN (
        VALUES
            (1, ST_Point(3427928, 5793243.85) ),
            (2, 'LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)' ↩
    ::geometry),
            (3, 'LINESTRING(1 2,3 4)'::geometry)
    ) As g(gid, geom)
    WHERE A.rid = 2
) AS foo;
```

Output

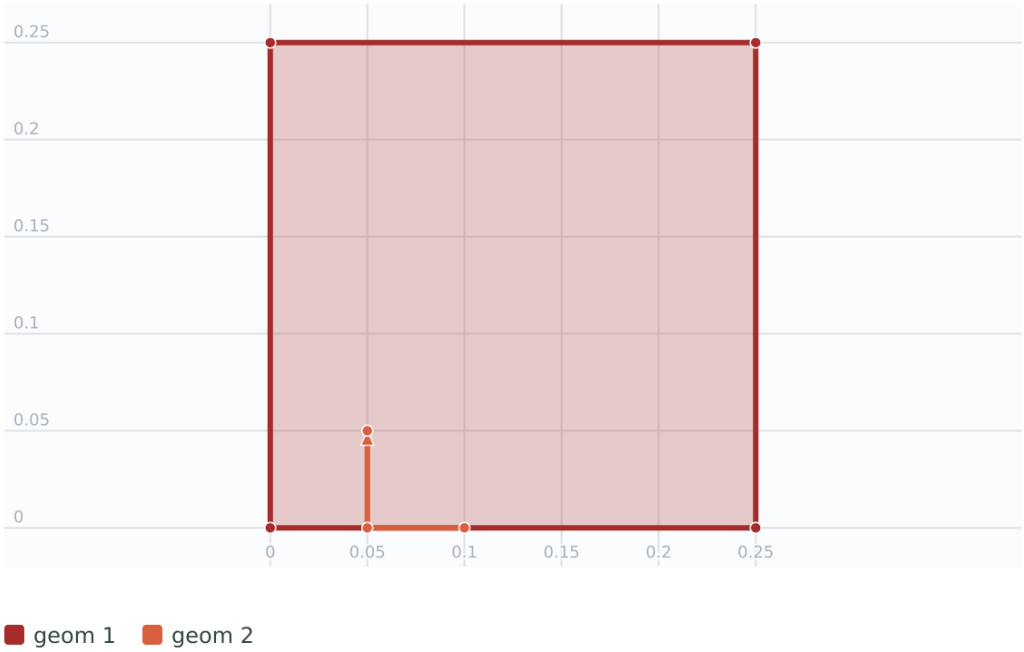
```
rid | gid |      geom_type       | value
-----+-----+----------------------+-------
   2 |   1 | POINT                |   249
   2 |   1 | POINT                |   253
   2 |   2 | POINT                |   254
   2 |   2 | POINT                |   251
   2 |   2 | POINT                |   253
   2 |   2 | LINESTRING           |   252
   2 |   2 | MULTILINESTRING      |   250
   2 |   3 | GEOMETRYCOLLECTION   |
```

The same line example rendered in a common coordinate frame. The raster footprint and complete input line remain visible behind the returned intersection pieces.

Code

```
WITH source AS (
    SELECT
        rast,
        ST_GeomFromText(
            'LINESTRING(3427927.85 5793243.75,'
            '3427927.8 5793243.75,3427927.8 5793243.8)'
        ) AS input_geometry
    FROM dummy_rast
    WHERE rid = 2
), intersection AS (
SELECT
    rast,
    input_geometry,
    ST_Collect(gv.geom) AS result_geometry
FROM source
CROSS JOIN LATERAL ST_Intersection(rast, input_geometry) AS gv
GROUP BY rast, input_geometry
), layers AS (
    SELECT
        'raster' AS title,
        ST_Translate(ST_Envelope(rast), -3427927.75, -5793243.75) AS geom
    FROM intersection
    UNION ALL
    SELECT
        'input',
        ST_Translate(input_geometry, -3427927.75, -5793243.75)
    FROM intersection
    UNION ALL
    SELECT
        'intersection',
        ST_Translate(result_geometry, -3427927.75, -5793243.75)
    FROM intersection
)
SELECT title, geom AS geom
FROM layers
ORDER BY CASE title
    WHEN 'raster' THEN 1
    WHEN 'input' THEN 2
    ELSE 3
END;
```

Figure



Voir aussi

geomval, ST_Intersects, ST_MapAlgebraExpr, ST_Clip, ST_AsText

11.12.5 ST_MapAlgebra (callback function version)

ST_MapAlgebra (callback function version) — Version avec fonction de rappel - Retourne un raster à une bande à partir d'un ou plusieurs rasters d'entrée, d'index de bandes et d'une fonction de rappel spécifiée par l'utilisateur.

Synopsis

raster **ST_MapAlgebra**(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=INTERSECTIO
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);

Description

Retourne un raster à une bande à partir d'un ou plusieurs rasters d'entrée, d'index de bandes et d'une fonction de rappel spécifiée par l'utilisateur.

rast,rast1,rast2, rastbandargset Rasters sur lesquels le traitement d'algèbre cartographique est évalué.

`rastbandargset` permet d'utiliser une opération d'algèbre cartographique sur plusieurs rasters et/ou plusieurs bandes. Voir l'exemple Variante 1.

nband, nband1, nband2 Numéros de bande du raster à évaluer. nband peut être un entier ou un tableau d'entiers désignant les bandes. nband1 est la bande sur rast1 et nband2 est la bande sur rast2 pour le cas de 2 rasters/2 bandes.

callbackfunc Le paramètre `callbackfunc` doit être le nom et la signature d'une fonction SQL ou PL/pgSQL, transformée en une regprocedure. Un exemple de fonction PL/pgSQL est le suivant :

Code

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], pos ↩
    integer[][], VARIADIC userargs text[])
    RETURNS double precision
    AS $$
    BEGIN
        RETURN 0;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The `callbackfunc` must have two or three arguments: a 3-dimension double precision array, a 2-dimension integer array and, optionally, a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The optional third argument `userargs` is for passing through any user-specified arguments.

The callback return value is used as the value of the output pixel. If the callback returns `NULL`, the output pixel is set to NODATA when the output band has a NODATA value. If the output band has no NODATA value, a `NULL` callback result raises an error.

Passer un argument de type regprocedure à une fonction SQL nécessite de passer la signature complète de la fonction, puis de la convertir en un type regprocedure. Pour passer la fonction PL/pgSQL ci-dessus en tant qu'argument, le code SQL de l'argument est le suivant :

Code

```
'sample_callbackfunc(double precision[], integer[], text[])'::regprocedure
```

Note that PostgreSQL array types do not encode dimensions in function signatures, so multidimensional callback arguments are written as double precision[] and integer[] in the regprocedure signature.

mask Un tableau à n dimensions (matrice) de nombres utilisés pour filtrer les cellules transmises à la fonction de rappel de l'algèbre cartographique. 0 signifie que la valeur d'une cellule voisine doit être traitée comme une absence de données et 1 signifie que la valeur doit être traitée comme une donnée. Si weight est définit à true, les valeurs sont utilisées comme multiplicateurs pour multiplier la valeur du pixel par cette valeur dans la position de voisinage.

weighted booléen (true/false) indiquant si la valeur d'un masque doit être pondérée (multipliée par la valeur originale) ou non (ne s'applique qu'aux variantes qui prennent un masque).

pixeltype Si `pixeltype` est spécifié, l'unique bande du nouveau raster sera de ce type de pixel. Si pixeltype est NULL ou omis, la nouvelle bande du raster aura le même pixeltype que la bande spécifiée du premier raster (pour les types d'étendue : INTERSECTION, UNION, FIRST, CUSTOM) ou que la bande spécifiée du raster approprié (pour les types d'étendue : SECOND, LAST). En cas de doute, spécifiez toujours `pixeltype`.

Le type de pixel résultant du raster de sortie doit correspondre à l'un des types énumérés dans ST_BandPixelType, ou être omis ou défini à NULL.

extenttype Les valeurs possibles sont INTERSECTION (par défaut), UNION, FIRST (premier, par défaut pour les variantes à un seul raster), SECOND, LAST (dernier), CUSTOM (personnalisé).

customextent Si `extentype` est CUSTOM, un raster doit être spécifié par le paramètre `customextent`. Voir l'exemple 4 de la variante 1.

distancex La distance en pixels de la cellule de référence dans la direction x. La largeur de la matrice résultante sera donc `2*distancex + 1`. Si elle n'est pas spécifiée, seule la cellule de référence est prise en compte (voisinage de 0).

distancey La distance en pixels de la cellule de référence dans la direction y. La hauteur de la matrice résultante sera `2*distancey + 1` Si elle n'est pas spécifiée, seule la cellule de référence est prise en compte (voisinage de 0).

userargs If the `callbackfunc` declares a third argument, that argument is a variadic text array. All trailing text arguments are passed through to the specified `callbackfunc`, and are contained in the `userargs` argument.



Note
Pour plus d'informations sur le mot-clé VARIADIC, veuillez vous référer à la documentation PostgreSQL et à la section "SQL Functions with Variable Numbers of Arguments" de Query Language (SQL) Functions.

La variante 1 accepte un tableau de `rastbandarg` permettant l'utilisation d'une opération d'algèbre cartographique sur de nombreux masters et/ou de nombreuses bandes. Voir l'exemple de la variante 1.

Les variantes 2 et 3 agissent sur une ou plusieurs bandes d'un même raster. Voir l'exemple des variantes 2 et 3.

La variante 4 utilise deux rasters avec une bande par raster. Voir l'exemple de la variante 4.

Disponibilité : 2.2.0 : Possibilité d'ajouter un masque

Disponibilité : 2.1.0

Exemples

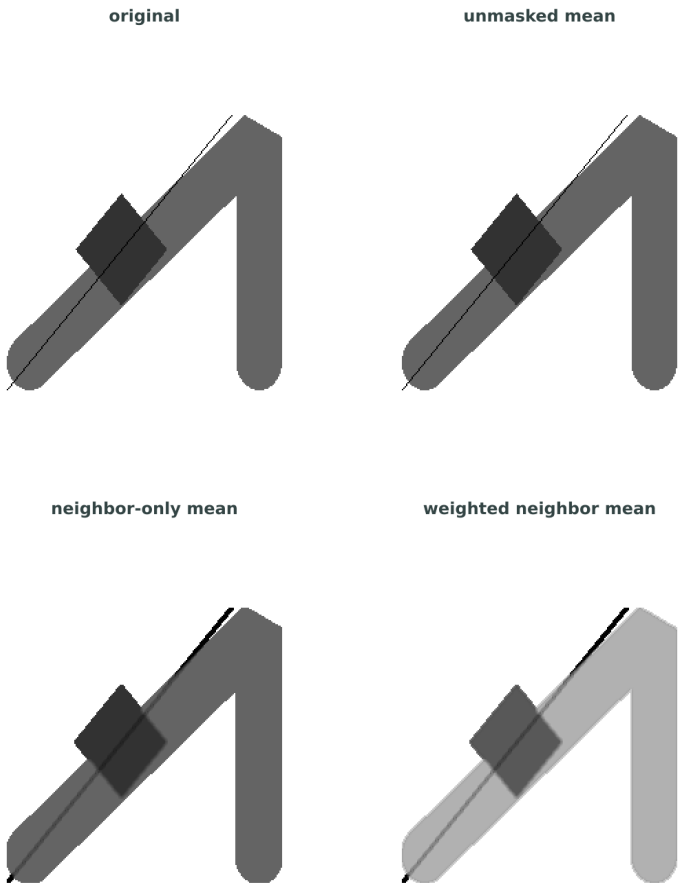
Compare an unmasked mean with neighbor-only and weighted-neighbor masks. The weighted mask multiplies neighboring values by 2 before computing the result.

Code

```
WITH foo AS (
    SELECT ST_SetBandNoDataValue(
        ST_SetValue(
            ST_SetValue(
                ST_AsRaster(
                    ST_Buffer('LINESTRING(50 50,100 90,100 50)'::geometry, 5, 'join=bevel'),
                    200, 200, ARRAY['8BUI'], ARRAY[100], ARRAY[0]
                ),
                ST_Buffer('POINT(70 70)'::geometry, 10, 'quad_segs=1'),
                50
            ),
            'LINESTRING(20 20,100 100,150 98)'::geometry,
            1
        ),
        0
    ) AS rast
), variants AS (
    SELECT 'original' AS title, rast
    FROM foo
    UNION ALL
    SELECT 'unmasked mean' AS title,
        ST_MapAlgebra(rast, 1, 'ST_Mean4ma(double precision[], int[], text[])'::regprocedure ↩
    ) AS rast
    FROM foo
    UNION ALL
    SELECT 'neighbor-only mean' AS title,
        ST_MapAlgebra(
            rast,
            1,
            'ST_Mean4ma(double precision[], int[], text[])'::regprocedure,
```

```
            '{{1,1,1}, {1,0,1}, {1,1,1}}'::double precision[],
            false
        ) AS rast
    FROM foo
    UNION ALL
    SELECT 'weighted neighbor mean' AS title,
        ST_MapAlgebra(
            rast,
            1,
            'ST_Mean4ma(double precision[], int[], text[])'::regprocedure,
            '{{2,2,2}, {2,0,2}, {2,2,2}}'::double precision[],
            true
        ) AS rast
    FROM foo
)
SELECT title, ST_AsPNG(rast) AS image
FROM variants;
```

Figure



Variant 1.

Un raster, une bande

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI',  ↩
    1, 0) AS rast
)
SELECT
    ST_MapAlgebra(ARRAY[ROW(rast, 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Un raster, plusieurs bandes

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg ↩
    [],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Plusieurs rasters, plusieurs bandes

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2) ↩
    ]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Complete example of tiles of a coverage with neighborhood.

Code

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI',  ↩
    1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0)  ↩
    AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0)  ↩
    AS rast UNION ALL
```

```
    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) ↩
     AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) ↩
     AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, 0) ↩
     AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100,  ↩
    0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200,  ↩
    0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300,  ↩
    0) AS rast
)
SELECT
    t1.rid,
    ST_MapAlgebra(ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
        '32BUI',
        'CUSTOM', t1.rast,
        1, 1
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
    AND t2.rid BETWEEN 0 AND 8
    AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast
```

Example like the prior one for tiles of a coverage with neighborhood but using only basic SQL features.

Code

```
WITH src AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI',  ↩
    1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0)  ↩
    AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0)  ↩
    AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10, 0) ↩
     AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20, 0) ↩
     AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30, 0) ↩
     AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100,  ↩
    0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200,  ↩
    0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300,  ↩
    0) AS rast
), foo AS (
    SELECT
        t1.rid,
        ST_Union(t2.rast) AS rast
    FROM src t1
    JOIN src t2
        ON ST_Intersects(t1.rast, t2.rast)
```

```
        AND t2.rid BETWEEN 0 AND 8
    WHERE t1.rid = 4
    GROUP BY t1.rid
), bar AS (
    SELECT
        t1.rid,
        ST_MapAlgebra(
            ARRAY[ROW(t2.rast, 1)]::rastbandarg[],
            'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
    FROM src t1
    JOIN foo t2
        ON t1.rid = t2.rid
)
SELECT
    rid,
    (ST_MetaData(rast)),
    (ST_BandMetaData(rast, 1)),
    ST_Value(rast, 1, 1, 1)
FROM bar;
```

Variants 2 and 3.

Un raster, plusieurs bandes

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast,
        ARRAY[3, 1, 3, 2]::integer[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Un raster, une bande

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1,  ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast,
        2,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Variant 4.

Deux rasters, deux bandes

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast,
        2,
        t2.rast,
        1,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Voir aussi

rastbandarg, ST_Union, ST_MapAlgebra (expression version)

11.12.6 ST_MapAlgebra (expression version)

ST_MapAlgebra (expression version) — Version avec expression - Retourne un raster à une bande à partir d'un ou deux rasters d'entrée, d'index de bandes et d'une ou plusieurs expressions SQL spécifiées par l'utilisateur.

Synopsis

raster **ST_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Description

Version avec expression - Retourne un raster à une bande à partir d'un ou deux rasters d'entrée, d'index de bandes et d'une ou plusieurs expressions SQL spécifiées par l'utilisateur.

Disponibilité : 2.1.0

Description : Variantes 1 et 2 (un raster)

Crée un nouveau raster à une bande formé en appliquant une opération algébrique PostgreSQL valide définie par `expression` sur le raster d'entrée (`rast`). Si `nband` n'est pas fourni, la bande 1 est utilisée. Le nouveau raster aura la même géoréférence, la même largeur et la même hauteur que le raster original mais n'aura qu'une seule bande.

Si `pixeltype` est fourni, le nouveau raster aura une bande de ce type de pixel. Si pixeltype est NULL, la nouvelle bande raster aura le même type de pixel que la bande d'entrée de `rast`.

— Mots clés possibles pour `expression`

1. `[rast]` - Valeur du pixel d'intérêt
2. `[rast.val]` - Valeur du pixel d'intérêt
3. `[rast.x]` - colonne du pixel d'intérêt (démarrant à 1)
4. `[rast.y]` - ligne du pixel d'intérêt (démarrant à 1)

Description : Variantes 3 et 4 (deux rasters)

Crée un nouveau raster à une bande formé en appliquant une opération algébrique PostgreSQL valide aux deux bandes définies par `expression` sur les deux bandes d'entrée `rast1` (et optionnellement `rast2`). Si aucune bande `band1`, `band2` n'est spécifiée, la bande 1 est utilisée. Le raster résultant sera aligné (échelle, obliquité et coins) sur la grille définie par le premier raster. Le raster résultant aura l'étendue définie par le paramètre `extenttype`.

`expression` Une expression algébrique PostgreSQL impliquant les deux rasters et les fonctions/opérateurs définis par PostgreSQL qui définiront la valeur du pixel lorsque les pixels intersectent, par exemple (([rast1] + [rast2])/2.0)::integer

`pixeltype` Le type de pixel résultant du raster de sortie. Doit être l'un de ceux listés dans ST_BandPixelType, omis ou NULL. S'il n'est pas fourni ou NULL, le type de pixel du premier raster sera utilisé par défaut.

`extenttype` Contrôle l'étendue du raster de sortie

1. `INTERSECTION` - L'étendue du nouveau raster est l'intersection des deux rasters. Il s'agit de la valeur par défaut.
2. `UNION` - L'étendue du nouveau raster est l'union des deux rasters.
3. `FIRST` - L'étendue du nouveau raster est la même que celle du premier raster.
4. `SECOND` - L'étendue du nouveau raster est la même que celle du second raster.

`nodata1expr` Une expression algébrique impliquant uniquement `rast2`, ou une constante qui définit la valeur à retourner lorsque les pixels de `rast1` sont des valeurs nodata et que les pixels correspondants dans rast2 ont des valeurs.

`nodata2expr` Une expression algébrique impliquant uniquement `rast1`, ou une constante qui définit la valeur à retourner lorsque les pixels de `rast2` sont des valeurs nodata et que les pixels correspondants dans rast1 ont des valeurs.

`nodatanodataval` Constante numérique à retourner lorsque les pixels de rast1 et de rast2 sont tous deux des valeurs nodata.

— Mots clés possibles dans `expression`, `nodata1expr` et `nodata2expr`

1. `[rast1]` - Valeur du pixel d'intérêt sur `rast1`
2. `[rast1.val]` - Valeur du pixel d'intérêt de `rast1`
3. `[rast1.x]` - Colonne du pixel d'intérêt de `rast1` (démarrant à 1)
4. `[rast1.y]` - Ligne du pixel d'intérêt de `rast1` (démarrant à 1)
5. `[rast2]` - Valeur du pixel d'intérêt sur `rast2`
6. `[rast2.val]` - Valeur du pixel d'intérêt de `rast2`
7. `[rast2.x]` - Colonne du pixel d'intérêt de `rast2` (démarrant à 1)
8. `[rast2.y]` - Ligne du pixel d'intérêt de `rast2` (démarrant à 1)

Exemples

Variants 1 and 2.

Code

```
WITH foo AS (
    SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ↩
    AS rast
)
SELECT
    ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

Variant 3 and 4.

Code

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1,
    0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast UNION
    ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1,
    0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast,
        2,
        t2.rast,
        1,
        '([rast2] + [rast1.val]) / 2'
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2;
```

Voir aussi

rastbandarg, ST_Union, ST_MapAlgebra (callback function version)

11.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Version avec raster à 1 bande : Crée un nouveau raster à une bande formé par l'application d'une opération algébrique PostgreSQL valide sur la bande d'entrée du raster et du type de pixel fourni. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

Description



AVERTISSEMENT
ST_MapAlgebraExpr est obsolète depuis 2.1.0. Utilisez plutôt ST_MapAlgebra (expression version).

Crée un nouveau raster à une bande formé par l'application d'une opération algébrique PostgreSQL valide définie par `expression` sur le raster d'entrée (`rast`). Si aucune `band` n'est spécifiée, la bande 1 est utilisée. Le nouveau raster aura la même géoréférence, la même largeur et la même hauteur que le raster original mais n'aura qu'une seule bande.

Si `pixeltype` est fourni, le nouveau raster aura une bande de ce type de pixel. Si pixeltype est NULL, la nouvelle bande raster aura le même type de pixel que la bande d'entrée de `rast`.

Dans l'expression, vous pouvez utiliser le mot clé `[rast]` pour faire référence à la valeur en pixels de la bande originale, `[rast.x]` pour faire référence à l'indice de colonne de pixels (basé sur 1), `[rast.y]` pour faire référence à l'indice de ligne de pixels (basé sur 1).

Disponibilité : 2.0.0

Exemples

Crée une nouveau raster à 1 bande qui est une fonction modulo 2 de la bande du raster original.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast, NULL, 'mod([rast]::numeric,2)') ↩
    WHERE rid = 2;

SELECT
    ST_Value(rast, 1, i, j) As origval,
    ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
origval | mapval
---------+--------
     253 |      1
     254 |      0
     253 |      1
     253 |      1
     254 |      0
     254 |      0
     250 |      0
     254 |      0
     254 |      0
```

Crée un nouveau raster à 1 bande de type de pixel 2BUI à partir d'un raster original reclassifié et avec la valeur nodata à 0.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
    map_rast2 = ST_MapAlgebraExpr(rast, '2BUI'::text, 'CASE WHEN [rast] BETWEEN 100 and 250 ↩
    THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END'::text ↩
    , '0')
WHERE rid = 2;

SELECT DISTINCT
    ST_Value(rast, 1, i, j) As origval,
    ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1, 5) AS j
WHERE rid = 2;
```

Output

```
origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3
```

Code

```
SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

Output

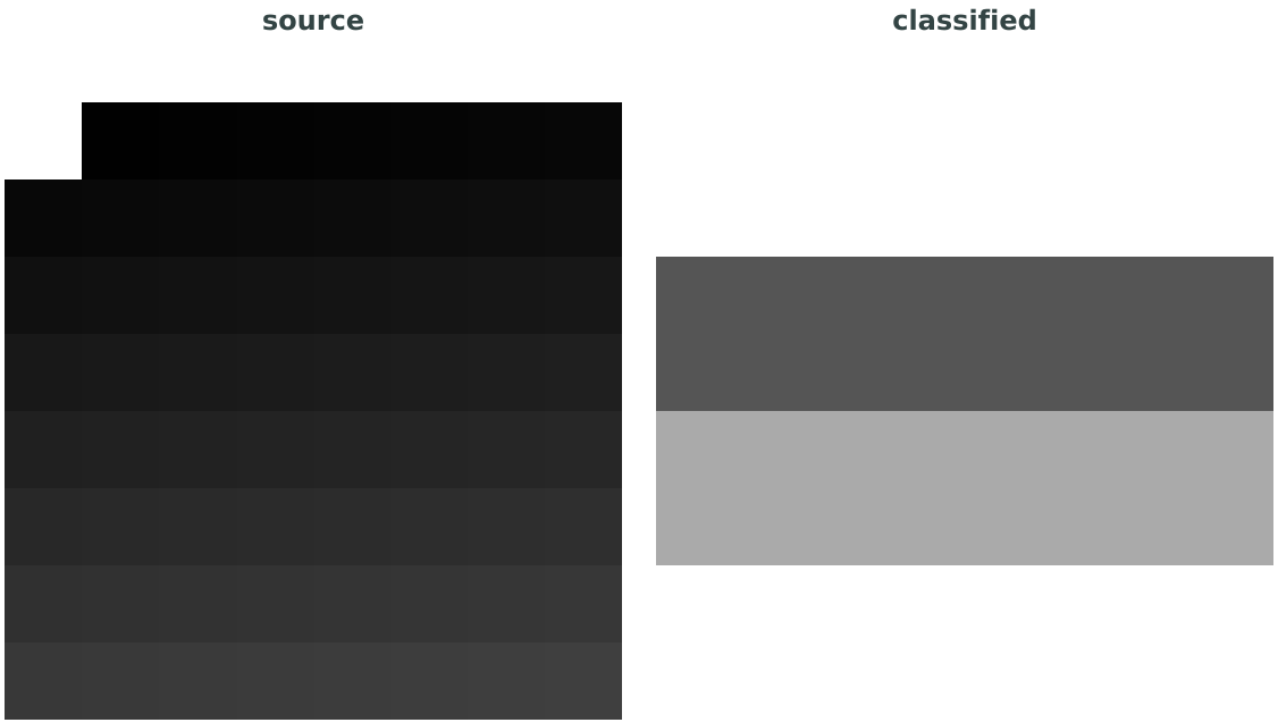
```
2BUI
```

A self-contained source raster and the raster produced by classifying its values into four display levels.

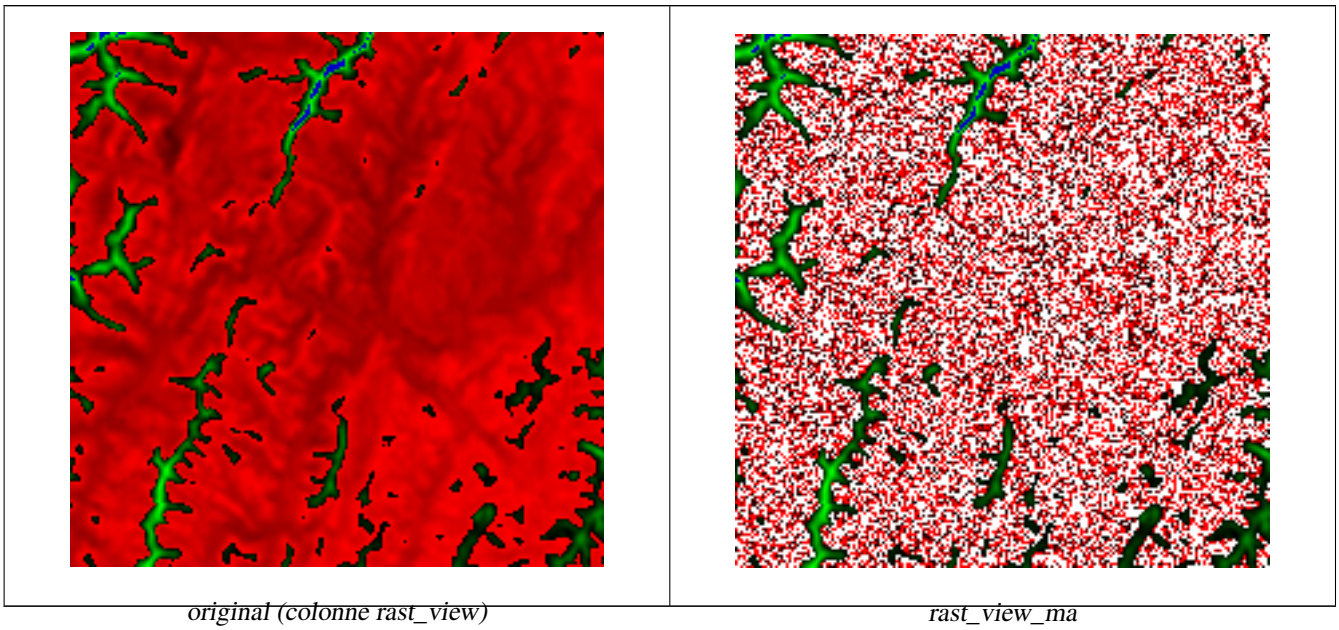
Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(8, 8, 0, 8, 1, -1, 0, 0, 0),
            1, '8BUI', 0, 0
        ),
        1, 1, 1,
        ARRAY[
            [0, 1, 2, 3, 4, 5, 6, 7],
            [8, 9, 10, 11, 12, 13, 14, 15],
            [16, 17, 18, 19, 20, 21, 22, 23],
            [24, 25, 26, 27, 28, 29, 30, 31],
            [32, 33, 34, 35, 36, 37, 38, 39],
            [40, 41, 42, 43, 44, 45, 46, 47],
            [48, 49, 50, 51, 52, 53, 54, 55],
            [56, 57, 58, 59, 60, 61, 62, 63]
        ]::double precision[][]
    ) AS rast
), variants AS (
    SELECT 'source' AS title, rast AS rendered
    FROM source
    UNION ALL
    SELECT
        'classified',
        ST_MapAlgebraExpr(rast, '8BUI', 'floor([rast] / 16) * 85')
    FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Map algebra altering the first band of a three-band raster.



original (colonne rast_view) *rast_view_ma*

Crée un nouveau raster à 3 bandes de même type de pixel que le raster original à 3 bandes, la première bande étant modifiée par l'algèbre cartographique et les 2 bandes restantes n'étant pas modifiées.

Code

```
SELECT
    ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(rast_view),
                ST_MapAlgebraExpr(rast_view, 1, NULL, 'tan([rast])*[rast]')
            ),
            ST_Band(rast_view, 2)
        ),
        ST_Band(rast_view, 3)
    )  As rast_view_ma
FROM wind
WHERE rid=167;
```

Voir aussi

ST_MapAlgebraExpr, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value

11.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Version avec 2 bandes : Crée un nouveau raster à une bande formé en appliquant une opération algébrique PostgreSQL valide sur les deux bandes du raster d'entrée et du type de pixel fourni. La bande 1 de chaque raster est utilisée si aucun numéro de bande n'est spécifié. Le raster résultant sera aligné (échelle, obliquité et coins) sur la grille définie par le premier raster et aura son étendue définie par le paramètre "extenttype". Les valeurs de "extenttype" peuvent être : INTERSECTION, UNION, FIRST, SECOND.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSEC
text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same_as_rast1_band,
text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Description



AVERTISSEMENT
ST_MapAlgebraExpr est obsolète depuis 2.1.0. Utilisez plutôt ST_MapAlgebra (expression version).

Crée un nouveau raster à une bande formé en appliquant une opération algébrique PostgreSQL valide aux deux bandes définies par `expression` sur les deux bandes d'entrée `rast1` (et optionnellement `rast2`). Si aucune bande `band1`, `band2` n'est spécifiée, la bande 1 est utilisée. Le raster résultant sera aligné (échelle, obliquité et coins) sur la grille définie par le premier raster. Le raster résultant aura l'étendue définie par le paramètre `extenttype`.

`expression` Une expression algébrique PostgreSQL impliquant les deux rasters et les fonctions/opérateurs définis par PostgreSQL qui définiront la valeur du pixel lorsque les pixels intersectent, par exemple (([rast1] + [rast2])/2.0)::integer

`pixeltype` Le type de pixel résultant du raster de sortie. Doit être l'un de ceux listés dans ST_BandPixelType, omis ou NULL. S'il n'est pas fourni ou NULL, le type de pixel du premier raster sera utilisé par défaut.

`extenttype` Contrôle l'étendue du raster de sortie

1. `INTERSECTION` - L'étendue du nouveau raster est l'intersection des deux rasters. Il s'agit de la valeur par défaut.
2. `UNION` - L'étendue du nouveau raster est l'union des deux rasters.
3. `FIRST` - L'étendue du nouveau raster est la même que celle du premier raster.

4. SECOND - L'étendue du nouveau raster est la même que celle du second raster.

nodata1expr Une expression algébrique impliquant uniquement rast2, ou une constante qui définit la valeur à retourner lorsque les pixels de rast1 sont des valeurs nodata et que les pixels correspondants dans rast2 ont des valeurs.

nodata2expr Une expression algébrique impliquant uniquement rast1, ou une constante qui définit la valeur à retourner lorsque les pixels de rast2 sont des valeurs nodata et que les pixels correspondants dans rast1 ont des valeurs.

nodatanodataval Constante numérique à retourner lorsque les pixels de rast1 et de rast2 sont tous deux des valeurs nodata.

Si le pixeltype est fourni, le nouveau raster aura une bande de ce type de pixel. Si le type de pixel est NULL ou si aucun type de pixel n'est spécifié, la nouvelle bande raster aura le même type de pixel que la bande d'entrée de rast1.

Utilisez le mot clé [rast1.val] [rast2.val] pour désigner la valeur en pixels des bandes raster d'origine et [rast1.x], [rast1.y] etc. pour désigner la position des pixels dans les colonnes et les lignes.

Disponibilité : 2.0.0

Exemples

2 Band Intersection and Union.

Crée une nouveau raster à 1 bande qui est une fonction modulo 2 de la bande du raster original.

Create a cool set of rasters.

Insert some cool shapes around Boston in Massachusetts state plane meters, then map their intersection and union.

Code

```
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930, 26986), 200, 200, ↩
     '8BUI', 0, 0));

INSERT INTO fun_shapes(fun_name, rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930), 26986) ↩
    , 1000),
            ref.rast, '8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
       ST_AsRaster(
           (
               SELECT ST_Collect(geom)
               FROM (
                   SELECT ST_Buffer(
                       ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random()*100) ↩
    , 26986),
                       random()*20
                   ) AS geom
                   FROM generate_series(1, 10) AS i,
                        generate_series(1, 10) AS j
               ) AS foo
           ),
           ref.rast, '8BUI', 200, 0
       )
FROM ref;

SELECT  ST_MapAlgebraExpr(area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[ ↩
    rast2.val]', '[rast1.val]') As interrast,
        ST_MapAlgebraExpr(area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]' ↩
    , '[rast1.val]') As unionrast
```

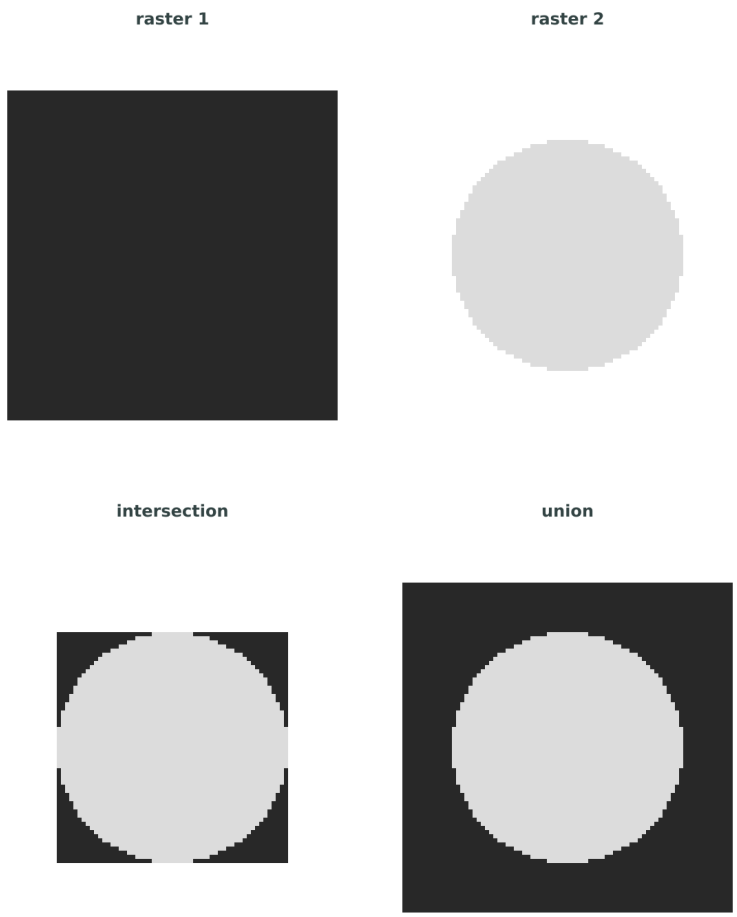
```
FROM
  (SELECT rast FROM fun_shapes WHERE fun_name = 'area') As area
CROSS JOIN
  (SELECT rast FROM fun_shapes WHERE fun_name = 'rand bubbles') As bub
```

The same intersection and union operations using two deterministic, self-contained rasters.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 40, 0
    ) AS rast
), inputs AS (
    SELECT
        rast AS raster_1,
        ST_AsRaster(
            ST_Buffer(ST_Point(40, 40), 28),
            rast,
            '8BUI', 220, 0
        ) AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'intersection', ST_MapAlgebraExpr(
        raster_1, raster_2,
        '[rast2.val]', '8BUI', 'INTERSECTION',
        '[rast2.val]', '[rast1.val]'
    ) FROM inputs
    UNION ALL
    SELECT 'union', ST_MapAlgebraExpr(
        raster_1, raster_2,
        '[rast2.val]', '8BUI', 'UNION',
        '[rast2.val]', '[rast1.val]'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY CASE title
    WHEN 'raster 1' THEN 1
    WHEN 'raster 2' THEN 2
    WHEN 'intersection' THEN 3
    ELSE 4
END;
```

Figure



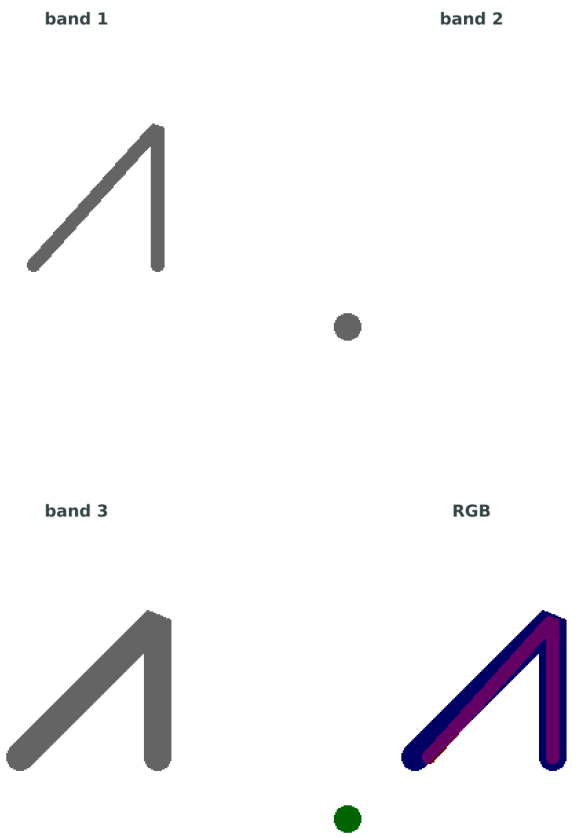
Overlaying rasters on a one-unit-per-pixel canvas as separate bands. We use ST_AsPNG to render the image so all single band ones look grey.

Code

```
WITH mygeoms AS (
    SELECT 2 AS bnum, ST_Buffer(ST_Point(1, 5), 10) AS geom
    UNION ALL
    SELECT 3,
        ST_Buffer(
            ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'),
            10,
            'join=bevel'
        )
    UNION ALL
    SELECT 1,
        ST_Buffer(
            ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'),
            5,
            'join=bevel'
        )
), canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(
```

```
            200,
            200,
            ST_XMin(e)::integer,
            ST_YMax(e)::integer,
            1, -1, 0, 0),
        '8BUI'::text,
        0
    ) AS rast
    FROM (
      SELECT ST_Extent(geom) AS e
      FROM mygeoms
    ) AS bounds
), rbands AS (
    SELECT ARRAY(
        SELECT ST_MapAlgebraExpr(
            canvas.rast,
            ST_AsRaster(m.geom, canvas.rast, '8BUI', 100),
            '[rast2.val]', '8BUI', 'FIRST',
            '[rast2.val]', '[rast1.val]'
        )
        FROM mygeoms AS m
        CROSS JOIN canvas
        ORDER BY m.bnum
    ) AS rasts
)
SELECT title, ST_AsPNG(rendered) AS image
FROM rbands
CROSS JOIN LATERAL (VALUES
    ('band 1', rasts[1]),
    ('band 2', rasts[2]),
    ('band 3', rasts[3]),
    ('RGB', ST_AddBand(ST_AddBand(rasts[1], rasts[2]), rasts[3]))
) AS variants(title, rendered);
```

Figure



Overlay a two-meter boundary of selected parcels over aerial imagery. The query first unions the parcel geometries, clips the source raster shards to the region, unions those smaller shards, and finally adds the first two bands plus a third-band map algebra overlay.

Create a new three-band raster composed of the first two clipped bands and the third band overlaid with the geometry. This query took 3.6 seconds on a 64-bit PostGIS Windows installation.

Code

```
WITH pr AS (
    SELECT ST_Clip(rast, ST_Expand(geom, 50)) AS rast, g.geom
    FROM aerials.o_2_boston AS r
    INNER JOIN (
        SELECT ST_Union(ST_Transform(geom, 26986)) AS geom
        FROM landparcels
        WHERE pid IN ('0303890000', '0303900000')
    ) AS g
        ON ST_Intersects(rast::geometry, ST_Expand(g.geom, 50))
), prunion AS (
    SELECT ST_AddBand(
        NULL,
        ARRAY[
            ST_Union(rast, 1),
            ST_Union(rast, 2),
```

```
            ST_Union(rast, 3)
        ]
    ) AS clipped,
    geom
    FROM pr
    GROUP BY geom
)
SELECT ST_AddBand(
    ST_Band(clipped, ARRAY[1, 2]),
    ST_MapAlgebraExpr(
     ST_Band(clipped, 3),
     ST_AsRaster(ST_Buffer(ST_Boundary(geom), 2), clipped, '8BUI', 250),
     '[rast2.val]',
     '8BUI',
     'FIRST',
     '[rast2.val]',
     '[rast1.val]') ) As rast
FROM prunion;
```

Parcel-boundary overlay.



The blue lines are the boundaries of selected parcels

The generated four-panel canvas example above shows the same band-overlay technique without requiring an external aerial-imagery or parcel table.

Voir aussi

ST_MapAlgebraExpr, ST_AddBand, ST_AsPNG, ST_AsRaster, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value, ST_Union, ST_Union

11.12.9 ST_MapAlgebraFct

ST_MapAlgebraFct — Version à 1 bande - Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide sur la bande d'entrée du raster et sur le type de pixel fourni. La bande 1 est utilisée si aucune bande n'est

spécifiée.

Synopsis

raster **ST_MapAlgebraFct**(raster rast, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, integer band, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

Description



AVERTISSEMENT
ST_MapAlgebraFct est obsolète depuis 2.1.0. Utilisez plutôt ST_MapAlgebra (callback function version).

Crée un nouveau raster à une bande formé en appliquant une fonction PostgreSQL valide spécifiée par `onerasteruserfunc` sur le raster d'entrée (`rast`). Si aucune bande `band` n'est spécifiée, la bande 1 est utilisée. Le nouveau raster aura la même géoréférence, la même largeur et la même hauteur que le raster original mais n'aura qu'une seule bande.

Si `pixeltype` est fourni, le nouveau raster aura une bande de ce type de pixel. Si pixeltype est NULL, la nouvelle bande raster aura le même type de pixel que la bande d'entrée de `rast`.

Le paramètre `onerasteruserfunc` doit être le nom et la signature d'une fonction SQL ou PL/pgSQL, transformée en regprocedure. Un exemple de fonction PL/pgSQL très simple et assez inutile est le suivant :

Code

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

La `fonction utilisateur` peut accepter deux ou trois arguments : une valeur flottante, un tableau d'entiers facultatif et un tableau de texte variable. Le premier argument est la valeur d'une cellule matricielle individuelle (quel que soit le type de données matricielles). Le deuxième argument est la position de la cellule de traitement actuelle sous la forme '{x,y}'. Le troisième argument indique que tous les paramètres restants de ST_MapAlgebraFct doivent être transmis à la `fonction utilisateur`.

Passer un argument de type regprodedure à une fonction SQL nécessite de passer la signature complète de la fonction, puis de la convertir en un type regprocedure. Pour passer la fonction PL/pgSQL ci-dessus en tant qu'argument, le code SQL de l'argument est le suivant :

Code

```
'simple_function(float,integer[],text[])'::regprocedure
```

Notez que l'argument contient le nom de la fonction, les types des arguments de la fonction, des guillemets autour du nom et des types d'arguments, ainsi qu'un cast vers un regprocedure.

Le troisième paramètre de la fonction `userfunction` est un tableau variadic text . Tous les arguments textuels supplémentaires de l'appel à ST_MapAlgebraFct sont transmis à `userfunction`, et sont contenus dans l'argument `args`.



Note
Pour plus d'informations sur le mot-clé VARIADIC, veuillez vous référer à la documentation PostgreSQL et à la section "SQL Functions with Variable Numbers of Arguments" de Query Language (SQL) Functions.



Note
Le paramètre text[] de la fonction `userfunction` est obligatoire, que vous choisissiez ou non de transmettre des arguments à la fonction de rappel.

Disponibilité : 2.0.0

Exemples

Crée une nouveau raster à 1 bande qui est une fonction modulo 2 de la bande du raster original.

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast, NULL, 'mod_fct(float,integer[],text ↩
    [])'::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast, 1, i, j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
origval | mapval
---------+--------
     253 |      1
     254 |      0
     253 |      1
     253 |      1
     254 |      0
     254 |      0
     250 |      0
     254 |      0
     254 |      0
```

Crée un nouveau raster à 1 bande de type de pixel 2BUI à partir d'un raster reclassifié avec la valeur nodata fixée via un paramètre passé à la fonction utilisateur (0).

Code

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
```

```
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel
> 252 THEN
        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↩
    [],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;
```

Output

```
 origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3
```

Code

```
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;
```

Output

```
 b1pixtyp
----------
 2BUI
```

Crée un nouveau raster à 3 bandes de même type de pixel que le raster original à 3 bandes, la première bande étant modifiée par l'algèbre cartographique et les 2 bandes restantes n'étant pas modifiées.

Code

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(rast_view),
            ST_MapAlgebraFct(rast_view, 1, NULL, 'rast_plus_tan(float,integer[],text[])':: ↩
    regprocedure)
```

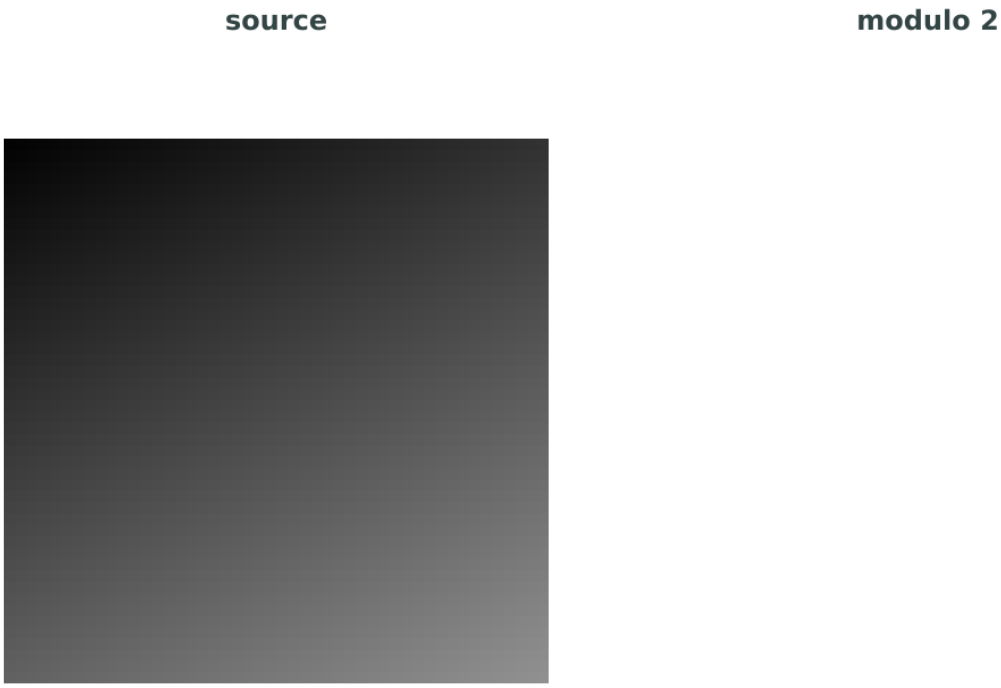
```
        ),
        ST_Band(rast_view, 2)
    ),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
```

The deprecated callback form above is equivalent to a modern ST_MapAlgebra (callback function version) expression for simple per-pixel transforms. This build-time example renders the original raster and a modulo-2 classification.

Code

```
WITH pixels AS (
    SELECT x, y, (x + y * 2)::double precision AS value
    FROM generate_series(1, 48) AS x
    CROSS JOIN generate_series(1, 48) AS y
), rows AS (
    SELECT y, array_agg(value ORDER BY x) AS values
    FROM pixels
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(48, 48, 0, 48, 1, -1, 0, 0, 0),
            1, '8BUI', 0, 0
        ),
        1, 1, 1, array_agg(values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), variants AS (
    SELECT 'source' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'modulo 2', ST_MapAlgebra(
        rast, 1,
        '8BUI',
        'mod([rast]::integer,2) * 255'
    )
    FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Voir aussi

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

11.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — Version à 2 bandes - Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide sur les 2 bandes d'entrée du raster et sur le type de pixel fourni. La bande 1 est utilisée si aucune bande n'est spécifiée. Le type d'étendue INTERSECTION est utilisé si non spécifié.

Synopsis

raster **ST_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster **ST_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

Description



AVERTISSEMENT
ST_MapAlgebraFct est obsolète depuis 2.1.0. Utilisez plutôt ST_MapAlgebra (callback function version).

Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide spécifiée par `tworastuserfunc` sur les rasters d'entrée `rast1`, `rast2`. Si aucune bande `band1` ou `band2` n'est spécifiée, la bande 1 est utilisée. Le nouveau raster aura la même géoréférence, la même largeur et la même hauteur que les rasters originaux, mais il n'aura qu'une seule bande.

Si `pixeltype` est fourni, le nouveau raster aura une bande de ce type de pixel. Si pixeltype est NULL ou omis, la nouvelle bande raster aura le même type de pixel que la bande d'entrée de `rast1`.

Le paramètre `tworastuserfunc` doit être le nom et la signature d'une fonction SQL ou PL/pgSQL, transformée en une regprocedure. Un exemple de fonction PL/pgSQL est le suivant :

Code

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ↩
    INTEGER[], VARIADIC args TEXT[])
RETURNS FLOAT
AS $$ BEGIN
    RETURN 0.0;
END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

La `tworastuserfunc` peut accepter trois ou quatre arguments : une valeur de double précision, une valeur de double précision, un tableau d'entiers facultatif et un tableau de texte variable. Le premier argument est la valeur d'une cellule matricielle individuelle dans `rast1` (quel que soit le type de données matricielles). Le deuxième argument est la valeur d'une cellule de trame individuelle dans `rast2`. Le troisième argument est la position de la cellule de traitement actuelle sous la forme '{x,y}'. Le quatrième argument indique que tous les paramètres restants de ST_MapAlgebraFct doivent être transmis au `tworastuserfunc`.

Passer un argument de type regprodedure à une fonction SQL nécessite de passer la signature complète de la fonction, puis de la convertir en un type regprocedure. Pour passer la fonction PL/pgSQL ci-dessus en tant qu'argument, le code SQL de l'argument est le suivant :

Code

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

Notez que l'argument contient le nom de la fonction, les types des arguments de la fonction, des guillemets autour du nom et des types d'arguments, ainsi qu'un cast vers un regprocedure.

Le quatrième paramètre de la fonction `tworastuserfunc` est un tableau variadic text . Tous les arguments textuels supplémentaires de l'appel à ST_MapAlgebraFct sont transmis à `tworastuserfunc`, et sont contenus dans l'argument `userargs`.



Note
Pour plus d'informations sur le mot-clé VARIADIC, veuillez vous référer à la documentation PostgreSQL et à la section "SQL Functions with Variable Numbers of Arguments" de Query Language (SQL) Functions.



Note
Le paramètre text[] de la fonction `tworastuserfunc` est obligatoire, que vous choisissiez ou non de transmettre des arguments à la fonction de rappel.

Disponibilité : 2.0.0

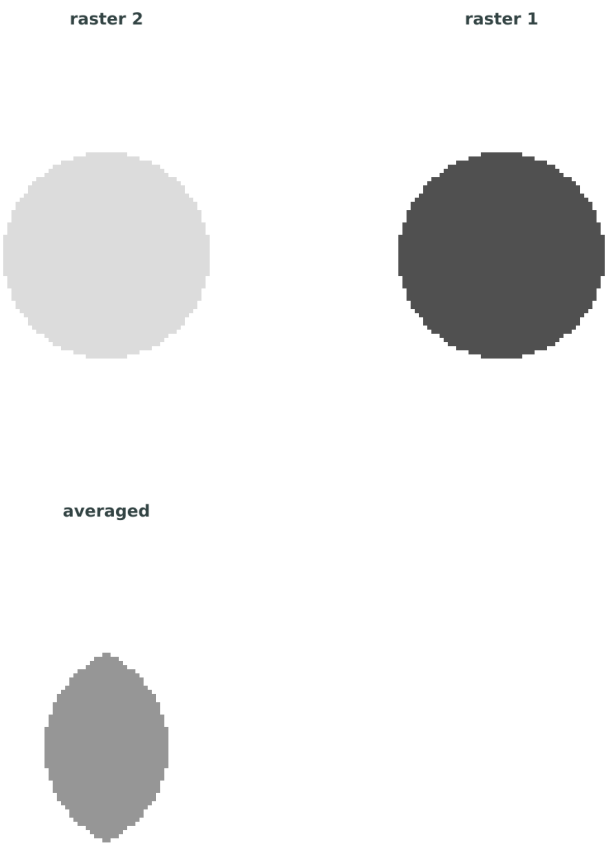
Exemple : Superposition de rasters sur un canevas sous forme de bandes distinctes

The deprecated two-raster callback form can often be expressed with the modern two-raster ST_MapAlgebra (callback function version) expression variant. This build-time example renders two input rasters and their averaged overlap.

Code

```
WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 0, 0
    ) AS rast
), inputs AS (
    SELECT
        ST_AsRaster(ST_Buffer(ST_Point(30, 40), 25), rast, '8BUI', 80, 0)
            AS raster_1,
        ST_AsRaster(ST_Buffer(ST_Point(50, 40), 25), rast, '8BUI', 220, 0)
            AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'averaged', ST_MapAlgebra(
        raster_1, 1,
        raster_2, 1,
        '([rast1] + [rast2]) / 2',
        '8BUI',
        'UNION'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Figure



Define our user defined function.

Code

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN ((rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN NULL;
            WHEN rast1 IS NULL THEN
                RETURN rast2;
            ELSE
                RETURN rast1;
```

```
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;
```

Prepare a test table of rasters.

Code

```
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
            UNION ALL
            SELECT 3 AS bnum,
                ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom,
    'big road' As descrip
            UNION ALL
            SELECT 1 As bnum,
                ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'),
    8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
            ),
   -- define our canvas to be 1 to 1 pixel to geometry
   canvas
    AS ( SELECT ST_AddBand(
        ST_MakeEmptyRaster(
            250,
            250,
            ST_XMin(e)::integer,
            ST_YMax(e)::integer,
            1, -1, 0, 0),
        '8BUI'::text,
        0) As rast
        FROM (SELECT ST_Extent(geom) As e,
                    Max(ST_SRID(geom)) As srid
                    from mygeoms
                    ) As foo
            )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
                FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;
```

Map algebra on single-band rasters and then collect the result with ST_AddBand.

Code

```
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union
    (canvas)'
    FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(
            m1.rast,
            m2.rast,
            'raster_mapalgebra_union(double precision, double precision, integer[], text[])'
    ::regprocedure,
            '8BUI',
            'FIRST')
                FROM map_shapes As m1 CROSS JOIN map_shapes As m2
```

```
    WHERE m1.descrip = 'canvas' AND m2.descrip <
> 'canvas' ORDER BY m2.bnum) As rasts) As foo;
```

A compact, executable callback example. The source rasters and their averaged overlap are rendered at build time.

Code

```
CREATE OR REPLACE FUNCTION pg_temp.raster_average(
    value_1 double precision,
    value_2 double precision,
    position integer[],
    VARIADIC userargs text[]
) RETURNS double precision
LANGUAGE SQL IMMUTABLE
AS 'SELECT COALESCE(($1 + $2) / 2.0, $1, $2)';

WITH canvas AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(80, 80, 0, 80, 1, -1, 0, 0, 0),
        1, '8BUI', 0, 0
    ) AS rast
), inputs AS (
    SELECT
        ST_AsRaster(ST_Buffer(ST_Point(30, 40), 25), rast, '8BUI', 80, 0)
            AS raster_1,
        ST_AsRaster(ST_Buffer(ST_Point(50, 40), 25), rast, '8BUI', 220, 0)
            AS raster_2
    FROM canvas
), variants AS (
    SELECT 'raster 1' AS title, raster_1 AS rendered FROM inputs
    UNION ALL
    SELECT 'raster 2', raster_2 FROM inputs
    UNION ALL
    SELECT 'averaged', ST_MapAlgebraFct(
        raster_1,
        raster_2,
        'pg_temp.raster_average(double precision,double precision,integer[],text[])':: ↩
    regprocedure,
        '8BUI',
        'UNION'
    ) FROM inputs
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

Fonction définie par l'utilisateur avec arguments supplémentaires

Code

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
    RETURNS double precision
    AS $$
    DECLARE
```

```
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN least(userargs[1]::integer,(rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN userargs[2]::integer;
            WHEN rast1 IS NULL THEN
                RETURN greatest(rast2, random()*userargs[3]::integer)::integer;
            ELSE
                RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(
    m1.rast,
    1,
    m1.rast,
    3,
    'raster_mapalgebra_userargs(double precision, double precision, integer[], text[])':: ↩
    regprocedure,
    '8BUI',
    'INTERSECT',
    '100', '200', '200', '0')
                FROM map_shapes As m1
    WHERE m1.descrip = 'map bands overlay fct union (canvas)';
```

The generated callback figure above demonstrates the raster alignment and overlap used by this variant; the exact callback result remains available as text.

Voir aussi

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

11.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — Version à 1 bande : Algèbre cartographique Plus proche voisin en utilisant une fonction PostgreSQL définie par l'utilisateur. Retourne un raster dont les valeurs sont le résultat d'une fonction utilisateur PLPGSQL prenant un voisinage des valeurs de la bande raster d'entrée.

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Description



AVERTISSEMENT
ST_MapAlgebraFctNgb est obsolète depuis 2.1.0. Utilisez plutôt ST_MapAlgebra (callback function version).

(version à un raster) Retourne un raster dont les valeurs sont le résultat d'une fonction utilisateur PLPGSQL prenant un voisinage de valeurs de la bande du raster d'entrée. La fonction utilisateur prend le voisinage des valeurs des pixels sous la forme d'un tableau de nombres. Pour chaque pixel, elle renvoie le résultat de la fonction utilisateur, en remplaçant la valeur du pixel actuellement traité par le résultat de la fonction utilisateur.

rast Raster sur lequel la fonction utilisateur est évaluée.

band Numéro de bande du raster à évaluer. La valeur par défaut est 1.

pixeltype Le type de pixel résultant du raster de sortie. Doit être l'un de ceux listés dans ST_BandPixelType, ou être omis ou défini à NULL. S'il n'est pas spécifié ou s'il est défini à NULL, le type de pixel du raster `rast` sera utilisé par défaut. Les résultats sont tronqués s'ils sont plus grands que ce qui est autorisé pour le type de pixel.

ngbwidth La largeur du voisinage, en nombre de cellules.

ngbheight La hauteur du voisinage, en nombre de cellules.

onerastngbuserfunc Fonction utilisateur PLPGSQL/psql à appliquer aux pixels du voisinage de la bande du raster. Le premier élément est un tableau bidimensionnel de nombres, représentant le voisinage rectangulaire du pixel

nodatamode Définit la valeur à transmettre à la fonction pour un pixel de voisinage qui est nodata ou NULL

'ignore' : toutes les valeurs NODATA rencontrées dans le voisinage sont ignorées par le calcul -- ce drapeau doit être envoyé à la fonction de rappel de l'utilisateur, et c'est la fonction de l'utilisateur qui décide comment l'ignorer.

'NULL' : toute valeur NODATA rencontrée dans le voisinage aura pour conséquence que le pixel résultant sera NULL -- la fonction de rappel de l'utilisateur est ignorée dans ce cas.

'value' : toutes les valeurs NODATA rencontrées dans le voisinage sont remplacées par le pixel de référence (celui qui se trouve au centre du voisinage). Notez que si cette valeur est NODATA, le comportement est le même que 'NULL' (pour le voisinage concerné)

args Arguments à transmettre à la fonction utilisateur.

Disponibilité : 2.0.0

Exemples

Les exemples utilisent le raster katrina chargé comme une seule tuile, tel que décrit dans https://gdal.org/user/drivers/raster/-postgisraster.html et ensuite traitée dans les exemples ST_Rescale

A simple 'callback' user function that averages up all the values in a neighborhood.

Code

```
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text[])
    RETURNS float AS
    $$
    DECLARE
        _matrix float[][];
        x1 integer;
        x2 integer;
        y1 integer;
        y2 integer;
        sum float;
    BEGIN
        _matrix := matrix;
        sum := 0;
        FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
            FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
                sum := sum + _matrix[x][y];
            END LOOP;
        END LOOP;
        RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
    END;
    $$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;
```

Apply the raster averaging callback within two pixels of each input pixel in the X and Y directions.

Code

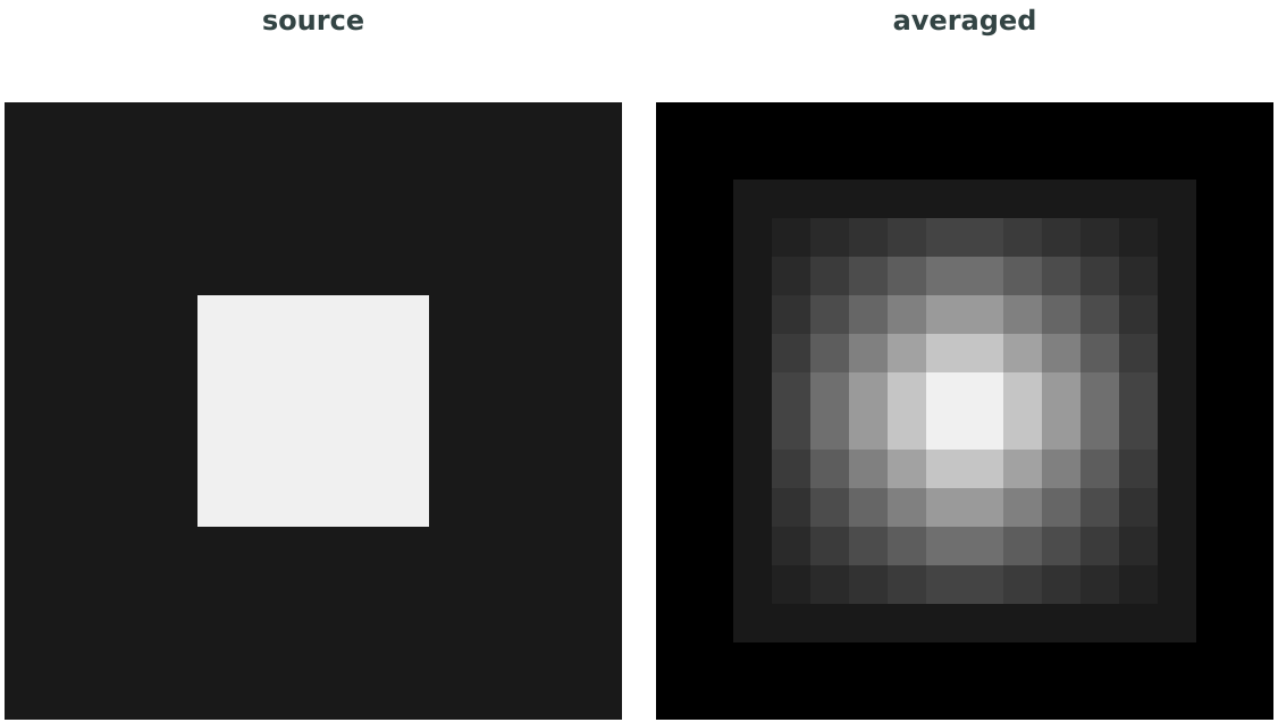
```
SELECT ST_MapAlgebraFctNgb(
        rast,
        1,
        '8BUI',
        4,
        4,
        'rast_avg(float[][], text, text[])'::regprocedure,
        'NULL',
        NULL) As nn_with_border
    FROM katrinas_rescaled
    limit 1;
```

A self-contained neighborhood average. The source raster has a sharp central square, while the result spreads those values across neighboring pixels. The source uses a non-NODATA background so `nodatamode ignore` averages background values with the square values, instead of treating the background as absent.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(16, 16, 0, 16, 1, -1, 0, 0, 0),
            1, '8BUI', 25, NULL
        ),
        1, 6, 6,
        ARRAY[
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240],
            [240, 240, 240, 240, 240, 240]
        ]::double precision[][]
    ) AS rast
), variants AS (
    SELECT 'source' AS title, rast AS rendered FROM source
    UNION ALL
    SELECT 'averaged', ST_SetBandNoDataValue(ST_MapAlgebraFctNgb(
        rast,
        1,
        '8BUI',
        2,
        2,
        'ST_Mean4ma(float[][],text,text[])'::regprocedure,
        'ignore',
        NULL
    ), NULL) FROM source
)
SELECT title, ST_AsPNG(rendered) AS image
FROM variants
ORDER BY title DESC;
```

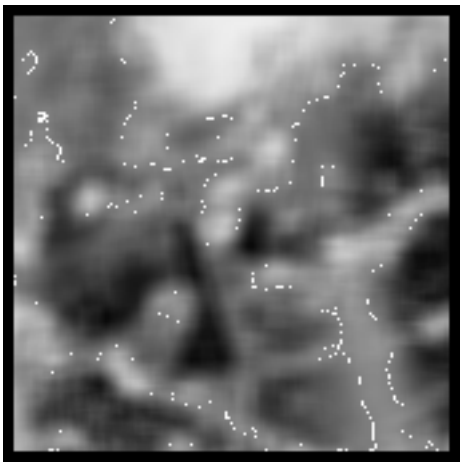
Figure



Neighborhood averaging.



Première bande du raster



Averaged result

Voir aussi

ST_MapAlgebraFct, ST_MapAlgebraExpr, ST_Rescale

11.12.12 ST_Reclass

ST_Reclass — Crée un nouveau raster composé de types de bandes reclassifiés par rapport à l'original. La bande n est la bande à modifier. Si nband n'est pas spécifié, la bande 1 est utilisée. Toutes les autres bandes sont retournées inchangées. Cas d'utilisation : convertir une bande 16BUI en 8BUI et ainsi de suite pour un rendu plus simple en tant que formats visualisables.

Synopsis

raster **ST_Reclass**(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster **ST_Reclass**(raster rast, reclassarg[] VARIADIC reclassargset);
raster **ST_Reclass**(raster rast, text reclassexpr, text pixeltype);

Description

Creates a new raster formed by applying a reclassification operation defined by the `reclassexpr` on the input raster (`rast`). Refer to reclassarg for the description of reclassification expressions. If no `band` is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. The bands of the new raster have pixel type of `pixeltype`. If `reclassargset` is specified then each `reclassarg` defines the type of the target band. Bands not designated are returned unchanged.

Disponibilité : 2.0.0

Exemples

Basic.

Crée un nouveau raster à partir de l'original où la bande 2 est convertie de 8BUI à 4BUI et où toutes les valeurs entre 101 et 254 sont fixées à nodata.

Code

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast, 2, '0-87:1-10, 88-100:11-15,  ↩
    101-254:0-0', '4BUI', 0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast, 2, i, j) As origval,
    ST_Value(reclass_rast, 2, i, j) As reclassval,
    ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
col | row | origval | reclassval | reclassval_include_nodata
-----+-----+---------+------------+--------------------------
   1 |   1 |      78 |          9 |                         9
   2 |   1 |      98 |         14 |                        14
   3 |   1 |     122 |            |                         0
   1 |   2 |      96 |         14 |                        14
   2 |   2 |     118 |            |                         0
   3 |   2 |     180 |            |                         0
   1 |   3 |      99 |         15 |                        15
   2 |   3 |     112 |            |                         0
   3 |   3 |     169 |            |                         0
```

Advanced using multiple reclassargs.

Crée un nouveau raster à partir de l'original où les bandes 1, 2, 3 sont respectivement converties en 1BB, 4BUI, 4BUI et reclassifiées. Cette opération utilise l'argument variadique `reclassarg` qui peut prendre en entrée un nombre variable de reclassargs (théoriquement autant de bandes que vous avez)

Code

```
UPDATE dummy_rast SET reclass_rast =
    ST_Reclass(
        rast,
        ROW(2, '0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI', NULL)::reclassarg,
        ROW(1, '0-253]:1, 254:0', '1BB', NULL)::reclassarg,
        ROW(3, '0-70]:1, (70-86:2, [86-150):3, [150-255:4', '4BUI', NULL)::reclassarg
        ) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast, 1, i, j) As ov1, ST_Value(reclass_rast, 1, i, j) ↩
    As rv1,
    ST_Value(rast, 2, i, j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
    ST_Value(rast, 3, i, j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1, 3) AS j
WHERE rid = 2;
```

Output

```
col | row | ov1 | rv1 | ov2 | rv2 | ov3 | rv3
----+-----+-----+-----+-----+-----+-----+-----
  1 |   1 | 253 |   1 |  78 |   9 |  70 |   1
  2 |   1 | 254 |   0 |  98 |  14 |  86 |   3
  3 |   1 | 253 |   1 | 122 |   0 | 100 |   3
  1 |   2 | 253 |   1 |  96 |  14 |  80 |   2
  2 |   2 | 254 |   0 | 118 |   0 | 108 |   3
  3 |   2 | 254 |   0 | 180 |   0 | 162 |   4
  1 |   3 | 250 |   1 |  99 |  15 |  90 |   3
  2 |   3 | 254 |   0 | 112 |   0 | 108 |   3
  3 |   3 | 254 |   0 | 169 |   0 | 175 |   4
```

Advanced Map a single band 32BF raster to multiple viewable bands.

Crée un nouveau raster à 3 bandes (8BUI, 8BUI, 8BUI visualisable) à partir d'un raster qui n'a qu'une seule bande de 32bf

Code

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
    set rast_view = ST_AddBand(
        NULL,
        ARRAY[
    ST_Reclass(rast, 1, '0.1-10]:1-10,9-10]:11,(11-33:0'::text, '8BUI'::text, 0),
    ST_Reclass(rast, 1, '11-33):0-255,[0-32:0,(34-1000:0'::text, '8BUI'::text, 0),
    ST_Reclass(rast, 1, '0-32]:0,(32-100:100-255'::text, '8BUI'::text, 0)
    ]
    );
```

Voir aussi

ST_AddBand, ST_Band, ST_BandPixelType, ST_MakeEmptyRaster, reclassarg, ST_Value

11.12.13 ST_ReclassExact

ST_ReclassExact — Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.

Synopsis

raster **ST_ReclassExact**(raster rast, double precision[] inputvalues, double precision[] outputvalues, integer bandnumber=1, text pixeltype=32BF, double precision nodatavalue=NULL);

Description

Creates a new raster formed by applying a reclassification operation defined by the `inputvalues` and `outputvalues` arrays. Pixel values found in the input array are mapped to the corresponding value in the output array. All other pixel values are mapped to the `nodatavalue`.

The output pixel type defaults to float, but can be specified using the `pixeltype` parameter. If no `bandnumber` is specified band 1 is assumed.

The new raster will have the same georeference, width, and height as the original raster. Bands not designated are returned unchanged.

Disponibilité : 3.6.0

Example: Basic

Create a small raster and map its pixels to new values.

Create a raster with four pixels: first row 1 2, second row 3 4.

Code

```
CREATE TABLE reclassexact (
    id integer,
    rast raster
);
INSERT INTO reclassexact (id, rast)
SELECT 1, ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(2, -- width in pixels
        2, -- height in pixels
        0, -- upper-left x-coordinate
        0, -- upper-left y-coordinate
        1, -- pixel size in x-direction
        -1, -- pixel size in y-direction (negative for north-up)
        0, -- skew in x-direction
        0, -- skew in y-direction
        4326  -- SRID (e.g., WGS 84)
      ),
      '32BUI'::text, -- pixel type (e.g., '32BF' for float, '8BUI' for unsigned 8-bit int)
      0.0, -- initial value for the band (e.g., 0.0 or a no-data value)
      -99            -- nodatavalue
    ),
    1, -- band number (usually 1 for single-band rasters)
    1, -- x origin for setting values (usually 1)
    1, -- y origin for setting values (usually 1)
    ARRAY[
      ARRAY[1, 2],
      ARRAY[3, 4]
    ]::double precision[][] -- 2D array of values
  );
```

Reclass the values to new values and dump the values of the new raster for display.

Code

```
WITH rc AS (
  SELECT ST_ReclassExact(rast, -- input raster
    ARRAY[4, 3, 2, 1], -- input map
    ARRAY[14, 13, 12, 11], -- output map
    1, -- band number to remap
    '32BUI'              -- output raster pixel type
    ) AS rast
```

```
  FROM reclassexact
  WHERE id = 1
  )
SELECT 'rce-1', (ST_DumpValues(rc.rast)).*
FROM rc;
```

Voir aussi

ST_Reclass, ST_AddBand, ST_Band, ST_MakeEmptyRaster

11.12.14 ST_Union

ST_Union — Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.

Synopsis

raster **ST_Union**(setof raster rast);
raster **ST_Union**(setof raster rast, unionarg[] unionargset);
raster **ST_Union**(setof raster rast, integer nband);
raster **ST_Union**(setof raster rast, text uniontype);
raster **ST_Union**(setof raster rast, integer nband, text uniontype);

Description

Retourne l'union d'un ensemble de tuiles raster en un seul raster, composé d'au moins une bande. L'étendue du raster résultant est l'étendue de l'ensemble. Dans le cas d'une intersection, la valeur résultante est définie par `uniontype` parmi : LAST (par défaut), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.



Note
Pour pouvoir calculer l'union de raster, tous les rasters doivent avoir le même alignement. Voir ST_SameAlignment et ST_NotSameAlignmentReason pour plus de détails. Une façon de résoudre les problèmes d'alignement est d'utiliser ST_Resample et d'utiliser le même raster de référence pour l'alignement.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Vitesse améliorée (entièrement en C).

Disponibilité : 2.1.0 Ajout de la variante ST_Union(rast, unionarg).

Amélioration : 2.1.0 ST_Union(rast) (variante 1) permet l'union sur toutes les bandes de tous les rasters d'entrée. Les versions précédentes de PostGIS ne prenaient en compte que la première bande.

Amélioration : 2.1.0 ST_Union(rast, uniontype) (variante 4) permet l'union de toutes les bandes de tous les rasters d'entrée.

Exemples

Reconstitute a single band chunked raster tile.

This creates a single band from first band of raster tiles. That form the original file system tile.

Code

```
SELECT filename, ST_Union(rast, 1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

Return a multi-band raster that is the union of tiles intersecting geometry.

This creates a multi band raster collecting all the tiles that intersect a line. Note that In 2.0, this would have just returned a single band raster. , new union works on all bands by default. This is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]::unionarg[].

Code

```
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771,230500 88772)', 26986) ) ↩
    ;
```

Select and reorder bands while unioning tiles that intersect a geometry.

Nous utilisons ici la syntaxe plus longue si nous ne voulons qu'un sous-ensemble de bandes ou si nous voulons changer l'ordre des bandes

This creates a multi band raster collecting all the tiles that intersect a line.

Code

```
SELECT ST_Union(rast, ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771,230500 88772)', 26986) ) ↩
    ;
```

Voir aussi

unionarg, ST_Envelope, ST_ConvexHull, ST_Clip, ST_Union

11.13 Fonctions de rappel intégrées d'algèbre cartographique

11.13.1 ST_Distinct4ma

ST_Distinct4ma — Fonction de traitement des données raster qui calcule le nombre de valeurs de pixels uniques dans un voisinage.

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule le nombre de valeurs de pixels uniques dans un voisinage de pixels.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Distinct4ma(float[][],text,text[]) ↩
    '::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |        3
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.2 ST_InvDistWeight4ma

ST_InvDistWeight4ma — Fonction de traitement des données raster qui interpole la valeur d'un pixel à partir de son voisinage.

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule une valeur interpolée pour un pixel en utilisant la méthode de la distance inverse pondérée.

Deux paramètres facultatifs peuvent être fournis par l'intermédiaire de `userargs`. Le premier paramètre est le facteur de puissance (variable k dans l'équation ci-dessous) entre 0 et 1, utilisé dans l'équation de la pondération de la distance inverse. S'il n'est pas spécifié, la valeur par défaut est 1. Le second paramètre est le pourcentage de pondération, appliqué uniquement lorsque la valeur du pixel d'intérêt est incluse dans la valeur interpolée du voisinage. S'il n'est pas spécifié et que le pixel d'intérêt a une valeur, cette valeur est retournée.

The basic inverse-distance weighted estimate is: z = (Σᵢ zᵢ / dᵢᵏ) / (Σᵢ 1 / dᵢ&#x1d4
Here `zᵢ` is a neighboring pixel value, `dᵢ` is its distance from the pixel being estimated, and `k` is a real-valued power factor between 0 and 1.



Note
Cette fonction est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , NULL::double precision, -9999),
        1, 1, 1, ARRAY[
            [10, 10, 10],
            [10, NULL, 20],
            [20, 20, 20]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(
    ST_MapAlgebra(
        rast,
        1,
        'ST_InvDistWeight4ma(double precision[][][],integer[][],text[])'::regprocedure,
        '32BF',
        'FIRST',
        NULL,
        1,
        1
    ),
    2, 2
) AS interpolated_value
FROM foo;
```

Output

```
15
```

Voir aussi

ST_MapAlgebra (callback function version), ST_MinDist4ma

11.13.3 ST_Max4ma

ST_Max4ma — Fonction de traitement des données raster qui calcule la valeur maximale d'un pixel dans un voisinage.

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule la valeur maximale d'un pixel dans un voisinage de pixels.

Pour la variante 2, une valeur de substitution pour les pixels NODATA peut être spécifiée en passant cette valeur à userargs.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Max4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |      254
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.4 ST_Mean4ma

ST_Mean4ma — Fonction de traitement des données raster qui calcule la valeur moyenne d'un pixel dans un voisinage.

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule la valeur moyenne d'un pixel dans un voisinage de pixels.

Pour la variante 2, une valeur de substitution pour les pixels NODATA peut être spécifiée en passant cette valeur à userargs.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Variant 1.

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_Mean4ma(float[][],text,text[])' ↩
    ::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Variant 2.

Code

```
SELECT
  rid,
  ST_Value(
      ST_MapAlgebra(rast, 1, 'ST_Mean4ma(double precision[][][], integer[][], text[])':: ↩
    regprocedure, '32BF', 'FIRST', NULL, 1, 1),
      2,
      2)
FROM dummy_rast
 WHERE rid = 2;
```

Output

```
rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Range4ma, ST_StdDev4ma

11.13.5 ST_Min4ma

ST_Min4ma — Fonction de traitement des données raster qui calcule la valeur minimale d'un pixel dans un voisinage.

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule la valeur minimale d'un pixel dans un voisinage de pixels.

Pour la variante 2, une valeur de substitution pour les pixels NODATA peut être spécifiée en passant cette valeur à userargs.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Min4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |      250
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.6 ST_MinDist4ma

ST_MinDist4ma — Fonction de traitement des données raster qui renvoie la distance minimale (en nombre de pixels) entre le pixel d'intérêt et un pixel voisin ayant une valeur.

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Retourne la plus petite distance (en nombre de pixels) entre le pixel d'intérêt et le pixel le plus proche ayant une valeur dans le voisinage.



Note
Le but de cette fonction est de fournir un point de données informatif qui aide à déduire l'utilité de la valeur interpolée du pixel d'intérêt à partir de ST_InvDistWeight4ma. Cette fonction est particulièrement utile lorsque le voisinage est peu peuplé.



Note
Cette fonction est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , NULL::double precision, -9999),
        1, 1, 1, ARRAY[
            [10, 10, 10],
            [10, NULL, 20],
            [20, 20, 20]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(
    ST_MapAlgebra(
        rast,
        1,
        'ST_MinDist4ma(double precision[][][],integer[][],text[])'::regprocedure,
        '32BF',
        'FIRST',
        NULL,
        1,
        1
    ),
    2, 2
) AS min_distance
FROM foo;
```

Output

```
1
```

Voir aussi

ST_MapAlgebra (callback function version), ST_InvDistWeight4ma

11.13.7 ST_Range4ma

ST_Range4ma — Fonction de traitement des données raster qui calcule la plage de valeurs des pixels dans un voisinage.

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule la plage de valeurs des pixels dans un voisinage de pixels.

Pour la variante 2, une valeur de substitution pour les pixels NODATA peut être spécifiée en passant cette valeur à userargs.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, NULL, 1, 1, 'ST_Range4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |        4
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.8 ST_StdDev4ma

ST_StdDev4ma — Fonction de traitement des données raster qui calcule l'écart type des valeurs des pixels dans un voisinage.

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule l'écart type des valeurs des pixels dans un voisinage de pixels.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_StdDev4ma(float[][],text,text[]) ↩
    '::regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid |     st_value
-----+------------------
   2 | 1.30170822143555
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

11.13.9 ST_Sum4ma

ST_Sum4ma — Fonction de traitement des données raster qui calcule la somme de toutes les valeurs de pixels dans un voisinage.

Synopsis

float8 **ST_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Description

Calcule la somme de toutes les valeurs des pixels dans un voisinage de pixels.

Pour la variante 2, une valeur de substitution pour les pixels NODATA peut être spécifiée en passant cette valeur à userargs.



Note
La variante 1 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebraFctNgb.



Note
La variante 2 est une fonction de rappel spécialisée à utiliser comme paramètre de rappel pour ST_MapAlgebra (callback function version).



AVERTISSEMENT
L'utilisation de la variante 1 est non recommandée, ST_MapAlgebraFctNgb étant dépréciée depuis 2.1.0.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Ajout de la variante 2

Exemples

Code

```
SELECT
    rid,
    ST_Value(ST_MapAlgebraFctNgb(rast, 1, '32BF', 1, 1, 'ST_Sum4ma(float[][],text,text[])':: ↩
    regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
```

Output

```
rid | st_value
-----+----------
   2 |     2279
(1 row)
```

Voir aussi

ST_MapAlgebraFctNgb, ST_MapAlgebra (callback function version), ST_Min4ma, ST_Max4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

11.14 Traitement des données raster : MNT (élévation)

11.14.1 ST_Aspect

ST_Aspect — Retourne l'exposition (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.

Synopsis

raster **ST_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
raster **ST_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);

Description

Retourne l'exposition (par défaut, en degrés) d'une bande raster d'élévation. Utilise l'algèbre cartographique et applique l'équation d'exposition aux pixels voisins.

`units` indique l'unité de l'exposition. Les valeurs possibles sont : RADIANS, DEGREES (par défaut).

Lorsque `units` = RADIANS, les valeurs sont comprises entre 0 et 2 * pi radians, mesurés dans le sens des aiguilles d'une montre à partir du nord.

Lorsque `units` = DEGREES, les valeurs sont comprises entre 0 et 360 degrés, mesurés dans le sens des aiguilles d'une montre à partir du nord.

Si la pente du pixel est nulle, l'exposition du pixel est -1.



Note
Pour plus d'informations sur la pente, l'exposition et l'ombrage, veuillez consulter ESRI - How hillshade works et ERDAS Field Guide - Aspect Images.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Utilise ST_MapAlgebra() et ajout du paramètre optionnel `interpolate_nodata`

Changement : 2.1.0 Dans les versions précédentes, les valeurs retournées étaient en radians. Désormais, les valeurs retournées sont par défaut en degrés

Exemples

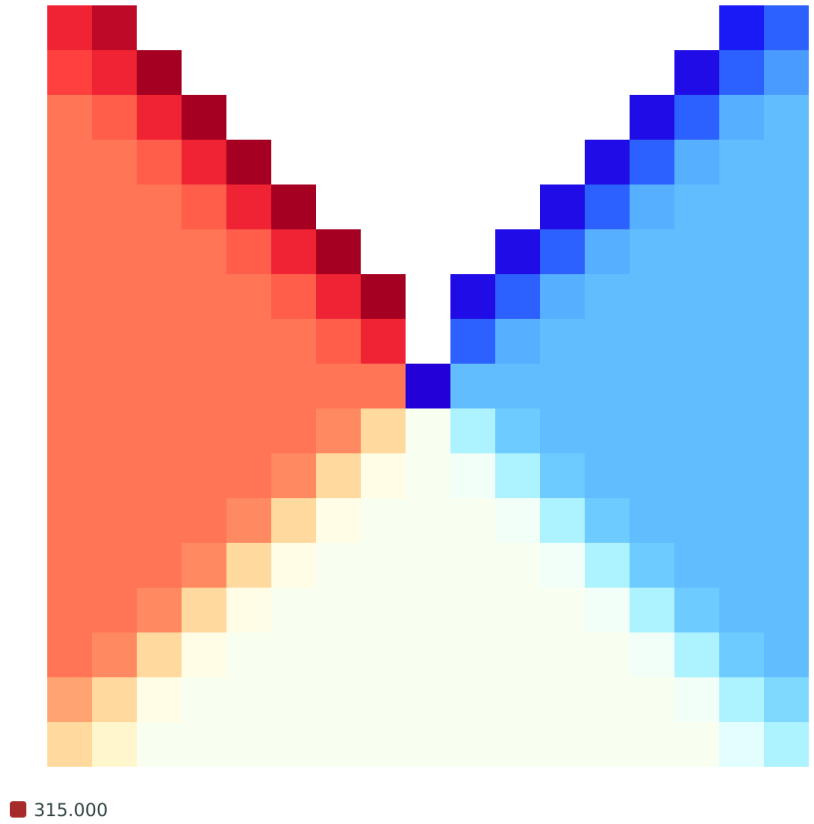
Variant 1.

This symmetric peak highlights the expected compass directions at the corners and the `-1` aspect at the summit. The generated PNG shows the full aspect surface without relying on a manual illustration. The generated source raster is large enough for the PNG to show the directional field around the summit.

Code

```
WITH pixels AS (
    SELECT
        x,
        y,
        (20 - greatest(abs(x - 9), abs(y - 9)))::double precision AS value
    FROM generate_series(1, 17) AS y
    CROSS JOIN generate_series(1, 17) AS x
), rows AS (
    SELECT
        y,
        array_agg(value ORDER BY x) AS row_values
    FROM pixels
    GROUP BY y
), source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(17, 17, 0, 0, 1, -1, 0, 0, 0),
            1, '32BF', 0, -9999
        ),
        1, 1, 1,
        array_agg(row_values ORDER BY y)::double precision[][]
    ) AS rast
    FROM rows
), aspect AS (
    SELECT ST_Aspect(rast, 1, '32BF') AS rast
    FROM source
)
SELECT
    round(ST_Value(rast, 1, 1, 1)::numeric, 3) AS northwest_degrees,
    round(ST_Value(rast, 1, 9, 9)::numeric, 3) AS summit_degrees,
    round(ST_Value(rast, 1, 17, 17)::numeric, 3) AS southeast_degrees,
    ST_AsPNG(ST_ColorMap(rast, 1, 'bluered')) AS image
FROM aspect;
```

Figure



Variant 2.

Complete example of tiles of a coverage.

Code

```
WITH foo AS (
    SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Voir aussi

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

11.14.2 ST_HillShade

ST_HillShade — Retourne l'hypothétique éclairage d'une bande raster d'élévation en utilisant les valeurs d'azimut, d'altitude, de luminosité et d'échelle fournies.

Synopsis

raster **ST_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Description

Retourne l'hypothétique éclairage d'une bande raster d'élévation en utilisant les données d'entrée d'azimut, d'altitude, de luminosité et d'échelle. Utilise l'algèbre cartographique et applique l'équation d'ombrage aux pixels voisins. Les valeurs des pixels retournées sont comprises entre 0 et 255.

`azimuth` est une valeur comprise entre 0 et 360 degrés, mesurée dans le sens des aiguilles d'une montre à partir du nord.

`altitude` est une valeur comprise entre 0 et 90 degrés, où 0 degré correspond à l'horizon et 90 degrés à la verticale.

`max_bright` est une valeur comprise entre 0 et 255, 0 correspondant à une absence de luminosité et 255 à une luminosité maximale.

`scale` est le rapport entre les unités verticales et les unités horizontales. Pour Feet:LatLon, utiliser scale=370400, pour Meters:LatLon utiliser scale=111120.

Si `interpolate_nodata` est TRUE, les valeurs des pixels NODATA du raster d'entrée seront interpolées à l'aide de ST_InvDistWeight4ma avant de calculer l'éclairage.



Note
Pour plus d'informations sur l'ombrage, veuillez consulter How hillshade works.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Utilise ST_MapAlgebra() et ajout du paramètre optionnel `interpolate_nodata`

Changement : 2.1.0 Dans les versions précédentes, l'azimut et l'altitude étaient exprimés en radians. Désormais, l'azimut et l'altitude sont exprimés en degrés

Exemples

Variant 1.

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
```

```
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_HillShade(rast, 1, '32BF'))
FROM foo
```

```
                                                
```

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Calcule la rugosité d'un MNT en soustrayant le maximum du minimum pour une zone donnée.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(ST_Roughness(rast, 1, '32BF'), 3, 3) AS roughness
FROM foo;
```

Output

```
13
```

Voir aussi

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

11.14.4 ST_Slope

ST_Slope — Retourne la pente (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.

Synopsis

raster **ST_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Description

Retourne la pente (par défaut, en degrés) d'une bande raster d'élévation. Utilise l'algèbre cartographique et applique l'équation de pente aux pixels voisins.

`units` indique l'unité de la pente. Les valeurs possibles sont : RADIANS, DEGREES (par défaut), PERCENT.

`scale` est le rapport entre les unités verticales et les unités horizontales. Pour Feet:LatLon, utiliser scale=370400, pour Meters:LatLon utiliser scale=111120.

Si `interpolate_nodata` est TRUE, les valeurs des pixels NODATA du raster d'entrée seront interpolées à l'aide de ST_InvDistWeight4ma avant de calculer la pente de la surface.



Note
Pour plus d'informations sur la pente, l'exposition et l'ombrage, veuillez consulter ESRI - How hillshade works et ERDAS Field Guide - Slope Images.

Disponibilité : 2.0.0

Amélioration : 2.1.0 Utilise ST_MapAlgebra() et ajout des paramètres optionnels `units`, `scale`, `interpolate_nodata`

Changement : 2.1.0 Dans les versions précédentes, les valeurs retournées étaient en radians. Désormais, les valeurs retournées sont par défaut en degrés

Exemples

Variant 1.

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

                            st_dumpvalues
```

Output

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------------------------------------


-------------------------------------------------------------------
 ↩
    (1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,35.2643890380808

{26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.2643890380859,36.8698959350586,35.26

5681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

Variant 2.

Complete example of tiles of a coverage.

Code

```
WITH foo AS (
    SELECT ST_Tile(ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Voir aussi

ST_MapAlgebra (callback function version), ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Aspect

11.14.5 ST_TPI

ST_TPI — Retourne un raster avec l'index de position topographique (TPI) calculé.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

Calcule l'index de position topographique (TPI), qui est défini comme la moyenne focale avec un rayon de un moins la cellule centrale.



Note
Cette fonction ne prend en charge qu'un rayon focal de un.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(ST_TPI(rast, 1, '32BF'), 3, 3) AS tpi
FROM foo;
```

Output

```
3.75
```

Voir aussi

ST_MapAlgebra (callback function version), ST_TRI, ST_Roughness, ST_Slope, ST_HillShade, ST_Aspect

11.14.6 ST_TRI

ST_TRI — Retourne un raster avec l'indice de rugosité du terrain (TRI) calculé.

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Description

L'indice de rugosité du terrain est calculé en comparant le pixel central à ses voisins, en prenant les valeurs absolues des différences et en faisant la moyenne du résultat.



Note
Cette fonction ne prend en charge qu'un rayon focal de un.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT ST_SetValues(ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF' ↩
    , 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 2, 3, 3],
```

```
            [1, 2, 4, 15, 3],
            [2, 4, 10, 7, 5],
            [3, 5, 7, 6, 4],
            [3, 3, 5, 4, 4]
        ]::double precision[][]
    ) AS rast
)
SELECT ST_Value(ST_TRI(rast, 1, '32BF'), 3, 3) AS tri
FROM foo;
```

Output

```
5
```

Voir aussi

ST_MapAlgebra (callback function version), ST_Roughness, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

11.14.7 ST_InterpolateRaster

ST_InterpolateRaster — Interpole une surface quadrillée à partir d'un ensemble de points 3-d, en utilisant les coordonnées X et Y des points sur la grille et la coordonnée Z des points pour l'élévation des points.

Synopsis

raster **ST_InterpolateRaster**(geometry input_points, text algorithm_options, raster template, integer template_band_num=1);

Description

Interpole une surface quadrillée à partir d'un ensemble de points 3-d, en utilisant les coordonnées X et Y des points sur la grille et la coordonnée Z des points pour l'élévation des points. 5 algorithmes d'interpolations sont disponibles : inverse de la distance, inverse de la distance avec plus proche voisin, moyenne mobile, plus proche voisin, et interpolation linéaire. Voir gdal_grid documentation pour plus de détails sur ces algorithmes et leurs paramètres. Pour plus d'informations sur le calcul des interpolations, voir le tutorial GDAL grid.

Les paramètres d'entrée sont :

`input_points` Les points pour effectuer l'interpolation. Toute géometrie avec une dimension Z est acceptée ; tous les points de la géométrie seront utilisés.

`algorithm_options` Une chaîne définissant l'algorithme et ses options, au format utilisable par gdal_grid. Par exemple, pour une interpolation en inverse de la distance avec un lissage de 2, la valeur sera "invdist:smoothing=2.0"

`template` Un gabarit raster pour la création du raster de sortie. Les largeur, hauteur, taille des pixels, étendue spatiale et type des pixels seront définis à partir de ce gabarit.

`template_band_num` Indice de la bande raster du gabarit raster à utiliser ; par défaut, la première bande sera utilisée.

Disponibilité : 3.2.0

Exemple

Code

```
SELECT ST_InterpolateRaster('MULTIPOINT(10.5 9.5 1000,11.5 8.5 1000,10.5 8.5 500,11.5 9.5 ↩
    500)'::geometry,
    'invdist:smoothing:2.0',
    ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```

Voir aussi

ST_Contour

11.14.8 ST_Contour

ST_Contour — Génère un ensemble de courbes de niveau vectorielles depuis la bande raster spécifiée, en utilisant l'algorithme de contour GDAL.

Synopsis

setof record **ST_Contour**(raster rast, integer bandnumber=1, double precision level_interval=100.0, double precision level_base=0.0, double precision[] fixed_levels=ARRAY[], boolean polygonize=false);

Description

Génère un ensemble de courbes de niveau vectorielles depuis la bande raster spécifiée, en utilisant l'algorithme de contour GDAL.

Si le paramètre `fixed_levels` est un tableau non vide, les paramètres `level_interval` et `level_base` sont ignorés.

Les paramètres d'entrée sont :

`rast` Le raster pour générer le contour de

`bandnumber` La bande pour générer le contour de

`level_interval` Intervalle d'élévation entre les courbes de niveau générées

`level_base` La "base" par rapport à laquelle les intervalles de contour sont appliqués ; elle est normalement égale à zéro, mais peut être différente. Pour générer des courbes de niveau de 10 m à 5, 15, 25, ... le LEVEL_BASE serait de 5.

`fixed_levels` Intervalle d'élévation entre les courbes de niveau générées

`polygonize` Si `true`, des polygones de contour seront créés, plutôt que des lignes de polygone.

La valeur de retour est un ensemble de résultats avec les attributs suivants :

`geom` La géométrie de la ligne de contour.

`id` Un identifiant unique de la ligne de contour, déterminée par GDAL.

`value` La valeur raster représentée par la ligne. Pour un modèle numérique de terrain, la valeur correspond à l'élévation du contour.

Disponibilité : 3.2.0

Exemple

Code

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT geom, id, value
FROM c;
```

Voir aussi

ST_InterpolateRaster

11.15 Traitement des données raster : raster vers géométrie

11.15.1 Box3D

Box3D — Retourne la représentation 3d de la boîte englobante du raster.

Synopsis

box3d **Box3D**(raster rast);

Description

Retourne la boîte représentant l'étendue du raster.

Le polygone est défini par les points des coins de la boîte englobante ((`MINX`, `MINY`), (`MAXX`, `MAXY`))

Changement : 2.0.0 Dans les versions antérieures à la version 2.0, utilisait le type box2d au lieu de box3d. Comme box2d est un type obsolète, il a été remplacé par box3d.

Exemples

Code

```
SELECT
    rid,
    Box3D(rast) AS rastbox
FROM dummy_rast;
```

Output

```
rid |        rastbox
----+--------------------------------------------------
1   | BOX3D(0.5 0.5 0,20.5 60.5 0)
2   | BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)
```

Voir aussi

ST_Envelope

11.15.2 ST_ConvexHull

ST_ConvexHull — Retourne l'enveloppe convexe du raster, en incluant les valeurs de pixels égales à BandNoDataValue. Pour les données raster de forme régulière et sans obliquité, cette fonction donne le même résultat que ST_Envelope ; elle n'est donc utile que pour les données raster de forme irrégulière ou inclinée.

Synopsis

geometry **ST_ConvexHull**(raster rast);

Description

Retourne l'enveloppe convexe du raster, en incluant les valeurs de pixels égales à BandNoDataValue. Pour les données raster de forme régulière et sans obliquité, cette fonction donne plus ou moins le même résultat que ST_Envelope ; elle n'est donc utile que pour les données raster de forme irrégulière ou inclinée.



Note
ST_Envelope arrondit les coordonnées à l'inférieur, et ajoute donc un petit tampon autour du raster. La réponse est donc subtilement différente de ST_ConvexHull qui n'arrondit pas les coordonnées.

Exemples

See `doc/development/internals/raster-affine.md` for the scale, skew, and rotation terms that make the convex hull and envelope differ for skewed rasters.

For regular non-skewed rasters, the envelope and convex hull are more or less the same.

Code

```
SELECT ST_ConvexHull(rast) As convhull, ST_Envelope(rast) As env
FROM dummy_rast WHERE rid=1;
```

Output

```
convhull                        |               env
--------------------------------------------------------+-----------------------------------
 POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0))
```

Now we skew the raster. Note how the convex hull and envelope are now different.

Code

```
SELECT ST_ConvexHull(rast) As convhull, ST_Envelope(rast) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
    FROM dummy_rast WHERE rid=1) As foo;
```

Output

```
convhull                        |               env
--------------------------------------------------------+-----------------------------------
 POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0))
```

Voir aussi

ST_Envelope, ST_MinConvexHull, ST_ConvexHull, ST_AsText

11.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — Retourne un ensemble d'enregistrements de type geomval (geom, val), à partir d'une bande raster donnée. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE);

Description

Il s'agit d'une fonction de retour d'ensemble (SRF). Elle retourne un ensemble d'enregistrements geomval, composés par une géométrie (geom) et une valeur de bande de pixels (val). Chaque polygone est l'union de tous les pixels de cette bande qui ont la même valeur de pixel, indiquée par val.

ST_DumpAsPolygon est utile pour polygoniser les données raster. C'est l'inverse d'un GROUP BY, en ce sens qu'elle crée de nouvelles lignes. Par exemple, cette fonction peut être utilisée pour décomposer un seul raster en plusieurs POLYGONS/MULTIPOLYGONS.

Changement 3.3.0, la validation et la correction sont désactivées pour améliorer les performances. Peut donc retourner des géométries non valides.

Changed 3.7.0, the polygonization honours PostgreSQL interrupts so cancellations and statement timeouts halt processing promptly.

Disponibilité : Nécessite GDAL 1.7 ou plus.



Note
Si une valeur nodata est définie pour une bande, les pixels ayant cette valeur ne seront pas retournés, sauf si exclude_nodata_value=false.



Note
Si vous ne voulez que le nombre de pixels ayant une valeur donnée dans un raster, il est plus rapide d'utiliser ST_ValueCount.



Note
Cette fonction est différente de ST_PixelAsPolygons, où une géométrie est retournée pour chaque pixel, quelle que soit la valeur du pixel.

Exemples

This syntax uses a LATERAL join.

Code

```
SELECT val,geom As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;
```

Output

```
val |                                                                        geomwkt
-----+-------------------------------------------------------------------------------
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,
        3427928 5793243.95,3427927.95 5793243.95))
 250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,
        3427927.8 5793243.9,3427927.75 5793243.9))
 250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,
        3427927.85 5793243.8, 3427927.8 5793243.8))
 251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,
        3427927.8 5793243.85,3427927.75 5793243.85))
```

Voir aussi

geomval, ST_AsRasterAgg, ST_Value, ST_Polygon, ST_ValueCount

11.15.4 ST_Envelope

ST_Envelope — Retourne la représentation polygonale de l'étendue du raster.

Synopsis

geometry **ST_Envelope**(raster rast);

Description

Retourne la représentation polygonale de l'étendue du raster, dans les unités du système de référence spatial défini par srid. Il s'agit d'une boîte de délimitation minimale float8 représentée sous la forme d'un polygone.

Le polygone est défini par les points des coins de la boîte englobante ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`))

Exemples

Code

```
SELECT rid, ST_Envelope(rast) As envgeomwkt
FROM dummy_rast;
```

Output

```
rid |                                        envgeomwkt
-----+-------------------------------------------------------------------------
   1 | POLYGON((0 0,20 0,20 60,0 60,0 0))
   2 | POLYGON((3427927 5793243,3427928 5793243,
        3427928 5793244,3427927 5793244, 3427927 5793243))
```

Voir aussi

ST_Envelope, ST_AsText, ST_SRID

11.15.5 ST_MinConvexHull

ST_MinConvexHull — Retourne la géométrie de l'enveloppe convexe du raster en excluant les pixels NODATA.

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

Description

Retourne la géométrie de l'enveloppe convexe du raster en excluant les pixels NODATA. Si `nband` est NULL, toutes les bandes du raster sont prises en compte.

Disponibilité : 2.1.0

Exemples

Code

```
WITH foo AS (
    SELECT
        ST_SetValues(ST_SetValues(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1,  ↩
    -1, 0, 0, 0), 1, '8BUI', 0, 0), 2, '8BUI', 1, 0),
                1, 1, 1,
                ARRAY[
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 1],
                    [0, 0, 0, 1, 1, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0]
                ]::double precision[][]
            ),
            2, 1, 1,
            ARRAY[
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0]
            ]::double precision[][]
        ) AS rast
)
SELECT title, geom AS geom
FROM foo
CROSS JOIN LATERAL (VALUES
    ('raster hull', ST_ConvexHull(rast)),
    ('all bands', ST_MinConvexHull(rast)),
    ('band 1', ST_MinConvexHull(rast, 1)),
    ('band 2', ST_MinConvexHull(rast, 2))
) AS variants(title, geom);
```

Output

```
title    |                geom
-------------+--------------------------------------
 raster hull | POLYGON((0 0,9 0,9 -9,0 -9,0 0))
 all bands   | POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3))
 band 1      | POLYGON((3 -3,9 -3,9 -6,3 -6,3 -3))
 band 2      | POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))
```

Figure



Voir aussi

ST_Envelope, ST_ConvexHull, ST_ConvexHull, ST_AsText

11.15.6 ST_Polygon

ST_Polygon — Retourne une géométrie multipolygonale formée par l'union des pixels dont la valeur est différente de nodata. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Description

Changement 3.3.0, la validation et la correction sont désactivées pour améliorer les performances. Peut donc retourner des géométries non valides.

Disponibilité : 0.1.6 Nécessite GDAL 1.7 ou plus.

Amélioration : 2.1.0 Vitesse améliorée (entièrement en C) et le multipolygone retourné est garanti d'être valide.

Changement : 2.1.0 Dans les versions précédentes, retournait parfois un polygone, retourne désormais toujours un multipolygon.

Exemples

Compare polygonization of every pixel with polygonization after zero is assigned as the NODATA value. The latter preserves the gap formed by zero-valued cells.

Code

```
WITH source AS (
    SELECT ST_SetValues(
        ST_AddBand(
            ST_MakeEmptyRaster(4, 4, 0, 4, 1, -1, 0, 0, 0),
            1, '8BUI', 1, NULL
        ),
        1, 1, 1,
        ARRAY[
            [1, 1, 1, 1],
            [1, 0, 0, 1],
            [1, 1, 0, 1],
            [1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
), variants AS (
    SELECT 'all pixels' AS title, ST_Polygon(rast) AS geom
    FROM source
    UNION ALL
    SELECT
        'zero excluded',
        ST_Polygon(ST_SetBandNoDataValue(rast, 1, 0))
    FROM source
)
SELECT title, geom AS geom
FROM variants
ORDER BY CASE title
    WHEN 'all pixels' THEN 1
    ELSE 2
END;
```

Figure



Voir aussi

ST_Value, ST_DumpAsPolygons

11.15.7 ST_IntersectionFractions

ST_IntersectionFractions — Calculates the fraction of each raster cell that is covered by a given geometry.

Synopsis

raster **ST_IntersectionFractions**(raster rast, geometry geom);

Description

Calculates the fraction of each raster cell that is covered by a given geometry. The first argument is a raster, which defines the grid geometry to use for the calculation. The extent and cell size are read from the raster parameter. The second argument is a geometry, which is overlaid with the grid, and each grid populated based on overlaying the geometry on the grid. For polygons, the value returned for each cell is the proportion of its area that is covered by the geometry. For linestrings, the value returned for each cell is the length contained in the cell.

Availability: 3.6.0 Requires GEOS 3.14 or higher.

Exemples

This rotated square polygon covers half of each cell in the raster.

Code

```
CREATE TABLE raster_proportions_rast (
    name text,
    rast raster
);

INSERT INTO raster_proportions_rast (name, rast) VALUES (
  '2x2 raster covering 0,0 to 10,10',
  ST_MakeEmptyRaster(2, 2, -- raster width/height in pixels
    0, 10, -- upper-left corner x/y coordinates
    5, -5, -- pixel width/height in ground units
    0, 0, -- skew x/y
    0        -- SRID
  ));

SELECT name, ST_DumpValues(
    ST_IntersectionFractions(
        rast,
        'POLYGON((5 0,0 5,5 10,10 5,5 0))'::geometry),
    1)
FROM raster_proportions_rast;
```

Output

```
2x2 raster covering 0,0 to 10,10
--------------------------------
 {{0.5,0.5},{0.5,0.5}}
```

Voir aussi

ST_MakeEmptyRaster

11.16 Opérateurs raster

11.16.1 &&

&& — Retourne TRUE si la boite englobante de A intersecte la boite englobante de B.

Synopsis

boolean **&&**(raster A , raster B);
boolean **&&**(raster A , geometry B);
boolean **&&**(geometry B , raster A);

Description

L'opérateur && retourne TRUE si la boîte englobante du raster/géométrie A intersecte la boîte englobante du raster/géométrie B.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les données raster.

Disponibilité : 2.0.0

Exemples

Code

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;
```

Output

```
a_rid | b_rid | intersect
-------+-------+---------
     2 |     2 | t
     2 |     3 | f
     2 |     1 | f
```

11.16.2 &<

&< — Retourne TRUE si la boîte englobante de A est à gauche de celle de B.

Synopsis

boolean **&<**(raster A , raster B);

Description

L'opérateur &< retourne TRUE si la boîte englobante du raster A chevauche ou est à gauche de la boîte englobante de B ou, plus précisément, si elle chevauche ou n'est PAS à droite de la boîte englobante de B.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les données raster.

Exemples

Code

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

```
a_rid | b_rid | overleft
------+-------+----------
    2 |     2 | t
    2 |     3 | f
    2 |     1 | f
    3 |     2 | t
    3 |     3 | t
    3 |     1 | f
    1 |     2 | t
    1 |     3 | t
    1 |     1 | t
```

11.16.3 &>

&> — Retourne TRUE si la boîte englobante de A est à droite de celle de B.

Synopsis

boolean **&>**(raster A , raster B);

Description

L'opérateur &> retourne TRUE si la boîte englobante du raster A chevauche ou est à droite de la boîte englobante du raster B ou, plus précisément, si elle chevauche ou n'est PAS à gauche de la boîte englobante du raster B.



Note
Cette opérande utilisera tous les index qui peuvent être disponibles sur les géométries.

Exemples

Code

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &
> B.rast As overright
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

```
 a_rid | b_rid | overright
-------+-------+-----------
     2 |     2 | t
     2 |     3 | t
     2 |     1 | t
     3 |     2 | f
     3 |     3 | t
     3 |     1 | f
     1 |     2 | f
     1 |     3 | t
     1 |     1 | t
```

11.16.4 =

= — Retourne TRUE si la boîte englobante de A est la même que celle de B. Utilise des boîtes englobantes en double précision.

Synopsis

boolean =(raster A , raster B);

Description

L'opérateur = retourne TRUE si la boîte englobante du raster A est la même que celle du raster B. PostgreSQL utilise les opérateurs =, <, et > définis pour les rasters pour effectuer des tris et comparaisons internes sur les rasters (par exemple, dans une clause GROUP BY ou ORDER BY).



Attention
Cet opérande n'utilisera PAS les index éventuellement disponibles sur les données raster. Utilisez plutôt ~=. Cet opérateur existe principalement pour permettre le regroupement par colonne raster.

Disponibilité : 2.1.0

Voir aussi

~=

11.16.5 @

@ — Retourne TRUE si la boîte englobante de A est contenue dans celle de B. Utilise des boîtes englobantes en double précision.

Synopsis

boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);

Description

L'opérateur @ retourne TRUE si la boîte englobante du raster/géométrie A est contenue dans la boîte englobante du raster/géométrie B.



Note
Cet opérande utilisera des index spatiaux sur les données raster.

Disponibilité : 2.0.0 ajout de raster @ raster, raster @ geometry

Disponibilité : 2.0.5 ajout de geometry @ raster

Voir aussi

~

11.16.6 ~=

~= — Renvoie TRUE si la boîte de délimitation de A est la même que celle de B.

Synopsis

boolean ~=(raster A , raster B);

Description

L'opérateur ~= retourne TRUE si la boîte englobante du raster A est la même que la boîte englobante du raster B.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les données raster.

Disponibilité : 2.0.0

Exemples

Un cas d'utilisation très utile est celui où l'on prend deux ensembles de rasters à bande unique qui sont du même morceau mais représentent des thèmes différents, et où l'on crée un raster multi-bandes

Code

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

Voir aussi

ST_AddBand, =

11.16.7 ~

~ — Retourne TRUE si la boîte englobante de A contient celle de B. Utilise des boîtes englobantes en double précision.

Synopsis

boolean ~(raster A , raster B);
boolean ~(geometry A , raster B);
boolean ~(raster B , geometry A);

Description

L'opérateur ~ retourne TRUE si la boîte englobante du raster/géométrie A contient la boîte englobante du raster/géométrie B.



Note
Cet opérande utilisera des index spatiaux sur les données raster.

Disponibilité : 2.0.0

Voir aussi

@

11.17 Relations spatiales entre raster et entre bandes raster

11.17.1 ST_Contains

ST_Contains — Retourne true si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA et si au moins un point de l'intérieur du raster rastB se trouve à l'intérieur du raster rastA.

Synopsis

boolean **ST_Contains**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Contains**(raster rastA , raster rastB);

Description

Le raster rastA contient rastB si et seulement si aucun point de rastB ne se trouve à l'extérieur de rastA et si au moins un point de l'intérieur de rastB se trouve à l'intérieur de rastA. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si un numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_Contains(ST_Polygon(raster), geometry) ou ST_Contains(geometry, ST_Polygon(raster)).



Note
ST_Contains() est l'inverse de ST_Within(). Ainsi, ST_Contains(rastA, rastB) implique ST_Within(rastB, rastA).

Disponibilité : 2.1.0

Exemples

Specified band numbers.

Code

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 1;
```

Output

```
NOTICE:  The first raster provided has no bands
 rid | rid | st_contains
-----+-----+-------------
   1 |   1 |
   1 |   2 | f
```

No band numbers specified.

Code

```
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 1;
```

Output

```
rid | rid | st_contains
-----+-----+-------------
   1 |   1 | t
   1 |   2 | f
```

Voir aussi

ST_Intersects, ST_Within

11.17.2 ST_ContainsProperly

ST_ContainsProperly — Retourne true si rastB intersecte l'intérieur de rastA, mais pas la frontière ou l'extérieur de rastA.

Synopsis

boolean **ST_ContainsProperly**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_ContainsProperly**(raster rastA , raster rastB);

Description

Le raster rastA contient proprement rastB si rastB intersecte l'intérieur de rastA mais pas la limite ou l'extérieur de rastA. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.

Le raster rastA ne contient pas proprement lui-même, mais il se contient lui-même.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_ContainsProperly(ST_Polygon(raster), geometry) ou ST_ContainsProperly(geometry, ST_Polygon(raster)).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS ↩
    JOIN dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_containsproperly
-----+-----+---------------------
   2 |   1 | f
   2 |   2 | f
```

Voir aussi

ST_Intersects, ST_Contains

11.17.3 ST_Covers

ST_Covers — Retourne true si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA.

Synopsis

boolean **ST_Covers**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Covers**(raster rastA , raster rastB);

Description

Le raster rastA recouvre le raster rastB si et seulement si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_Covers(ST_Polygon(raster), geometry) ou ST_Covers(geometry, ST_Polygon(raster)).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_covers
-----+-----+-----------
   2 |   1 | f
   2 |   2 | t
```

Voir aussi

ST_Intersects, ST_CoveredBy

11.17.4 ST_CoveredBy

ST_CoveredBy — Retourne true si aucun point du raster rastA ne se trouve à l'extérieur du raster rastB.

Synopsis

boolean **ST_CoveredBy**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_CoveredBy**(raster rastA , raster rastB);

Description

Le raster rastA est couvert par rastB si et seulement si aucun point de rastA ne se trouve à l'extérieur de rastB. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_CoveredBy(ST_Polygon(raster), geometry) ou ST_CoveredBy(geometry, ST_Polygon(raster)).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_coveredby
-----+-----+--------------
   2 |   1 | f
   2 |   2 | t
```

Voir aussi

ST_Intersects, ST_Covers

11.17.5 ST_Disjoint

ST_Disjoint — Retourne true si le raster rastA n'intersecte pas spatialement le rastB.

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Disjoint**(raster rastA , raster rastB);

Description

Les rasters rastA et rastB sont disjoints s'ils ne partagent aucun espace. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction n'utilise PAS d'index.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_Disjoint(ST_Polygon(raster), geometry).

Disponibilité : 2.1.0

Exemples

Rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint.

Code

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
NOTICE:  The second raster provided has no bands
 rid | rid | st_disjoint
-----+-----+-------------
   2 |   1 |
   2 |   2 | f
```

This time, without specifying band numbers.

Code

```
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_disjoint
-----+-----+-------------
   2 |   1 | t
   2 |   2 | f
```

Voir aussi

ST_Intersects

11.17.6 ST_Intersects

ST_Intersects — Retourne vrai si le raster rastA intersecte spatialement le raster rastB.

Synopsis

boolean **ST_Intersects**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Intersects**(raster rastA , raster rastB);
boolean **ST_Intersects**(raster rast , integer nband , geometry geommin);
boolean **ST_Intersects**(raster rast , geometry geommin , integer nband=NULL);
boolean **ST_Intersects**(geometry geommin , raster rast , integer nband=NULL);

Description

Retourne vrai si le raster rastA intersecte spatialement le raster rastB. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.

Amélioration : 2.0.0 ajout de l'intersection raster/raster.



AVERTISSEMENT
Changement : 2.1.0 Le fonctionnement des variantes ST_Intersects(raster, geometry) a changé pour correspondre au fonctionnement de ST_Intersects(geometry, raster).

Exemples

Different bands of same raster.

Code

```
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;
```

Output

```
t
```

Voir aussi

ST_Intersection, ST_Disjoint

11.17.7 ST_Overlaps

ST_Overlaps — Retourne true si les raster rastA et rastB intersectent mais que l'un ne contient pas complètement l'autre.

Synopsis

boolean **ST_Overlaps**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Overlaps**(raster rastA , raster rastB);

Description

Retourne true si le raster rastA recouvre spatialement le raster rastB. Cela signifie que rastA et rastB intersectent mais que l'un ne contient pas complètement l'autre. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_Overlaps(ST_Polygon(raster), geometry).

Disponibilité : 2.1.0

Exemples

Comparing different bands of same raster.

Code

```
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;
```

Output

```
f
```

Voir aussi

ST_Intersects

11.17.8 ST_Touches

ST_Touches — Retourne true si les raster rastA et rastB ont au moins un point en commun mais que leurs intérieurs n'intersectent pas.

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

Description

Retourne true si le raster rastA touche spatialement le raster rastB. Cela signifie que rastA et rastB ont au moins un point en commun mais que leurs intérieurs n'intersectent pas. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cette fonction utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_Touches(ST_Polygon(raster), geometry).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_touches
-----+-----+------------
   2 |   1 | f
   2 |   2 | f
```

Voir aussi

ST_Intersects

11.17.9 ST_SameAlignment

ST_SameAlignment — Retourne true si les rasters ont les mêmes skew, scale, spatial ref, et offset (les pixels peuvent être placés sur la même grille sans être coupés) et false si ce n'est pas le cas, avec une NOTICE détaillant le problème.

Synopsis

boolean **ST_SameAlignment**(raster rastA , raster rastB);
boolean **ST_SameAlignment**(double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2 , double precision skewy2);
boolean **ST_SameAlignment**(raster set rastfield);

Description

Version non agrégée (variantes 1 et 2) : Retourne true si les deux rasters (fournis directement ou créés en utilisant les valeurs pour upperleft, scale, skew et srid) ont la même échelle, le même skew, le même srid et si au moins un des quatre coins de n'importe quel pixel d'un raster tombe sur n'importe quel coin de la grille de l'autre raster. Retourne false si ce n'est pas le cas et un NOTICE détaillant le problème d'alignement.

Version agrégée (variante 3) : A partir d'un ensemble de rasters, retourne vrai si tous les rasters de l'ensemble sont alignés. La fonction ST_SameAlignment() est une fonction "agrégée" dans la terminologie de PostgreSQL. Cela signifie qu'elle opère sur des lignes de données, de la même manière que les fonctions SUM() et AVG().

Disponibilité : 2.0.0

Amélioration : 2.1.0 ajout de la variante agrégée

Exemples : Rasters

Code

```
SELECT ST_SameAlignment(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;
```

Output

```
t
```

Code

```
SELECT ST_SameAlignment(A.rast, b.rast)
FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;
```

Output

```
NOTICE:  The two rasters provided have different SRIDs
NOTICE:  The two rasters provided have different SRIDs
 t
 f
 f
 f
```

Voir aussi

Section 10.1, ST_NotSameAlignmentReason, ST_MakeEmptyRaster

11.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — Retourne un texte indiquant si les rasters sont alignés et, s'ils ne le sont pas, la raison du problème.

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

Description

Retourne un texte indiquant si les rasters sont alignés et, s'ils ne le sont pas, la raison du problème.



Note
S'il y a plusieurs raisons pour lesquelles les rasters ne sont pas alignés, une seule raison (le premier test qui a échoué) sera retournée.

Disponibilité : 2.1.0

Exemples

Code

```
SELECT
    ST_SameAlignment(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    ),
    ST_NotSameAlignmentReason(ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    )
;
```

Output

```
st_samealignment |            st_notsamealignmentreason
-----------------+-------------------------------------------------
 f               | The rasters have different scales on the X axis
(1 row)
```

Voir aussi

Section 10.1, ST_SameAlignment

11.17.11 ST_Within

ST_Within — Retourne true si aucun point du raster rastA ne se trouve à l'extérieur du raster rastB et si au moins un point de l'intérieur du raster rastA se trouve à l'intérieur du raster rastB.

Synopsis

boolean **ST_Within**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Within**(raster rastA , raster rastB);

Description

Le raster rastA est compris dans rastB si et seulement si aucun point de rastA ne se trouve à l'extérieur de rastB et si au moins un point de l'intérieur de rastA se trouve à l'intérieur de rastB. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_Within(ST_Polygon(raster), geometry) ou ST_Within(geometry, ST_Polygon(raster)).



Note
ST_Within() est l'inverse de ST_Contains(). Ainsi, ST_Within(rastA, rastB) implique ST_Contains(rastB, rastA).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_within
-----+-----+-----------
   2 |   1 | f
   2 |   2 | t
```

Voir aussi

ST_Intersects, ST_Contains, ST_DWithin, ST_DFullyWithin

11.17.12 ST_DWithin

ST_DWithin — Retourne true si les rasters rastA et rastB se trouvent à une distance donnée l'un de l'autre.

Synopsis

boolean **ST_DWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

Retourne true si les rasters rastA et rastB sont à la distance spécifiée l'un de l'autre. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.

La distance est spécifiée en unités définies par le système de référence spatiale des rasters. Pour que cette fonction ait un sens, les rasters sources doivent avoir la même projection de coordonnées et le même SRID.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_DWithin(ST_Polygon(raster), geometry).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS ↩
    JOIN dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_dwithin
-----+-----+------------
   2 |   1 | f
   2 |   2 | t
```

Voir aussi

ST_Within, ST_DFullyWithin

11.17.13 ST_DFullyWithin

ST_DFullyWithin — Retourne true si les rasters rastA et rastB se trouvent entièrement à une distance donnée l'un de l'autre.

Synopsis

boolean **ST_DFullyWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DFullyWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Description

Retourne true si les rasters rastA et rastB sont entièrement à une distance donnée l'un de l'autre. Si le numéro de bande n'est pas spécifié (ou si NULL), seule l'enveloppe convexe du raster est prise en compte dans le test. Si le numéro de bande est spécifié, seuls les pixels ayant une valeur (différente de NODATA) sont pris en compte dans le test.

La distance est spécifiée en unités définies par le système de référence spatiale des rasters. Pour que cette fonction ait un sens, les rasters sources doivent avoir la même projection de coordonnées et le même SRID.



Note
Cet opérande utilisera tous les index qui peuvent être disponibles sur les données raster.



Note
Pour tester la relation spatiale entre un raster et une géométrie, utilisez ST_Polygon sur le raster, par exemple ST_DFullyWithin(ST_Polygon(raster), geometry).

Disponibilité : 2.1.0

Exemples

Code

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 ↩
    CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;
```

Output

```
rid | rid | st_dfullywithin
-----+-----+-----------------
   2 |   1 | f
   2 |   2 | t
```

Voir aussi

ST_Within, ST_DWithin

11.18 Astuces raster

11.18.1 Rasters out-DB

11.18.1.1 Répertoire contenant de nombreux fichiers

Lorsque GDAL ouvre un fichier, il parcourt l'entièreté du répertoire du fichier pour construire un catalogue d'autres fichiers. Si ce répertoire contient de nombreux fichiers (par exemple des milliers, des millions), l'ouverture de ce fichier devient extrêmement lente (en particulier si ce fichier se trouve sur un lecteur réseau tel que NFS).

Pour contrôler ce comportement, GDAL fournit la variable d'environnement suivante : GDAL_DISABLE_READDIR_ON_OPEN. Définissez `GDAL_DISABLE_READDIR_ON_OPEN` à `TRUE` pour désactiver l'analyse des répertoires.

Sous Ubuntu (et en supposant que vous utilisez les paquets PostgreSQL pour Ubuntu), `GDAL_DISABLE_READDIR_ON_OPEN` peut être défini dans */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment* (où POSTGRESQL_VERSION est la version de PostgreSQL, par ex.9.6 et CLUSTER_NAME est le nom du cluster, par exemple maindb). Vous pouvez également définir les variables d'environnement PostGIS ici.

Code

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
#  VARIABLE = simple_value
#  VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

11.18.1.2 Nombre maximum de fichiers ouverts

Le nombre maximum de fichiers ouverts autorisé par Linux et PostgreSQL est généralement prudent (typiquement 1024 fichiers ouverts par processus), étant donné l'hypothèse que le système est utilisé par des utilisateurs humains. Pour les rasters Out-DB, une seule requête valide peut facilement dépasser cette limite (par exemple, un jeu de données de 10 ans avec un raster pour chaque jour contenant les températures minimales et maximales et nous voulons connaître les valeurs min et max absolues pour un pixel dans ce jeu de données).

Le changement le plus simple à effectuer est le paramètre suivant de PostgreSQL : max_files_per_process. La valeur par défaut est de 1000, ce qui est beaucoup trop faible pour les données raster Out-DB. Une valeur de départ sûre pourrait être 65536, mais cela dépend vraiment de vos ensembles de données et des requêtes exécutées sur ces ensembles de données. Ce paramètre ne peut être défini qu'au démarrage du serveur et probablement uniquement dans le fichier de configuration de PostgreSQL (par exemple */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf* dans les environnements Ubuntu).

Code

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536           # min 25
                                        # (change requires restart)
...
```

La principale modification à apporter concerne les limites des fichiers ouverts du noyau Linux. Il y a deux parties à cela :

— Nombre maximal de fichiers ouverts pour l'ensemble du système
— Nombre maximal de fichiers ouverts par processus

11.18.1.2.1 Nombre maximal de fichiers ouverts pour l'ensemble du système

L'exemple suivant vous permet de connaître le nombre maximum de fichiers ouverts sur l'ensemble du système :

Code

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

Si la valeur retournée n'est pas assez importante, ajoutez un fichier dans */etc/sysctl.d/* comme dans l'exemple suivant :

Code

```
$ echo "fs.file-max = 6145324" >> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

11.18.1.2.2 Nombre maximal de fichiers ouverts par processus

Nous devons augmenter le nombre maximum de fichiers ouverts par processus pour les processus du serveur PostgreSQL.

Pour connaître le nombre maximal de fichiers ouverts utilisé par les processus de service PostgreSQL, suivez l'exemple suivant (assurez-vous que PostgreSQL est en cours d'exécution) :

Code

```
$ ps aux | grep postgres
```

Output

```
postgres 31713  0.0  0.4 179012 17564 pts/0    S    Dec26   0:03 /home/dustymugs/devel/postgresql/sandbox/10/usr/local/ ↩
     bin/postgres -D /home/dustymugs/devel/postgresql/sandbox/10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss   Dec26   0:01 postgres: checkpointer process
postgres 31717  0.0  0.2 179144  9416 ?        Ss   Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss   Dec26   0:06 postgres: wal writer process
postgres 31719  0.0  0.1 179568  7252 ?        Ss   Dec26   0:03 postgres: autovacuum launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss   Dec26   0:09 postgres: stats collector process
postgres 31721  0.0  0.1 179308  6052 ?        Ss   Dec26   0:00 postgres: bgworker: logical replication launcher
```

Code

```
$ cat /proc/31718/limits
```

Output

```
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             15738                15738                processes
Max open files            1024                 4096                 files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       15738                15738                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
```

Dans l'exemple ci-dessus, nous avons inspecté la limite des fichiers ouverts pour le processus 31718. Peu importe le processus PostgreSQL, n'importe lequel fera l'affaire. La réponse qui nous intéresse est *Max open files*.

Nous voulons augmenter *Soft Limit* et *Hard Limit* de *Max open files* pour qu'il soit supérieur à la valeur que nous avons spécifiée pour le paramètre PostgreSQL `max_files_per_process`. Dans notre exemple, nous avons fixé `max_files_per_process` à 65536.

Sous Ubuntu (et en supposant que vous utilisez les paquets PostgreSQL pour Ubuntu), la façon la plus simple de changer les paramètres *Soft Limit* et *Hard Limit* est d'éditer **/etc/init.d/postgresql** (SysV) ou **/lib/systemd/system/postgresql*.service** (systemd).

Commençons par le cas de SysV Ubuntu où nous ajoutons **ulimit -H -n 262144** et **ulimit -n 131072** à **/etc/init.d/postgresql**.

Code

```
...
case "$1" in
    start|stop|restart|reload)
        if [ "$1" = "start" ]; then
            create_socket_directory
        fi
    if [ -z "`pg_lsclusters -h`" ]; then
        log_warning_msg 'No PostgreSQL clusters exist; see "man pg_createcluster"'
        exit 0
    fi

    ulimit -H -n 262144
    ulimit -n 131072

    for v in $versions; do
        $1 $v || EXIT=$?
    done
    exit ${EXIT:-0}
        ;;
    status)
...
```

Passons maintenant au cas de systemd Ubuntu. Nous ajouterons **LimitNOFILE=131072** à chaque fichier **/lib/systemd/system/-postgresql*.service** dans la section **[Service]**.

Code

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target
...
```

Après avoir effectué les changements nécessaires à systemd, assurez-vous de recharger le démon

Code

```
systemctl daemon-reload
```

Chapitre 12

PostGIS Extras

Ce chapitre documente les fonctionnalités trouvées dans le dossier extras des fichiers tarballs et du dépôt de sources de PostGIS. Celles-ci ne sont pas toujours packagées avec les versions binaires de PostGIS, mais sont généralement basées sur PL/pgSQL ou des scripts shell standard qui peuvent être exécutés tels quels.

Chapitre 13

Index des fonctions spéciales de PostGIS

13.1 Fonctions d'agrégation de PostGIS

Les fonctions ci-dessous sont des fonctions d'agrégation spatiale qui sont utilisées de la même manière que les fonctions d'agrégation SQL telles que `sum` et `average`.

— CG_3DUnion - Réaliser l'union 3D en utilisant postgis_sfcgal.
— ST_3DExtent - Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.
— ST_AsFlatGeobuf - Renvoie une représentation FlatGeobuf d'un ensemble de lignes.
— ST_AsGeobuf - Retourne une représentation Geobuf d'un ensemble de lignes.
— ST_AsMVT - Fonction d'agrégation renvoyant une représentation MVT d'un ensemble de lignes.
— ST_ClusterDBSCAN - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie d'entrée en utilisant l'algorithme DBSCAN.
— ST_ClusterIntersecting - Fonction d'agrégation qui regroupe les géométries en entrée en ensembles connectés.
— ST_ClusterIntersectingWin - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, en regroupant les géométries en entrée en ensembles connectés.
— ST_ClusterKMeans - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée en utilisant l'algorithme K-means.
— ST_ClusterRelateWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
— ST_ClusterWithin - Fonction agrégée qui regroupe les géométries en fonction de la distance de séparation.
— ST_ClusterWithinWin - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, regroupement en utilisant la distance de séparation.
— ST_Collect - Crée une géométrie GeometryCollection ou Multi à partir d'un ensemble de géométries.
— ST_CoverageClean - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
— ST_CoverageInvalidEdges - Fonction window qui trouve les endroits où les polygones ne forment pas une couverture valide.
— ST_CoverageSimplify - Fonction window qui simplifie les bords d'une couverture polygonale.
— ST_CoverageUnion - Calcule l'union d'un ensemble de polygones formant une couverture en supprimant les arêtes communes.
— ST_Extent - Fonction agrégée qui renvoie la boîte de délimitation des géométries.
— ST_MakeLine - Crée une LineString à partir de géométries Point, MultiPoint ou LineString.
— ST_MemUnion - Fonction d'agrégation qui fusionne les géométries d'une manière efficace sur le plan de la mémoire mais plus lente
— ST_MinimumSpanningTree - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
— ST_Polygonize - Calcule une collection de polygones formés à partir du tracé d'un ensemble de géométries.
— ST_SameAlignment - Retourne true si les rasters ont les mêmes skew, scale, spatial ref, et offset (les pixels peuvent être placés sur la même grille sans être coupés) et false si ce n'est pas le cas, avec une NOTICE détaillant le problème.

— ST_Union - Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.
— ST_Union - Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— TopoElementArray_Agg - Renvoie un topoelementarray pour un ensemble de tableaux de type, element_id (topoelements).

13.2 Fonctions Window PostGIS

Les fonctions ci-dessous sont des fonctions window spatiale qui sont utilisées de la même manière que les fonctions de fenêtre SQL telles que `row_number()`, `lead()`, et `lag()`. Elles doivent être suivies d'une clause `OVER()`.

— ST_ClusterDBSCAN - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie d'entrée en utilisant l'algorithme DBSCAN.
— ST_ClusterIntersectingWin - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, en regroupant les géométries en entrée en ensembles connectés.
— ST_ClusterKMeans - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée en utilisant l'algorithme K-means.
— ST_ClusterRelateWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
— ST_ClusterWithinWin - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, regroupement en utilisant la distance de séparation.
— ST_CoverageClean - Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
— ST_CoverageInvalidEdges - Fonction window qui trouve les endroits où les polygones ne forment pas une couverture valide.
— ST_CoverageSimplify - Fonction window qui simplifie les bords d'une couverture polygonale.
— ST_MinimumSpanningTree - Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.

13.3 Fonctions de PostGIS compatibles avec SQL-MM

Les fonctions ci-dessous sont des fonctions PostGIS conformes à la norme SQL/MM 3

— CG_3DArea - Calcule la surface des géométries de surface 3D. Retourne 0 pour les solides.
— CG_3DDifference - Effectuer une différence 3D
— CG_3DIntersection - Réaliser une intersection 3D
— CG_3DUnion - Réaliser l'union 3D en utilisant postgis_sfcgal.
— CG_Volume - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
— ST_3DDWithin - Teste si deux géométries 3D se trouvent à une distance 3D donnée
— ST_3DDistance - Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DIntersects - Teste si deux géométries se croisent dans l'espace en 3D - uniquement pour les points, les lignes, les polygones, les surfaces polyédriques (aire)
— ST_3DLength - Renvoie la longueur 3D d'une géométrie linéaire.
— ST_3DPerimeter - Renvoie le périmètre 3D d'une géométrie polygonale.
— ST_AddEdgeModFace - Ajoutez une nouvelle arête et, si elle divise une face, modifiez la face d'origine et ajoutez une nouvelle face.
— ST_AddEdgeNewFaces - Ajoutez une nouvelle arête et, si elle divise une face, supprimez la face d'origine et remplacez-la par deux nouvelles faces.
— ST_AddIsoEdge - Ajoute une arête isolée définie par la géométrie alinestring à une topologie reliant deux nœuds isolés existants anode et anothernode et renvoie l'identifiant de l'arête de la nouvelle arête.
— ST_AddIsoNode - Ajoute un noeud isolé à une face dans une topologie et renvoie le nodeid du nouveau noeud. Si la face est nulle, le noeud est quand même créé.
— ST_Area - Renvoie l'aire d'une géométrie polygonale.

— ST_AsBinary - Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsGML - Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsText - Renvoie la représentation Well-Known Text (WKT) de la géométrie/geography sans métadonnées SRID.
— ST_Boundary - Renvoie la limite d'une géométrie.
— ST_Buffer - Calcule une géométrie couvrant tous les points situés à une distance donnée d'une géométrie.
— ST_Centroid - Renvoie le centre géométrique d'une géométrie.
— ST_ChangeEdgeGeom - Modifie la forme d'une arête sans affecter la structure de la topologie.
— ST_Contains - Tests si chaque point de B est situé dans A, et que leurs intérieurs ont un point commun
— ST_ConvexHull - Calcule l'enveloppe convexe d'une géométrie.
— ST_CoordDim - Renvoie la dimension des coordonnées d'une géométrie.
— ST_CreateTopoGeo - Ajoute une collection de géométries à une topologie vide donnée et renvoie un message détaillant le succès.
— ST_Crosses - Teste si deux géométries ont en commun certains points intérieurs, mais pas tous
— ST_CurveN - Renvoie la Nième courbe composante d'une CompoundCurve.
— ST_CurveToLine - Convertit une géométrie contenant des courbes en une géométrie linéaire.
— ST_Difference - Calcule une géométrie représentant la partie de la géométrie A qui n'intersecte pas la géométrie B.
— ST_Dimension - Renvoie la dimension topologique d'une géométrie.
— ST_Disjoint - Teste si deux géométries n'ont pas de points communs
— ST_Distance - Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_EndPoint - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
— ST_Envelope - Renvoie une géométrie représentant la boîte de délimitation d'une géométrie.
— ST_Equals - Teste si deux géométries comprennent le même ensemble de points
— ST_ExteriorRing - Renvoie une ligne représentant l'anneau extérieur d'un polygone.
— ST_GMLToSQL - Returns a geometry from GML. Alias for ST_GeomFromGML.
— ST_GeomCollFromText - Crée une collection Geometry à partir de la collection WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.
— ST_GeomFromText - Retourne un objet ST_Geometry à partir de sa représentation textuelle Well-Known Text (WKT).
— ST_GeomFromWKB - Retourne un objet de type geometry à partir de sa représentation binaire Well-Know Binary (WKB) et d'un SRID optionnel.
— ST_GeometryFromText - Returns a geometry from Well-Known Text (WKT). Alias for ST_GeomFromText.
— ST_GeometryN - Renvoie un élément d'une collection de géométries.
— ST_GeometryType - Renvoie le type SQL-MM d'une géométrie sous forme de texte.
— ST_GetFaceEdges - Renvoie un ensemble d'arêtes ordonnées qui délimitent aface.
— ST_GetFaceGeometry - Renvoie le polygone dans la topologie donnée avec l'identifiant de face spécifié.
— ST_InitTopoGeo - Crée un nouveau schéma topologique et l'enregistre dans la table topology.topology.
— ST_InteriorRingN - Renvoie le Nième anneau intérieur (trou) d'un polygone.
— ST_Intersection - Calcule une géométrie représentant la partie partagée des géométries A et B.
— ST_Intersects - Teste si deux géométries se croisent (elles ont au moins un point en commun)
— ST_IsClosed - Teste si les points de départ et d'arrivée d'une LineString coïncident. Pour une PolyhedralSurface, teste si elle est fermée (volumétrique).
— ST_IsEmpty - Teste si une géométrie est vide.
— ST_IsRing - Teste si une ligne est fermée et simple.
— ST_IsSimple - Teste si une géométrie n'a pas de points d'auto-intersection ou d'auto-tangente.
— ST_IsValid - Teste si une géométrie est bien formée en 2D.
— ST_Length - Renvoie la longueur 2D d'une géométrie linéaire.
— ST_LineFromText - Construit une géométrie à partir d'une représentation WKT avec le SRID donné. Si aucun SRID n'est donné, la valeur par défaut est 0.
— ST_LineFromWKB - Creates a LINESTRING from a WKB representation, optionally using a given SRID.
— ST_LinestringFromWKB - Creates a LINESTRING from a WKB representation, optionally using a given SRID.
— ST_LocateAlong - Renvoie le(s) point(s) d'une géométrie qui correspond(ent) à une valeur de mesure.

— ST_LocateBetween - Renvoie les parties d'une géométrie qui correspondent à un intervalle de mesure.
— ST_M - Renvoie la coordonnée M d'un point.
— ST_MLineFromText - Retourne un objet de type ST_MultiLineString à partir de sa représentation WKT.
— ST_MPointFromText - Créé une Geometry depuis un WKT avec le SRID donné. Si le SRID n'est pas fourni, il sera défini par défaut à 0.
— ST_MPolyFromText - Crée une géométrie multi-polygone à partir de WKT avec le SRID donné. Si le SRID n'est pas donné, la valeur par défaut est 0.
— ST_ModEdgeHeal - Répare deux arêtes en supprimant le nœud qui les relie, en modifiant la première arête et en supprimant la seconde. Renvoie l'identifiant du nœud supprimé.
— ST_ModEdgeSplit - Fractionner une arête en créant un nouveau nœud le long d'une arête existante, en modifiant l'arête d'origine et en ajoutant une nouvelle arête.
— ST_MoveIsoNode - Déplace un nœud isolé dans une topologie d'un point à un autre. Si la nouvelle géométrie apoint existe en tant que noeud, une erreur est générée. Retourne la description du déplacement.
— ST_NewEdgeHeal - Répare deux arêtes en supprimant le nœud qui les relie, en supprimant les deux arêtes et en les remplaçant par une arête dont la direction est la même que la première arête fournie.
— ST_NewEdgesSplit - Fractionne une arête en créant un nouveau nœud le long d'une arête existante, en supprimant l'arête d'origine et en la remplaçant par deux nouvelles arêtes. Renvoie l'identifiant du nouveau nœud créé qui relie les nouvelles arêtes.
— ST_NumCurves - Renvoie le nombre de courbes composantes d'une CompoundCurve.
— ST_NumGeometries - Renvoie le nombre d'éléments dans une collection de géométrie.
— ST_NumInteriorRings - Renvoie le nombre d'anneaux intérieurs (trous) d'un polygone.
— ST_NumPatches - Returns the number of faces in a PolyhedralSurface or TIN.
— ST_NumPoints - Renvoie le nombre de points dans une LineString ou une CircularString.
— ST_OrderingEquals - Teste si deux géométries représentent la même géométrie et ont des points dans le même ordre directionnel
— ST_Overlaps - Teste si deux géométries ont la même dimension et se croisent, mais si chacune a au moins un point qui n'est pas dans l'autre
— ST_PatchN - Returns the Nth face of a PolyhedralSurface or TIN.
— ST_Perimeter - Renvoie la longueur de la limite d'une géométrie polygonale ou d'une géographie.
— ST_Point - Crée un point avec des valeurs X, Y et SRID.
— ST_PointFromText - Construit une géométrie point à partir d'une représentation WKT et le SRID donné. Si aucun SRID n'est donné, la valeur par défaut est 0.
— ST_PointFromWKB - Construit une géométrie depuis la représentation binaire WKB et le SRID donné
— ST_PointN - Renvoie le Nième point de la première LineString ou LineString circulaire d'une géométrie.
— ST_PointOnSurface - Calcule un point dont on garantit qu'il se trouve dans un polygone ou sur une géométrie.
— ST_Polygon - Crée un polygone à partir d'une LineString avec un SRID spécifié.
— ST_PolygonFromText - Créé une Geometry depuis un WKT avec le SRID donné. Si le SRID n'est pas fourni, il sera défini par défaut à 0.
— ST_Relate - Teste si deux géométries ont une relation topologique correspondant à un modèle de matrice d'intersection, ou calcule leur matrice d'intersection
— ST_RemEdgeModFace - Supprime une arête et, si l'arête sépare deux faces, supprime une face et modifie l'autre face pour couvrir l'espace des deux.
— ST_RemEdgeNewFace - Enlève une arête et, si l'arête enlevée séparait deux faces, supprime les faces originales et les remplace par une nouvelle face.
— ST_RemoveIsoEdge - Supprime une arête isolée et renvoie la description de l'action. Si l'arête n'est pas isolée, une exception est levée.
— ST_RemoveIsoNode - Supprime un noeud isolé et renvoie la description de l'action. Si le noeud n'est pas isolé (début ou fin d'une arête), une exception est levée.
— ST_SRID - Renvoie l'identifiant de référence spatiale d'une géométrie.
— ST_StartPoint - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
— ST_SymDifference - Calcule une géométrie représentant les parties des géométries A et B qui ne s'intersectent pas.
— ST_Touches - Teste si deux géométries ont au moins un point en commun, mais que leurs intérieurs ne se croisent pas

— ST_Transform - Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent.
— ST_Union - Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.
— ST_WKBToSQL - Returns a geometry from Well-Known Binary (WKB). SQL/MM alias for ST_GeomFromWKB without an SRID argument.
— ST_WKTToSQL - Returns a geometry from Well-Known Text (WKT). SQL/MM alias for ST_GeomFromText.
— ST_Within - Tests si chaque point de A se trouve dans B, et que leurs intérieurs ont un point commun
— ST_X - Renvoie la coordonnée X d'un point.
— ST_Y - Renvoie la coordonnée Y d'un point.
— ST_Z - Renvoie la coordonnée Z d'un point.

13.4 Fonctions d'aide au type geography de PostGIS

Les fonctions et opérateurs ci-dessous sont des fonctions/opérateurs PostGIS qui prennent en entrée ou renvoient en sortie un objet de type geography.



Note
Les fonctions avec un (T) ne sont pas des fonctions géodésiques natives, et utilisent un appel ST_Transform vers et depuis la géométrie pour effectuer l'opération. Par conséquent, elles peuvent ne pas se comporter comme prévu lorsque l'on passe au-dessus de la ligne de base, des pôles, et pour les grandes géométries ou les paires de géométries qui couvrent plus d'une zone UTM. Transformation de base - (favorisant l'UTM, l'azimut Lambert (Nord/Sud), et se rabattant sur le mercator dans le pire des cas)

— ST_Area - Renvoie l'aire d'une géométrie polygonale.
— ST_AsBinary - Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsEWKT - Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.
— ST_AsGML - Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsGeoJSON - Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_AsKML - Renvoyer la géométrie sous forme d'élément KML.
— ST_AsSVG - Renvoie les données de chemin SVG pour une géométrie.
— ST_AsText - Renvoie la représentation Well-Known Text (WKT) de la géométrie/geography sans métadonnées SRID.
— ST_Azimuth - Renvoie l'azimut basé sur le nord d'une ligne entre deux points.
— ST_Buffer - Calcule une géométrie couvrant tous les points situés à une distance donnée d'une géométrie.
— ST_Centroid - Renvoie le centre géométrique d'une géométrie.
— ST_ClosestPoint - Renvoie le point 2D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne la plus courte d'une géométrie à l'autre.
— ST_CoveredBy - Tests si chaque point de A se trouve dans B
— ST_Covers - Tests si chaque point de B est situé dans A
— ST_DWithin - Teste si deux géométries se trouvent à une distance donnée
— ST_Distance - Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_GeogFromText - Retourne un objet de type geography à partir de sa représentation Well-Know Text (WKT ou EWKT).
— ST_GeogFromWKB - Retourne un objet de type geography à partir de sa représentation binaire Well-Know Binary (WKB ou EWKB).
— ST_GeographyFromText - Retourne un objet de type geography à partir de sa représentation Well-Know Text (WKT ou EWKT).
— = - Renvoie TRUE si les coordonnées et l'ordre des coordonnées de la géométrie/géographie A sont les mêmes que les coordonnées et l'ordre des coordonnées de la géométrie/géographie B.
— ST_Intersection - Calcule une géométrie représentant la partie partagée des géométries A et B.
— ST_Intersects - Teste si deux géométries se croisent (elles ont au moins un point en commun)

— ST_Length - Renvoie la longueur 2D d'une géométrie linéaire.
— ST_LineInterpolatePoint - Renvoie un point interpolé le long d'une ligne à un emplacement fractionnaire.
— ST_LineInterpolatePoints - Renvoie des points interpolés le long d'une ligne à un intervalle fractionnaire.
— ST_LineLocatePoint - Renvoie l'emplacement fractionnaire du point le plus proche d'un point sur une ligne.
— ST_LineSubstring - Renvoie la partie d'une ligne située entre deux emplacements fractionnaires.
— ST_Perimeter - Renvoie la longueur de la limite d'une géométrie polygonale ou d'une géographie.
— ST_Project - Renvoie un point projeté à partir d'un point de départ en fonction d'une distance et d'un azimut.
— ST_Segmentize - Renvoie une geometry/geography modifiée dont aucun segment ne dépasse une distance donnée.
— ST_ShortestLine - Renvoie la ligne 2D la plus courte entre deux géométries
— ST_Summary - Renvoie un résumé textuel du contenu d'une géométrie.
— <-> - Renvoie la distance en 2D entre A et B.
— && - Renvoi VRAI si la boite englobante 2D de A intersecte la boite englobante 2D de B.

13.5 Fonctions de support des données raster de PostGIS

Les fonctions et opérateurs ci-dessous sont des fonctions/opérateurs PostGIS qui prennent en entrée ou renvoient en sortie un objet de type raster. Ils sont classés par ordre alphabétique.

— Box3D - Retourne la représentation 3d de la boîte englobante du raster.
— @ - Retourne TRUE si la boîte englobante de A est contenue dans celle de B. Utilise des boîtes englobantes en double précision.
— ~ - Retourne TRUE si la boîte englobante de A contient celle de B. Utilise des boîtes englobantes en double précision.
— = - Retourne TRUE si la boîte englobante de A est la même que celle de B. Utilise des boîtes englobantes en double précision.
— && - Retourne TRUE si la boite englobante de A intersecte la boite englobante de B.
— &< - Retourne TRUE si la boîte englobante de A est à gauche de celle de B.
— &> - Retourne TRUE si la boîte englobante de A est à droite de celle de B.
— ~= - Renvoie TRUE si la boîte de délimitation de A est la même que celle de B.
— ST_Retile - Retourne un ensemble de tuiles configurées à partir d'une couverture raster composée de tuiles arbitraires.
— ST_AddBand - Retourne un raster avec la/les nouvelle(s) bande(s) ajouté(s) à un index donné, de type et valeur initiale donnés. Si aucun index n'est spécifié, la bande est ajoutée à la fin.
— ST_AsBinary/ST_AsWKB - Retourne la représentation Well-Known Binary (WKB) de ce raster.
— ST_AsGDALRaster - Retourne la tuile raster dans le format GDAL raster spécifié. Les formats raster disponibles sont ceux supportés par votre bibliothèque compilée. Utilisez ST_GDALDrivers() pour obtenir la liste des formats supportés par votre bibliothèque.
— ST_AsHexWKB - Retourne la représentation hexadécimale Well-Known Binary (WKB) de ce raster.
— ST_AsJPEG - Retourne les bandes sélectionnées du raster sous la forme d'une image JPEG (sous forme de tableau d'octets). Si aucune bande n'est spécifiée, et que le raster a 1 ou plus de 3 bandes, seule la première bande est utilisée. Si le raster a exactement 3 bandes, les 3 bandes sont utilisées et mappées en RGB.
— ST_AsPNG - Retourne les bandes sélectionnées du raster sous la forme d'une image PNG (sous forme de tableau d'octets). Si aucune bande n'est spécifiée et que le raster a 1, 3 ou 4 bandes, toutes les bandes sont utilisées. Si aucune bande n'est spécifiée et que le raster a 2 ou plus de 4 bandes, seule la bande 1 est utilisée. Les bandes sont mappées en RGB ou RGBA.
— ST_AsRaster - Convertit une géométrie PostGIS en un raster PostGIS.
— ST_AsRasterAgg - Aggregate. Rend les géométries PostGIS dans un nouveau raster.
— ST_AsTIFF - Retourne les bandes sélectionnées du raster sous la forme d'une seule image TIFF (sous forme de tableau d'octets). Si aucune bande n'est spécifiée ou si l'une des bandes spécifiées n'existe pas dans le raster, toutes les bandes sont utilisées.
— ST_Aspect - Retourne l'exposition (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_Band - Retourne une ou plusieurs bandes d'un raster existant en tant que nouveau raster. Utile pour construire de nouveaux rasters à partir de rasters existants.
— ST_BandFileSize - Retourne la taille du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.

— ST_BandFileTimestamp - Retourne le timestamp du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_BandIsNoData - Retourne true si la bande ne contient que des valeurs nodata.
— ST_BandMetaData - Retourne les méta-données de base d'une bande raster spécifique. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_BandNoDataValue - Retourne la valeur qui représente l'absence de valeur (nodata) pour cette bande. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_BandPath - Retourne le chemin système du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_BandPixelType - Retourne le type de pixel d'une bande. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_Clip - Renvoie le raster découpé par la géométrie d'entrée. Si le numéro de bande n'est pas spécifié, toutes les bandes sont traitées. Si crop n'est pas spécifié ou est TRUE, le raster de sortie est rogné. Si touched est défini sur TRUE, les pixels touchés sont inclus, sinon seulement si le centre du pixel est dans la géométrie, il est inclus.
— ST_ColorMap - Crée un nouveau raster comprenant jusqu'à quatre bandes 8BUI (niveaux de gris, RGB, RGBA) à partir du raster source et d'une bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_Contains - Retourne true si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA et si au moins un point de l'intérieur du raster rastB se trouve à l'intérieur du raster rastA.
— ST_ContainsProperly - Retourne true si rastB intersecte l'intérieur de rastA, mais pas la frontière ou l'extérieur de rastA.
— ST_Contour - Génère un ensemble de courbes de niveau vectorielles depuis la bande raster spécifiée, en utilisant l'algorithme de contour GDAL.
— ST_ConvexHull - Retourne l'enveloppe convexe du raster, en incluant les valeurs de pixels égales à BandNoDataValue. Pour les données raster de forme régulière et sans obliquité, cette fonction donne le même résultat que ST_Envelope ; elle n'est donc utile que pour les données raster de forme irrégulière ou inclinée.
— ST_Count - Renvoie le nombre de pixels dans une bande donnée d'un raster ou d'une couverture raster. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels dont la valeur est différente de la valeur nodata seront comptés.
— ST_CountAgg - Agrégat. Renvoie le nombre de pixels dans une bande donnée d'un ensemble de rasters. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels différents de la valeur NODATA seront comptés.
— ST_CoveredBy - Retourne true si aucun point du raster rastA ne se trouve à l'extérieur du raster rastB.
— ST_Covers - Retourne true si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA.
— ST_DFullyWithin - Retourne true si les rasters rastA et rastB se trouvent entièrement à une distance donnée l'un de l'autre.
— ST_DWithin - Retourne true si les rasters rastA et rastB se trouvent à une distance donnée l'un de l'autre.
— ST_Disjoint - Retourne true si le raster rastA n'intersecte pas spatialement le rastB.
— ST_DumpAsPolygons - Retourne un ensemble d'enregistrements de type geomval (geom, val), à partir d'une bande raster donnée. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.
— ST_DumpValues - Retourne les valeurs d'une bande raster spécifiée, sous forme d'un tableau à deux dimensions.
— ST_Envelope - Retourne la représentation polygonale de l'étendue du raster.
— ST_FromGDALRaster - Retourne un raster depuis un fichier raster supporté par GDAL.
— ST_GeoReference - Retourne les méta-données de géo-référencement au format GDAL (par défaut) ou ESRI, tel qu'utilisé généralement dans les fichiers world file.
— ST_Grayscale - Crée un nouveau raster à 1 bande 8BUI à partir du raster source et des bandes spécifiées représentant les composantes rouge, vert et bleu
— ST_HasNoBand - Retourne true si le raster n'a pas la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_Height - Retourne la hauteur du raster en pixels.
— ST_HillShade - Retourne l'hypothétique éclairage d'une bande raster d'élévation en utilisant les valeurs d'azimut, d'altitude, de luminosité et d'échelle fournies.
— ST_Histogram - Retourne un ensemble d'enregistrements résumant une distribution de données raster ou de couverture raster, dans des classes distinctes. Le nombre de classes est calculé automatiquement s'il n'est pas spécifié.
— ST_InterpolateRaster - Interpole une surface quadrillée à partir d'un ensemble de points 3-d, en utilisant les coordonnées X et Y des points sur la grille et la coordonnée Z des points pour l'élévation des points.
— ST_Intersection - Retourne un raster ou un ensemble de paires (géométrie, valeur de pixel) représentant la partie partagée de deux rasters ou l'intersection géométrique d'une vectorisation du raster et d'une géométrie.

— ST_IntersectionFractions - Calculates the fraction of each raster cell that is covered by a given geometry.
— ST_Intersects - Retourne vrai si le raster rastA intersecte spatialement le raster rastB.
— ST_IsEmpty - Retourne true si le raster est vide (largeur = 0 et hauteur = 0). Sinon, retourne false.
— ST_MakeEmptyCoverage - Couvre une zone géo-référencée avec une grille raster de tuiles vides.
— ST_MakeEmptyRaster - Retourne un raster vide (sans aucune bande) de dimension donnée (width & height), de coin supérieur gauche à X et Y, de paramètres de taille de pixel données (scalex, scaley, skewx & skewy) et de système de référence spatial (srid) donné. Si un raster est spécifié, retourne un nouveau raster de même taille, alignement et SRID. Si srid n'est pas spécifié, le système de référence spatial est défini à inconnu (0).
— ST_MapAlgebra (callback function version) - Version avec fonction de rappel - Retourne un raster à une bande à partir d'un ou plusieurs rasters d'entrée, d'index de bandes et d'une fonction de rappel spécifiée par l'utilisateur.
— ST_MapAlgebraExpr - Version avec raster à 1 bande : Crée un nouveau raster à une bande formé par l'application d'une opération algébrique PostgreSQL valide sur la bande d'entrée du raster et du type de pixel fourni. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_MapAlgebraExpr - Version avec 2 bandes : Crée un nouveau raster à une bande formé en appliquant une opération algébrique PostgreSQL valide sur les deux bandes du raster d'entrée et du type de pixel fourni. La bande 1 de chaque raster est utilisée si aucun numéro de bande n'est spécifié. Le raster résultant sera aligné (échelle, obliquité et coins) sur la grille définie par le premier raster et aura son étendue définie par le paramètre "extenttype". Les valeurs de "extenttype" peuvent être : INTERSECTION, UNION, FIRST, SECOND.
— ST_MapAlgebraFct - Version à 1 bande - Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide sur la bande d'entrée du raster et sur le type de pixel fourni. La bande 1 est utilisée si aucune bande n'est spécifiée.
— ST_MapAlgebraFct - Version à 2 bandes - Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide sur les 2 bandes d'entrée du raster et sur le type de pixel fourni. La bande 1 est utilisée si aucune bande n'est spécifiée. Le type d'étendue INTERSECTION est utilisé si non spécifié.
— ST_MapAlgebraFctNgb - Version à 1 bande : Algèbre cartographique Plus proche voisin en utilisant une fonction PostgreSQL définie par l'utilisateur. Retourne un raster dont les valeurs sont le résultat d'une fonction utilisateur PLPGSQL prenant un voisinage des valeurs de la bande raster d'entrée.
— ST_MapAlgebra (expression version) - Version avec expression - Retourne un raster à une bande à partir d'un ou deux rasters d'entrée, d'index de bandes et d'une ou plusieurs expressions SQL spécifiées par l'utilisateur.
— ST_MemSize - Retourne l'espace utilisé par le raster (en octets).
— ST_MetaData - Retourne les méta-données de base de l'objet raster : taille des pixels, rotation, coin haut/bas gauche, etc.
— ST_MinConvexHull - Retourne la géométrie de l'enveloppe convexe du raster en excluant les pixels NODATA.
— ST_NearestValue - Retourne la valeur la plus proche différent de NODATA pour une bande raster spécifiée au pixel donné par columnx et rowy, ou à un point géométrique spécifié dans le même système de référence spatial que le raster.
— ST_Neighborhood - Retourne un tableau 2-D de double avec les valeurs non NODATA autour du pixel de la bande spécifiée, aux coordonnées spécifiées par columnX & rowY ou par un point géométrique dans le même système de référence spatial que le raster.
— ST_NotSameAlignmentReason - Retourne un texte indiquant si les rasters sont alignés et, s'ils ne le sont pas, la raison du problème.
— ST_NumBands - Retourne le nombre de bandes de l'objet raster.
— ST_Overlaps - Retourne true si les raster rastA et rastB intersectent mais que l'un ne contient pas complètement l'autre.
— ST_PixelAsCentroid - Retourne le centroïde (point géométrique) de la zone représentée par un pixel.
— ST_PixelAsCentroids - Retourne le centroïde (point géométrique) pour chaque pixel de la bande raster, avec sa valeur et les coordonnées raster X et Y. Le point géométrique est le centroïde de la zone représentée par un pixel.
— ST_PixelAsPoint - Retourne le point géométrique du coin supérieur gauche du pixel.
— ST_PixelAsPoints - Retourne un point géométrique pour chaque pixel de la bande raster, avec sa valeur et ses coordonnées raster X et Y. Les coordonnées du points sont ceux du coin supérieur gauche du pixel.
— ST_PixelAsPolygon - Retourne la géométrie polygonale qui délimite le pixel pour une ligne et colonne spécifiées.
— ST_PixelAsPolygons - Retourne la géométrie polygonale qui délimite chaque pixel de la bande raster, avec la valeur et les coordonnées raster X et Y de chaque pixel.
— ST_PixelHeight - Retourne la hauteur d'un pixel, dans l'unité du système de référence spatial.
— ST_PixelOfValue - Retourne les coordonnées columnx, rowy du pixel dont la valeur est égale à la valeur recherchée.
— ST_PixelWidth - Retourne la largeur d'un pixel, dans l'unité du système de référence spatial.

— ST_Polygon - Retourne une géométrie multipolygonale formée par l'union des pixels dont la valeur est différente de nodata. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.
— ST_Quantile - Calcule les quantiles d'un raster ou d'une couverture raster, dans le contexte de l'échantillon ou de la population. Ainsi, une valeur peut être examinée pour se situer au percentile de 25%, 50% ou 75% du raster.
— ST_RastFromHexWKB - Retourne un raster à partir d'un raster Well-Known Binary (WKB) en hexadécimal.
— ST_RastFromWKB - Retourne un raster à partir d'un raster Well-Known Binary (WKB).
— ST_RasterToWorldCoord - Retourne le coin supérieur gauche du raster, sous forme de coordonnées X et Y (longitude et latitude) d'un colonne et d'une ligne. Les numéros de colonne et de ligne commencent à 1.
— ST_RasterToWorldCoordX - Retourne la coordonnée X du coin supérieur gauche du raster à column et row. Les numéros de colonne et de ligne commencent à 1.
— ST_RasterToWorldCoordY - Retourne la coordonnée Y du coin supérieur gauche du raster à column et row. Les numéros de colonne et de ligne commencent à 1.
— ST_Reclass - Crée un nouveau raster composé de types de bandes reclassifiés par rapport à l'original. La bande n est la bande à modifier. Si nband n'est pas spécifié, la bande 1 est utilisée. Toutes les autres bandes sont retournées inchangées. Cas d'utilisation : convertir une bande 16BUI en 8BUI et ainsi de suite pour un rendu plus simple en tant que formats visualisables.
— ST_ReclassExact - Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
— ST_Resample - Rééchantillonne un raster, en utilisant l'algorithme spécifié, les nouvelles dimensions, un coin arbitraire de la grille et un ensemble de paramètres de géo-référencement définis ou empruntés à un autre raster.
— ST_Rescale - Rééchantillonne un raster en ajustant juste son échelle (ou la taille des pixels). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline), Lanczos, Max ou Min. La valeur par défaut est NearestNeighbor.
— ST_Resize - Redimensionne un raster à une nouvelle largeur/hauteur
— ST_Reskew - Rééchantillonne un raster en ajustant simplement son obliquité (skew, ou paramètre de rotation). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.
— ST_Rotation - Retourne l'angle de rotation du raster en radians.
— ST_Roughness - Retourne un raster avec la rugosité d'un MNT.
— ST_SRID - Retourne l'identifiant du système de référence spatial du raster, tel que défini dans la table spatial_ref_sys.
— ST_SameAlignment - Retourne true si les rasters ont les mêmes skew, scale, spatial ref, et offset (les pixels peuvent être placés sur la même grille sans être coupés) et false si ce n'est pas le cas, avec une NOTICE détaillant le problème.
— ST_ScaleX - Renvoie la composante X de la largeur du pixel dans l'unité du système de référence spatial.
— ST_ScaleY - Renvoie la composante Y de la hauteur du pixel dans l'unité du système de référence spatial.
— ST_SetBandIndex - Met à jour le numéro de bande externe d'une bande out-db
— ST_SetBandIsNoData - Définit la valeur du flag isnodata de la bande à TRUE.
— ST_SetBandNoDataValue - Définit la valeur pour l'absence de données (nodata) pour la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée. Pour indiquer qu'une bande n'a pas de valeur nodata, définir la valeur nodata = NULL.
— ST_SetBandPath - Met à jour le chemin externe et le numéro de bande d'une bande out-db
— ST_SetGeoReference - Définit les 6 paramètres de géo-référencement en un seul appel. Les nombres doivent être séparés par un espace. Accepte les formats GDAL (par défaut) ou ESRI.
— ST_SetM - Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée M copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.
— ST_SetRotation - Définit la rotation du raster en radians.
— ST_SetSRID - Modifie le SRID d'un raster à une valeur définie dans la table spatial_ref_sys.
— ST_SetScale - Définit la résolution des pixels en X et Y en unité du système de référence spatial : nombre d'unités/pixel en largeur/hauteur.
— ST_SetSkew - Définit l'obliquité X et Y (skew, ou paramètre de rotation). Si une seule valeur est spécifiée, la même valeur est utilisée pour X et pour Y.
— ST_SetUpperLeft - Modifie les coordonnées du pixel du coin supérieur gauche du raster, selon les coordonnées X et Y projetées.
— ST_SetValue - Retourne un nouveau raster en modifiant la valeur du pixel pour la bande spécifiée et aux coordonnées columnx, rowy, ou pour tous les pixels qui intersectent une géométrie spécifiée. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié.

— ST_SetValues - Retourne un nouveau raster en modifiant les valeurs de certains pixels d'une bande spécifiée.
— ST_SetZ - Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée Z copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.
— ST_SkewX - Retourne l'obliquité géo-référencée X (paramètre de rotation).
— ST_SkewY - Retourne l'obliquité géo-référencée Y (paramètre de rotation).
— ST_Slope - Retourne la pente (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_SnapToGrid - Rééchantillonne un raster en l'accrochant sur une grille. Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.
— ST_Summary - Retourne un résumé du contenu du raster sous forme de texte.
— ST_SummaryStats - Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.
— ST_SummaryStatsAgg - Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.
— ST_TPI - Retourne un raster avec l'index de position topographique (TPI) calculé.
— ST_TRI - Retourne un raster avec l'indice de rugosité du terrain (TRI) calculé.
— ST_Tile - Retourne un ensemble de rasters issus de la division d'un raster d'entrée selon les dimensions spécifiées pour les rasters de sortie.
— ST_Touches - Retourne true si les raster rastA et rastB ont au moins un point en commun mais que leurs intérieurs n'intersectent pas.
— ST_Transform - Reprojette un raster depuis un système de référence spatial vers un autre, en utilisant l'algorithme de rééchantillonnage spécifié. Les algorithmes possibles sont NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.
— ST_Union - Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— ST_UpperLeftX - Retourne la coordonnée X du coin supérieur gauche du raster projeté dans le système de référence spatial.
— ST_UpperLeftY - Retourne la coordonnée Y du coin supérieur gauche du raster projeté dans le système de référence spatial.
— ST_Value - Retourne la valeur d'une bande raster spécifiée au pixel donné par columnx, rowy, ou à un point géométrique spécifié. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié. Si exclude_nodata_value vaut false, tous les pixels y compris ceux ayant la valeur nodata sont considérés comme intersectés et leur valeur sera retournée. Si exclude_nodata_value n'est pas spécifié, la valeur est lue depuis les méta-données du raster.
— ST_ValueCount - Retourne un ensemble d'enregistrements contenant une valeur de pixels et le nombre de pixels ayant cette valeur dans la bande du raster spécifié (ou de la couverture raster). Si aucune bande n'est spécifiée, la bande 1 est utilisée. Par défaut, les pixels de valeur nodata ne sont pas comptés, et toutes les autres valeurs sont retournées, avec leur valeur arrondies à l'entier le plus proche.
— ST_Width - Retourne la largeur du raster en pixels.
— ST_Within - Retourne true si aucun point du raster rastA ne se trouve à l'extérieur du raster rastB et si au moins un point de l'intérieur du raster rastA se trouve à l'intérieur du raster rastB.
— ST_WorldToRasterCoord - Retourne le coin supérieur gauche comme colonne et ligne, en fonction de coordonnées géométriques X et Y (longitude et latitude) ou un point géométrique dans le système de référence spatial du raster.
— ST_WorldToRasterCoordX - Retourne la colonne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw) exprimés dans le système de référence spatial du raster.
— ST_WorldToRasterCoordY - Retourne la ligne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw) exprimés dans le système de référence spatial du raster.
— UpdateRasterSRID - Change le SRID de tous les rasters dans la table et colonne en paramètres.

13.6 Fonctions PostGIS de dump Geometry / Geography / Raster

Les fonctions ci-dessous sont des fonctions PostGIS qui prennent en entrée ou renvoient en sortie un ensemble ou un seul objet de type geometry_dump ou geomval.

— ST_DumpAsPolygons - Retourne un ensemble d'enregistrements de type geomval (geom, val), à partir d'une bande raster donnée. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.

— ST_Intersection - Retourne un raster ou un ensemble de paires (géométrie, valeur de pixel) représentant la partie partagée de deux rasters ou l'intersection géométrique d'une vectorisation du raster et d'une géométrie.
— ST_Dump - Renvoie un ensemble de lignes geometry_dump pour les composants d'une géométrie.
— ST_DumpPoints - Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_DumpRings - Renvoie un ensemble de lignes geometry_dump pour les anneaux extérieurs et intérieurs d'un polygone.
— ST_DumpSegments - Renvoie un ensemble de lignes geometry_dump pour les segments d'une géométrie.

13.7 Fonctions Box de PostGIS

Les fonctions ci-dessous sont des fonctions PostGIS qui prennent en entrée ou renvoient en sortie la famille box* des types spatiaux PostGIS. La famille de types box comprend box2d, et box3d

— Box2D - Renvoie une BOX2D représentant l'étendue 2D d'une géométrie.
— Box3D - Renvoie une BOX3D représentant l'étendue 3D d'une géométrie.
— MakeTopologyPrecise - Snap topology vertices to precision grid.
— Box3D - Retourne la représentation 3d de la boîte englobante du raster.
— ST_3DExtent - Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.
— ST_3DMakeBox - Crée un BOX3D défini par deux géométries de points 3D.
— ST_AsMVTGeom - Transforme une géométrie dans l'espace de coordonnées d'une tuile MVT.
— ST_AsTWKB - Renvoie la géométrie sous forme de TWKB, diminutif de "Tiny Well-Known Binary"
— ST_Box2dFromGeoHash - Retourne une BOX2D à partir d'une chaîne GeoHash.
— ST_ClipByBox2D - Calcule la partie d'une géométrie située à l'intérieur d'un rectangle.
— ST_EstimatedExtent - Renvoie l'étendue estimée d'une table spatiale.
— ST_Expand - Renvoie une boîte de délimitation développée à partir d'une autre boîte de délimitation ou d'une géométrie.
— ST_Extent - Fonction agrégée qui renvoie la boîte de délimitation des géométries.
— ST_MakeBox2D - Crée un BOX2D défini par deux géométries de points 2D.
— ST_RemoveIrrelevantPointsForView - Supprime les points qui ne sont pas pertinents pour le rendu d'une vue rectangulaire spécifique d'une géométrie.
— ST_XMax - Retourne les maxima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_XMin - Retourne les minima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_XSize - Returns the X size of a 2D or 3D bounding box or a geometry.
— ST_YMax - Retourne les maxima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_YMin - Retourne les minima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_YSize - Returns the Y size of a 2D or 3D bounding box or a geometry.
— ST_ZMax - Retourne les maxima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_ZMin - Retourne les minima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_ZSize - Returns the Z size of a 2D or 3D bounding box or a geometry.
— RemoveUnusedPrimitives - Supprime les primitives topologiques qui ne sont pas nécessaires pour définir les objets TopoGeometry existants.
— ValidateTopology - Renvoie un ensemble d'objets validatetopology_returntype détaillant les problèmes liés à la topologie.
— ValidateTopologyPrecision - Returns non-precise vertices in the topology.
— ~(box2df,box2df) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient une autre boîte de délimitation de précision flottante 2D (BOX2DF).
— ~(box2df,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient la boîte de délimitation 2D d'une géométrie.
— ~(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D d'une géométrie contient une boîte de délimitation de précision flottante 2D (GIDX).
— @(box2df,box2df) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans une autre boîte de délimitation de précision flottante 2D.
— @(box2df,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans la boîte de délimitation 2D d'une géométrie.

— @(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D d'une géométrie est contenue dans une boîte de délimitation 2D à précision flottante (BOX2DF).
— &&(box2df,box2df) - Renvoie TRUE si deux boîtes de délimitation 2D à précision flottante (BOX2DF) se croisent.
— &&(box2df,geometry) - Renvoie TRUE si une boîte de délimitation 2D de précision flottante (BOX2DF) intersecte la boîte de délimitation 2D (mise en cache) d'une géométrie.
— &&(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D (en cache) d'une géométrie intersecte une boîte de délimitation 2D de précision flottante (BOX2DF).

13.8 Fonctions PostGIS supportant la 3D

Les fonctions ci-dessous sont des fonctions PostGIS qui n'éliminent pas l'indice Z.

— AddGeometryColumn - Ajoute une colonne de géométrie à une table existante.
— Box3D - Renvoie une BOX3D représentant l'étendue 3D d'une géométrie.
— CG_3DAlphaWrapping - Computes a 3D Alpha-wrapping strictly enclosing a geometry.
— CG_3DArea - Calcule la surface des géométries de surface 3D. Retourne 0 pour les solides.
— CG_3DConvexHull - Calcule l'enveloppe convexe 3D d'une géométrie.
— CG_3DDifference - Effectuer une différence 3D
— CG_3DIntersection - Réaliser une intersection 3D
— CG_3DRotate - Fait pivoter une géométrie dans l'espace 3D autour d'un vecteur d'axe.
— CG_3DScale - Met à l'échelle une géométrie en fonction de facteurs distincts le long des axes X, Y et Z.
— CG_3DScaleAroundCenter - Met à l'échelle une géométrie dans l'espace 3D autour d'un point central spécifié.
— CG_3DTranslate - Déplace une géométrie en fonction de décalages donnés dans l'espace 3D.
— CG_3DUnion - Réaliser l'union 3D en utilisant postgis_sfcgal.
— CG_ConstrainedDelaunayTriangles - Renvoie une triangulation de Delaunay contrainte autour de la géométrie d'entrée donnée.
— CG_Extrude - Extruder une surface vers un volume
— CG_ForceLHR - Force l'orientation LHR d'un objet
— CG_IsPlanar - Vérifie si une surface est planaire ou non
— CG_IsSolid - Teste si la géométrie est un solide. Aucun contrôle de validité n'est effectué.
— CG_MakeSolid - Transformer la géométrie dans un solide. Aucune vérification n'est effectuée. Pour obtenir un solide valide, la géométrie d'entrée doit être une surface polyédrique fermée ou un TIN fermé.
— CG_NurbsCurveApproximate - Creates an approximating NURBS curve fitting data points within a tolerance
— CG_NurbsCurveDerivative - Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
— CG_NurbsCurveInterpolate - Creates an interpolating NURBS curve passing through all given data points
— CG_Orientation - Détermine l'orientation d'une surface
— CG_RotateX - Fait pivoter une géométrie autour de l'axe X d'un angle donné.
— CG_RotateY - Fait pivoter une géométrie autour de l'axe Y d'un angle donné.
— CG_RotateZ - Fait pivoter une géométrie autour de l'axe Z d'un angle donné.
— CG_Simplify - Reduces the complexity of a geometry while preserving essential features and Z/M values.
— CG_Tesselate - Effectue la tessellation de la surface d'un polygone ou d'une surface polyédrique et renvoie un TIN ou une collection de TINS
— CG_Visibility - Calculer un polygone de visibilité à partir d'un point ou d'un segment dans une géométrie polygonale
— CG_Volume - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
— DropGeometryColumn - Supprime une colonne géométrique d'une table spatiale.
— GeometryType - Renvoie le type d'une géométrie sous forme de texte.
— ST_3DClosestPoint - Renvoie le point 3D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne 3D la plus courte.
— ST_3DDFullyWithin - Teste si deux géométries 3D sont entièrement comprises dans une distance 3D donnée
— ST_3DDWithin - Teste si deux géométries 3D se trouvent à une distance 3D donnée

— ST_3DDistance - Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DExtent - Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.
— ST_3DInterpolatePoint - Returns the interpolated measure of a geometry closest to a point in 3D.
— ST_3DIntersects - Teste si deux géométries se croisent dans l'espace en 3D - uniquement pour les points, les lignes, les polygones, les surfaces polyédriques (aire)
— ST_3DLength - Renvoie la longueur 3D d'une géométrie linéaire.
— ST_3DLineInterpolatePoint - Renvoie un point interpolé le long d'une ligne 3D à un emplacement fractionnaire.
— ST_3DLongestLine - Renvoie la ligne 3D la plus longue entre deux géométries
— ST_3DMaxDistance - Renvoie la distance maximale cartésienne 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DPerimeter - Renvoie le périmètre 3D d'une géométrie polygonale.
— ST_3DShortestLine - Renvoie la ligne 3D la plus courte entre deux géométries
— ST_AddMeasure - Interpole les mesures le long d'une géométrie linéaire.
— ST_AddPoint - Ajoute un point à une LineString.
— ST_Affine - Appliquer une transformation affine 3D à une géométrie.
— ST_AsBinary - Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsEWKB - Renvoie la représentation Extended Well-Known Binary (EWKB) de la géométrie avec les métadonnées SRID.
— ST_AsEWKT - Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.
— ST_AsGML - Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsGeoJSON - Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_AsHEXEWKB - Renvoie une géométrie au format HEXEWKB (en tant que texte) en utilisant l'encodage little-endian (NDR) ou big-endian (XDR).
— ST_AsKML - Renvoyer la géométrie sous forme d'élément KML.
— ST_AsX3D - Renvoie une géométrie au format X3D xml node element : ISO-IEC-19776-1.2-X3DEncodings-XML
— ST_Boundary - Renvoie la limite d'une géométrie.
— ST_BoundingDiagonal - Retourne la diagonale de la boîte englobante pour la géométrie en argument.
— ST_CPAWithin - Teste si le point d'approche le plus proche de deux trajectoires se trouve dans la distance spécifiée.
— ST_CatmullRomSmoothing - Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
— ST_ChaikinSmoothing - Renvoie une version lissée d'une géométrie, en utilisant l'algorithme Chaikin
— ST_ClosestPointOfApproach - Renvoie une mesure au point d'approche le plus proche de deux trajectoires.
— ST_Collect - Crée une géométrie GeometryCollection ou Multi à partir d'un ensemble de géométries.
— ST_ConvexHull - Calcule l'enveloppe convexe d'une géométrie.
— ST_CoordDim - Renvoie la dimension des coordonnées d'une géométrie.
— ST_CurveN - Renvoie la Nième courbe composante d'une CompoundCurve.
— ST_CurveToLine - Convertit une géométrie contenant des courbes en une géométrie linéaire.
— ST_DelaunayTriangles - Renvoie la triangulation de Delaunay des sommets d'une géométrie.
— ST_Difference - Calcule une géométrie représentant la partie de la géométrie A qui n'intersecte pas la géométrie B.
— ST_DistanceCPA - Renvoie la distance entre le point d'approche le plus proche de deux trajectoires.
— ST_Dump - Renvoie un ensemble de lignes geometry_dump pour les composants d'une géométrie.
— ST_DumpPoints - Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_DumpRings - Renvoie un ensemble de lignes geometry_dump pour les anneaux extérieurs et intérieurs d'un polygone.
— ST_DumpSegments - Renvoie un ensemble de lignes geometry_dump pour les segments d'une géométrie.
— ST_EndPoint - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
— ST_ExteriorRing - Renvoie une ligne représentant l'anneau extérieur d'un polygone.
— ST_FlipCoordinates - Renvoie une version d'une géométrie dont les axes X et Y sont inversés.
— ST_Force2D - Forcer les géométries à passer en "mode bidimensionnel".
— ST_ForceCurve - Retransformation d'une géométrie dans son type de courbure, le cas échéant.

— ST_ForcePolygonCCW - Oriente tous les anneaux extérieurs dans le sens inverse des aiguilles d'une montre et tous les anneaux intérieurs dans le sens des aiguilles d'une montre.
— ST_ForcePolygonCW - Oriente tous les anneaux extérieurs dans le sens des aiguilles d'une montre et tous les anneaux intérieurs dans le sens inverse des aiguilles d'une montre.
— ST_ForceRHR - Force l'orientation des sommets d'un polygone à suivre la règle de la main droite.
— ST_ForceSFS - Force geometries to use SFS 1.1 or 1.2 geometry types.
— ST_Force3D - Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.
— ST_Force3DZ - Forcer les géométries en mode XYZ.
— ST_Force4D - Forcer les géométries en mode XYZM.
— ST_ForceCollection - Convertir la géométrie en une GEOMETRYCOLLECTION.
— ST_GeomFromEWKB - Retourne un objet ST_Geometry à partir de sa représentation binaire étendue (Extended Well-Known Binary representation, EWKB).
— ST_GeomFromEWKT - Retourne un objet ST_Geometry à partir de sa représentation textuelle étendue (Extended Well-Known Text representation, EWKT).
— ST_GeomFromGML - Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry
— ST_GeomFromGeoJSON - Prend en entrée une géométrie au format geojson et renvoie un objet Postgis de type geometry
— ST_GeomFromKML - Prend en entrée une géométrie au format KML et renvoie un objet Postgis de type geometry
— ST_GeometricMedian - Renvoie la médiane géométrique d'un MultiPoint.
— ST_GeometryN - Renvoie un élément d'une collection de géométries.
— ST_GeometryType - Renvoie le type SQL-MM d'une géométrie sous forme de texte.
— ST_HasArc - Teste si une géométrie contient un arc de cercle
— ST_HasM - Vérifie si une géométrie a une dimension M (mesure).
— ST_HasZ - Vérifie si une géométrie possède une dimension Z.
— ST_InteriorRingN - Renvoie le Nième anneau intérieur (trou) d'un polygone.
— ST_InterpolatePoint - Renvoie la mesure interpolée d'une géométrie la plus proche d'un point.
— ST_Intersection - Calcule une géométrie représentant la partie partagée des géométries A et B.
— ST_IsClosed - Teste si les points de départ et d'arrivée d'une LineString coïncident. Pour une PolyhedralSurface, teste si elle est fermée (volumétrique).
— ST_IsCollection - Teste si une géométrie est un type de collection de géométrie.
— ST_IsPolygonCCW - Teste si les polygones ont des anneaux extérieurs orientés dans le sens inverse des aiguilles d'une montre et des anneaux intérieurs orientés dans le sens des aiguilles d'une montre.
— ST_IsPolygonCW - Teste si les polygones ont des anneaux extérieurs orientés dans le sens des aiguilles d'une montre et des anneaux intérieurs orientés dans le sens inverse des aiguilles d'une montre.
— ST_IsSimple - Teste si une géométrie n'a pas de points d'auto-intersection ou d'auto-tangente.
— ST_IsValidTrajectory - Teste si la géométrie est une trajectoire valide.
— ST_LengthSpheroid - Renvoie la longueur/périmètre 2D ou 3D d'une géométrie lon/lat sur un sphéroïde.
— ST_LineFromMultiPoint - Crée une LineString à partir d'une géométrie MultiPoint.
— ST_LineInterpolatePoint - Renvoie un point interpolé le long d'une ligne à un emplacement fractionnaire.
— ST_LineInterpolatePoints - Renvoie des points interpolés le long d'une ligne à un intervalle fractionnaire.
— ST_LineSubstring - Renvoie la partie d'une ligne située entre deux emplacements fractionnaires.
— ST_LineToCurve - Convertit une géométrie linéaire en une géométrie courbe.
— ST_LocateBetweenElevations - Renvoie les parties d'une géométrie qui se trouvent dans un intervalle d'élévation (Z).
— ST_M - Renvoie la coordonnée M d'un point.
— ST_MMax - Returns the M maxima of a geometry.
— ST_MMin - Returns the M minima of a geometry.
— ST_MSize - Returns the M size of a geometry.
— ST_MakeLine - Crée une LineString à partir de géométries Point, MultiPoint ou LineString.
— ST_MakePoint - Crée un point 2D, 3DZ ou 4D.
— ST_MakePolygon - Creates a Polygon or CurvePolygon from a shell and optional list of holes.
— ST_MakeValid - Tente de rendre valide une géométrie invalide sans perdre de sommets.

— ST_MemSize - Renvoie la quantité d'espace mémoire que prend une géométrie.
— ST_MemUnion - Fonction d'agrégation qui fusionne les géométries d'une manière efficace sur le plan de la mémoire mais plus lente
— ST_NDims - Renvoie la dimension des coordonnées d'une géométrie.
— ST_NPoints - Retourne le nombre de points (vertex) d'un objet géométrique.
— ST_NRings - Renvoie le nombre d'anneaux dans une géométrie polygonale.
— ST_Node - Nœuds d'une collection de lignes.
— ST_NumCurves - Renvoie le nombre de courbes composantes d'une CompoundCurve.
— ST_NumGeometries - Renvoie le nombre d'éléments dans une collection de géométrie.
— ST_NumPatches - Returns the number of faces in a PolyhedralSurface or TIN.
— ST_PatchN - Returns the Nth face of a PolyhedralSurface or TIN.
— ST_PointFromWKB - Construit une géométrie depuis la représentation binaire WKB et le SRID donné
— ST_PointN - Renvoie le Nième point de la première LineString ou LineString circulaire d'une géométrie.
— ST_PointOnSurface - Calcule un point dont on garantit qu'il se trouve dans un polygone ou sur une géométrie.
— ST_Points - Renvoie un MultiPoint contenant les coordonnées d'une géométrie.
— ST_Polygon - Crée un polygone à partir d'une LineString avec un SRID spécifié.
— ST_RemovePoint - Supprime un point d'une ligne.
— ST_RemoveRepeatedPoints - Renvoie une version d'une géométrie dont les points en double ont été supprimés.
— ST_Reverse - Retourne la géométrie avec l'ordre des sommets inversé.
— ST_Rotate - Fait pivoter une géométrie autour d'un point d'origine.
— ST_RotateX - Fait pivoter une géométrie autour de l'axe X.
— ST_RotateY - Fait pivoter une géométrie autour de l'axe Y.
— ST_RotateZ - Fait pivoter une géométrie autour de l'axe Z.
— ST_Scale - Met à l'échelle une géométrie en fonction de facteurs donnés.
— ST_Scroll - Modifier le point de départ d'une LineString fermée.
— ST_SetPoint - Remplacer le point d'une ligne par un point donné.
— ST_ShiftLongitude - Décale les coordonnées de longitude d'une géométrie entre -180..180 et 0..360.
— ST_SnapToGrid - Accrocher tous les points de la géométrie d'entrée à une grille régulière.
— ST_StartPoint - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
— ST_SwapOrdinates - Renvoie une version de la géométrie donnée avec les valeurs d'ordonnées permutées.
— ST_SymDifference - Calcule une géométrie représentant les parties des géométries A et B qui ne s'intersectent pas.
— ST_TransScale - Traduit et met à l'échelle une géométrie en fonction des paramètres offset et factor spécifiés.
— ST_Translate - Traduit une géométrie en fonction de décalages donnés.
— ST_UnaryUnion - Calcule l'union des composantes d'une seule géométrie.
— ST_Union - Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.
— ST_WrapX - Enveloppe une géométrie autour d'une valeur X.
— ST_X - Renvoie la coordonnée X d'un point.
— ST_XMax - Retourne les maxima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_XMin - Retourne les minima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_XSize - Returns the X size of a 2D or 3D bounding box or a geometry.
— ST_Y - Renvoie la coordonnée Y d'un point.
— ST_YMax - Retourne les maxima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_YMin - Retourne les minima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_YSize - Returns the Y size of a 2D or 3D bounding box or a geometry.
— ST_Z - Renvoie la coordonnée Z d'un point.
— ST_ZMFlag - Retourne un code indiquant la dimension des coordonnées ZM d'une géométrie.
— ST_ZMax - Retourne les maxima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_ZMin - Retourne les minima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_ZSize - Returns the Z size of a 2D or 3D bounding box or a geometry.
— Equals - Retourne vrai si deux topogeometries sont composées des mêmes primitives topologiques.

— Intersects - Retourne true si une paire de primitives des deux topogeometries s'intersectent.
— UpdateGeometrySRID - Met à jour le SRID de tous les éléments d'une colonne géométrique et les métadonnées de la table.
— &&& - Renvoie TRUE si la boîte de délimitation n-D de A intersecte la boîte de délimitation n-D de B.
— &&&(geometry,gidx) - Renvoie TRUE si la boîte de délimitation n-D (en cache) d'une géométrie intersecte une boîte de délimitation de précision flottante n-D (GIDX).
— &&&(gidx,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante n-D (GIDX) intersecte la boîte de délimitation n-D (mise en cache) d'une géométrie.
— &&&(gidx,gidx) - Renvoie TRUE si deux boîtes de délimitation (GIDX) de précision flottante n-D se croisent.

13.9 Fonctions d'aide aux géométries courbes de PostGIS

Les fonctions ci-dessous sont des fonctions PostGIS qui peuvent utiliser CIRCULARSTRING, CURVEPOLYGON et d'autres types de géométrie courbe

— AddGeometryColumn - Ajoute une colonne de géométrie à une table existante.
— Box2D - Renvoie une BOX2D représentant l'étendue 2D d'une géométrie.
— Box3D - Renvoie une BOX3D représentant l'étendue 3D d'une géométrie.
— DropGeometryColumn - Supprime une colonne géométrique d'une table spatiale.
— GeometryType - Renvoie le type d'une géométrie sous forme de texte.
— PostGIS_AddBBox - Ajoute une bounding box à la géométrie.
— PostGIS_DropBBox - Supprime le cache de la boîte de délimitation de la géométrie.
— PostGIS_HasBBox - Renvoie TRUE si la bbox de cette géométrie est en cache, sinon FALSE.
— ST_3DExtent - Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.
— ST_Affine - Appliquer une transformation affine 3D à une géométrie.
— ST_AsBinary - Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsEWKB - Renvoie la représentation Extended Well-Known Binary (EWKB) de la géométrie avec les métadonnées SRID.
— ST_AsEWKT - Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.
— ST_AsHEXEWKB - Renvoie une géométrie au format HEXEWKB (en tant que texte) en utilisant l'encodage little-endian (NDR) ou big-endian (XDR).
— ST_AsSVG - Renvoie les données de chemin SVG pour une géométrie.
— ST_AsText - Renvoie la représentation Well-Known Text (WKT) de la géométrie/geography sans métadonnées SRID.
— ST_ClusterDBSCAN - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie d'entrée en utilisant l'algorithme DBSCAN.
— ST_ClusterWithin - Fonction agrégée qui regroupe les géométries en fonction de la distance de séparation.
— ST_ClusterWithinWin - Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, regroupement en utilisant la distance de séparation.
— ST_Collect - Crée une géométrie GeometryCollection ou Multi à partir d'un ensemble de géométries.
— ST_CoordDim - Renvoie la dimension des coordonnées d'une géométrie.
— ST_CurveToLine - Convertit une géométrie contenant des courbes en une géométrie linéaire.
— ST_Distance - Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_Dump - Renvoie un ensemble de lignes geometry_dump pour les composants d'une géométrie.
— ST_DumpPoints - Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_EndM - Returns the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.
— ST_EndPoint - Returns the last point of a LineString, CircularLineString, or NURBSCurve.
— ST_EstimatedExtent - Renvoie l'étendue estimée d'une table spatiale.
— ST_FlipCoordinates - Renvoie une version d'une géométrie dont les axes X et Y sont inversés.
— ST_Force2D - Forcer les géométries à passer en "mode bidimensionnel".
— ST_ForceCurve - Retransformation d'une géométrie dans son type de courbure, le cas échéant.
— ST_ForceSFS - Force geometries to use SFS 1.1 or 1.2 geometry types.

— ST_Force3D - Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.
— ST_Force3DM - Forcer les géométries en mode XYM.
— ST_Force3DZ - Forcer les géométries en mode XYZ.
— ST_Force4D - Forcer les géométries en mode XYZM.
— ST_ForceCollection - Convertir la géométrie en une GEOMETRYCOLLECTION.
— ST_GeoHash - Retourne une représentation GeoHash de la géométrie.
— ST_GeogFromWKB - Retourne un objet de type geography à partir de sa représentation binaire Well-Know Binary (WKB ou EWKB).
— ST_GeomFromEWKB - Retourne un objet ST_Geometry à partir de sa représentation binaire étendue (Extended Well-Known Binary representation, EWKB).
— ST_GeomFromEWKT - Retourne un objet ST_Geometry à partir de sa représentation textuelle étendue (Extended Well-Known Text representation, EWKT).
— ST_GeomFromText - Retourne un objet ST_Geometry à partir de sa représentation textuelle Well-Known Text (WKT).
— ST_GeomFromWKB - Retourne un objet de type geometry à partir de sa représentation binaire Well-Know Binary (WKB) et d'un SRID optionnel.
— ST_GeometryN - Renvoie un élément d'une collection de géométries.
— = - Renvoie TRUE si les coordonnées et l'ordre des coordonnées de la géométrie/géographie A sont les mêmes que les coordonnées et l'ordre des coordonnées de la géométrie/géographie B.
— &<| - Renvoie TRUE si la boîte englobante de A chevauche ou est inférieure à celle de B.
— ST_HasArc - Teste si une géométrie contient un arc de cercle
— ST_Intersects - Teste si deux géométries se croisent (elles ont au moins un point en commun)
— ST_IsClosed - Teste si les points de départ et d'arrivée d'une LineString coïncident. Pour une PolyhedralSurface, teste si elle est fermée (volumétrique).
— ST_IsCollection - Teste si une géométrie est un type de collection de géométrie.
— ST_IsEmpty - Teste si une géométrie est vide.
— ST_LineToCurve - Convertit une géométrie linéaire en une géométrie courbe.
— ST_MMax - Returns the M maxima of a geometry.
— ST_MMin - Returns the M minima of a geometry.
— ST_MSize - Returns the M size of a geometry.
— ST_MemSize - Renvoie la quantité d'espace mémoire que prend une géométrie.
— ST_NPoints - Retourne le nombre de points (vertex) d'un objet géométrique.
— ST_NRings - Renvoie le nombre d'anneaux dans une géométrie polygonale.
— ST_PointFromWKB - Construit une géométrie depuis la représentation binaire WKB et le SRID donné
— ST_PointN - Renvoie le Nième point de la première LineString ou LineString circulaire d'une géométrie.
— ST_Points - Renvoie un MultiPoint contenant les coordonnées d'une géométrie.
— ST_Rotate - Fait pivoter une géométrie autour d'un point d'origine.
— ST_RotateZ - Fait pivoter une géométrie autour de l'axe Z.
— ST_SRID - Renvoie l'identifiant de référence spatiale d'une géométrie.
— ST_Scale - Met à l'échelle une géométrie en fonction de facteurs donnés.
— ST_SetEndM - Sets the M coordinate of the last point of a LineString, CircularLineString, or NURBSCurve.
— ST_SetSRID - Définir le SRID d'une géométrie.
— ST_SetStartM - Sets the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.
— ST_StartM - Returns the M coordinate of the first point of a LineString, CircularLineString, or NURBSCurve.
— ST_StartPoint - Returns the first point of a LineString, CircularLineString, or NURBSCurve.
— ST_Summary - Renvoie un résumé textuel du contenu d'une géométrie.
— ST_SwapOrdinates - Renvoie une version de la géométrie donnée avec les valeurs d'ordonnées permutées.
— ST_TransScale - Traduit et met à l'échelle une géométrie en fonction des paramètres offset et factor spécifiés.
— ST_Transform - Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent.
— ST_Translate - Traduit une géométrie en fonction de décalages donnés.
— ST_XMax - Retourne les maxima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.

— ST_XMin - Retourne les minima X d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_XSize - Returns the X size of a 2D or 3D bounding box or a geometry.
— ST_YMax - Retourne les maxima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_YMin - Retourne les minima Y d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_YSize - Returns the Y size of a 2D or 3D bounding box or a geometry.
— ST_ZMFlag - Retourne un code indiquant la dimension des coordonnées ZM d'une géométrie.
— ST_ZMax - Retourne les maxima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_ZMin - Retourne les minima Z d'une boîte de délimitation 2D ou 3D ou d'une géométrie.
— ST_ZSize - Returns the Z size of a 2D or 3D bounding box or a geometry.
— UpdateGeometrySRID - Met à jour le SRID de tous les éléments d'une colonne géométrique et les métadonnées de la table.
— ~(box2df,box2df) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient une autre boîte de délimitation de précision flottante 2D (BOX2DF).
— ~(box2df,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient la boîte de délimitation 2D d'une géométrie.
— ~(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D d'une géométrie contient une boîte de délimitation de précision flottante 2D (GIDX).
— && - Renvoi VRAI si la boite englobante 2D de A intersecte la boite englobante 2D de B.
— &&& - Renvoie TRUE si la boîte de délimitation n-D de A intersecte la boîte de délimitation n-D de B.
— @(box2df,box2df) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans une autre boîte de délimitation de précision flottante 2D.
— @(box2df,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans la boîte de délimitation 2D d'une géométrie.
— @(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D d'une géométrie est contenue dans une boîte de délimitation 2D à précision flottante (BOX2DF).
— &&(box2df,box2df) - Renvoie TRUE si deux boîtes de délimitation 2D à précision flottante (BOX2DF) se croisent.
— &&(box2df,geometry) - Renvoie TRUE si une boîte de délimitation 2D de précision flottante (BOX2DF) intersecte la boîte de délimitation 2D (mise en cache) d'une géométrie.
— &&(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D (en cache) d'une géométrie intersecte une boîte de délimitation 2D de précision flottante (BOX2DF).
— &&&(geometry,gidx) - Renvoie TRUE si la boîte de délimitation n-D (en cache) d'une géométrie intersecte une boîte de délimitation de précision flottante n-D (GIDX).
— &&&(gidx,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante n-D (GIDX) intersecte la boîte de délimitation n-D (mise en cache) d'une géométrie.
— &&&(gidx,gidx) - Renvoie TRUE si deux boîtes de délimitation (GIDX) de précision flottante n-D se croisent.

13.10 Fonctions de support des surfaces polyédriques de PostGIS

Les fonctions ci-dessous sont des fonctions PostGIS qui peuvent utiliser les géométries POLYHEDRALSURFACE, POLYHEDRALSURFACEM

— Box2D - Renvoie une BOX2D représentant l'étendue 2D d'une géométrie.
— Box3D - Renvoie une BOX3D représentant l'étendue 3D d'une géométrie.
— CG_3DArea - Calcule la surface des géométries de surface 3D. Retourne 0 pour les solides.
— CG_3DConvexHull - Calcule l'enveloppe convexe 3D d'une géométrie.
— CG_3DDifference - Effectuer une différence 3D
— CG_3DIntersection - Réaliser une intersection 3D
— CG_3DUnion - Réaliser l'union 3D en utilisant postgis_sfcgal.
— CG_ApproximateMedialAxis - Calculer l'axe médian approximatif d'une géométrie aréolaire.
— CG_Extrude - Extruder une surface vers un volume
— CG_ForceLHR - Force l'orientation LHR d'un objet
— CG_GenerateFlatRoof - Generate a flat (box) roof from a footprint polygon.

— CG_GenerateGableRoof - Generate a gable roof from a footprint polygon.
— CG_GenerateHippedRoof - Generate a hipped roof from a footprint polygon.
— CG_GenerateRoof - Generate a roof of the requested type from a footprint polygon.
— CG_GenerateSkillionRoof - Generate a skillion (single-slope) roof from a footprint polygon.
— CG_IsPlanar - Vérifie si une surface est planaire ou non
— CG_IsSolid - Teste si la géométrie est un solide. Aucun contrôle de validité n'est effectué.
— CG_MakeSolid - Transformer la géométrie dans un solide. Aucune vérification n'est effectuée. Pour obtenir un solide valide, la géométrie d'entrée doit être une surface polyédrique fermée ou un TIN fermé.
— CG_PolygonRepair - Repair an invalid polygon or multipolygon.
— CG_StraightSkeleton - Calcule un squelette (straight skeleton) à partir d'une géométrie
— CG_Tesselate - Effectue la tessellation de la surface d'un polygone ou d'une surface polyédrique et renvoie un TIN ou une collection de TINS
— CG_Visibility - Calculer un polygone de visibilité à partir d'un point ou d'un segment dans une géométrie polygonale
— CG_Volume - Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
— GeometryType - Renvoie le type d'une géométrie sous forme de texte.
— ST_3DClosestPoint - Renvoie le point 3D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne 3D la plus courte.
— ST_3DDFullyWithin - Teste si deux géométries 3D sont entièrement comprises dans une distance 3D donnée
— ST_3DDWithin - Teste si deux géométries 3D se trouvent à une distance 3D donnée
— ST_3DDistance - Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DExtent - Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.
— ST_3DIntersects - Teste si deux géométries se croisent dans l'espace en 3D - uniquement pour les points, les lignes, les polygones, les surfaces polyédriques (aire)
— ST_3DLongestLine - Renvoie la ligne 3D la plus longue entre deux géométries
— ST_3DMaxDistance - Renvoie la distance maximale cartésienne 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DShortestLine - Renvoie la ligne 3D la plus courte entre deux géométries
— ST_Affine - Appliquer une transformation affine 3D à une géométrie.
— ST_Area - Renvoie l'aire d'une géométrie polygonale.
— ST_AsBinary - Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsEWKB - Renvoie la représentation Extended Well-Known Binary (EWKB) de la géométrie avec les métadonnées SRID.
— ST_AsEWKT - Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.
— ST_AsGML - Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsX3D - Renvoie une géométrie au format X3D xml node element : ISO-IEC-19776-1.2-X3DEncodings-XML
— ST_CoordDim - Renvoie la dimension des coordonnées d'une géométrie.
— ST_Dimension - Renvoie la dimension topologique d'une géométrie.
— ST_Dump - Renvoie un ensemble de lignes geometry_dump pour les composants d'une géométrie.
— ST_DumpPoints - Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_Expand - Renvoie une boîte de délimitation développée à partir d'une autre boîte de délimitation ou d'une géométrie.
— ST_Extent - Fonction agrégée qui renvoie la boîte de délimitation des géométries.
— ST_FlipCoordinates - Renvoie une version d'une géométrie dont les axes X et Y sont inversés.
— ST_Force2D - Forcer les géométries à passer en "mode bidimensionnel".
— ST_ForceRHR - Force l'orientation des sommets d'un polygone à suivre la règle de la main droite.
— ST_ForceSFS - Force geometries to use SFS 1.1 or 1.2 geometry types.
— ST_Force3D - Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.
— ST_Force3DZ - Forcer les géométries en mode XYZ.
— ST_ForceCollection - Convertir la géométrie en une GEOMETRYCOLLECTION.

— ST_GeomFromEWKB - Retourne un objet ST_Geometry à partir de sa représentation binaire étendue (Extended Well-Known Binary representation, EWKB).
— ST_GeomFromEWKT - Retourne un objet ST_Geometry à partir de sa représentation textuelle étendue (Extended Well-Known Text representation, EWKT).
— ST_GeomFromGML - Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry
— ST_GeometryN - Renvoie un élément d'une collection de géométries.
— ST_GeometryType - Renvoie le type SQL-MM d'une géométrie sous forme de texte.
— = - Renvoie TRUE si les coordonnées et l'ordre des coordonnées de la géométrie/géographie A sont les mêmes que les coordonnées et l'ordre des coordonnées de la géométrie/géographie B.
— &<| - Renvoie TRUE si la boîte englobante de A chevauche ou est inférieure à celle de B.
— ~= - Renvoie TRUE si la boîte de délimitation de A est la même que celle de B.
— ST_IsClosed - Teste si les points de départ et d'arrivée d'une LineString coïncident. Pour une PolyhedralSurface, teste si elle est fermée (volumétrique).
— ST_MemSize - Renvoie la quantité d'espace mémoire que prend une géométrie.
— ST_NPoints - Retourne le nombre de points (vertex) d'un objet géométrique.
— ST_NumGeometries - Renvoie le nombre d'éléments dans une collection de géométrie.
— ST_NumPatches - Returns the number of faces in a PolyhedralSurface or TIN.
— ST_PatchN - Returns the Nth face of a PolyhedralSurface or TIN.
— ST_RemoveRepeatedPoints - Renvoie une version d'une géométrie dont les points en double ont été supprimés.
— ST_Reverse - Retourne la géométrie avec l'ordre des sommets inversé.
— ST_Rotate - Fait pivoter une géométrie autour d'un point d'origine.
— ST_RotateX - Fait pivoter une géométrie autour de l'axe X.
— ST_RotateY - Fait pivoter une géométrie autour de l'axe Y.
— ST_RotateZ - Fait pivoter une géométrie autour de l'axe Z.
— ST_Scale - Met à l'échelle une géométrie en fonction de facteurs donnés.
— ST_ShiftLongitude - Décale les coordonnées de longitude d'une géométrie entre -180..180 et 0..360.
— ST_Summary - Renvoie un résumé textuel du contenu d'une géométrie.
— ST_SwapOrdinates - Renvoie une version de la géométrie donnée avec les valeurs d'ordonnées permutées.
— ST_Transform - Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent.
— ~(box2df,box2df) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient une autre boîte de délimitation de précision flottante 2D (BOX2DF).
— ~(box2df,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient la boîte de délimitation 2D d'une géométrie.
— ~(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D d'une géométrie contient une boîte de délimitation de précision flottante 2D (GIDX).
— && - Renvoi VRAI si la boite englobante 2D de A intersecte la boite englobante 2D de B.
— &&& - Renvoie TRUE si la boîte de délimitation n-D de A intersecte la boîte de délimitation n-D de B.
— @(box2df,box2df) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans une autre boîte de délimitation de précision flottante 2D.
— @(box2df,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans la boîte de délimitation 2D d'une géométrie.
— @(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D d'une géométrie est contenue dans une boîte de délimitation 2D à précision flottante (BOX2DF).
— &&(box2df,box2df) - Renvoie TRUE si deux boîtes de délimitation 2D à précision flottante (BOX2DF) se croisent.
— &&(box2df,geometry) - Renvoie TRUE si une boîte de délimitation 2D de précision flottante (BOX2DF) intersecte la boîte de délimitation 2D (mise en cache) d'une géométrie.
— &&(geometry,box2df) - Renvoie TRUE si la boîte de délimitation 2D (en cache) d'une géométrie intersecte une boîte de délimitation 2D de précision flottante (BOX2DF).
— &&&(geometry,gidx) - Renvoie TRUE si la boîte de délimitation n-D (en cache) d'une géométrie intersecte une boîte de délimitation de précision flottante n-D (GIDX).

— &&&(gidx,geometry) - Renvoie TRUE si une boîte de délimitation de précision flottante n-D (GIDX) intersecte la boîte de délimitation n-D (mise en cache) d'une géométrie.
— &&&(gidx,gidx) - Renvoie TRUE si deux boîtes de délimitation (GIDX) de précision flottante n-D se croisent.

13.11 Matrice d'aide aux fonctions de PostGIS

Vous trouverez ci-dessous une liste alphabétique des fonctions spatiales spécifiques de PostGIS et des types de données spatiales qu'elles utilisent ou de la conformité OGC/SQL à laquelle elles tentent de se conformer.

— Un ✓ signifie que la fonction fonctionne nativement avec le type ou le sous-type.
— Un signifie qu'il fonctionne mais avec une transformation intégrée utilisant la géométrie, la transformation vers une référence spatiale "meilleur srid", puis la transformation à nouveau. Les résultats peuvent ne pas être conformes aux attentes pour les grandes zones ou les zones aux pôles et peuvent accumuler des erreurs en virgule flottante.
— Un signifie que la fonction fonctionne avec le type en raison d'un transfert automatique vers un autre type, tel que box3d, plutôt que d'une prise en charge directe du type.
— Un signifie que la fonction n'est disponible que si PostGIS a été compilé avec le support SFCGAL.
— geom - Support géométrique 2D de base (x,y).
— geog - Support géographique de base en 2D (x,y).
— 2.5D - géométries de base en 2D dans un espace 3 D/4D (avec coordonnée Z ou M).
— PS - Surfaces polyédriques
— T - Triangles et surfaces irrégulières triangulées (TIN)

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_Collect	✓		✓	✓			
ST_LineFromMulti	✓ t		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_Square	✓						
ST_Letters	✓						
ST_MakeNurbsCur	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiago	✓		✓				
ST_CoordDim	✓		✓	✓	✓	✓	✓
ST_Dimension	✓				✓	✓	✓
ST_Dump	✓		✓	✓		✓	✓
ST_DumpPoints	✓		✓	✓		✓	✓
ST_DumpSegments	✓		✓				✓
ST_DumpRings	✓		✓				
ST_EndPoint	✓		✓	✓	✓		
ST_Envelope	✓				✓		
ST_ExteriorRing	✓		✓		✓		
ST_GeometryN	✓		✓	✓	✓	✓	✓
ST_GeometryType	✓		✓		✓	✓	
ST_HasArc	✓		✓	✓			
ST_InteriorRingN	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometrie	✓		✓		✓	✓	✓

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_NumInteriorRin	✓				✓		
ST_NumInteriorRin	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	✓
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_StartM	✓			✓			
ST_EndM	✓			✓			
ST_SetStartM	✓			✓			
ST_SetEndM	✓			✓			
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_ZMFlag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_ControlPoints	✓						
ST_Degree	✓						
ST_Weights	✓						
ST_Knots	✓						
ST_NumControlPoi	✓						
ST_NurbsCurveIsR	✓ ıal						
ST_IsNurbsCurve	✓						
ST_Evaluate	✓						
ST_NurbsToLineSt	✓						
ST_AddPoint	✓		✓				
ST_CollectionExtra	✓						
ST_CollectionHom	✓ ıize						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonC	✓		✓				
ST_ForcePolygonC	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordi	✓ s						
ST_RemovePoint	✓		✓				
ST_RemoveRepeate	✓ ints		✓			✓	
ST_RemoveIrreleva	✓ intsForView						
ST_RemoveSmallPa	✓						
ST_Reverse	✓		✓			✓	
ST_Segmentize	✓	✓					
ST_SetPoint	✓		✓				
ST_ShiftLongitude	✓		✓			✓	✓
ST_WrapX	✓		✓				
ST_SnapToGrid	✓		✓				
ST_Snap	✓						
ST_SwapOrdinates	✓		✓	✓		✓	✓
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_MakeValid	✓		✓				
ST_InverseTransfor	✓ peline						

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipel	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							
postgis_srs_search	✓						
ST_BdPolyFromTe	✓						
ST_BdMPolyFromT	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFrom	✓				✓		
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromMA	✓ 1						
ST_GeometryFrom	✓				✓		
ST_GeomFromText	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromTex	✓				✓		
ST_MPointFromTe	✓				✓		
ST_MPolyFromTex	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromTe	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWI	✓		✓	✓		✓	✓
ST_GeomFromWK	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LinestringFrom	✓ B				✓		
ST_PointFromWKB	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeo	h						
ST_GeomFromGeo	✓ ı						
ST_GeomFromGM	✓		✓			✓	✓
ST_GeomFromGeo	✓ N		✓				

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_GeomFromKML	✓		✓				
ST_GeomFromTWK	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓						
ST_AsFlatGeobuf	☒						
ST_AsGeobuf	☒						
ST_AsGeoJSON	✓	✓	✓				
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVTGeom	✓						
ST_AsMVT	☒						
ST_AsSVG	✓	✓		✓			
ST_AsTWKB	✓						
ST_AsX3D	✓		✓			✓	✓
ST_GeoHash	✓			✓			
&&	✓	✓		✓		✓	
&&(geometry,box2df)	✓			✓		✓	
&&(box2df,geometry)	✓			✓		✓	
&&(box2df,box2df)	☒			✓		✓	
&&&	✓		✓	✓		✓	✓
&&&(geometry,gidx)	✓		✓	✓		✓	✓
&&&(gidx,geometry)	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
&<|	✓			✓		✓	
&>	✓						
<<	✓						
<<|	✓						
=	✓	✓		✓		✓	
>>	✓						
@	✓						
@(geometry,box2df	✓			✓		✓	
@(box2df,geometry	✓			✓		✓	
@(box2df,box2df)	✓			✓		✓	
|&>	✓						
|>>	✓						
~	✓						
~(geometry,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(box2df,box2df)	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
|=|	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProper	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDi ion	✓						
ST_OrderingEquals	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_RelateMatch							
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithi	✓		✓			✓	
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_PointInsideCirc	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
ST_DistanceSphere	✓						
ST_DistanceSphero	✓						
ST_FrechetDistance	✓						
ST_HausdorffDistar	✓						
ST_Length	✓	✓			✓		
ST_Length2D	✓						
ST_3DLength	✓		✓		✓		
ST_LengthSpheroid	✓		✓				
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClear	✓						
ST_MinimumClear	✓ Line						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓		✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						
ST_Subdivide	✓						
ST_SymDifference	✓		✓		✓		
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Buffer	✓				✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmooth	✓		✓				
ST_CatmullRomSm	✓	ning	✓				
ST_ConcaveHull	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTriang	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoints	✓						
ST_GeometricMedi	✓		✓				
ST_LineMerge	✓						
ST_MaximumInscri	✓	Circle					
ST_LargestEmptyC	✓						
ST_MinimumBoun	✓	Circle					
ST_MinimumBoun	✓	Radius					
ST_OrientedEnvelo	✓						
ST_OffsetCurve	✓						
ST_PointOnSurface	✓		✓		✓		
ST_Polygonize	✓						
ST_ReducePrecisio	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPreserv	✓	pology					

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_SimplifyPolygo	✓ ll						
ST_SimplifyVW	✓						
ST_SetEffectiveAre	✓						
ST_TriangulatePoly	✓						
ST_VoronoiLines	✓						
ST_VoronoiPolygor	✓						
ST_CoverageInvalic	✓ es						
ST_CoverageClean	✓						
ST_CoverageSimpli	✓						
ST_CoverageEdges	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCA	✓			✓			
ST_ClusterIntersect	✓						
ST_ClusterIntersect	✓ Vin						
ST_ClusterRelateW	✓						
ST_MinimumSpanr	✓ Tree						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinW	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExten	✓			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_3DMakeBox	✓						
ST_XMax	☑		✓	✓			
ST_XMin	☑		✓	✓			
ST_YMax	☑		✓	✓			
ST_YMin	☑		✓	✓			
ST_ZMax	☑		✓	✓			
ST_ZMin	☑		✓	✓			
ST_XSize	☑		✓	✓			
ST_YSize	☑		✓	✓			
ST_ZSize	☑		✓	✓			
ST_MMin	✓		✓	✓			
ST_MMax	✓		✓	✓			
ST_MSize	✓		✓	✓			
ST_LineInterpolatel	✓ t	✓	✓				
ST_3DLineInterpol	✓ oint		✓				
ST_LineInterpolatel	✓ ts	✓	✓				
ST_LineLocatePoin	✓	✓					
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenl	✓ ations		✓				
ST_InterpolatePoint	✓		✓				
ST_3DInterpolatePo	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajecto	✓		✓				
ST_ClosestPointOf	✓ oach		✓				
ST_DistanceCPA	✓		✓				
ST_CPAWithin	✓		✓				
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
postgis.gdal_cpl_debug							
PostGIS_AddBBox	✓			✓			
PostGIS_DropBBox	✓			✓			
PostGIS_HasBBox	✓			✓			

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
postgis_sfcgal_version							
postgis_sfcgal_full_version							
CG_ForceLHR							
CG_IsPlanar							
CG_IsSolid							
CG_MakeSolid							
CG_Orientation							
CG_Area							
CG_3DArea							
CG_Volume							
CG_Intersection							
CG_Intersects							
CG_3DIntersects							
CG_Difference							
CG_3DDifference							
CG_Distance							
CG_3DDistance							
CG_3DConvexHul							
CG_3DIntersectior							
CG_Union							
CG_3DUnion							
CG_AlphaShape							
CG_ApproxConve	 ion						
CG_ApproximateM	 Axis						
CG_GenerateFlatR							

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
CG_GenerateHipp	 f						
CG_GenerateGabl							
CG_GenerateSkilli	 of						
CG_GenerateRoof							
CG_PolygonRepai							
CG_ConstrainedD	 yTriangles						
CG_Extrude							
CG_ExtrudeStraigl	 eton						
CG_GreeneApprox	 exPartition						
CG_MinkowskiSu							
CG_OptimalAlpha							
CG_OptimalConve	 tion						
CG_StraightSkelet							
CG_Tesselate							
CG_Triangulate							
CG_Visibility							
CG_YMonotonePa							
CG_StraightSkelet	 ition						
CG_3DBuffer							
CG_Rotate							
CG_2DRotate							
CG_3DRotate							
CG_RotateX							
CG_RotateY							

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
CG_RotateZ							
CG_Scale							
CG_3DScale							
CG_3DScaleArour ter							
CG_Translate							
CG_3DTranslate							
CG_Simplify							
CG_3DAlphaWrap							
CG_NurbsCurveIn ate							
CG_NurbsCurveA mate							
CG_NurbsCurveD ve							
getfaceedges_returntype							
TopoGeometry							
validatetopology_returntype							
TopoElement							
TopoElementArray							
AddTopoGeometryColumn							
RenameTopoGeometryColumn							
DropTopology							
RenameTopology							
DropTopoGeometryColumn							
FixCorruptTopoGeometryColumn							
Populate_Topology_Layer							
TopologySummary							
ValidateTopology	✓						
ValidateTopologyRelation							
ValidateTopologyPr ion	✓						
MakeTopologyPreci	✓						
FindTopology	✓						
FindLayer	✓						
TotalTopologySize							
UpgradeTopology							
CreateTopology							
CopyTopology							
ST_InitTopoGeo					✓		
ST_CreateTopoGeo	✓				✓		

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
TopoGeo_AddPoint	✓						
TopoGeo_AddLineS	✓ g						
TopoGeo_AddPolyg	✓						
TopoGeo_LoadGeo	✓ y						
ST_AddIsoNode	✓				✓		
ST_AddIsoEdge	✓				✓		
ST_AddEdgeNewFa	✓				✓		
ST_AddEdgeModFa	✓				✓		
ST_RemEdgeNewFace					✓		
ST_RemEdgeModFace					✓		
ST_ChangeEdgeGe	✓				✓		
ST_ModEdgeSplit	✓				✓		
ST_ModEdgeHeal					✓		
ST_NewEdgeHeal					✓		
ST_MoveIsoNode	✓				✓		
ST_NewEdgesSplit	✓				✓		
ST_RemoveIsoNode					✓		
ST_RemoveIsoEdge					✓		
GetEdgeByPoint	✓						
GetFaceByPoint	✓						
GetFaceContainingI	✓ t						
GetNodeByPoint	✓						
GetTopologyID							
GetTopologySRID							
GetTopologyName							
ST_GetFaceEdges					✓		
ST_GetFaceGeomet	✓				✓		
GetRingEdges							
GetNodeEdges							
FindVertexSegmentPairsBelowDistance							
Polygonize							
AddNode	✓						
AddEdge	✓						
AddFace	✓						
ST_Simplify	✓						
RemoveUnusedPrim	✓ es						
CreateTopoGeom	✓						

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
toTopoGeom	✓						
TopoElementArray_Agg							
TopoElement	✓						
clearTopoGeom	✓						
TopoGeom_addEle	✓						
TopoGeom_remEle	✓ t						
TopoGeom_addTop	✓ om						
toTopoGeom							
GetTopoGeomElementArray							
GetTopoGeomElements							
ST_SRID	✓						
AsGML	✓						
AsTopoJSON	✓						
Equals	✓		✓				
Intersects	✓		✓				
geomval							
addbandarg							
rastbandarg							
raster							
reclassarg							
summarystats							
unionarg							
AddRasterConstraints							
DropRasterConstraints							
AddOverviewConstraints							
DropOverviewConstraints							
PostGIS_GDAL_Version							
PostGIS_Raster_Lib_Build_Date							
PostGIS_Raster_Lib_Version							
ST_GDALDrivers							
UpdateRasterSRID							
ST_CreateOverview							
ST_AddBand							
ST_AsRaster	✓						
ST_AsRasterAgg	✓						
ST_Band							
ST_MakeEmptyCoverage							
ST_MakeEmptyRaster							
ST_Tile							
ST_Retile	✓						
ST_FromGDALRaster							
ST_GeoReference							
ST_Height							
ST_IsEmpty							
ST_MemSize							
ST_MetaData							
ST_NumBands							

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_PixelHeight							
ST_PixelWidth							
ST_ScaleX							
ST_ScaleY							
ST_RasterToWorldCoord							
ST_RasterToWorldCoordX							
ST_RasterToWorldCoordY							
ST_Rotation							
ST_SkewX							
ST_SkewY							
ST_SRID							
ST_Summary							
ST_UpperLeftX							
ST_UpperLeftY							
ST_Width							
ST_WorldToRaster(✓)d							
ST_WorldToRaster(✓)dX							
ST_WorldToRaster(✓)dY							
ST_BandMetaData							
ST_BandNoDataValue							
ST_BandIsNoData							
ST_BandPath							
ST_BandFileSize							
ST_BandFileTimestamp							
ST_BandPixelType							
ST_MinPossibleValue							
ST_HasNoBand							
ST_PixelAsPolygon(✓)							
ST_PixelAsPolygons							
ST_PixelAsPoint ✓							
ST_PixelAsPoints							
ST_PixelAsCentroid(✓)							
ST_PixelAsCentroids							
ST_Value ✓							
ST_NearestValue ✓							
ST_SetZ ✓							
ST_SetM ✓							
ST_Neighborhood ✓							
ST_SetValue ✓							
ST_SetValues							
ST_DumpValues							
ST_PixelOfValue							
ST_SetGeoReference							
ST_SetRotation							
ST_SetScale							
ST_SetSkew							
ST_SetSRID							
ST_SetUpperLeft							

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_Resample							
ST_Rescale							
ST_Reskew							
ST_SnapToGrid							
ST_Resize							
ST_Transform							
ST_SetBandNoDataValue							
ST_SetBandIsNoData							
ST_SetBandPath							
ST_SetBandIndex							
ST_Count							
ST_CountAgg							
ST_Histogram							
ST_Quantile							
ST_SummaryStats							
ST_SummaryStatsAgg							
ST_ValueCount							
ST_RastFromWKB							
ST_RastFromHexWKB							
ST_AsBinary/ST_AsWKB							
ST_AsHexWKB							
ST_AsGDALRaster							
ST_AsJPEG							
ST_AsPNG							
ST_AsTIFF							
ST_Clip	✓						
ST_ColorMap							
ST_Grayscale							
ST_Intersection	✓						
ST_MapAlgebra (callback function version)							
ST_MapAlgebra (expression version)							
ST_MapAlgebraExpr							
ST_MapAlgebraExpr							
ST_MapAlgebraFct							
ST_MapAlgebraFct							
ST_MapAlgebraFctNgb							
ST_Reclass							
ST_ReclassExact							
ST_Union							
ST_Distinct4ma							
ST_InvDistWeight4ma							
ST_Max4ma							
ST_Mean4ma							
ST_Min4ma							
ST_MinDist4ma							
ST_Range4ma							
ST_StdDev4ma							
ST_Sum4ma							
ST_Aspect							

Fonction	geom	geog	2.5D	Courbes	SQL MM	PS	T
ST_HillShade							
ST_Roughness							
ST_Slope							
ST_TPI							
ST_TRI							
ST_InterpolateRaste	✓						
ST_Contour							
Box3D	☑						
ST_ConvexHull	✓						
ST_DumpAsPolygons							
ST_Envelope	✓						
ST_MinConvexHull	✓						
ST_Polygon	✓						
ST_IntersectionFrac	✓ s						
&&	✓						
&<							
&>							
=							
@	✓						
~=							
~	✓						
ST_Contains							
ST_ContainsProperly							
ST_Covers							
ST_CoveredBy							
ST_Disjoint							
ST_Intersects	✓						
ST_Overlaps							
ST_Touches							
ST_SameAlignment							
ST_NotSameAlignmentReason							
ST_Within							
ST_DWithin							
ST_DFullyWithin							

13.12 Fonctions PostGIS nouvelles, améliorées ou modifiées

13.12.1 Fonctions PostGIS nouvelles ou améliorées en 3.7

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.7

— CG_ApproximateMedialAxis - Availability: 3.7.0 - projected parameter. Requires SFCGAL >= 2.3.0 for projected result; falls back to non-projected with a notice on older versions. Calculer l'axe médian approximatif d'une géométrie aréolaire.

— CG_GenerateFlatRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a flat (box) roof from a footprint polygon.

— CG_GenerateGableRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a gable roof from a footprint polygon.

— CG_GenerateHippedRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a hipped roof from a footprint polygon.

— CG_GenerateRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a roof of the requested type from a footprint polygon.
— CG_GenerateSkillionRoof - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Generate a skillion (single-slope) roof from a footprint polygon.
— CG_NurbsCurveApproximate - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an approximating NURBS curve fitting data points within a tolerance
— CG_NurbsCurveDerivative - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Computes the derivative (tangent, curvature) of a NURBS curve at a given parameter
— CG_NurbsCurveInterpolate - Availability: 3.7.0 - requires SFCGAL >= 2.3.0. Creates an interpolating NURBS curve passing through all given data points
— CG_PolygonRepair - Availability: 3.7.0 - requires SFCGAL >= 2.3.0 and CGAL >= 6.0. Repair an invalid polygon or multipolygon.
— FindVertexSegmentPairsBelowDistance - Availability: 3.7 Find pairs of topology vertex/segment that are closer than tolerated distance
— ST_3DInterpolatePoint - Availability: 3.7.0 Returns the interpolated measure of a geometry closest to a point in 3D.
— ST_CatmullRomSmoothing - Availability: 3.7.0 Returns a smoothed version of a geometry, using the Catmull-Rom spline algorithm
— ST_ClusterRelateWin - Availability: 3.7.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets using the relate pattern to determine whether the geometries are connected.
— ST_ControlPoints - Availability: 3.7.0 Returns the control points of a NURBS curve as a MULTIPOINT geometry.
— ST_CoverageEdges - Availability: 3.7.0 Computes the unique edges of a polygonal coverage.
— ST_Degree - Availability: 3.7.0 Returns the polynomial degree of a NURBS curve.
— ST_Evaluate - Availability: 3.7.0 Evaluates a NURBS curve at a specific parameter value and returns the resulting point.
— ST_IsNurbsCurve - Availability: 3.7.0 Checks if a geometry is a NURBS curve.
— ST_Knots - Availability: 3.7.0 Returns the knot vector of a NURBS curve.
— ST_MSize - Availability: 3.7.0 Returns the M size of a geometry.
— ST_MakeLine - Availability: 3.7.0 - Support for MultiLineString input elements was introduced Crée une LineString à partir de géométries Point, MultiPoint ou LineString.
— ST_MakeNurbsCurve - Availability: 3.7.0 Creates a NURBS (Non-Uniform Rational B-Spline) curve from control points, optional weights, and optional knot vector.
— ST_MinimumSpanningTree - Availability: 3.7.0 Window function that returns a tree id for each input geometry, clustering input geometries into connected trees using the Minimum Spanning Tree algorithm.
— ST_NumControlPoints - Availability: 3.7.0 Returns the number of control points in a NURBS curve.
— ST_NurbsCurveIsRational - Availability: 3.7.0 Checks if a NURBS curve is rational (has weights).
— ST_NurbsToLineString - Availability: 3.7.0 Converts a NURBS curve to a LINESTRING by uniform sampling.
— ST_Weights - Availability: 3.7.0 Returns the weight array of a rational NURBS curve.
— ST_XSize - Availability: 3.7.0 Returns the X size of a 2D or 3D bounding box or a geometry.
— ST_YSize - Availability: 3.7.0 Returns the Y size of a 2D or 3D bounding box or a geometry.
— ST_ZSize - Availability: 3.7.0 Returns the Z size of a 2D or 3D bounding box or a geometry.

Fonctions améliorées dans PostGIS 3.7

— ST_AsX3D - Enhanced: 3.7.0: Support for curve geometries. NURBSCURVE is emitted as X3D NurbsCurve; other curve types are linearized before output. Renvoie une géométrie au format X3D xml node element : ISO-IEC-19776-1.2-X3DEncodings-XML
— ST_GMLToSQL - Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced. Returns a geometry from GML. Alias for ST_GeomFromGML.
— ST_GeomFromGML - Enhanced: 3.7.0 support for GML ArcString, curved Ring, and CurvePolygon geometries was introduced. Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry
— ST_LongestLine - Enhanced: 3.7.0 - support for a single geometry input. Renvoie la ligne 2D la plus longue entre deux géométries.
— ST_MakePolygon - Enhanced: 3.7.0 - Support for curved input rings was introduced. Creates a Polygon or CurvePolygon from a shell and optional list of holes.

— ST_MaxDistance - Enhanced: 3.7.0 - support for a single geometry input. Renvoie la plus grande distance 2D entre deux géométries en unités projetées.
— ST_Value - Enhanced: 3.7.0 resample accepts nearest-neighbor boundary options "nearest-UL", "nearest-UR", "nearest-LL", and "nearest-LR". Retourne la valeur d'une bande raster spécifiée au pixel donné par columnx, rowy, ou à un point géométrique spécifié. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié. Si exclude_nodata_value vaut false, tous les pixels y compris ceux ayant la valeur nodata sont considérés comme intersectés et leur valeur sera retournée. Si exclude_nodata_value n'est pas spécifié, la valeur est lue depuis les méta-données du raster.
— TopoGeo_AddLineString - Enhanced: 3.7.0 added support for limiting the number of new edges created in the topology. Ajoute une ligne à une topologie existante en utilisant une tolérance et en divisant éventuellement les arêtes/faces existantes.

Fonctions modifiées dans PostGIS 3.7

— ST_AsGeoJSON - Changed: 3.7.0 added warning about duplicate keys Renvoyer une géométrie ou un élément au format GeoJSON.
— toTopoGeom - Changed: 3.7.0 changes meaning of tolerance 0 to really mean 0 and -1 (new default) to request using the topology precision. Convertit une géométrie simple en une géométrie topographique.

13.12.2 Fonctions PostGIS nouvelles ou améliorées en 3.6

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.6

— CG_2DRotate - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Fait pivoter une géométrie d'un angle donné autour d'un point spécifié en 2D.
— CG_3DAlphaWrapping - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Computes a 3D Alpha-wrapping strictly enclosing a geometry.
— CG_3DBuffer - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Calcule un tampon 3D autour d'une géométrie.
— CG_3DRotate - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Fait pivoter une géométrie dans l'espace 3D autour d'un vecteur d'axe.
— CG_3DScale - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Met à l'échelle une géométrie en fonction de facteurs distincts le long des axes X, Y et Z.
— CG_3DScaleAroundCenter - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Met à l'échelle une géométrie dans l'espace 3D autour d'un point central spécifié.
— CG_3DTranslate - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Déplace une géométrie en fonction de décalages donnés dans l'espace 3D.
— CG_Rotate - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Fait pivoter une géométrie d'un angle donné autour de l'origine (0,0).
— CG_RotateX - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Fait pivoter une géométrie autour de l'axe X d'un angle donné.
— CG_RotateY - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Fait pivoter une géométrie autour de l'axe Y d'un angle donné.
— CG_RotateZ - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Fait pivoter une géométrie autour de l'axe Z d'un angle donné.
— CG_Scale - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Met à l'échelle une géométrie uniformément dans toutes les dimensions selon un facteur donné.
— CG_Simplify - Availability: 3.6.0 - requires SFCGAL >= 2.1.0 Reduces the complexity of a geometry while preserving essential features and Z/M values.
— CG_StraightSkeletonPartition - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0. Calcule la partition du squelette droit d'un polygone.
— CG_Translate - Disponibilité : 3.6.0 - nécessite SFCGAL >= 2.0.0 Déplace une géométrie en fonction de décalages donnés dans l'espace 2D.
— FixCorruptTopoGeometryColumn - Availability: 3.6.1 Fixes topogeometry corruption caused by upgrade to postgis_topology 3.6.0 and higher
— MakeTopologyPrecise - Disponibilité : 3.6.0 Snap topology vertices to precision grid.
— ST_AsRasterAgg - Disponibilité : 3.6.0 Aggregate. Rend les géométries PostGIS dans un nouveau raster.

— ST_CoverageClean - Availability: 3.6.0 - requires GEOS >= 3.14.0 Computes a clean (edge matched, non-overlapping, gap-cleared) polygonal coverage, given a non-clean input.
— ST_IntersectionFractions - Availability: 3.6.0 Requires GEOS 3.14 or higher. Calculates the fraction of each raster cell that is covered by a given geometry.
— ST_ReclassExact - Disponibilité : 3.6.0 Creates a new raster composed of bands reclassified from original, using a 1:1 mapping from values in the original band to new values in the destination band.
— TotalTopologySize - Disponibilité : 3.6.0 Total disk space used by the specified topology, including all indexes and TOAST data.
— UpgradeTopology - Disponibilité : 3.6.0 Upgrades the specified topology to support large ids (int8) for topology and primitive ids.
— ValidateTopologyPrecision - Disponibilité : 3.6.0 Returns non-precise vertices in the topology.
— postgis.gdal_cpl_debug - Disponibilité : 3.6.0 A boolean configuration to turn logging of GDAL debug messages on and off.

Fonctions améliorées dans PostGIS 3.6

— CreateTopology - Enhanced: 3.6.0 added topoid and usesLargeIDs parameters. Crée un nouveau schéma topologique et l'enregistre dans la table topology.topology.

Fonctions modifiées dans PostGIS 3.6

— ST_GeometryN - Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and are not decomposed into patches; use ST_PatchN to access faces. Renvoie un élément d'une collection de géométries.
— ST_NumGeometries - Changed: 3.6.0 TIN and PolyhedralSurface are treated as unitary geometries and return 1; use ST_NumPatches to count faces. Renvoie le nombre d'éléments dans une collection de géométrie.

13.12.3 Fonctions PostGIS nouvelles ou améliorées en 3.5

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.5

— CG_3DArea - Disponibilité : 3.5.0 Calcule la surface des géométries de surface 3D. Retourne 0 pour les solides.
— CG_3DConvexHull - Disponibilité : 3.5.0 Calcule l'enveloppe convexe 3D d'une géométrie.
— CG_3DDifference - Disponibilité : 3.5.0 Effectuer une différence 3D
— CG_3DDistance - Disponibilité : 3.5.0 Calcule la distance 3D minimale entre deux géométries
— CG_3DIntersection - Disponibilité : 3.5.0 Réaliser une intersection 3D
— CG_3DIntersects - Disponibilité : 3.5.0 Teste si deux géométries 3D intersectent
— CG_3DUnion - Disponibilité : 3.5.0 Réaliser l'union 3D en utilisant postgis_sfcgal.
— CG_AlphaShape - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.4.1. Calcul d'une forme Alpha entourant une géométrie
— CG_ApproxConvexPartition - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0. Calcul de la partition convexe approximative de la géométrie du polygone
— CG_ApproximateMedialAxis - Disponibilité : 3.5.0 Calculer l'axe médian approximatif d'une géométrie aréolaire.
— CG_Area - Disponibilité : 3.5.0 Calcule la surface d'une géométrie
— CG_Difference - Disponibilité : 3.5.0 Calcul la différence géométrique entre deux géométries
— CG_Distance - Disponibilité : 3.5.0 Calcule la distance minimale entre deux géométries
— CG_Extrude - Disponibilité : 3.5.0 Extruder une surface vers un volume
— CG_ExtrudeStraightSkeleton - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0. Extrusion de squelette droit
— CG_ForceLHR - Disponibilité : 3.5.0 Force l'orientation LHR d'un objet
— CG_GreeneApproxConvexPartition - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0. Calcul de la partition convexe approximative de la géométrie du polygone
— CG_Intersection - Disponibilité : 3.5.0 Calcul l'intersection de deux géométries
— CG_Intersects - Disponibilité : 3.5.0 Teste si deux géométries se croisent (elles ont au moins un point en commun)
— CG_IsPlanar - Disponibilité : 3.5.0 Vérifie si une surface est planaire ou non
— CG_IsSolid - Disponibilité : 3.5.0 Teste si la géométrie est un solide. Aucun contrôle de validité n'est effectué.
— CG_MakeSolid - Disponibilité : 3.5.0 Transformer la géométrie dans un solide. Aucune vérification n'est effectuée. Pour obtenir un solide valide, la géométrie d'entrée doit être une surface polyédrique fermée ou un TIN fermé.

— CG_MinkowskiSum - Disponibilité : 3.5.0 Effectue la somme de Minkowski
— CG_OptimalAlphaShape - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.4.1. Calcule une forme alpha entourant une géométrie en utilisant une valeur alpha "optimale".
— CG_OptimalConvexPartition - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0. Calcul d'une partition convexe optimale de la géométrie du polygone
— CG_Orientation - Disponibilité : 3.5.0 Détermine l'orientation d'une surface
— CG_StraightSkeleton - Disponibilité : 3.5.0 Calcule un squelette (straight skeleton) à partir d'une géométrie
— CG_Tesselate - Disponibilité : 3.5.0 Effectue la tessellation de la surface d'un polygone ou d'une surface polyédrique et renvoie un TIN ou une collection de TINS
— CG_Triangulate - Disponibilité : 3.5.0 Triangule une géométrie polygonale
— CG_Union - Disponibilité : 3.5.0 Calcule l'union de deux géométries
— CG_Visibility - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0. Calculer un polygone de visibilité à partir d'un point ou d'un segment dans une géométrie polygonale
— CG_Volume - Disponibilité : 3.5.0 Computes the volume of a 3D geometry. Closed surfaces can have non-zero volume.
— CG_YMonotonePartition - Disponibilité : 3.5.0 - nécessite SFCGAL >= 1.5.0. Calcul de la partition y-monotone de la géométrie du polygone
— ST_HasM - Disponibilité : 3.5.0 Vérifie si une géométrie a une dimension M (mesure).
— ST_HasZ - Disponibilité : 3.5.0 Vérifie si une géométrie possède une dimension Z.
— ST_RemoveIrrelevantPointsForView - Disponibilité : 3.5.0 Supprime les points qui ne sont pas pertinents pour le rendu d'une vue rectangulaire spécifique d'une géométrie.
— ST_RemoveSmallParts - Disponibilité : 3.5.0 Supprime les petites parties (anneaux de polygone ou lignes) d'une géométrie.
— TopoGeo_LoadGeometry - Disponibilité : 3.5.0 Charge une géométrie dans une topologie existante, en effectuant de l'accrochage et des divisions si nécessaire.

Fonctions améliorées dans PostGIS 3.5

— ST_Clip - Amélioration : 3.5.0 - ajout de l'argument touched. Renvoie le raster découpé par la géométrie d'entrée. Si le numéro de bande n'est pas spécifié, toutes les bandes sont traitées. Si crop n'est pas spécifié ou est TRUE, le raster de sortie est rogné. Si touched est défini sur TRUE, les pixels touchés sont inclus, sinon seulement si le centre du pixel est dans la géométrie, il est inclus.

Fonctions modifiées dans PostGIS 3.5

— ST_AsGeoJSON - Modifié : la version 3.5.0 permet de spécifier la colonne contenant l'identifiant de l'élément Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_DFullyWithin - Modifié : 3.5.0 : la logique implémentée utilise désormais un test de confinement dans un tampon, plutôt que l'algorithme ST_MaxDistance. Les résultats seront différents de ceux des versions précédentes, mais devraient être plus proches des attentes de l'utilisateur. Teste si une géométrie se trouve entièrement à une distance d'une autre géométrie

13.12.4 Fonctions PostGIS nouvelles ou améliorées en 3.4

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.4

— PostGIS_GEOS_Compiled_Version - Disponibilité : 3.4.0 Renvoie le numéro de version de la librairie GEOS avec laquelle PostGIS a été construit.
— PostGIS_PROJ_Compiled_Version - Disponibilité : 3.5.0 Renvoie le numéro de version de la bibliothèque PROJ avec laquelle PostGIS a été construit.
— RenameTopoGeometryColumn - Disponibilité : 3.4.0 Renomme une colonne topogeometry
— RenameTopology - Disponibilité : 3.4.0 Renomme une topologie
— ST_ClusterIntersectingWin - Disponibilité : 3.4.0 Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, en regroupant les géométries en entrée en ensembles connectés.
— ST_ClusterWithinWin - Disponibilité : 3.4.0 Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée, regroupement en utilisant la distance de séparation.
— ST_CoverageInvalidEdges - Disponibilité : 3.4.0 Fonction window qui trouve les endroits où les polygones ne forment pas une couverture valide.

— ST_CoverageSimplify - Disponibilité : 3.4.0 Fonction window qui simplifie les bords d'une couverture polygonale.
— ST_CoverageUnion - Disponibilité : 3.4.0 - nécessite GEOS >= 3.8.0 Calcule l'union d'un ensemble de polygones formant une couverture en supprimant les arêtes communes.
— ST_InverseTransformPipeline - Disponibilité : 3.4.0 Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent en utilisant l'inverse d'un pipeline de transformation de coordonnées défini.
— ST_LargestEmptyCircle - Disponibilité : 3.4.0. Calcule le plus grand cercle ne recouvrant pas une géométrie.
— ST_LineExtend - Disponibilité : 3.4.0 Renvoie une ligne prolongée vers l'avant et vers l'arrière selon les distances spécifiées.
— ST_TransformPipeline - Disponibilité : 3.4.0 Retourner une nouvelle géométrie avec des coordonnées transformées dans un système de référence spatial différent à l'aide d'un pipeline de transformation de coordonnées défini.
— TopoElement - Disponibilité : 3.4.0 Convertit une topogeometry en un topoelement.
— postgis_srs - Disponibilité : 3.4.0 Renvoyer une fiche de métadonnées pour l'autorité et le srid demandés.
— postgis_srs_all - Disponibilité : 3.4.0 Renvoie des enregistrements de métadonnées pour chaque système de référence spatiale dans la base de données Proj sous-jacente.
— postgis_srs_codes - Disponibilité : 3.4.0 Renvoie la liste des codes SRS associés à l'autorité donnée.
— postgis_srs_search - Disponibilité : 3.4.0 Renvoyer les enregistrements de métadonnées pour les systèmes de coordonnées projetées dont les zones d'utilisation contiennent entièrement le paramètre bounds.

Fonctions améliorées dans PostGIS 3.4

— PostGIS_Full_Version - Amélioration : 3.4.0 inclut désormais les configurations supplémentaires PROJ NETWORK_ENABLED, URL_ENDPOINT et DATABASE_PATH pour l'emplacement proj.db Donne des informations complètes sur la version de PostGIS et la configuration du packaging.
— PostGIS_PROJ_Version - Amélioration : 3.4.0 inclut désormais PROJ NETWORK_ENABLED, URL_ENDPOINT et DATABASE_PATH pour l'emplacement proj.db Renvoie le numéro de version de la librairie PROJ4.
— ST_AsSVG - Amélioration : 3.4.0 pour prendre en charge tous les types de courbes Renvoie les données de chemin SVG pour une géométrie.
— ST_ClosestPoint - Amélioré : 3.4.0 - Prise en charge de la geography. Renvoie le point 2D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne la plus courte d'une géométrie à l'autre.
— ST_LineSubstring - Amélioration : 3.4.0 - La prise en charge de la géographie a été introduite. Renvoie la partie d'une ligne située entre deux emplacements fractionnaires.
— ST_Project - Amélioration : 3.4.0 Autorise les arguments géométriques et la forme en deux points omettant l'azimut. Renvoie un point projeté à partir d'un point de départ en fonction d'une distance et d'un azimut.
— ST_Resample - Améliration : 3.4.0 ajout des options de rééchantillonnage max et min Rééchantillonne un raster, en utilisant l'algorithme spécifié, les nouvelles dimensions, un coin arbitraire de la grille et un ensemble de paramètres de géo-référencement définis ou empruntés à un autre raster.
— ST_Rescale - Améliration : 3.4.0 ajout des options de rééchantillonnage max et min Rééchantillonne un raster en ajustant juste son échelle (ou la taille des pixels). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline), Lanczos, Max ou Min. La valeur par défaut est NearestNeighbor.
— ST_ShortestLine - Amélioré : 3.4.0 - Prise en charge de la geography. Renvoie la ligne 2D la plus courte entre deux géométries

Fonctions modifiées dans PostGIS 3.4

— PostGIS_Extensions_Upgrade - Modifié : 3.4.0 pour ajouter l'argument target_version. Packages et mises à jour des extensions PostGIS (par exemple postgis_raster, postgis_topology, postgis_sfcgal) vers la version donnée ou la plus récente.

13.12.5 Fonctions PostGIS nouvelles ou améliorées en 3.3

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.3

— RemoveUnusedPrimitives - Disponibilité : 3.3.0 Supprime les primitives topologiques qui ne sont pas nécessaires pour définir les objets TopoGeometry existants.
— ST_AsMARC21 - Disponibilité : 3.3.0 Renvoie la géométrie sous forme d'enregistrement MARC21/XML avec un champ de données géographiques (034).

— ST_GeomFromMARC21 - Disponibilité : 3.3.0, nécessite libxml2 2.6+ Prend les données géographiques MARC21/XML en entrée et renvoie un objet géométrique PostGIS.
— ST_Letters - Disponibilité : 3.3.0 Renvoie les lettres d'entrée rendues sous forme de géométrie avec une position de départ par défaut à l'origine et une hauteur de texte par défaut de 100.
— ST_SimplifyPolygonHull - Disponibilité : 3.3.0. Calcule une enveloppe extérieure ou intérieure simplifiée préservant la topologie d'une géométrie polygonale.
— ST_TriangulatePolygon - Disponibilité : 3.3.0. Calcule la triangulation de Delaunay contrainte des polygones
— postgis_sfcgal_full_version - Disponibilité : 3.3.0 Retourne la version complète de SFCGAL en cours d'utilisation, y compris les versions CGAL et Boost

Fonctions améliorées dans PostGIS 3.3

— ST_ConcaveHull - Amélioré : 3.3.0, implémentation native de GEOS activée pour GEOS 3.11+ Calcule une géométrie éventuellement concave qui contient tous les sommets de la géométrie d'entrée
— ST_LineMerge - Amélioration : 3.3.0 accepte un paramètre direct. Renvoie les lignes formées par la couture d'une MultiLineString.

Fonctions modifiées dans PostGIS 3.3

— PostGIS_Extensions_Upgrade - Modifié : 3.3.0 support pour les mises à jour à partir de n'importe quelle version de PostGIS. Ne fonctionne pas sur tous les systèmes. Packages et mises à jour des extensions PostGIS (par exemple postgis_raster, postgis_topology, postgis_sfcgal) vers la version donnée ou la plus récente.

13.12.6 Fonctions PostGIS nouvelles ou améliorées en 3.2

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.2

— FindLayer - Disponibilité : 3.2.0 Renvoie un enregistrement topology.layer par différents moyens.
— FindTopology - Disponibilité : 3.2.0 Renvoie un enregistrement topologique par différents moyens.
— GetFaceContainingPoint - Disponibilité : 3.2.0 Recherche la face contenant un point.
— ST_AsFlatGeobuf - Disponibilité : 3.2.0 Renvoie une représentation FlatGeobuf d'un ensemble de lignes.
— ST_Contour - Disponibilité : 3.2.0 Génère un ensemble de courbes de niveau vectorielles depuis la bande raster spécifiée, en utilisant l'algorithme de contour GDAL.
— ST_DumpSegments - Disponibilité : 3.2.0 Renvoie un ensemble de lignes geometry_dump pour les segments d'une géométrie.
— ST_FromFlatGeobuf - Disponibilité : 3.2.0 Lit les données FlatGeobuf.
— ST_FromFlatGeobufToTable - Disponibilité : 3.2.0 Crée une table basée sur la structure des données FlatGeobuf.
— ST_InterpolateRaster - Disponibilité : 3.2.0 Interpole une surface quadrillée à partir d'un ensemble de points 3-d, en utilisant les coordonnées X et Y des points sur la grille et la coordonnée Z des points pour l'élévation des points.
— ST_SRID - Disponibilité : 3.2.0 Renvoie l'identifiant de référence spatiale d'une topogeometry.
— ST_Scroll - Disponibilité : 3.2.0 Modifier le point de départ d'une LineString fermée.
— ST_SetM - Disponibilité : 3.2.0 Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée M copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.
— ST_SetZ - Disponibilité : 3.2.0 Retourne une géométrie avec les mêmes coordonnées X/Y que la géométrie d'entrée, et avec la coordonnée Z copiée depuis les valeurs du raster selon l'algorithme d'interpolation en paramètre.
— TopoGeom_addTopoGeom - Disponibilité : 3.2 Ajoute un élément d'une TopoGeometry à la définition d'une autre TopoGeometry.
— ValidateTopologyRelation - Disponibilité : 3.2.0 Renvoie des informations sur les enregistrements de relations topologiques non valides
— postgis.gdal_vsi_options - Disponibilité : 3.2.0 Une chaîne de configuration pour définir les options utilisées lors de l'utilisation d'un raster out-db.

Fonctions améliorées dans PostGIS 3.2

— GetFaceByPoint - Amélioration : 3.2.0 mise en œuvre plus efficace et contrat plus clair, arrêt du fonctionnement avec des topologies non valides. Recherche la face intersectant un point donné.

— ST_ClusterKMeans - Amélioré : 3.2.0 Support pour max_radius Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée en utilisant l'algorithme K-means.
— ST_MakeValid - Amélioration : 3.2.0, ajout d'options d'algorithme, 'linework' et 'structure' qui nécessite GEOS >= 3.10.0. Tente de rendre valide une géométrie invalide sans perdre de sommets.
— ST_PixelAsCentroid - Amélioration : 3.2.0 Plus rapide, désormais implémenté en C. Retourne le centroïde (point géométrique) de la zone représentée par un pixel.
— ST_PixelAsCentroids - Amélioration : 3.2.0 Plus rapide, désormais implémenté en C. Retourne le centroïde (point géométrique) pour chaque pixel de la bande raster, avec sa valeur et les coordonnées raster X et Y. Le point géométrique est le centroïde de la zone représentée par un pixel.
— ST_Point - Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie. Crée un point avec des valeurs X, Y et SRID.
— ST_PointM - Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie. Crée un point avec des valeurs X, Y, M et SRID.
— ST_PointZ - Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie. Crée un point avec des valeurs X, Y, Z et SRID.
— ST_PointZM - Amélioration : 3.2.0 srid a été ajouté comme argument optionnel supplémentaire. Les anciennes installations nécessitent une combinaison avec ST_SetSRID pour marquer le srid sur la géométrie. Crée un point avec des valeurs X, Y, Z, M et SRID.
— ST_RemovePoint - Amélioration : 3.2.0 Supprime un point d'une ligne.
— ST_RemoveRepeatedPoints - Amélioration : 3.2.0 Renvoie une version d'une géométrie dont les points en double ont été supprimés.
— ST_StartPoint - Amélioré : 3.2.0 renvoie un point pour toutes les géométries. Le comportement précédent renvoyait NULL si l'entrée n'était pas une LineString. Returns the first point of a LineString, CircularLineString, or NURBSCurve.
— ST_Value - Amélioration : 3.2.0 ajout du paramètre optionnel resample. Retourne la valeur d'une bande raster spécifiée au pixel donné par columnx, rowy, ou à un point géométrique spécifié. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié. Si exclude_nodata_value vaut false, tous les pixels y compris ceux ayant la valeur nodata sont considérés comme intersectés et leur valeur sera retournée. Si exclude_nodata_value n'est pas spécifié, la valeur est lue depuis les méta-données du raster.
— TopoGeo_AddLineString - Amélioration : la version 3.2.0 a ajouté la possibilité de renvoyer un identifiant signé. Ajoute une ligne à une topologie existante en utilisant une tolérance et en divisant éventuellement les arêtes/faces existantes.

Fonctions modifiées dans PostGIS 3.2

— ST_Boundary - Modifié : 3.2.0 support pour TIN, n'utilise pas geos, ne linéarise pas les courbes Renvoie la limite d'une géométrie.
— ValidateTopology - Modifié : 3.2.0 a ajouté le paramètre optionnel bbox, effectuant des vérifications de l'étiquetage des faces et de la liaison des arêtes. Renvoie un ensemble d'objets validatetopology_returntype détaillant les problèmes liés à la topologie.

13.12.7 Fonctions PostGIS nouvelles ou améliorées en 3.1

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.1

— ST_Hexagon - Disponibilité : 3.1.0 Renvoie un seul hexagone, en utilisant la taille du bord et les coordonnées de la cellule fournies dans l'espace de la grille de l'hexagone.
— ST_HexagonGrid - Disponibilité : 3.1.0 Renvoie un ensemble d'hexagones et d'indices de cellules qui couvrent complètement les limites de l'argument géométrie.
— ST_MaximumInscribedCircle - Disponibilité : 3.1.0. Calcule le plus grand cercle contenu dans une géométrie.
— ST_ReducePrecision - Disponibilité : 3.1.0. Renvoie une géométrie valide dont les points sont arrondis en fonction de la tolérance de la grille.
— ST_Square - Disponibilité : 3.1.0 Renvoie un seul carré, en utilisant la taille du bord et la coordonnée de la cellule fournies dans l'espace de la grille du carré.

— ST_SquareGrid - Disponibilité : 3.1.0 Renvoie un ensemble de carrés de grille et d'indices de cellules qui couvrent complètement les limites de l'argument géométrie.

Fonctions améliorées dans PostGIS 3.1

— ST_AsEWKT - Amélioré : support du paramètre optionnel de précision dans la version 3.1.0. Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.
— ST_ClusterKMeans - Amélioration : 3.1.0 Prise en charge des géométries et des poids en 3D Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée en utilisant l'algorithme K-means.
— ST_Difference - Amélioration : 3.1.0 accepte un paramètre gridSize. Calcule une géométrie représentant la partie de la géométrie A qui n'intersecte pas la géométrie B.
— ST_Intersection - Amélioration : 3.1.0 accepte un paramètre gridSize Calcule une géométrie représentant la partie partagée des géométries A et B.
— ST_MakeValid - Amélioration : 3.1.0, suppression des coordonnées avec des valeurs NaN. Tente de rendre valide une géométrie invalide sans perdre de sommets.
— ST_Subdivide - Amélioration : 3.1.0 accepte un paramètre gridSize. Calcule une subdivision rectiligne d'une géométrie.
— ST_SymDifference - Amélioration : 3.1.0 accepte un paramètre gridSize. Calcule une géométrie représentant les parties des géométries A et B qui ne s'intersectent pas.
— ST_TileEnvelope - Amélioré : 3.1.0 Ajout d'un paramètre de marge. Crée un polygone rectangulaire dans Web Mercator (SRID:3857) en utilisant le système de tuiles XYZ.
— ST_UnaryUnion - Amélioration : 3.1.0 accepte un paramètre gridSize. Calcule l'union des composantes d'une seule géométrie.
— ST_Union - Amélioration : 3.1.0 accepte un paramètre gridSize. Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.

Fonctions modifiées dans PostGIS 3.1

— ST_Count - Changement : 3.1.0 - Suppression des variantes ST_Count(rastertable, rastercolumn, ...). Utiliser à la place. Renvoie le nombre de pixels dans une bande donnée d'un raster ou d'une couverture raster. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels dont la valeur est différente de la valeur nodata seront comptés.
— ST_Force3D - Modifié : 3.1.0. Ajout de la prise en charge pour pouvoir passer une valeur Z non nulle. Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.
— ST_Force3DM - Modifié : 3.1.0. Ajout de la prise en charge de pouvoir passer une valeur M non nulle. Forcer les géométries en mode XYM.
— ST_Force3DZ - Modifié : 3.1.0. Ajout de la prise en charge pour pouvoir passer une valeur Z non nulle. Forcer les géométries en mode XYZ.
— ST_Force4D - Modifié : 3.1.0. Ajout de la prise en charge de pouvoir passer des valeurs Z et M non nulles. Forcer les géométries en mode XYZM.
— ST_Histogram - Changement : 3.1.0 Suppression de la variante ST_Histogram(table_name, column_name). Retourne un ensemble d'enregistrements résumant une distribution de données raster ou de couverture raster, dans des classes distinctes. Le nombre de classes est calculé automatiquement s'il n'est pas spécifié.
— ST_Quantile - Changement : 3.1.0 Suppression de la variante ST_Quantile(table_name, column_name). Calcule les quantiles d'un raster ou d'une couverture raster, dans le contexte de l'échantillon ou de la population. Ainsi, une valeur peut être examinée pour se situer au percentile de 25%, 50% ou 75% du raster.
— ST_SummaryStats - Changement : 3.1.0 Suppression des variantes ST_SummaryStats(rastertable, rastercolumn, ...). Utiliser à la place. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.

13.12.8 Fonctions PostGIS nouvelles ou améliorées en 3.0

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 3.0

— CG_ConstrainedDelaunayTriangles - Disponibilité : 3.0.0 Renvoie une triangulation de Delaunay contrainte autour de la géométrie d'entrée donnée.

— ST_3DLineInterpolatePoint - Disponibilité : 3.0.0 Renvoie un point interpolé le long d'une ligne 3D à un emplacement fractionnaire.
— ST_TileEnvelope - Disponibilité : 3.0.0 Crée un polygone rectangulaire dans Web Mercator (SRID:3857) en utilisant le système de tuiles XYZ.

Fonctions améliorées dans PostGIS 3.0

— ST_AsMVT - Amélioration : 3.0 - ajout de la prise en charge du Feature ID. Fonction d'agrégation renvoyant une représentation MVT d'un ensemble de lignes.
— ST_Contains - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Tests si chaque point de B est situé dans A, et que leurs intérieurs ont un point commun
— ST_ContainsProperly - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Tests si chaque point de B se trouve à l'intérieur de A
— ST_CoveredBy - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Tests si chaque point de A se trouve dans B
— ST_Covers - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Tests si chaque point de B est situé dans A
— ST_Crosses - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Teste si deux géométries ont en commun certains points intérieurs, mais pas tous
— ST_CurveToLine - Amélioration : la version 3.0.0 a mis en place un nombre minimum de segments par arc linéarisé afin d'éviter une rupture topologique. Convertit une géométrie contenant des courbes en une géométrie linéaire.
— ST_Disjoint - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Teste si deux géométries n'ont pas de points communs
— ST_Equals - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Teste si deux géométries comprennent le même ensemble de points
— ST_GeneratePoints - Amélioration : 3.0.0, ajout du paramètre seed Génère un multipoint de points aléatoires contenus dans un polygone ou un multipolygone.
— ST_GeomFromGeoJSON - Amélioré : 3.0.0 La géométrie parsée prend par défaut la valeur SRID=4326 si elle n'est pas spécifiée autrement. Prend en entrée une géométrie au format geojson et renvoie un objet Postgis de type geometry
— ST_LocateBetween - Amélioration : 3.0.0 - ajout de la prise en charge du POLYGONE, du TIN et du TRIANGLE. Renvoie les parties d'une géométrie qui correspondent à un intervalle de mesure.
— ST_LocateBetweenElevations - Amélioration : 3.0.0 - ajout de la prise en charge du POLYGONE, du TIN et du TRIANGLE. Renvoie les parties d'une géométrie qui se trouvent dans un intervalle d'élévation (Z).
— ST_Overlaps - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Teste si deux géométries ont la même dimension et se croisent, mais si chacune a au moins un point qui n'est pas dans l'autre
— ST_Relate - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Teste si deux géométries ont une relation topologique correspondant à un modèle de matrice d'intersection, ou calcule leur matrice d'intersection
— ST_Segmentize - Amélioration : 3.0.0 La segmentation de géométrie produit désormais des sous-segments de longueur égale Renvoie une geometry/geography modifiée dont aucun segment ne dépasse une distance donnée.
— ST_Touches - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Teste si deux géométries ont au moins un point en commun, mais que leurs intérieurs ne se croisent pas
— ST_Within - Amélioration : 3.0.0 a permis la prise en charge de GEOMETRYCOLLECTION Tests si chaque point de A se trouve dans B, et que leurs intérieurs ont un point commun

Fonctions modifiées dans PostGIS 3.0

— PostGIS_Extensions_Upgrade - Modifié : 3.0.0 pour repackager les extensions libres et supporter postgis_raster. Packages et mises à jour des extensions PostGIS (par exemple postgis_raster, postgis_topology, postgis_sfcgal) vers la version donnée ou la plus récente.
— ST_3DDistance - Modifié : 3.0.0 - Version SFCGAL supprimée Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DIntersects - Modifié : 3.0.0 SFCGAL backend supprimé, GEOS backend supporte les TINs. Teste si deux géométries se croisent dans l'espace en 3D - uniquement pour les points, les lignes, les polygones, les surfaces polyédriques (aire)
— ST_Area - Modifié : 3.0.0 - ne dépend plus de SFCGAL. Renvoie l'aire d'une géométrie polygonale.
— ST_AsGeoJSON - Modifié : la version 3.0.0 prend en charge les enregistrements en tant que données d'entrée Renvoyer une géométrie ou un élément au format GeoJSON.

— ST_AsGeoJSON - Modifié : 3.0.0 SRID de sortie si ce n'est pas EPSG:4326. Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_AsKML - Modifié : 3.0.0 - Suppression de la signature de la variante "versioned" Renvoyer la géométrie sous forme d'élément KML.
— ST_Distance - Modifié : 3.0.0 - ne dépend plus de SFCGAL. Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_Intersection - Modifié : 3.0.0 ne dépend pas de SFCGAL. Calcule une géométrie représentant la partie partagée des géométries A et B.
— ST_Intersects - Modifié : 3.0.0 La version SFCGAL a été supprimée et la prise en charge native des TINS 2D a été ajoutée. Teste si deux géométries se croisent (elles ont au moins un point en commun)
— ST_Union - Modifié : 3.0.0 ne dépend pas de SFCGAL. Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.

13.12.9 Fonctions PostGIS nouvelles ou améliorées en 2.5

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 2.5

— PostGIS_Extensions_Upgrade - Disponibilité : 2.5.0 Packages et mises à jour des extensions PostGIS (par exemple postgis_raster, postgis_topology, postgis_sfcgal) vers la version donnée ou la plus récente.
— ST_Angle - Disponibilité : 2.5.0 Renvoie l'angle entre deux vecteurs définis par 3 ou 4 points, ou 2 lignes.
— ST_AsHexWKB - Disponibilité : 2.5.0 Retourne la représentation hexadécimale Well-Known Binary (WKB) de ce raster.
— ST_BandFileSize - Disponibilité : 2.5.0 Retourne la taille du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_BandFileTimestamp - Disponibilité : 2.5.0 Retourne le timestamp du fichier d'une bande stockée sur le système de fichier. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_ChaikinSmoothing - Disponibilité : 2.5.0 Renvoie une version lissée d'une géométrie, en utilisant l'algorithme Chaikin
— ST_FilterByM - Disponibilité : 2.5.0 Supprime les vertices en fonction de leur valeur M
— ST_Grayscale - Disponibilité : 2.5.0 Crée un nouveau raster à 1 bande 8BUI à partir du raster source et des bandes spécifiées représentant les composantes rouge, vert et bleu
— ST_LineInterpolatePoints - Disponibilité : 2.5.0 Renvoie des points interpolés le long d'une ligne à un intervalle fractionnaire.
— ST_OrientedEnvelope - Disponibilité : 2.5.0. Renvoie un rectangle de surface minimale contenant une géométrie.
— ST_QuantizeCoordinates - Disponibilité : 2.5.0 Met à zéro les bits de poids faible des coordonnées
— ST_RastFromHexWKB - Disponibilité : 2.5.0 Retourne un raster à partir d'un raster Well-Known Binary (WKB) en hexadécimal.
— ST_RastFromWKB - Disponibilité : 2.5.0 Retourne un raster à partir d'un raster Well-Known Binary (WKB).
— ST_SetBandIndex - Disponibilité : 2.5.0 Met à jour le numéro de bande externe d'une bande out-db
— ST_SetBandPath - Disponibilité : 2.5.0 Met à jour le chemin externe et le numéro de bande d'une bande out-db

Fonctions améliorées dans PostGIS 2.5

— ST_AsBinary/ST_AsWKB - Amélioration : 2.5.0 Ajout de ST_AsWKB Retourne la représentation Well-Known Binary (WKB) de ce raster.
— ST_AsMVT - Amélioration : 2.5.0 - ajout de la prise en charge des requêtes parallèles. Fonction d'agrégation renvoyant une représentation MVT d'un ensemble de lignes.
— ST_AsText - Amélioration : 2.5 - introduction de la précision des paramètres optionnels. Renvoie la représentation Well-Known Text (WKT) de la géométrie/geography sans métadonnées SRID.
— ST_BandMetaData - Amélioration : 2.5.0 ajout de outdbbandnum, filesize et filetimestamp pour les rasters outdb. Retourne les méta-données de base d'une bande raster spécifique. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_Buffer - Amélioration : 2.5.0 - La prise en charge de la géométrie ST_Buffer a été améliorée pour permettre la spécification de la mise en mémoire tampon latérale side=bothlleftlright. Calcule une géométrie couvrant tous les points situés à une distance donnée d'une géométrie.
— ST_GeomFromGeoJSON - Amélioration : 2.5.0 peut maintenant accepter json et jsonb comme entrées. Prend en entrée une géométrie au format geojson et renvoie un objet Postgis de type geometry

— ST_GeometricMedian - Amélioré : 2.5.0 Ajout de la prise en charge de M comme poids des points. Renvoie la médiane géométrique d'un MultiPoint.
— ST_Intersects - Amélioré : 2.5.0 Supporte GEOMETRYCOLLECTION. Teste si deux géométries se croisent (elles ont au moins un point en commun)
— ST_OffsetCurve - Amélioration : 2.5 - ajout de la prise en charge de GEOMETRYCOLLECTION et MULTILINESTRING Renvoie une ligne décalée par rapport à une distance et un côté donnés à partir d'une ligne en entrée.
— ST_Scale - Amélioration : la prise en charge de la mise à l'échelle par rapport à une origine locale (paramètre origin) a été introduite dans la version 2.5.0. Met à l'échelle une géométrie en fonction de facteurs donnés.
— ST_Split - Amélioration : la prise en charge de la division d'un polygone par une ligne multiple a été introduite dans la version 2.5.0. Renvoie une collection de géométries créées en divisant une géométrie par une autre géométrie.
— ST_Subdivide - Amélioration : 2.5.0 réutilise les points existants lors de la division d'un polygone, le nombre de vertex est réduit de 8 à 5. Calcule une subdivision rectiligne d'une géométrie.

Fonctions modifiées dans PostGIS 2.5

— ST_GDALDrivers - Changement : 2.5.0 - ajout des colonnes can_read et can_write. Retourne la liste des formats raster supportés par PostGIS via GDAL. Seuls les formats avec can_write=True peuvent être utilisés par ST_AsGDALRaster

13.12.10 Fonctions PostGIS nouvelles ou améliorées en 2.4

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 2.4

— ST_AsGeobuf - Disponibilité : 2.4.0 Retourne une représentation Geobuf d'un ensemble de lignes.
— ST_AsMVT - Disponibilité : 2.4.0 Fonction d'agrégation renvoyant une représentation MVT d'un ensemble de lignes.
— ST_AsMVTGeom - Disponibilité : 2.4.0 Transforme une géométrie dans l'espace de coordonnées d'une tuile MVT.
— ST_Centroid - Disponibilité : la prise en charge du type geography a été introduite dans la version 2.4.0. Renvoie le centre géométrique d'une géométrie.
— ST_ForcePolygonCCW - Disponibilité : 2.4.0 Oriente tous les anneaux extérieurs dans le sens inverse des aiguilles d'une montre et tous les anneaux intérieurs dans le sens des aiguilles d'une montre.
— ST_ForcePolygonCW - Disponibilité : 2.4.0 Oriente tous les anneaux extérieurs dans le sens des aiguilles d'une montre et tous les anneaux intérieurs dans le sens inverse des aiguilles d'une montre.
— ST_FrechetDistance - Disponibilité : 2.4.0 - nécessite GEOS >= 3.7.0 Renvoie la distance de Fréchet entre deux géométries.
— ST_IsPolygonCCW - Disponibilité : 2.4.0 Teste si les polygones ont des anneaux extérieurs orientés dans le sens inverse des aiguilles d'une montre et des anneaux intérieurs orientés dans le sens des aiguilles d'une montre.
— ST_IsPolygonCW - Disponibilité : 2.4.0 Teste si les polygones ont des anneaux extérieurs orientés dans le sens des aiguilles d'une montre et des anneaux intérieurs orientés dans le sens inverse des aiguilles d'une montre.
— ST_MakeEmptyCoverage - Disponibilité : 2.4.0 Couvre une zone géo-référencée avec une grille raster de tuiles vides.

Fonctions améliorées dans PostGIS 2.4

— ST_AsTWKB - Amélioration : 2.4.0 amélioration de la mémoire et de la vitesse. Renvoie la géométrie sous forme de TWKB, diminutif de "Tiny Well-Known Binary"
— ST_Covers - Amélioration : 2.4.0 Ajout de la prise en charge des polygones dans les polygones et des lignes dans les polygones pour le type geography Tests si chaque point de B est situé dans A
— ST_CurveToLine - Amélioration : 2.4.0 a ajouté la prise en charge de la tolérance de l'écart maximal et de l'angle maximal, ainsi que de la sortie symétrique. Convertit une géométrie contenant des courbes en une géométrie linéaire.
— ST_Project - Amélioration : 2.4.0 Autorise les distances négatives et les azimuts non normalisés. Renvoie un point projeté à partir d'un point de départ en fonction d'une distance et d'un azimut.
— ST_Reverse - Amélioration : la prise en charge des courbes a été introduite dans la version 2.4.0. Retourne la géométrie avec l'ordre des sommets inversé.

Fonctions modifiées dans PostGIS 2.4

— = - Modifié : 2.4.0, dans les versions précédentes, il s'agissait d'une égalité de boîte de délimitation et non d'une égalité géométrique. Si vous avez besoin d'une égalité de boîte de délimitation, utilisez à la place. Renvoie TRUE si les coordonnées et l'ordre des coordonnées de la géométrie/géographie A sont les mêmes que les coordonnées et l'ordre des coordonnées de la géométrie/géographie B.
— ST_Node - Modifié : 2.4.0 cette fonction utilise GEOSNode en interne au lieu de GEOSUnaryUnion. Cela peut entraîner un ordre et une direction différents des lignes résultantes par rapport à PostGIS < 2.4. Nœuds d'une collection de lignes.

13.12.11 Fonctions PostGIS nouvelles ou améliorées en 2.3

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 2.3

- &&&(geometry,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si la boîte de délimitation n-D (en cache) d'une géométrie intersecte une boîte de délimitation de précision flottante n-D (GIDX).
- &&&(gidx,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si une boîte de délimitation de précision flottante n-D (GIDX) intersecte la boîte de délimitation n-D (mise en cache) d'une géométrie.
- &&&(gidx,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si deux boîtes de délimitation (GIDX) de précision flottante n-D se croisent.
- &&(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si deux boîtes de délimitation 2D à précision flottante (BOX2DF) se croisent.
- &&(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si une boîte de délimitation 2D de précision flottante (BOX2DF) intersecte la boîte de délimitation 2D (mise en cache) d'une géométrie.
- &&(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si la boîte de délimitation 2D (en cache) d'une géométrie intersecte une boîte de délimitation 2D de précision flottante (BOX2DF).
- @(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans une autre boîte de délimitation de précision flottante 2D.
- @(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) est contenue dans la boîte de délimitation 2D d'une géométrie.
- @(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si la boîte de délimitation 2D d'une géométrie est contenue dans une boîte de délimitation 2D à précision flottante (BOX2DF).
- Populate_Topology_Layer - Disponibilité : 2.3.0 Ajoute les entrées manquantes à la table topology.layer en lisant les métadonnées des tables topo.
- ST_ClusterDBSCAN - Disponibilité : 2.3.0 Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie d'entrée en utilisant l'algorithme DBSCAN.
- ST_ClusterKMeans - Disponibilité : 2.3.0 Fonction Window qui renvoie un identifiant de cluster pour chaque géométrie en entrée en utilisant l'algorithme K-means.
- ST_GeneratePoints - Disponibilité : 2.3.0 Génère un multipoint de points aléatoires contenus dans un polygone ou un multipolygone.
- ST_GeometricMedian - Disponibilité : 2.3.0 Renvoie la médiane géométrique d'un MultiPoint.
- ST_MakeLine - Disponibilité : 2.3.0 - La prise en charge des éléments d'entrée MultiPoint a été introduite Crée une LineString à partir de géométries Point, MultiPoint ou LineString.
- ST_MinimumBoundingRadius - Disponibilité - 2.3.0 Renvoie le point central et le rayon du plus petit cercle contenant une géométrie.
- ST_MinimumClearance - Disponibilité : 2.3.0 Renvoie la clearance (le dégagement) d'une géométrie, une mesure de la robustesse d'une géométrie.
- ST_MinimumClearanceLine - Disponibilité : 2.3.0 - nécessite GEOS >= 3.6.0 Renvoie la chaîne de lignes à deux points couvrant le dégagement (clearance) minimum d'une géométrie.
- ST_Normalize - Disponibilité : 2.3.0 Renvoie la géométrie sous sa forme canonique.
- ST_Points - Disponibilité : 2.3.0 Renvoie un MultiPoint contenant les coordonnées d'une géométrie.
- ST_VoronoiLines - Disponibilité : 2.3.0 Renvoie les limites des polygones de Voronoï des sommets d'une géométrie.
- ST_VoronoiPolygons - Disponibilité : 2.3.0 Renvoie les cellules de la représentation de Voronoï des sommets d'une géométrie.
- ST_WrapX - Disponibilité : 2.3.0 nécessite GEOS Enveloppe une géométrie autour d'une valeur X.
- TopoGeom_addElement - Disponibilité : 2.3 Ajoute un élément à la définition d'une TopoGeometry.
- TopoGeom_remElement - Disponibilité : 2.3 Supprime un élément de la définition d'une TopoGeometry.
- ~(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient une autre boîte de délimitation de précision flottante 2D (BOX2DF).

— ~(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si une boîte de délimitation de précision flottante 2D (BOX2DF) contient la boîte de délimitation 2D d'une géométrie.

— ~(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Renvoie TRUE si la boîte de délimitation 2D d'une géométrie contient une boîte de délimitation de précision flottante 2D (GIDX).

Fonctions améliorées dans PostGIS 2.3

— ST_Contains - Amélioré : 2.3.0 Amélioration du court-circuit PIP étendu à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones. Tests si chaque point de B est situé dans A, et que leurs intérieurs ont un point commun

— ST_Covers - Amélioration : 2.3.0 Amélioration du court-circuit PIP pour la géométrie étendue à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones. Tests si chaque point de B est situé dans A

— ST_Expand - Amélioration : 2.3.0 : prise en charge de l'expansion d'une boîte par différentes quantités dans différentes dimensions. Renvoie une boîte de délimitation développée à partir d'une autre boîte de délimitation ou d'une géométrie.

— ST_Intersects - Amélioré : 2.3.0 Amélioration du court-circuit PIP étendu à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones. Teste si deux géométries se croisent (elles ont au moins un point en commun)

— ST_Segmentize - Amélioration : 2.3.0 La segmentation d'objets geography produit désormais des sous-segments de longueur égale Renvoie une geometry/geography modifiée dont aucun segment ne dépasse une distance donnée.

— ST_Transform - Amélioration : la version 2.3.0 a introduit la prise en charge du texte PROJ.4 direct. Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent.

— ST_Within - Amélioration : 2.3.0 Amélioration du court-circuit PIP pour la géométrie étendue à la prise en charge des multipoints avec peu de points. Les versions précédentes ne prenaient en charge que les points dans les polygones. Tests si chaque point de A se trouve dans B, et que leurs intérieurs ont un point commun

Fonctions modifiées dans PostGIS 2.3

— ST_PointN - Modifié : 2.3.0 : indexation négative disponible (-1 est le dernier point) Renvoie le Nième point de la première LineString ou LineString circulaire d'une géométrie.

13.12.12 Fonctions PostGIS nouvelles ou améliorées en 2.2

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 2.2

— <<->> - Availability: 2.2.0. Renvoie la distance n-D entre les géométries A et B ou les boîtes englobantes

— ST_AsEncodedPolyline - Disponibilité : 2.2.0 Renvoie une polyligne encodée à partir d'une géométrie LineString.

— ST_AsTWKB - Disponibilité : 2.2.0 Renvoie la géométrie sous forme de TWKB, diminutif de "Tiny Well-Known Binary"

— ST_BoundingDiagonal - Disponibilité : 2.2.0 Retourne la diagonale de la boîte englobante pour la géométrie en argument.

— ST_CPAWithin - Disponibilité : 2.2.0 Teste si le point d'approche le plus proche de deux trajectoires se trouve dans la distance spécifiée.

— ST_ClipByBox2D - Disponibilité : 2.2.0 Calcule la partie d'une géométrie située à l'intérieur d'un rectangle.

— ST_ClosestPointOfApproach - Disponibilité : 2.2.0 Renvoie une mesure au point d'approche le plus proche de deux trajectoires.

— ST_ClusterIntersecting - Disponibilité : 2.2.0 Fonction d'agrégation qui regroupe les géométries en entrée en ensembles connectés.

— ST_ClusterWithin - Disponibilité : 2.2.0 Fonction agrégée qui regroupe les géométries en fonction de la distance de séparation.

— ST_CountAgg - Disponibilité : 2.2.0 Agrégat. Renvoie le nombre de pixels dans une bande donnée d'un ensemble de rasters. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels différents de la valeur NODATA seront comptés.

— ST_CreateOverview - Disponibilité : 2.2.0 Créé une version à plus faible résolution d'une colonne raster.

— ST_DistanceCPA - Disponibilité : 2.2.0 Renvoie la distance entre le point d'approche le plus proche de deux trajectoires.

— ST_ForceCurve - Disponibilité : 2.2.0 Retransformation d'une géométrie dans son type de courbure, le cas échéant.

— ST_IsValidTrajectory - Disponibilité : 2.2.0 Teste si la géométrie est une trajectoire valide.

— ST_LineFromEncodedPolyline - Disponibilité : 2.2.0 Crée une LineString depuis une polyligne encodée ("Encoded Polyline").
— ST_MapAlgebra (callback function version) - Disponibilité : 2.2.0 : Possibilité d'ajouter un masque Version avec fonction de rappel - Retourne un raster à une bande à partir d'un ou plusieurs rasters d'entrée, d'index de bandes et d'une fonction de rappel spécifiée par l'utilisateur.
— ST_MemSize - Disponibilité : 2.2.0 Retourne l'espace utilisé par le raster (en octets).
— ST_RemoveRepeatedPoints - Disponibilité : 2.2.0 Renvoie une version d'une géométrie dont les points en double ont été supprimés.
— ST_Retile - Disponibilité : 2.2.0 Retourne un ensemble de tuiles configurées à partir d'une couverture raster composée de tuiles arbitraires.
— ST_SetEffectiveArea - Disponibilité : 2.2.0 Définit la surface effective de chaque sommet, en utilisant l'algorithme Visvalingam-Whyatt.
— ST_SimplifyVW - Disponibilité : 2.2.0 Renvoie une représentation simplifiée d'une géométrie, en utilisant l'algorithme de Visvalingam-Whyatt
— ST_Subdivide - Disponibilité : 2.2.0 Calcule une subdivision rectiligne d'une géométrie.
— ST_SummaryStatsAgg - Disponibilité : 2.2.0 Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed if no band is specified.
— ST_SwapOrdinates - Disponibilité : 2.2.0 Renvoie une version de la géométrie donnée avec les valeurs d'ordonnées permutées.
— postgis.enable_outdb_rasters - Disponibilité : 2.2.0 Une option de configuration booléenne pour permettre l'accès aux bandes matricielles de out-db.
— postgis.gdal_datapath - Disponibilité : 2.2.0 Une option de configuration pour régler la valeur de l'option GDAL_DATA de GDAL. Si elle n'est pas assignée, la valeur de la variable d'environnement GDAL_DATA est utilisée.
— postgis.gdal_enabled_drivers - Disponibilité : 2.2.0 Option de configuration permettant de définir les drivers GDAL activés dans l'environnement PostGIS. Affecte la variable de configuration GDAL GDAL_SKIP.
— |=| - Availability: 2.2.0. Renvoie la distance entre les trajectoires A et B à leur point d'approche le plus proche.

Fonctions améliorées dans PostGIS 2.2

— <-> - Enhanced: 2.2.0 -- True KNN ("K nearest neighbor") behavior for geometry and geography. Note for geography KNN is based on sphere rather than spheroid. Renvoie la distance en 2D entre A et B.
— AsTopoJSON - Amélioration : 2.2.1 ajout de la prise en charge des entrées "puntal" Renvoie la représentation TopoJSON d'une topogeometry.
— ST_Area - Amélioration : 2.2.0 - mesure sur sphéroïde effectuée avec GeographicLib pour une meilleure précision et robustesse. Nécessite PROJ >= 4.9.0 pour profiter de la nouvelle fonctionnalité. Renvoie l'aire d'une géométrie polygonale.
— ST_AsX3D - Amélioration : 2.2.0 : Prise en charge des coordonnées géographiques et de l'inversion des axes (x/y, long/-lat). Voir les options pour plus de détails. Renvoie une géométrie au format X3D xml node element : ISO-IEC-19776-1.2-X3DEncodings-XML
— ST_Azimuth - Amélioration : 2.2.0 mesure sur sphéroïde effectuée avec GeographicLib pour améliorer la précision et la robustesse. Nécessite PROJ >= 4.9.0 pour profiter de la nouvelle fonctionnalité. Renvoie l'azimut basé sur le nord d'une ligne entre deux points.
— ST_Distance - Amélioration : 2.2.0 - mesure sur sphéroïde effectuée avec GeographicLib pour une meilleure précision et robustesse. Nécessite PROJ >= 4.9.0 pour profiter de la nouvelle fonctionnalité. Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_Scale - Amélioration : La prise en charge de la mise à l'échelle de toutes les dimensions (paramètre factor) a été introduite dans la version 2.2.0. Met à l'échelle une géométrie en fonction de facteurs donnés.
— ST_Split - Amélioration : la version 2.2.0 prend en charge la division d'une ligne par une limite multiligne, multipoint ou (multi)polygone. Renvoie une collection de géométries créées en divisant une géométrie par une autre géométrie.
— ST_Summary - Amélioré : 2.2.0 Ajout de la prise en charge des TIN et des courbes Renvoie un résumé textuel du contenu d'une géométrie.

Fonctions modifiées dans PostGIS 2.2

— <-> - Changed: 2.2.0 -- Old hybrid-syntax workarounds may be slower once true KNN is available. See examples below. Renvoie la distance en 2D entre A et B.
— ST_3DClosestPoint - Modifié : 2.2.0 - si 2 géométries 2D sont saisies, un point 2D est renvoyé (au lieu de l'ancien comportement supposant 0 pour Z manquant). Dans le cas de 2D et 3D, Z n'est plus supposé être 0 pour Z manquant. Renvoie le point 3D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne 3D la plus courte.

— ST_3DDistance - Modifié : 2.2.0 - Dans le cas de la 2D et de la 3D, Z n'est plus considéré comme égal à 0 en cas de Z manquant. Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DLongestLine - Modifié : 2.2.0 - si 2 géométries 2D sont saisies, un point 2D est renvoyé (au lieu de l'ancien comportement supposant 0 pour Z manquant). Dans le cas de 2D et 3D, Z n'est plus supposé être 0 pour Z manquant. Renvoie la ligne 3D la plus longue entre deux géométries
— ST_3DMaxDistance - Modifié : 2.2.0 - Dans le cas de la 2D et de la 3D, Z n'est plus considéré comme égal à 0 en cas de Z manquant. Renvoie la distance maximale cartésienne 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DShortestLine - Modifié : 2.2.0 - si 2 géométries 2D sont saisies, un point 2D est renvoyé (au lieu de l'ancien comportement supposant 0 pour Z manquant). Dans le cas de 2D et 3D, Z n'est plus supposé être 0 pour Z manquant. Renvoie la ligne 3D la plus courte entre deux géométries
— ST_DistanceSphere - Modifié : 2.2.0 Dans les versions antérieures, cette fonction s'appelait ST_Distance_Sphere Renvoie la distance minimale en mètres entre deux géométries lon/lat en utilisant un modèle de terre sphérique.
— ST_DistanceSpheroid - Modifié : 2.2.0 Dans les versions précédentes, cette fonction était appelée ST_Distance_Sphéroïde Renvoie la distance minimale entre deux géométries lon/lat en utilisant un modèle de terre sphéroïdale.
— ST_Equals - Modifié : 2.2.0 Retourne vrai même pour les géométries invalides si elles sont binairement égales Teste si deux géométries comprennent le même ensemble de points
— ST_LengthSpheroid - Modifié : 2.2.0 Dans les versions précédentes, cette fonction s'appelait ST_Length_Spheroid et avait l'alias ST_3DLength_Spheroid Renvoie la longueur/périmètre 2D ou 3D d'une géométrie lon/lat sur un sphéroïde.
— ST_MemSize - Modifié : 2.2.0 nom modifié en ST_MemSize pour respecter la convention de nommage. Renvoie la quantité d'espace mémoire que prend une géométrie.
— ST_PointInsideCircle - Modifié : 2.2.0 Dans les versions précédentes, cette fonction était appelée ST_Point_Inside_Circle Teste si un point géométrique se trouve à l'intérieur d'un cercle défini par un centre et un rayon
— ValidateTopology - Modifié : 2.2.0 les valeurs pour id1 et id2 ont été échangées pour "edge crosses node" pour être cohérent avec la description de l'erreur. Renvoie un ensemble d'objets validatetopology_returntype détaillant les problèmes liés à la topologie.

13.12.13 Fonctions PostGIS nouvelles ou améliorées en 2.1

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 2.1

— = - Disponibilité : 2.1.0 Retourne TRUE si la boîte englobante de A est la même que celle de B. Utilise des boîtes englobantes en double précision.
— AsTopoJSON - Disponibilité : 2.1.0 Renvoie la représentation TopoJSON d'une topogeometry.
— ST_Box2dFromGeoHash - Disponibilité : 2.1.0 Retourne une BOX2D à partir d'une chaîne GeoHash.
— ST_ColorMap - Disponibilité : 2.1.0 Crée un nouveau raster comprenant jusqu'à quatre bandes 8BUI (niveaux de gris, RGB, RGBA) à partir du raster source et d'une bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_Contains - Disponibilité : 2.1.0 Retourne true si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA et si au moins un point de l'intérieur du raster rastB se trouve à l'intérieur du raster rastA.
— ST_ContainsProperly - Disponibilité : 2.1.0 Retourne true si rastB intersecte l'intérieur de rastA, mais pas la frontière ou l'extérieur de rastA.
— ST_CoveredBy - Disponibilité : 2.1.0 Retourne true si aucun point du raster rastA ne se trouve à l'extérieur du raster rastB.
— ST_Covers - Disponibilité : 2.1.0 Retourne true si aucun point du raster rastB ne se trouve à l'extérieur du raster rastA.
— ST_DFullyWithin - Disponibilité : 2.1.0 Retourne true si les rasters rastA et rastB se trouvent entièrement à une distance donnée l'un de l'autre.
— ST_DWithin - Disponibilité : 2.1.0 Retourne true si les rasters rastA et rastB se trouvent à une distance donnée l'un de l'autre.
— ST_DelaunayTriangles - Disponibilité : 2.1.0 Renvoie la triangulation de Delaunay des sommets d'une géométrie.
— ST_Disjoint - Disponibilité : 2.1.0 Retourne true si le raster rastA n'intersecte pas spatialement le rastB.

— ST_DumpValues - Disponibilité : 2.1.0 Retourne les valeurs d'une bande raster spécifiée, sous forme d'un tableau à deux dimensions.
— ST_ForceSFS - Disponibilité : 2.1.0 Force geometries to use SFS 1.1 or 1.2 geometry types.
— ST_FromGDALRaster - Disponibilité : 2.1.0 Retourne un raster depuis un fichier raster supporté par GDAL.
— ST_GeomFromGeoHash - Disponibilité : 2.1.0 Retourne une geometry depuis une chaîne GeoHash.
— ST_InvDistWeight4ma - Disponibilité : 2.1.0 Fonction de traitement des données raster qui interpole la valeur d'un pixel à partir de son voisinage.
— ST_MapAlgebra (callback function version) - Disponibilité : 2.1.0 Version avec fonction de rappel - Retourne un raster à une bande à partir d'un ou plusieurs rasters d'entrée, d'index de bandes et d'une fonction de rappel spécifiée par l'utilisateur.
— ST_MapAlgebra (expression version) - Disponibilité : 2.1.0 Version avec expression - Retourne un raster à une bande à partir d'un ou deux rasters d'entrée, d'index de bandes et d'une ou plusieurs expressions SQL spécifiées par l'utilisateur.
— ST_MinConvexHull - Disponibilité : 2.1.0 Retourne la géométrie de l'enveloppe convexe du raster en excluant les pixels NODATA.
— ST_MinDist4ma - Disponibilité : 2.1.0 Fonction de traitement des données raster qui renvoie la distance minimale (en nombre de pixels) entre le pixel d'intérêt et un pixel voisin ayant une valeur.
— ST_NearestValue - Disponibilité : 2.1.0 Retourne la valeur la plus proche différent de NODATA pour une bande raster spécifiée au pixel donné par columnx et rowy, ou à un point géométrique spécifié dans le même système de référence spatial que le raster.
— ST_Neighborhood - Disponibilité : 2.1.0 Retourne un tableau 2-D de double avec les valeurs non NODATA autour du pixel de la bande spécifiée, aux coordonnées spécifiées par columnX & rowY ou par un point géométrique dans le même système de référence spatial que le raster.
— ST_NotSameAlignmentReason - Disponibilité : 2.1.0 Retourne un texte indiquant si les rasters sont alignés et, s'ils ne le sont pas, la raison du problème.
— ST_Overlaps - Disponibilité : 2.1.0 Retourne true si les raster rastA et rastB intersectent mais que l'un ne contient pas complètement l'autre.
— ST_PixelAsCentroid - Disponibilité : 2.1.0 Retourne le centroïde (point géométrique) de la zone représentée par un pixel.
— ST_PixelAsCentroids - Disponibilité : 2.1.0 Retourne le centroïde (point géométrique) pour chaque pixel de la bande raster, avec sa valeur et les coordonnées raster X et Y. Le point géométrique est le centroïde de la zone représentée par un pixel.
— ST_PixelAsPoint - Disponibilité : 2.1.0 Retourne le point géométrique du coin supérieur gauche du pixel.
— ST_PixelAsPoints - Disponibilité : 2.1.0 Retourne un point géométrique pour chaque pixel de la bande raster, avec sa valeur et ses coordonnées raster X et Y. Les coordonnées du points sont ceux du coin supérieur gauche du pixel.
— ST_PixelOfValue - Disponibilité : 2.1.0 Retourne les coordonnées columnx, rowy du pixel dont la valeur est égale à la valeur recherchée.
— ST_PointFromGeoHash - Disponibilité : 2.1.0 Retourne un point à partir d'une chaîne GeoHash.
— ST_RasterToWorldCoord - Disponibilité : 2.1.0 Retourne le coin supérieur gauche du raster, sous forme de coordonnées X et Y (longitude et latitude) d'un colonne et d'une ligne. Les numéros de colonne et de ligne commencent à 1.
— ST_Resize - Disponibilité : 2.1.0 Nécessite GDAL 1.6.1+ Redimensionne un raster à une nouvelle largeur/hauteur
— ST_Roughness - Disponibilité : 2.1.0 Retourne un raster avec la rugosité d'un MNT.
— ST_SetValues - Disponibilité : 2.1.0 Retourne un nouveau raster en modifiant les valeurs de certains pixels d'une bande spécifiée.
— ST_Simplify - Disponibilité : 2.1.0 Renvoie une version géométrique "simplifiée" de la TopoGeometry donnée en utilisant l'algorithme de Douglas-Peucker.
— ST_Summary - Disponibilité : 2.1.0 Retourne un résumé du contenu du raster sous forme de texte.
— ST_TPI - Disponibilité : 2.1.0 Retourne un raster avec l'index de position topographique (TPI) calculé.
— ST_TRI - Disponibilité : 2.1.0 Retourne un raster avec l'indice de rugosité du terrain (TRI) calculé.
— ST_Tile - Disponibilité : 2.1.0 Retourne un ensemble de rasters issus de la division d'un raster d'entrée selon les dimensions spécifiées pour les rasters de sortie.
— ST_Touches - Disponibilité : 2.1.0 Retourne true si les raster rastA et rastB ont au moins un point en commun mais que leurs intérieurs n'intersectent pas.

— ST_Union - Disponibilité : 2.1.0 Ajout de la variante ST_Union(rast, unionarg). Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— ST_Within - Disponibilité : 2.1.0 Retourne true si aucun point du raster rastA ne se trouve à l'extérieur du raster rastB et si au moins un point de l'intérieur du raster rastA se trouve à l'intérieur du raster rastB.
— ST_WorldToRasterCoord - Disponibilité : 2.1.0 Retourne le coin supérieur gauche comme colonne et ligne, en fonction de coordonnées géométriques X et Y (longitude et latitude) ou un point géométrique dans le système de référence spatial du raster.
— UpdateRasterSRID - Disponibilité : 2.1.0 Change le SRID de tous les rasters dans la table et colonne en paramètres.
— clearTopoGeom - Disponibilité : 2.1 Efface le contenu d'une géométrie topo.
— postgis_sfcgal_version - Disponibilité : 2.1.0 Retourne la version de SFCGAL utilisée

Fonctions améliorées dans PostGIS 2.1

— ST_AddBand - Amélioration : 2.1.0 ajout du paramètre addbandarg. Retourne un raster avec la/les nouvelle(s) bande(s) ajouté(s) à un index donné, de type et valeur initiale donnés. Si aucun index n'est spécifié, la bande est ajoutée à la fin.
— ST_AddBand - Amélioration : 2.1.0 introduction du support des bandes out-db. Retourne un raster avec la/les nouvelle(s) bande(s) ajouté(s) à un index donné, de type et valeur initiale donnés. Si aucun index n'est spécifié, la bande est ajoutée à la fin.
— ST_AsBinary/ST_AsWKB - Amélioration : 2.1.0 Ajout de outasin Retourne la représentation Well-Known Binary (WKB) de ce raster.
— ST_AsGML - Amélioration : 2.1.0 La prise en charge des identifiants a été introduite pour GML 3. Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_Aspect - Amélioration : 2.1.0 Utilise ST_MapAlgebra() et ajout du paramètre optionnel interpolate_nodata Retourne l'exposition (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_Boundary - Amélioration : 2.1.0 introduction du support pour Triangle Renvoie la limite d'une géométrie.
— ST_Clip - Amélioration : 2.1.0 Réécrit en C Renvoie le raster découpé par la géométrie d'entrée. Si le numéro de bande n'est pas spécifié, toutes les bandes sont traitées. Si crop n'est pas spécifié ou est TRUE, le raster de sortie est rogné. Si touched est défini sur TRUE, les pixels touchés sont inclus, sinon seulement si le centre du pixel est dans la géométrie, il est inclus.
— ST_DWithin - Amélioration : la version 2.1.0 a amélioré la vitesse pour le type geography. Voir Making Geography faster pour plus de détails. Teste si deux géométries se trouvent à une distance donnée
— ST_DWithin - Amélioration : la prise en charge des géométries courbes a été introduite dans la version 2.1.0. Teste si deux géométries se trouvent à une distance donnée
— ST_Distance - Amélioration : la version 2.1.0 a amélioré la vitesse pour le type geography. Voir Making Geography faster pour plus de détails. Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_Distance - Amélioration : 2.1.0 - la prise en charge des géométries courbes a été introduite. Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_Distinct4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule le nombre de valeurs de pixels uniques dans un voisinage.
— ST_DumpPoints - Amélioré : 2.1.0 Vitesse plus rapide. Réimplémentation en C natif. Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_HillShade - Amélioration : 2.1.0 Utilise ST_MapAlgebra() et ajout du paramètre optionnel interpolate_nodata Retourne l'hypothétique éclairage d'une bande raster d'élévation en utilisant les valeurs d'azimut, d'altitude, de luminosité et d'échelle fournies.
— ST_MakeValid - Amélioration : 2.1.0, ajout du support pour GEOMETRYCOLLECTION et MULTIPOINT. Tente de rendre valide une géométrie invalide sans perdre de sommets.
— ST_Max4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule la valeur maximale d'un pixel dans un voisinage.
— ST_Mean4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule la valeur moyenne d'un pixel dans un voisinage.
— ST_Min4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule la valeur minimale d'un pixel dans un voisinage.
— ST_PixelAsPolygons - Amélioration : 2.1.0 ajout du paramètre optionnel exclude_nodata_value. Retourne la géométrie polygonale qui délimite chaque pixel de la bande raster, avec la valeur et les coordonnées raster X et Y de chaque pixel.
— ST_Polygon - Amélioration : 2.1.0 Vitesse améliorée (entièrement en C) et le multipolygone retourné est garanti d'être valide. Retourne une géométrie multipolygonale formée par l'union des pixels dont la valeur est différente de nodata. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.

— ST_Range4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule la plage de valeurs des pixels dans un voisinage.
— ST_SameAlignment - Amélioration : 2.1.0 ajout de la variante agrégée Retourne true si les rasters ont les mêmes skew, scale, spatial ref, et offset (les pixels peuvent être placés sur la même grille sans être coupés) et false si ce n'est pas le cas, avec une NOTICE détaillant le problème.
— ST_Segmentize - Amélioration : la prise en charge des objets de type geography a été introduite dans la version 2.1.0. Renvoie une geometry/geography modifiée dont aucun segment ne dépasse une distance donnée.
— ST_SetGeoReference - Amélioration : 2.1.0 Ajout de la variante ST_SetGeoReference(raster, double precision, ...) Définit les 6 paramètres de géo-référencement en un seul appel. Les nombres doivent être séparés par un espace. Accepte les formats GDAL (par défaut) ou ESRI.
— ST_SetValue - Amélioration : 2.1.0 La variante de ST_SetValue() avec la géométrie supporte n'importe quelle type de géométrie, pas uniquement des points. La variante avec la géométrie est une enveloppe autour de la variante de ST_SetValues() avec geomval[] Retourne un nouveau raster en modifiant la valeur du pixel pour la bande spécifiée et aux coordonnées columnx, rowy, ou pour tous les pixels qui intersectent une géométrie spécifiée. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié.
— ST_Slope - Amélioration : 2.1.0 Utilise ST_MapAlgebra() et ajout des paramètres optionnels units, scale, interpolate_nodata Retourne la pente (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_StdDev4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule l'écart type des valeurs des pixels dans un voisinage.
— ST_Sum4ma - Amélioration : 2.1.0 Ajout de la variante 2 Fonction de traitement des données raster qui calcule la somme de toutes les valeurs de pixels dans un voisinage.
— ST_Summary - Amélioré : 2.1.0 Indicateur S pour indiquer si le système de référence spatiale est connu Renvoie un résumé textuel du contenu d'une géométrie.
— ST_Transform - Amélioration : 2.1.0 Ajout de la variante ST_Transform(rast, alignto) Reprojette un raster depuis un système de référence spatial vers un autre, en utilisant l'algorithme de rééchantillonnage spécifié. Les algorithmes possibles sont NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.
— ST_Union - Amélioration : 2.1.0 Vitesse améliorée (entièrement en C). Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— ST_Union - Amélioration : 2.1.0 ST_Union(rast) (variante 1) permet l'union sur toutes les bandes de tous les rasters d'entrée. Les versions précédentes de PostGIS ne prenaient en compte que la première bande. Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— ST_Union - Amélioration : 2.1.0 ST_Union(rast, uniontype) (variante 4) permet l'union de toutes les bandes de tous les rasters d'entrée. Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— toTopoGeom - Amélioré : 2.1.0 ajoute la version d'une TopoGeometry existante. Convertit une géométrie simple en une géométrie topographique.

Fonctions modifiées dans PostGIS 2.1

— ST_Aspect - Changement : 2.1.0 Dans les versions précédentes, les valeurs retournées étaient en radians. Désormais, les valeurs retournées sont par défaut en degrés Retourne l'exposition (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_EstimatedExtent - Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Estimated_Extent. Renvoie l'étendue estimée d'une table spatiale.
— ST_Force2D - Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_2D. Forcer les géométries à passer en "mode bidimensionnel".
— ST_Force3D - Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_3D. Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.
— ST_Force3DM - Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_3DM. Forcer les géométries en mode XYM.
— ST_Force3DZ - Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_3DZ. Forcer les géométries en mode XYZ.
— ST_Force4D - Modifié : 2.1.0. Jusqu'à la version 2.0.x, elle s'appelait ST_Force_4D. Forcer les géométries en mode XYZM.
— ST_ForceCollection - Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Force_Collection. Convertir la géométrie en une GEOMETRYCOLLECTION.

— ST_HillShade - Changement : 2.1.0 Dans les versions précédentes, l'azimut et l'altitude étaient exprimés en radians. Désormais, l'azimut et l'altitude sont exprimés en degrés Retourne l'hypothétique éclairage d'une bande raster d'élévation en utilisant les valeurs d'azimut, d'altitude, de luminosité et d'échelle fournies.

— ST_LineInterpolatePoint - Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Line_Interpolate_Point. Renvoie un point interpolé le long d'une ligne à un emplacement fractionnaire.

— ST_LineLocatePoint - Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Line_Locate_Point. Renvoie l'emplacement fractionnaire du point le plus proche d'un point sur une ligne.

— ST_LineSubstring - Modifié : 2.1.0. Jusqu'à la version 2.0.x, cette fonction était appelée ST_Line_Substring. Renvoie la partie d'une ligne située entre deux emplacements fractionnaires.

— ST_PixelAsCentroids - Changement : 2.1.1 Changement du comportement de behavior of exclude_nodata_value. Retourne le centroïde (point géométrique) pour chaque pixel de la bande raster, avec sa valeur et les coordonnées raster X et Y. Le point géométrique est le centroïde de la zone représentée par un pixel.

— ST_PixelAsPoints - Changement : 2.1.1 Changement du comportement de behavior of exclude_nodata_value. Retourne un point géométrique pour chaque pixel de la bande raster, avec sa valeur et ses coordonnées raster X et Y. Les coordonnées du points sont ceux du coin supérieur gauche du pixel.

— ST_PixelAsPolygons - Changement : 2.1.1 Changement du comportement de behavior of exclude_nodata_value. Retourne la géométrie polygonale qui délimite chaque pixel de la bande raster, avec la valeur et les coordonnées raster X et Y de chaque pixel.

— ST_Polygon - Changement : 2.1.0 Dans les versions précédentes, retournait parfois un polygone, retourne désormais toujours un multipolygon. Retourne une géométrie multipolygonale formée par l'union des pixels dont la valeur est différente de nodata. Si aucune bande bandnum n'est spécifiée, la bande 1 est utilisée.

— ST_RasterToWorldCoordX - Changement : 2.1.0 Jusqu'à la version 2.0.x, cette fonction était appelée ST_Raster2WorldCoordX Retourne la coordonnée X du coin supérieur gauche du raster à column et row. Les numéros de colonne et de ligne commencent à 1.

— ST_RasterToWorldCoordY - Changement : 2.1.0 Jusqu'à la version 2.0.x, cette fonction était appelée ST_Raster2WorldCoordY Retourne la coordonnée Y du coin supérieur gauche du raster à column et row. Les numéros de colonne et de ligne commencent à 1.

— ST_Rescale - Changement : 2.1.0 Fonctionne avec les rasters sans SRID Rééchantillonne un raster en ajustant juste son échelle (ou la taille des pixels). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline), Lanczos, Max ou Min. La valeur par défaut est NearestNeighbor.

— ST_Reskew - Changement : 2.1.0 Fonctionne avec les rasters sans SRID Rééchantillonne un raster en ajustant simplement son obliquité (skew, ou paramètre de rotation). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.

— ST_Segmentize - Modifié : 2.1.0 Suite à l'introduction de la prise en charge du type geography, l'utilisation ST_Segmentize('LINEST
2, 3 4)', 0.5) provoque une erreur de fonction ambiguë. L'entrée doit être correctement typée en tant que geometry ou geography. Utilisez ST_GeomFromText, ST_GeogFromText ou un cast vers le type requis (par exemple, ST_Segmentize('LINESTRING(1
2, 3 4)'::geometry, 0.5)) Renvoie une geometry/geography modifiée dont aucun segment ne dépasse une distance donnée.

— ST_Slope - Changement : 2.1.0 Dans les versions précédentes, les valeurs retournées étaient en radians. Désormais, les valeurs retournées sont par défaut en degrés Retourne la pente (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.

— ST_SnapToGrid - Changement : 2.1.0 Fonctionne avec les rasters sans SRID Rééchantillonne un raster en l'accrochant sur une grille. Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.

— ST_WorldToRasterCoordX - Changement : 2.1.0 Dans les versions précédentes, cette fonctionne était appelée ST_World2RasterCoor
Retourne la colonne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw) exprimés dans le système de référence spatial du raster.

— ST_WorldToRasterCoordY - Changement : 2.1.0 Dans les versions précédentes, cette fonctionne était appelée ST_World2RasterCoor
Retourne la ligne dans le raster du point géométrique (pt) ou des coordonnées X et Y (xw, yw) exprimés dans le système de référence spatial du raster.

13.12.14 Fonctions PostGIS nouvelles ou améliorées en 2.0

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 2.0

— && - Disponibilité : 2.0.0 Retourne TRUE si la boite englobante de A intersecte la boite englobante de B.
— &&& - Disponibilité : 2.0.0 Renvoie TRUE si la boîte de délimitation n-D de A intersecte la boîte de délimitation n-D de B.
— <#> - Availability: 2.0.0. Renvoie la distance 2D entre les boîtes de délimitation A et B.
— <-> - Availability: 2.0.0 -- Weak KNN provides nearest neighbors based on geometry centroid distances instead of true distances. Exact results for points, inexact for all other types. Renvoie la distance en 2D entre A et B.
— @ - Disponibilité : 2.0.0 ajout de raster @ raster, raster @ geometry Retourne TRUE si la boîte englobante de A est contenue dans celle de B. Utilise des boîtes englobantes en double précision.
— @ - Disponibilité : 2.0.5 ajout de geometry @ raster Retourne TRUE si la boîte englobante de A est contenue dans celle de B. Utilise des boîtes englobantes en double précision.
— AddEdge - Disponibilité : 2.0.0 Ajoute une arête linéaire à la table des arêtes et les points de départ et d'arrivée associés à la table des nœuds de points du schéma topologique spécifié en utilisant la géométrie linéaire spécifiée et renvoie l'identifiant de l'arête nouvelle (ou existante).
— AddFace - Disponibilité : 2.0.0 Enregistre une primitive de face dans une topologie et obtient son identifiant.
— AddNode - Disponibilité : 2.0.0 Ajoute un nœud ponctuel à la table des nœuds dans le schéma topologique spécifié et renvoie le nodeid du nouveau nœud. Si le point existe déjà en tant que nœud, l'identifiant du nœud existant est renvoyé.
— AddOverviewConstraints - Disponibilité : 2.0.0 Marque une colonne raster comme étant un aperçu d'une autre colonne.
— AddRasterConstraints - Disponibilité : 2.0.0 Ajoute des contraintes raster à une table pour une colonne spécifique pour contraindre le système de référence spatial, l'échelle, la taille des blocs, l'alignement, le nombre de bandes, le type de pixels, la contrainte d'unicité spatiale. La table doit être chargée avec des données pour que les contraintes puissent être inférées. Retourne true si les contraintes ont été ajoutées, ou émet une notice sinon.
— AsGML - Disponibilité : 2.0.0 Renvoie la représentation GML d'une topogeometry.
— CopyTopology - Disponibilité : 2.0.0 Copie une topologie (nœuds, arêtes, faces, couches et TopoGeometries) dans un nouveau schéma
— DropOverviewConstraints - Disponibilité : 2.0.0 Supprime les contraintes d'aperçu de la colonne.
— DropRasterConstraints - Disponibilité : 2.0.0 Supprime des contraintes raster PostGIS raster sur une colonne raster d'une table. Utile pour recharger des données ou mettre à jour les données d'une colonne raster.
— GetEdgeByPoint - Disponibilité : 2.0.0 Trouve l'identifiant d'une arête qui coupe un point donné.
— GetFaceByPoint - Disponibilité : 2.0.0 Recherche la face intersectant un point donné.
— GetNodeByPoint - Disponibilité : 2.0.0 Recherche l'identifiant d'un nœud à un point donné.
— GetNodeEdges - Disponibilité : 2.0 Renvoie un ensemble ordonné d'arêtes incidentes au nœud donné.
— GetRingEdges - Disponibilité : 2.0.0 Renvoie l'ensemble ordonné des identifiants d'arêtes signés rencontrés en évoluant sur un côté d'arête donné.
— GetTopoGeomElements - Disponibilité : 2.0.0 Renvoie un ensemble d'objets topoelement contenant les éléments topologiques element_id,element_type de la TopoGeometry donnée (éléments primitifs).
— GetTopologySRID - Disponibilité : 2.0.0 Renvoie le SRID d'une topologie dans la table topology.topology en fonction du nom de la topologie.
— Polygonize - Disponibilité : 2.0.0 Recherche et enregistre toutes les faces définies par les arêtes de la topologie.
— ST_3DClosestPoint - Disponibilité : 2.0.0 Renvoie le point 3D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne 3D la plus courte.
— ST_3DDFullyWithin - Disponibilité : 2.0.0 Teste si deux géométries 3D sont entièrement comprises dans une distance 3D donnée
— ST_3DDWithin - Disponibilité : 2.0.0 Teste si deux géométries 3D se trouvent à une distance 3D donnée
— ST_3DDistance - Disponibilité : 2.0.0 Renvoie la distance cartésienne minimale en 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DIntersects - Disponibilité : 2.0.0 Teste si deux géométries se croisent dans l'espace en 3D - uniquement pour les points, les lignes, les polygones, les surfaces polyédriques (aire)
— ST_3DLongestLine - Disponibilité : 2.0.0 Renvoie la ligne 3D la plus longue entre deux géométries

— ST_3DMaxDistance - Disponibilité : 2.0.0 Renvoie la distance maximale cartésienne 3D (basée sur la référence spatiale) entre deux géométries en unités projetées.
— ST_3DShortestLine - Disponibilité : 2.0.0 Renvoie la ligne 3D la plus courte entre deux géométries
— ST_AddEdgeModFace - Disponibilité : 2.0 Ajoutez une nouvelle arête et, si elle divise une face, modifiez la face d'origine et ajoutez une nouvelle face.
— ST_AddEdgeNewFaces - Disponibilité : 2.0 Ajoutez une nouvelle arête et, si elle divise une face, supprimez la face d'origine et remplacez-la par deux nouvelles faces.
— ST_AsGDALRaster - Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0. Retourne la tuile raster dans le format GDAL raster spécifié. Les formats raster disponibles sont ceux supportés par votre bibliothèque compilée. Utilisez ST_GDALDrivers() pour obtenir la liste des formats supportés par votre bibliothèque.
— ST_AsJPEG - Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0. Retourne les bandes sélectionnées du raster sous la forme d'une image JPEG (sous forme de tableau d'octets). Si aucune bande n'est spécifiée, et que le raster a 1 ou plus de 3 bandes, seule la première bande est utilisée. Si le raster a exactement 3 bandes, les 3 bandes sont utilisées et mappées en RGB.
— ST_AsLatLonText - Disponibilité : 2.0 Renvoie la représentation en degrés, minutes et secondes du point donné.
— ST_AsPNG - Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0. Retourne les bandes sélectionnées du raster sous la forme d'une image PNG (sous forme de tableau d'octets). Si aucune bande n'est spécifiée et que le raster a 1, 3 ou 4 bandes, toutes les bandes sont utilisées. Si aucune bande n'est spécifiée et que le raster a 2 ou plus de 4 bandes, seule la bande 1 est utilisée. Les bandes sont mappées en RGB ou RGBA.
— ST_AsRaster - Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0. Convertit une géométrie PostGIS en un raster PostGIS.
— ST_AsTIFF - Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0. Retourne les bandes sélectionnées du raster sous la forme d'une seule image TIFF (sous forme de tableau d'octets). Si aucune bande n'est spécifiée ou si l'une des bandes spécifiées n'existe pas dans le raster, toutes les bandes sont utilisées.
— ST_AsX3D - Disponibilité : 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Renvoie une géométrie au format X3D xml node element : ISO-IEC-19776-1.2-X3DEncodings-XML
— ST_Aspect - Disponibilité : 2.0.0 Retourne l'exposition (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_Band - Disponibilité : 2.0.0 Retourne une ou plusieurs bandes d'un raster existant en tant que nouveau raster. Utile pour construire de nouveaux rasters à partir de rasters existants.
— ST_BandIsNoData - Disponibilité : 2.0.0 Retourne true si la bande ne contient que des valeurs nodata.
— ST_Clip - Disponibilité : 2.0.0 Renvoie le raster découpé par la géométrie d'entrée. Si le numéro de bande n'est pas spécifié, toutes les bandes sont traitées. Si crop n'est pas spécifié ou est TRUE, le raster de sortie est rogné. Si touched est défini sur TRUE, les pixels touchés sont inclus, sinon seulement si le centre du pixel est dans la géométrie, il est inclus.
— ST_CollectionHomogenize - Disponibilité : 2.0.0 Renvoie la représentation la plus simple d'une collection de géométries.
— ST_ConcaveHull - Disponibilité : 2.0.0 Calcule une géométrie éventuellement concave qui contient tous les sommets de la géométrie d'entrée
— ST_Count - Disponibilité : 2.0.0 Renvoie le nombre de pixels dans une bande donnée d'un raster ou d'une couverture raster. Si aucune bande n'est spécifiée, la valeur par défaut est la bande 1. Si exclude_nodata_value est true, seuls les pixels dont la valeur est différente de la valeur nodata seront comptés.
— ST_CreateTopoGeo - Disponibilité : 2.0 Ajoute une collection de géométries à une topologie vide donnée et renvoie un message détaillant le succès.
— ST_Distinct4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule le nombre de valeurs de pixels uniques dans un voisinage.
— ST_FlipCoordinates - Disponibilité : 2.0.0 Renvoie une version d'une géométrie dont les axes X et Y sont inversés.
— ST_GDALDrivers - Disponibilité : 2.0.0 - nécessite GDAL >= 1.6.0. Retourne la liste des formats raster supportés par PostGIS via GDAL. Seuls les formats avec can_write=True peuvent être utilisés par ST_AsGDALRaster
— ST_GeomFromGeoJSON - Disponibilité : 2.0.0 nécessite JSON-C >= 0.9 Prend en entrée une géométrie au format geojson et renvoie un objet Postgis de type geometry
— ST_GetFaceEdges - Disponibilité : 2.0 Renvoie un ensemble d'arêtes ordonnées qui délimitent aface.
— ST_HasNoBand - Disponibilité : 2.0.0 Retourne true si le raster n'a pas la bande spécifiée. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_HillShade - Disponibilité : 2.0.0 Retourne l'hypothétique éclairage d'une bande raster d'élévation en utilisant les valeurs d'azimut, d'altitude, de luminosité et d'échelle fournies.
— ST_Histogram - Disponibilité : 2.0.0 Retourne un ensemble d'enregistrements résumant une distribution de données raster ou de couverture raster, dans des classes distinctes. Le nombre de classes est calculé automatiquement s'il n'est pas spécifié.

— ST_InterpolatePoint - Disponibilité : 2.0.0 Renvoie la mesure interpolée d'une géométrie la plus proche d'un point.
— ST_IsEmpty - Disponibilité : 2.0.0 Retourne true si le raster est vide (largeur = 0 et hauteur = 0). Sinon, retourne false.
— ST_IsValidDetail - Disponibilité : 2.0.0 Renvoie une ligne valid_detail indiquant si une géométrie est valide ou sinon une raison et une localisation.
— ST_IsValidReason - Disponibilité : la version 2.0 prend des flags. Renvoie un texte indiquant si une géométrie est valide, ou la raison de son invalidité.
— ST_MakeLine - Disponibilité : 2.0.0 - La prise en charge des éléments d'entrée LineString a été introduite Crée une LineString à partir de géométries Point, MultiPoint ou LineString.
— ST_MakeValid - Disponibilité : 2.0.0 Tente de rendre valide une géométrie invalide sans perdre de sommets.
— ST_MapAlgebraExpr - Disponibilité : 2.0.0 Version avec raster à 1 bande : Crée un nouveau raster à une bande formé par l'application d'une opération algébrique PostgreSQL valide sur la bande d'entrée du raster et du type de pixel fourni. Si aucune bande n'est spécifiée, la bande 1 est utilisée.
— ST_MapAlgebraExpr - Disponibilité : 2.0.0 Version avec 2 bandes : Crée un nouveau raster à une bande formé en appliquant une opération algébrique PostgreSQL valide sur les deux bandes du raster d'entrée et du type de pixel fourni. La bande 1 de chaque raster est utilisée si aucun numéro de bande n'est spécifié. Le raster résultant sera aligné (échelle, obliquité et coins) sur la grille définie par le premier raster et aura son étendue définie par le paramètre "extenttype". Les valeurs de "extenttype" peuvent être : INTERSECTION, UNION, FIRST, SECOND.
— ST_MapAlgebraFct - Disponibilité : 2.0.0 Version à 1 bande - Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide sur la bande d'entrée du raster et sur le type de pixel fourni. La bande 1 est utilisée si aucune bande n'est spécifiée.
— ST_MapAlgebraFct - Disponibilité : 2.0.0 Version à 2 bandes - Crée un nouveau raster à une bande formé par l'application d'une fonction PostgreSQL valide sur les 2 bandes d'entrée du raster et sur le type de pixel fourni. La bande 1 est utilisée si aucune bande n'est spécifiée. Le type d'étendue INTERSECTION est utilisé si non spécifié.
— ST_MapAlgebraFctNgb - Disponibilité : 2.0.0 Version à 1 bande : Algèbre cartographique Plus proche voisin en utilisant une fonction PostgreSQL définie par l'utilisateur. Retourne un raster dont les valeurs sont le résultat d'une fonction utilisateur PLPGSQL prenant un voisinage des valeurs de la bande raster d'entrée.
— ST_Max4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule la valeur maximale d'un pixel dans un voisinage.
— ST_Mean4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule la valeur moyenne d'un pixel dans un voisinage.
— ST_Min4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule la valeur minimale d'un pixel dans un voisinage.
— ST_ModEdgeHeal - Disponibilité : 2.0 Répare deux arêtes en supprimant le nœud qui les relie, en modifiant la première arête et en supprimant la seconde. Renvoie l'identifiant du nœud supprimé.
— ST_MoveIsoNode - Disponibilité : 2.0.0 Déplace un nœud isolé dans une topologie d'un point à un autre. Si la nouvelle géométrie apoint existe en tant que noeud, une erreur est générée. Retourne la description du déplacement.
— ST_NewEdgeHeal - Disponibilité : 2.0 Répare deux arêtes en supprimant le nœud qui les relie, en supprimant les deux arêtes et en les remplaçant par une arête dont la direction est la même que la première arête fournie.
— ST_Node - Disponibilité : 2.0.0 Nœuds d'une collection de lignes.
— ST_NumPatches - Disponibilité : 2.0.0 Returns the number of faces in a PolyhedralSurface or TIN.
— ST_OffsetCurve - Disponibilité : 2.0 Renvoie une ligne décalée par rapport à une distance et un côté donnés à partir d'une ligne en entrée.
— ST_PatchN - Disponibilité : 2.0.0 Returns the Nth face of a PolyhedralSurface or TIN.
— ST_Perimeter - Disponibilité 2.0.0 : La prise en charge du type geography a été introduite Renvoie la longueur de la limite d'une géométrie polygonale ou d'une géographie.
— ST_PixelAsPolygon - Disponibilité : 2.0.0 Retourne la géométrie polygonale qui délimite le pixel pour une ligne et colonne spécifiées.
— ST_PixelAsPolygons - Disponibilité : 2.0.0 Retourne la géométrie polygonale qui délimite chaque pixel de la bande raster, avec la valeur et les coordonnées raster X et Y de chaque pixel.
— ST_Project - Disponibilité : 2.0.0 Renvoie un point projeté à partir d'un point de départ en fonction d'une distance et d'un azimut.
— ST_Quantile - Disponibilité : 2.0.0 Calcule les quantiles d'un raster ou d'une couverture raster, dans le contexte de l'échantillon ou de la population. Ainsi, une valeur peut être examinée pour se situer au percentile de 25%, 50% ou 75% du raster.

— ST_Range4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule la plage de valeurs des pixels dans un voisinage.
— ST_Reclass - Disponibilité : 2.0.0 Crée un nouveau raster composé de types de bandes reclassifiés par rapport à l'original. La bande n est la bande à modifier. Si nband n'est pas spécifié, la bande 1 est utilisée. Toutes les autres bandes sont retournées inchangées. Cas d'utilisation : convertir une bande 16BUI en 8BUI et ainsi de suite pour un rendu plus simple en tant que formats visualisables.
— ST_RelateMatch - Disponibilité : 2.0.0 Teste si une matrice d'intersection DE-9IM correspond à un modèle de matrice d'intersection
— ST_RemEdgeModFace - Disponibilité : 2.0 Supprime une arête et, si l'arête sépare deux faces, supprime une face et modifie l'autre face pour couvrir l'espace des deux.
— ST_RemEdgeNewFace - Disponibilité : 2.0 Enlève une arête et, si l'arête enlevée séparait deux faces, supprime les faces originales et les remplace par une nouvelle face.
— ST_Resample - Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+ Rééchantillonne un raster, en utilisant l'algorithme spécifié, les nouvelles dimensions, un coin arbitraire de la grille et un ensemble de paramètres de géo-référencement définis ou empruntés à un autre raster.
— ST_Rescale - Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+ Rééchantillonne un raster en ajustant juste son échelle (ou la taille des pixels). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline), Lanczos, Max ou Min. La valeur par défaut est NearestNeighbor.
— ST_Reskew - Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+ Rééchantillonne un raster en ajustant simplement son obliquité (skew, ou paramètre de rotation). Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.
— ST_SameAlignment - Disponibilité : 2.0.0 Retourne true si les rasters ont les mêmes skew, scale, spatial ref, et offset (les pixels peuvent être placés sur la même grille sans être coupés) et false si ce n'est pas le cas, avec une NOTICE détaillant le problème.
— ST_SetBandIsNoData - Disponibilité : 2.0.0 Définit la valeur du flag isnodata de la bande à TRUE.
— ST_SharedPaths - Disponibilité : 2.0.0 Renvoie une collection contenant les chemins partagés par les deux lignes/multilignes en entrée.
— ST_Slope - Disponibilité : 2.0.0 Retourne la pente (par défaut, en degrés) d'une bande raster d'élévation. Utile pour l'analyse de terrain.
— ST_Snap - Disponibilité : 2.0.0 Accrocher les segments et les sommets de la géométrie d'entrée aux sommets d'une géométrie de référence.
— ST_SnapToGrid - Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+ Rééchantillonne un raster en l'accrochant sur une grille. Les nouvelles valeurs des pixels sont calculées en utilisant l'algorithme de rééchantillonnage NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est Nearest-Neighbor.
— ST_Split - Disponibilité : 2.0.0 nécessite GEOS Renvoie une collection de géométries créées en divisant une géométrie par une autre géométrie.
— ST_StdDev4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule l'écart type des valeurs des pixels dans un voisinage.
— ST_Sum4ma - Disponibilité : 2.0.0 Fonction de traitement des données raster qui calcule la somme de toutes les valeurs de pixels dans un voisinage.
— ST_SummaryStats - Disponibilité : 2.0.0 Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed if no band is specified.
— ST_Transform - Disponibilité : 2.0.0 Nécessite GDAL 1.6.1+ Reprojette un raster depuis un système de référence spatial vers un autre, en utilisant l'algorithme de rééchantillonnage spécifié. Les algorithmes possibles sont NearestNeighbor (plus proche voisin), Bilinear (Bilinéaire), Cubic (Cubique), CubicSpline (Cubique Spline) ou Lanczos. La valeur par défaut est NearestNeighbor.
— ST_UnaryUnion - Disponibilité : 2.0.0 Calcule l'union des composantes d'une seule géométrie.
— ST_Union - Disponibilité : 2.0.0 Retourne l'union d'un ensemble de tuiles raster, en un seul raster composé de 1 ou plusieurs bandes.
— ST_ValueCount - Disponibilité : 2.0.0 Retourne un ensemble d'enregistrements contenant une valeur de pixels et le nombre de pixels ayant cette valeur dans la bande du raster spécifié (ou de la couverture raster). Si aucune bande n'est spécifiée, la

bande 1 est utilisée. Par défaut, les pixels de valeur nodata ne sont pas comptés, et toutes les autres valeurs sont retournées, avec leur valeur arrondies à l'entier le plus proche.

— TopoElementArray_Agg - Disponibilité : 2.0.0 Renvoie un topoelementarray pour un ensemble de tableaux de type, element_id (topoelements).

— TopoGeo_AddLineString - Disponibilité : 2.0.0 Ajoute une ligne à une topologie existante en utilisant une tolérance et en divisant éventuellement les arêtes/faces existantes.

— TopoGeo_AddPoint - Disponibilité : 2.0.0 Ajoute un point à une topologie existante en utilisant une tolérance et en divisant éventuellement une arête existante.

— TopoGeo_AddPolygon - Disponibilité : 2.0.0 Ajoute un polygone à une topologie existante en utilisant une tolérance et en divisant éventuellement les arêtes/faces existantes. Renvoie les identifiants des faces.

— TopologySummary - Disponibilité : 2.0.0 Prend un nom de topologie et fournit des totaux récapitulatifs des types d'objets dans la topologie.

— toTopoGeom - Disponibilité : 2.0 Convertit une géométrie simple en une géométrie topographique.

— ~ - Disponibilité : 2.0.0 Retourne TRUE si la boîte englobante de A contient celle de B. Utilise des boîtes englobantes en double précision.

— ~= - Disponibilité : 2.0.0 Renvoie TRUE si la boîte de délimitation de A est la même que celle de B.

Fonctions améliorées dans PostGIS 2.0

— && - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Renvoi VRAI si la boite englobante 2D de A intersecte la boite englobante 2D de B.

— AddGeometryColumn - Amélioration : 2.0.0 introduction du paramètre use_typmod. Le comportement par défaut est de créer une colonne géométrique avec modificateur de type au lieu de contraintes sur la colonne. Ajoute une colonne de géométrie à une table existante.

— Box2D - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie une BOX2D représentant l'étendue 2D d'une géométrie.

— Box3D - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie une BOX3D représentant l'étendue 3D d'une géométrie.

— CreateTopology - Enhanced: 2.0 added support for hasZ. Crée un nouveau schéma topologique et l'enregistre dans la table topology.topology.

— GeometryType - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie le type d'une géométrie sous forme de texte.

— Populate_Geometry_Columns - Amélioration : 2.0.0 L'argument optionnel use_typmod a été introduit pour controler si les colonnes sont créés avec des modificateurs de type ou des contraintes de vérification. Assure que les colonnes géométriques sont définies avec des modificateurs de type ou qu'elles sont soumises à des contraintes spatiales appropriées.

— ST_3DExtent - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.

— ST_Affine - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Appliquer une transformation affine 3D à une géométrie.

— ST_Area - Amélioration : 2.0.0 - la prise en charge des surfaces polyédriques 2D a été introduite. Renvoie l'aire d'une géométrie polygonale.

— ST_AsBinary - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.

— ST_AsBinary - Amélioration : 2.0.0 le support pour des dimensions de coordonnées plus élevées a été introduit. Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.

— ST_AsBinary - Amélioration : 2.0.0 le support pour spécifier endian avec geography a été introduit. Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.

— ST_AsEWKB - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie la représentation Extended Well-Known Binary (EWKB) de la géométrie avec les métadonnées SRID.

— ST_AsEWKT - Amélioration : la version 2.0.0 a introduit la prise en charge de la geography, des surfaces polyédriques, des triangles et des TIN. Renvoie la représentation Well-Known Text (WKT) de la géométrie avec les métadonnées SRID.

— ST_AsGML - Amélioration : la prise en charge du préfixe 2.0.0 a été introduite. L'option 4 pour GML3 a été introduite pour permettre l'utilisation de LineString au lieu de Curve tag pour les lignes. La prise en charge GML3 des surfaces polyédriques et des TINS a été introduite. L'option 32 a été introduite pour produire la boîte. Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.

— ST_AsKML - Amélioré : 2.0.0 - Ajout d'un préfixe namespace, utilisation d'arguments par défaut et d'arguments nommés Renvoyer la géométrie sous forme d'élément KML.
— ST_Azimuth - Amélioration : la prise en charge du type geography a été introduite dans la version 2.0.0. Renvoie l'azimut basé sur le nord d'une ligne entre deux points.
— ST_Dimension - Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Ne renvoie plus une exception si une GEOMETRY EMPTY est passée. Renvoie la dimension topologique d'une géométrie.
— ST_Dump - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie un ensemble de lignes geometry_dump pour les composants d'une géométrie.
— ST_DumpPoints - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_Expand - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie une boîte de délimitation développée à partir d'une autre boîte de délimitation ou d'une géométrie.
— ST_Extent - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Fonction agrégée qui renvoie la boîte de délimitation des géométries.
— ST_Force2D - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Forcer les géométries à passer en "mode bidimensionnel".
— ST_Force3D - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Force les géométries en mode XYZ. Il s'agit d'un alias de ST_Force3DZ.
— ST_Force3DZ - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Forcer les géométries en mode XYZ.
— ST_ForceCollection - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Convertir la géométrie en une GEOMETRYCOLLECTION.
— ST_ForceRHR - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Force l'orientation des sommets d'un polygone à suivre la règle de la main droite.
— ST_GMLToSQL - Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Returns a geometry from GML. Alias for ST_GeomFromGML.
— ST_GMLToSQL - Amélioration : 2.0.0 paramètre optionnel de srid par défaut ajouté. Returns a geometry from GML. Alias for ST_GeomFromGML.
— ST_GeomFromEWKB - Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Retourne un objet ST_Geometry à partir de sa représentation binaire étendue (Extended Well-Known Binary representation, EWKB).
— ST_GeomFromEWKT - Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Retourne un objet ST_Geometry à partir de sa représentation textuelle étendue (Extended Well-Known Text representation, EWKT).
— ST_GeomFromGML - Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry
— ST_GeomFromGML - Amélioration : 2.0.0 paramètre optionnel de srid par défaut ajouté. Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry
— ST_GeometryN - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie un élément d'une collection de géométries.
— ST_GeometryType - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Renvoie le type SQL-MM d'une géométrie sous forme de texte.
— ST_IsClosed - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Teste si les points de départ et d'arrivée d'une LineString coïncident. Pour une PolyhedralSurface, teste si elle est fermée (volumétrique).
— ST_MakeEnvelope - Amélioré : 2.0 : La possibilité de spécifier une enveloppe sans spécifier un SRID a été introduite. Crée un polygone rectangulaire à partir des coordonnées minimales et maximales.
— ST_MakeValid - Amélioré : 2.0.1, améliorations de la vitesse Tente de rendre valide une géométrie invalide sans perdre de sommets.
— ST_NPoints - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Retourne le nombre de points (vertex) d'un objet géométrique.
— ST_NumGeometries - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Renvoie le nombre d'éléments dans une collection de géométrie.
— ST_Relate - Amélioration : 2.0.0 - ajout de la prise en charge de la spécification de boundary node rule. Teste si deux géométries ont une relation topologique correspondant à un modèle de matrice d'intersection, ou calcule leur matrice d'intersection
— ST_Rotate - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Fait pivoter une géométrie autour d'un point d'origine.

— ST_Rotate - Amélioration : 2.0.0 des paramètres supplémentaires ont été ajoutés pour spécifier l'origine de la rotation. Fait pivoter une géométrie autour d'un point d'origine.
— ST_RotateX - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Fait pivoter une géométrie autour de l'axe X.
— ST_RotateY - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Fait pivoter une géométrie autour de l'axe Y.
— ST_RotateZ - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Fait pivoter une géométrie autour de l'axe Z.
— ST_Scale - Amélioration : 2.0.0 introduction du support TIN, Triangles et surfaces polyédriques. Met à l'échelle une géométrie en fonction de facteurs donnés.
— ST_ShiftLongitude - Amélioration : 2.0.0 introduction du support TIN et surfaces polyédriques. Décale les coordonnées de longitude d'une géométrie entre -180..180 et 0..360.
— ST_Summary - Amélioré : la version 2.0.0 a ajouté la prise en charge du type geography Renvoie un résumé textuel du contenu d'une géométrie.
— ST_Transform - Amélioration : 2.0.0 introduction du support des surfaces polyédriques. Renvoie une nouvelle géométrie dont les coordonnées ont été transformées dans un système de référence spatial différent.
— ST_Value - Amélioration : 2.0.0 ajout du paramètre optionnel exclude_nodata_value. Retourne la valeur d'une bande raster spécifiée au pixel donné par columnx, rowy, ou à un point géométrique spécifié. Le numéro de bande démarre à 1, et la bande 1 est utilisée si non spécifié. Si exclude_nodata_value vaut false, tous les pixels y compris ceux ayant la valeur nodata sont considérés comme intersectés et leur valeur sera retournée. Si exclude_nodata_value n'est pas spécifié, la valeur est lue depuis les méta-données du raster.
— ValidateTopology - Amélioration : 2.0.0 détection plus efficace des croisements d'arêtes et correction des faux positifs qui existaient dans les versions précédentes. Renvoie un ensemble d'objets validatetopology_returntype détaillant les problèmes liés à la topologie.

Fonctions modifiées dans PostGIS 2.0

— AddGeometryColumn - Changement : 2.0.0 Cette fonction ne met plus à jour geometry_columns maintenant que geometry_columns est une vue basée sur le catalogue système. Par défaut, elle ne créée plus de contraintes mais utilise le modificateur de type de PostgreSQL. Ainsi, par exemple, créer une colonne de type POINT WGS84 est désormais équivalent à : ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326) ; Ajoute une colonne de géométrie à une table existante.
— AddGeometryColumn - Changement : 2.0.0 Si l'ancien mécanisme basé sur les contraintes est nécessaire, utiliser le paramètre use_typmod avec la valeur false. Ajoute une colonne de géométrie à une table existante.
— AddGeometryColumn - Changement : 2.0.0 Les vues ne peuvent plus être enregistrées dans geometry_columns. Cependant, les vues construites à partir de tables contenant des géométries définies avec le modificateur de type et n'utilisant pas de fonctions d'encapsulation seront enregistrées dans la vue geometry_columns car elles héritent du mécanisme des tables dont elles sont issues. Les vues utilisant des fonctions renvoyant d'autres géométries doivent être transtypées vers des géométries avec modificateur de type pour pouvoir être correctement référencées dans la vue geometry_columns. Cf. . Ajoute une colonne de géométrie à une table existante.
— Box3D - Changement : 2.0.0 Dans les versions antérieures à la version 2.0, utilisait le type box2d au lieu de box3d. Comme box2d est un type obsolète, il a été remplacé par box3d. Retourne la représentation 3d de la boîte englobante du raster.
— DropGeometryColumn - Changed: 2.0.0. Retained for backward compatibility after geometry_columns became a view of the system catalogs. Supprime une colonne géométrique d'une table spatiale.
— DropGeometryTable - Changed: 2.0.0. Retained for backward compatibility after geometry_columns became a view of the system catalogs. Supprime une table et toutes ces références dans geometry_columns.
— Populate_Geometry_Columns - Changement : 2.0.0 Par défaut, utilise les modificateurs de type au lieu de contraintes de vérification pour contraindre les types géométriques. Le comportement basé sur les contraintes peut être activé en mettant le nouveau paramètre use_typmod à false. Assure que les colonnes géométriques sont définies avec des modificateurs de type ou qu'elles sont soumises à des contraintes spatiales appropriées.
— ST_3DExtent - Modifié : 2.0.0 Dans les versions précédentes, cette fonction était appelée ST_Extent3D Fonction d'agrégation qui renvoie la boîte de délimitation 3D des géométries.
— ST_3DLength - Modifié : 2.0.0 Dans les versions précédentes, cette fonction était appelée ST_Length3D Renvoie la longueur 3D d'une géométrie linéaire.
— ST_3DMakeBox - Modifié : 2.0.0 Dans les versions précédentes, cette fonction s'appelait ST_MakeBox3D Crée un BOX3D défini par deux géométries de points 3D.

— ST_3DPerimeter - Modifié : 2.0.0 Dans les versions antérieures, il s'appelait ST_Perimeter3D Renvoie le périmètre 3D d'une géométrie polygonale.
— ST_AsBinary - Changed: 2.0.0 Inputs to this function can not be unknown -- must be geometry. Constructs such as ST_AsBinary('PO 2)') are no longer valid and you will get an n ST_AsBinary(unknown) is not unique error. Code like that needs to be changed to ST_AsBinary('POINT(1 2)'::geometry);. If that is not possible, then install legacy.sql. Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsGML - Modifié : 2.0.0 utiliser les args nommés par défaut Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsGeoJSON - Modifié : 2.0.0 supporte les args par défaut et les args nommés. Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_AsSVG - Modifié : 2.0.0 pour utiliser les args par défaut et supporter les args nommés Renvoie les données de chemin SVG pour une géométrie.
— ST_EndPoint - Modifié : 2.0.0 ne fonctionne plus avec les MultiLineStrings à géométrie unique. Dans les anciennes versions de PostGIS, une MultiLineString à géométrie unique fonctionnait avec cette fonction et renvoyait le point final. Dans la version 2.0.0, elle renvoie NULL comme toute autre MultiLineString. L'ancien comportement était une fonctionnalité non documentée, mais les personnes qui supposaient que leurs données étaient stockées en tant que LINESTRING peuvent voir ces dernières retourner NULL dans la version 2.0.0. Returns the last point of a LineString, CircularLineString, or NURBSCurve.
— ST_GDALDrivers - Changement : 2.0.6, 2.1.3 - aucun driver n'est activé par défaut, sauf si la variable GUC ou d'environnement gdal_enabled_drivers est définie. Retourne la liste des formats raster supportés par PostGIS via GDAL. Seuls les formats avec can_write=True peuvent être utilisés par ST_AsGDALRaster
— ST_GeomFromText - Changement : 2.0.0 dans les version précédentes de PostGIS ST_GeomFromText('GEOMETRYCOL etait autorisé. C'est désormais interdit dans PostGIS 2.0.0 pour respecter la norme SQL/MM. La forme privilégiée désormais est : ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') Retourne un objet ST_Geometry à partir de sa représentation textuelle Well-Known Text (WKT).
— ST_GeometryN - Changement : 2.0.0. Les versions antérieures renvoient NULL pour les géometries simples (un seul objet). Renvoie désormais la géométrie pour le cas ST_GeometryN(..,1). Renvoie un élément d'une collection de géométries.
— ST_IsEmpty - Modifié : 2.0.0 Dans les versions précédentes de PostGIS, ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY était autorisé. Ceci est maintenant illégal dans PostGIS 2.0.0 pour mieux se conformer aux normes SQL/MM Teste si une géométrie est vide.
— ST_Length - Modifié : 2.0.0 Rupture -- dans les versions précédentes, appliquer ceci à un MULTI/POLYGONE de type geography donnait le périmètre du POLYGONE/MULTIPOLYGONE. Dans la version 2.0.0, cette fonction a été modifiée pour retourner 0 afin d'être en ligne avec le comportement de la géométrie. Veuillez utiliser ST_Perimeter si vous souhaitez obtenir le périmètre d'un polygone Renvoie la longueur 2D d'une géométrie linéaire.
— ST_LocateAlong - Modifié : 2.0.0 dans les versions précédentes, cette fonction était appelée ST_Locate_Along_Measure. Renvoie le(s) point(s) d'une géométrie qui correspond(ent) à une valeur de mesure.
— ST_LocateBetween - Modifié : 2.0.0 - dans les versions précédentes, cette fonction s'appelait ST_Locate_Between_Measures. Renvoie les parties d'une géométrie qui correspondent à un intervalle de mesure.
— ST_ModEdgeSplit - Modifié : 2.0 - Dans les versions antérieures, cette fonction était mal nommée ST_ModEdgesSplit Fractionner une arête en créant un nouveau nœud le long d'une arête existante, en modifiant l'arête d'origine et en ajoutant une nouvelle arête.
— ST_NumGeometries - Modifié : 2.0.0 Dans les versions précédentes, cette fonction renvoyait NULL si la géométrie n'était pas de type collection/MULTI. 2.0.0+ renvoie maintenant 1 pour les géométries simples, par exemple POLYGONE, LINESTRING, POINT. Renvoie le nombre d'éléments dans une collection de géométrie.
— ST_NumInteriorRings - Modifié : 2.0.0 - dans les versions antérieures, il permettait de passer un MULTIPOLYGONE, renvoyant le nombre d'anneaux intérieurs du premier POLYGONE. Renvoie le nombre d'anneaux intérieurs (trous) d'un polygone.
— ST_PointN - Modifié : la version 2.0.0 ne fonctionne plus avec les multi-lignes à géométrie unique. Dans les anciennes versions de PostGIS, une multi-ligne d'une seule ligne fonctionnait parfaitement avec cette fonction et renvoyait le point de départ. Dans la version 2.0.0, elle renvoie simplement NULL comme n'importe quelle autre multi-ligne. Renvoie le Nième point de la première LineString ou LineString circulaire d'une géométrie.
— ST_ScaleX - Changement : 2.0.0. Dans les versions WKTRaster, cette fonction était appelée ST_PixelSizeX. Renvoie la composante X de la largeur du pixel dans l'unité du système de référence spatial.
— ST_ScaleY - Changement : 2.0.0. Dans les versions WKTRaster, cette fonction était appelée ST_PixelSizeY. Renvoie la composante Y de la hauteur du pixel dans l'unité du système de référence spatial.

— ST_SetScale - Changement : 2.0.0 Dans les versions WKTRaster, cette fonction était appelée ST_SetPixelSize. Définit la résolution des pixels en X et Y en unité du système de référence spatial : nombre d'unités/pixel en largeur/hauteur.
— ST_StartPoint - Modifié : 2.0.0 ne fonctionne plus avec les MultiLineStrings à géométrie unique. Dans les anciennes versions de PostGIS, une MultiLineString à géométrie unique fonctionnait sans problème avec cette fonction et renvoyait le point de départ. Dans la version 2.0.0, elle renvoie simplement NULL comme toute autre MultiLineString. L'ancien comportement était une fonctionnalité non documentée, mais les personnes qui supposaient que leurs données étaient stockées en tant que LINESTRING peuvent voir ces données retourner NULL dans la version 2.0.0. Returns the first point of a LineString, CircularLineString, or NURBSCurve.

13.12.15 Fonctions PostGIS nouvelles ou améliorées en 1.5

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 1.5

— && - Disponibilité : 1.5.0 le support de la géographie a été introduit. Renvoi VRAI si la boite englobante 2D de A intersecte la boite englobante 2D de B.
— PostGIS_LibXML_Version - Disponibilité : 1.5 Renvoie le numéro de version de la librairie libxml2.
— ST_AddMeasure - Disponibilité : 1.5.0 Interpole les mesures le long d'une géométrie linéaire.
— ST_AsBinary - Disponibilité : 1.5.0 Le support de la géographie a été introduit. Renvoie la représentation OGC/ISO Well-Known Binary (WKB) de la géométrie/geography sans les métadonnées SRID.
— ST_AsGML - Disponibilité : 1.5.0 Le support de la géographie a été introduit. Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsGeoJSON - Disponibilité : 1.5.0 Le support de la géographie a été introduit. Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_AsText - Disponibilité : 1.5 - le support de la geography a été introduit. Renvoie la représentation Well-Known Text (WKT) de la géométrie/geography sans métadonnées SRID.
— ST_Buffer - Disponibilité : 1.5 - ST_Buffer a été amélioré pour prendre en charge différents types de terminaisons et de jointures. Ceux-ci sont utiles, par exemple, pour convertir les lignes de route en routes polygonales avec des bords plats ou carrés au lieu de bords arrondis. Un petit wrapper pour la geography a été ajouté. Calcule une géométrie couvrant tous les points situés à une distance donnée d'une géométrie.
— ST_ClosestPoint - Disponibilité : 1.5.0 Renvoie le point 2D sur g1 qui est le plus proche de g2. Il s'agit du premier point de la ligne la plus courte d'une géométrie à l'autre.
— ST_CollectionExtract - Disponibilité : 1.5.0 Pour une collection de géométries spécifiée, renvoie une multi-géométrie contenant uniquement des éléments d'un type spécifié.
— ST_Covers - Disponibilité : 1.5 - le support de la geography a été introduit. Tests si chaque point de B est situé dans A
— ST_DFullyWithin - Disponibilité : 1.5.0 Teste si une géométrie se trouve entièrement à une distance d'une autre géométrie
— ST_DWithin - Disponibilité : la prise en charge du type geography a été introduite dans la version 1.5.0 Teste si deux géométries se trouvent à une distance donnée
— ST_Distance - Disponibilité : 1.5.0 La prise en charge du type geography a été introduite dans la version 1.5. Amélioration de la vitesse pour les géométries planaires afin de mieux gérer les géométries de grande taille ou à nombreux sommets Renvoie la distance entre deux valeurs de geometry ou geography.
— ST_DistanceSphere - Disponibilité : 1.5 - la prise en charge d'autres types de géométrie que les points a été introduite. Les versions précédentes ne fonctionnaient qu'avec des points. Renvoie la distance minimale en mètres entre deux géométries lon/lat en utilisant un modèle de terre sphérique.
— ST_DistanceSpheroid - Disponibilité : 1.5 - la prise en charge d'autres types de géométrie que les points a été introduite. Les versions précédentes ne fonctionnaient qu'avec des points. Renvoie la distance minimale entre deux géométries lon/lat en utilisant un modèle de terre sphéroïdale.
— ST_DumpPoints - Disponibilité : 1.5.0 Renvoie un ensemble de lignes geometry_dump pour les coordonnées dans une géométrie.
— ST_Envelope - Disponibilité : 1.5.0 changement pour renvoyer un type double précision à la place de float4 Renvoie une géométrie représentant la boîte de délimitation d'une géométrie.
— ST_Expand - Disponibilité : 1.5.0 comportement modifié pour afficher les coordonnées en double précision au lieu des coordonnées float4. Renvoie une boîte de délimitation développée à partir d'une autre boîte de délimitation ou d'une géométrie.

— ST_GMLToSQL - Disponibilité : 1.5, nécessite libxml2 1.6+ Returns a geometry from GML. Alias for ST_GeomFromGML
— ST_GeomFromGML - Disponibilité : 1.5, nécessite libxml2 1.6+ Prend en paramètre une représentation GML d'une géométrie et renvoie un objet PostGIS de type geometry
— ST_GeomFromKML - Disponibilité : 1.5, nécessite libxml2 2.6+ Prend en entrée une géométrie au format KML et renvoie un objet Postgis de type geometry
— ST_HausdorffDistance - Disponibilité : 1.5.0 Renvoie la distance de Hausdorff entre deux géométries.
— ST_Intersection - Disponibilité : La version 1.5 a introduit la prise en charge du type de données geography. Calcule une géométrie représentant la partie partagée des géométries A et B.
— ST_Intersects - Disponibilité : la version 1.5 a introduit la prise en charge du type geography. Teste si deux géométries se croisent (elles ont au moins un point en commun)
— ST_Length - Disponibilité : 1.5.0 La prise en charge du type geography a été introduite dans la version 1.5. Renvoie la longueur 2D d'une géométrie linéaire.
— ST_LongestLine - Disponibilité : 1.5.0 Renvoie la ligne 2D la plus longue entre deux géométries.
— ST_MakeEnvelope - Disponibilité : 1.5 Crée un polygone rectangulaire à partir des coordonnées minimales et maximales.
— ST_MaxDistance - Disponibilité : 1.5.0 Renvoie la plus grande distance 2D entre deux géométries en unités projetées.
— ST_ShortestLine - Disponibilité : 1.5.0 Renvoie la ligne 2D la plus courte entre deux géométries
— ~= - Disponibilité : 1.5.0 comportement changé Renvoie TRUE si la boîte de délimitation de A est la même que celle de B.

13.12.16 Fonctions PostGIS nouvelles ou améliorées en 1.4

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 1.4
— Populate_Geometry_Columns - Disponibilité : 1.4.0 Assure que les colonnes géométriques sont définies avec des modificateurs de type ou qu'elles sont soumises à des contraintes spatiales appropriées.
— ST_Collect - Disponibilité : 1.4.0 - ST_Collect(geomarray) a été introduit. ST_Collect a été amélioré pour gérer plus de géométries plus rapidement. Crée une géométrie GeometryCollection ou Multi à partir d'un ensemble de géométries.
— ST_ContainsProperly - Disponibilité : 1.4.0 Tests si chaque point de B se trouve à l'intérieur de A
— ST_GeoHash - Disponibilité : 1.4.0 Retourne une représentation GeoHash de la géométrie.
— ST_IsValidReason - Disponibilité : 1.4 Renvoie un texte indiquant si une géométrie est valide, ou la raison de son invalidité.
— ST_LineCrossingDirection - Disponibilité : 1.4 Renvoie un nombre indiquant le comportement de croisement de deux LineStrings
— ST_LocateBetweenElevations - Disponibilité : 1.4.0 Renvoie les parties d'une géométrie qui se trouvent dans un intervalle d'élévation (Z).
— ST_MakeLine - Disponibilité : 1.4.0 - création de ST_MakeLine(geomarray). L'agrégat spatial ST_MakeLine amélioré pour supporter plus de points plus rapidement. Crée une LineString à partir de géométries Point, MultiPoint ou LineString.
— ST_MinimumBoundingCircle - Disponibilité : 1.4.0 Retourne le plus petit cercle polygonal qui contient une géométrie.
— ST_Union - Disponibilité : 1.4.0 - ST_Union a été amélioré. ST_Union(geomarray) a été introduit ainsi qu'une collecte d'agrégats plus rapide dans PostgreSQL. Calcule une géométrie représentant l'union des ensembles de points des géométries d'entrée.

13.12.17 Fonctions PostGIS nouvelles ou améliorées en 1.3

Les fonctions indiquées ci-dessous sont des fonctions PostGIS qui ont été ajoutées ou améliorées.

Fonctions nouvelles dans PostGIS 1.3
— ST_AsGML - Disponibilité : 1.3.2 Renvoyer la géométrie en tant qu'élément GML version 2 ou 3.
— ST_AsGeoJSON - Disponibilité : 1.3.4 Renvoyer une géométrie ou un élément au format GeoJSON.
— ST_CurveToLine - Disponibilité : 1.3.0 Convertit une géométrie contenant des courbes en une géométrie linéaire.
— ST_LineToCurve - Disponibilité : 1.3.0 Convertit une géométrie linéaire en une géométrie courbe.
— ST_SimplifyPreserveTopology - Disponibilité : 1.3.3 Renvoie une représentation simplifiée et valide d'une géométrie, en utilisant l'algorithme de Douglas-Peucker.

Chapitre 14

Rapporter un problème

14.1 Rapporter un problème logiciel

Rapporter un problème est effectivement fondamental afin d'aider le développement de PostGIS. Un rapport de bug efficace permet à l'équipe des développeurs de PostGIS de reproduire le problème. Le rapport est donc pertinent s'il contient le script qui le déclenche et toutes les informations à propos de l'environnement dans lequel il a été détecté. De bonnes informations peuvent être obtenus par l'exécution de `SELECT postgis_full_version()` [pour postgis] et `SELECT version()` [pour postgresql].

Si vous n'utilisez pas la dernière version, il vaut la peine de jeter un coup d'œil à son journal des changements de la version d'abord, pour savoir si votre bug a déjà été corrigé.

L'utilisation du PostGIS bug tracker permettra de s'assurer que vos rapports ne sont pas rejetés et vous tiendra informé du processus de traitement. Avant de signaler un nouveau bug, veuillez interroger la base de données pour voir s'il s'agit d'un bug connu, et si c'est le cas, veuillez ajouter toute nouvelle information que vous possédez à son sujet.

Vous pouvez lire l'article de Simon Tatham sur Comment signaler efficacement les bugs avant de déposer un nouveau rapport.

14.2 Signaler les problèmes de documentation

La documentation doit refléter fidèlement les caractéristiques et le comportement du logiciel. Si ce n'est pas le cas, cela peut être dû à un bug du logiciel ou à une erreur ou une lacune dans la documentation.

Les problèmes de documentation peuvent également être signalés au PostGIS bug tracker.

Si votre révision est triviale, décrivez-la simplement dans une nouvelle question du bug tracker, en précisant son emplacement dans la documentation.

Si vos modifications sont plus importantes, il est préférable d'utiliser un correctif. Il s'agit d'un processus en quatre étapes sous Unix (en supposant que vous ayez déjà installé git) :

Before editing or building the manual, read the repository's documentation development guide. It covers the executable example, generated figure, validation, HTML, and PDF workflows used by the manual.

1. Clonez le dépôt git de PostGIS. Sous Unix, tapez :

   git clone https://git.osgeo.org/gitea/postgis/postgis.git

   Cela sera stocké dans le répertoire postgis
2. Faites vos changements sur la documentation avec votre éditeur de texte favori. Sur Unix, tapez (pour exemple) :

   vim doc/postgis.xml

   Remarquez que la documentation est écrite avec DocBook XML au lieu de HTML, donc si vous ne vous sentez pas familier avec cela, vous devriez suivre les exemples dans la suite de la documentation.

3. Faites un correctif contenant les différences avec une copie de la documentation principale. Sur Unix, tapez :

   git diff doc/postgis.xml > doc.patch
4. Ajoutez le correctif à un nouveau ticket dans le système de suivi de bogues.

Chapitre 15

Credits and Acknowledgments

PostGIS is built and sustained by a worldwide community. This chapter records the people, organizations, funders, and upstream projects that have contributed code, documentation, infrastructure, testing, support, and funding.

15.1 Project Steering Committee

The PostGIS Project Steering Committee (PSC) coordinates the general direction, release cycles, documentation, and outreach efforts for the PostGIS project. In addition the PSC provides general user support, accepts and approves patches from the general PostGIS community and votes on miscellaneous issues involving PostGIS such as developer commit access, new PSC members or significant API changes.

Regina Obe CI and website maintenance, Windows production and experimental builds, documentation, alignment of PostGIS with PostgreSQL releases, X3D support, management functions.

Darafei Praliaskouski Index improvements, bug fixing and geometry/geography function improvements, SFCGAL, raster, GitHub curation, and ci maintenance.

Paul Ramsey (Chair) Co-founder of PostGIS project. General bug fixing, geography support, geography and geometry index support (2D, 3D, nD index and anything spatial index), underlying geometry internal structures, GEOS functionality integration and alignment with GEOS releases, alignment of PostGIS with PostgreSQL releases, loader/dumper, and Shapefile GUI loader.

Sandro Santilli Bug fixes and maintenance, ci maintenance, git mirror management, management functions, integration of new GEOS functionality and alignment with GEOS releases, topology support, and raster framework and low level API functions.

15.2 Core Contributors Present

Nicklas Avén Distance function enhancements (including 3D distance and relationship functions) and additions, Tiny WKB (TWKB) output format and general user support

Loïc Bartoletti SFCGAL enhancements and maintenance and ci support

Dan Baston Geometry clustering function additions, other geometry algorithm enhancements, GEOS enhancements and general user support

Martin Davis GEOS enhancements and documentation

Björn Harrtell MapBox Vector Tile, GeoBuf, and Flatgeobuf functions. Gitea testing and GitLab experimentation.

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

Laurențiu Nicola Bug fixes, maintenance, stability, and performance

15.3 Core Contributors Past

Raúl Marín Rodríguez Prior PSC Member, MVT support, Bug fixing, Performance and stability improvements, GitHub curation, alignment of PostGIS with PostgreSQL releases

Bborie Park Prior PSC Member. Raster development, integration with GDAL, raster loader, user support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

Mark Cave-Ayland Prior PSC Member. Coordinated bug fixing and maintenance effort, spatial index selectivity and binding, loader/dumper, and Shapefile GUI Loader, integration of new and new function enhancements.

Jorge Arévalo Raster development, GDAL driver support, loader

Olivier Courtin (Emeritus) Input/output XML (KML,GML)/GeoJSON functions, 3D support and bug fixes.

Chris Hodgson Prior PSC Member. General development, site and buildbot maintenance, OSGeo incubation management

Mateusz Loskot CMake support for PostGIS, built original raster loader in python and low level raster API functions

Kevin Neufeld Prior PSC Member. Documentation and documentation support tools, buildbot maintenance, advanced user support on PostGIS newsgroup, and PostGIS maintenance function enhancements.

Dave Blasby The original developer/Co-founder of PostGIS. Dave wrote the server side objects, index bindings, and many of the server side analytical functions.

Jeff Lounsbury Original development of the Shapefile loader/dumper.

Mark Leslie Ongoing maintenance and development of core functions. Enhanced curve support. Shapefile GUI loader.

Pierre Racine Architect of PostGIS raster implementation. Raster overall architecture, prototyping, programming support

David Zwarg Raster development (mostly map algebra analytic functions)

15.4 PostGIS Raster Origins

PostGIS raster discussions date to 2002 and included early PGCHIP experiments. Xing Lin developed the **geotiff2pgraster** PGRaster prototype during Google Summer of Code 2007. The WKT Raster project began in 2008, Jorge Arévalo developed the first GDAL driver prototype during Google Summer of Code 2009, and the 2010 FOSS4G code sprint agreed to make WKT Raster the PostGIS raster type. Raster support shipped as part of PostGIS 2.0 in 2012.

The early implementation team included Pierre Racine, Mateusz Loskot, Sandro Santilli, Jorge Arévalo, Bborie Park, David Zwarg, and Regina Obe. Their work established the raster type, loader, GDAL integration, raster/vector operations, map algebra, documentation, and testing that became the current raster subsystem.

Other recorded contributions include Jeff Adams's early sprint work, Qing Liu's raster distance analysis, Maurício Carvalho Mathias de Paulo's QGIS raster plugin and related paper with Lubia Vinhas, and Ignacio Brodin's gvSIG raster plugin and presentation with Jorge Sanz. Bryce L. Nordgren created early raster coverage and seamless-architecture diagrams. Pierre Racine wrote the original raster/vector tutorial and created its walkthrough and early API diagrams.

The historical project planning record credits Steve Cumming and Université Laval with funding Pierre Racine's work from June 2008 through July 2012; Martin Daly and Cadcorp with funding about six months of Mateusz Loskot's work through June 2009; Tyler Erickson and Michigan Tech Research Institute with a $1,200 USD contribution; Miguel Lizondo and Deimos Space with support for about two years of Jorge Arévalo's work through June 2011; and the Center for Vectorborne Diseases at UC Davis with funding Bborie Park's work from March 2010 through July 2012.

15.5 Other Contributors

Individual Contributors

@amwps290 (GitHub user)
@kannes (GitHub user: Hannes)
@sunhm89 (GitHub user)
@wewindy (GitHub user: Chen)
Adrien Berchet
Alex Bodnaru
Alex Mayrhofer
Andrea Peri
Andreas Forø Tollefsen
Andreas Neumann
Andreas Schild
Andrew Gierth
Andrew Harvey
Anne Ghisla
Antoine Bajolet
Ariel Mashraki
Arno Renevier
Arthur Bazin
Arthur Lesuisse
Artur Zakirov
Ayo Adesugba
Barbara Phillipot
Baris Ergun
Bas Couwenberg
Ben Jubb
Benjamin Morel
Bernd Reiß
Bernhard Reiter
Bill Mill
Björn Esser
Brian Hamlin
Bruce Rindahl
Bruno Wolff III
Bryce L. Nordgren
Carl Anderson
Carlos López Quintanilla
Charles Karney
Charlie Savage
Chris Mayo
Christian Schroeder
Christoph Berg
Christoph Moench-Tegeder
Dan Faudemer
Dan Levy
Dane Springmeyer
Daniel Nylander
Dapeng Wang
Daryl Herzmann
Dave Fuhry
David G. Johnston
David Garnier
David Skea
David Techer
Dennis Tighe
Denys Kovshun
Devrim Gündüz
Dian M Fay
Dmitry Vasilyev
Edouard Choinière
Eduin Carrillo
Egor Ivkov
Eric Ridge
Esteban Zimanyi
Eugene Antimirov
Florian Weimer
François Bonzon
Frank Warmerdam
George Silva
Gerald Fenoy
Gino Lucrezi
Giovanni Zezza
Grant Hutchins
Greg Troxel
Guillaume Lelarge
Giuseppe Broccolo
Han Wang
Hannes Janetzek
Hans Lemuet
Haribabu Kommi
Harrissou Sant-anna
Havard Tveite
Ignacio Brodin
IIDA Tetsushi
Ingvild Nystuen
Jackie Leng
James Addison
James Marca
Jan Katins
Jan Tojnar
Jason Smith
Jean Felder
Jeff Adams
Jelte Fennema
Jim Jones
Joe Conway
Jonne Savolainen
Jorge Gustavo Rocha
Jorge Sanz
Jose Carlos Martinez Llari
Jostein Leira
Jörg Habenicht
Juan Sebastian Jimenez Aldana
Juergen E. Fischer
Julien Monticolo
Julien Rouhaud
Kashif Rasul
Klaus Foerster
Kris Jurka
Kristian Thy
Lars Aksel Opsahl
Laurențiu Nicola
Laurenz Albe
Lars Roessiger
Lauri Kajan
Leo Hsu
Loic Dachary
Louis Descoteaux
Luca Delucchi
Luca S. Percich
Lucas C. Villa Real
Ludovic Hirlimann
Lubia Vinhas
Maksim Korotkov
Manuel Morillo
Maria Arias de Reyna
Marc Ducobu
Mark Sondheim
Markus Schaber
Maurício Carvalho Mathias de Paulo
Maxim Kochetkov
Maxime Guillaud
Maxime van Noppen
Maxime Schoemans
Megan Ma
Mehmet Ince
Michael Fuhr
Michal Páleník
Mike Toews
Mitchell Henke
Nathan Wagner
Nathaniel Clay
Nicolas Roelandt
Nikita Shulga
Nikolai Berkoff
Norman Barker
Norman Vine
Patricia Tozer
Paul Norman
Paulo Cesar Coronado
Pavel Seleznev
Pawel Ostrowski
Peter Nguyen
Pierre Cardinal
Qing Liu
Rafal Magda
Ralph Mason
Rémi Cura
Richard Greenwood
Robert Coup
Rodrigo Smarzaro
Roger Crew
Ron Mayer
Sam Peters
Sarath Kumar
Sergei Shoulbakov
Sergey Fedoseev
Shinichi Sugiyama
Shoaib Burq
Silvio Grosso
Simon Greener
Stefan Corneliu Petrea
Steffen Macke
Stepan Kuzmin
Stephen Frost
Stephen Woodbridge
Steven Ottens
Svyatoslav Bogdanov
Talha Rizwan
Temuri Doghonadze
Teramoto Ikuhiro
Thomas Muguet
Thomas Petazzoni
Tobias Bussmann
Tom Glancy
Tom van Tilburg
Victor Collod
Vincent Bre
Vincent Mora
Vincent Picavet
Volf Tomáš
Xing Lin
Yoichi Kayama

Corporate Sponsors These are corporate entities that have contributed developer time, hosting, or direct monetary funding to the PostGIS project. In alphabetical order:

- Aiven
- Arrival 3D
- Associazione Italiana per l'Informazione Geografica Libera (GFOSS.it)
- AusVet
- Avencia
- Azavea
- Boundless
- Cadcorp
- Camptocamp
- Carto
- Crunchy Data
- City of Boston (DND)
- City of Helsinki
- Clever Elephant Solutions
- Cooperativa Alveo
- Deimos Space
- Faunalia
- Geographic Data BC
- Google
- HighGo
- Hunter Systems Group
- IITM Pravartak Security Team
- INIA-CSIC
- ISciences, LLC
- Jirotech (formerly LISAsoft)
- Koordinates
- Kontur
- Lidwala Consulting Engineers
- Logical Tracking & Tracing International AG
- Maponics
- Maumap
- Michigan Tech Research Institute
- Natural Resources Canada
- Norwegian Forest and Landscape Institute
- Norwegian Institute of Bioeconomy Research (NIBIO)
- OSGeo
- Oslandia
- Palantir Technologies
- Paragon Corporation
- PlanetScale
- Postgres Pro
- R3 GIS
- Refractions Research
- Regione Toscana - SITA
- Safe Software
- Sirius Corporation plc
- Stadt Uster
- UC Davis Center for Vectorborne Diseases

— Université Laval
— U.S. Census Bureau
— U.S. Department of State (HIU)
— Zonar Systems

Crowd Funding Campaigns Crowd funding campaigns are campaigns we run to get badly wanted features funded that can service a large number of people. Each campaign is specifically focused on a particular feature or set of features. Each sponsor chips in a small fraction of the needed funding and with enough people/organizations contributing, we have the funds to pay for the work that will help many. If you have an idea for a feature you think many others would be willing to co-fund, please post to the PostGIS newsgroup your thoughts and together we can make it happen.

PostGIS has experimented with community fundraising for features in the past. We used PledgeBank and we got two successful campaigns out of it.

postgistopology - 10 plus sponsors each contributed $250 USD to build toTopoGeometry function and beef up topology support in 2.0.0. It happened.

postgis64windows - 20 someodd sponsors each contributed $100 USD to pay for the work needed to work out PostGIS 64-bit issues on windows. It happened.

External Libraries and Platforms PostgreSQL, the extensible database system on which PostGIS is built. PostgreSQL provides the SQL engine, transactions, query planner, extension framework, and index access methods used by PostGIS.

The GEOS geometry operations library.

The PROJ coordinate transformation and geodesic calculation library.

The GDAL Geospatial Data Abstraction Library used to power much of the raster functionality introduced in PostGIS 2. In kind, improvements needed in GDAL to support PostGIS are contributed back to the GDAL project.

LibXML2 for XML-based spatial formats, and JSON-C for GeoJSON input.

The GNU Multiple Precision Arithmetic Library (GMP) for precision-sensitive topology operations.

protobuf-c for Mapbox Vector Tile and Geobuf output.

The iconv API, supplied by the C library or GNU libiconv, for character encoding conversion in the shapefile loader.

SFCGAL for advanced 2D and 3D functions, together with the CGAL algorithms that SFCGAL exposes.

Bundled Libraries and Source Code PostGIS ships selected source from FlatGeobuf and FlatBuffers to support FlatGeobuf input and output.

Wagyu and Mapbox Geometry provide polygon clipping and validation for Mapbox Vector Tile output.

Ryu provides round-trip-safe floating-point formatting, and uthash provides in-process hash tables.

The shapefile loader and dumper include code derived from ShapeLib, originally by Frank Warmerdam and Even Rouault. They also include public-domain `getopt` code from AT&T.

`liblwgeom/lookup3.c` contains the public-domain `lookup3` hash function by Bob Jenkins.

Annexe A

Notes de version

A.1 PostGIS 3.7.0

This version requires GEOS 3.10+, PostgreSQL 14-19beta2, Proj 6.1+, libgmp. To take advantage of all features postgis extension features, GEOS 3.15+ is needed. To take advantage of all postgis_sfcgal extension features SFCGAL 2.3+ is needed.

A.1.1 Changements avec rupture

#5688, [topology] topology building functions now interpret tolerance 0 (old default) to really mean 0 and want -1 (new default) to mean "use topology precision" (Sandro Santilli)

#6024, Remove support for PostgreSQL 12 and 13 (Regina Obe)

#6040, ST_GeoHash round trip through ST_GeomFromGeoHash (Paul Ramsey)

#6053, [address_standardizer] Extension removed and moved to https://github.com/postgis/address_standardizer (Paul Ramsey)

#6052, [tiger_geocoder] Extension removed and moved to https://git.osgeo.org/postgis/postgis_tiger_geocoder (Regina Obe)

#6079, Remove support for GEOS < 3.10 (Sandro Santilli)

#1793, Remove ST_MemCollect aggregate (Darafei Praliaskouski)

GH-848, CG_AlphaShape now returns a MultiPolygon (Jean Felder)

A.1.2 New Features

#1124, #2935, #3033, #4658, #4659, [loader] Rework loader arguments: shp2pgsql can create UNLOGGED tables and choose the feature id column name, shp2pgsql --drop-table can emit DROP TABLE before prepare output, raster2pgsql/shp2pgsql expose loader actions as long options, both loaders accept long aliases for existing options, and

-if-not-exists makes creation actions idempotent (Darafei Praliaskouski)

#4208, Add single-geometry variants of ST_MaxDistance and ST_LongestLine (Darafei Praliaskouski)

[topology] FindVertexSegmentPairsBelowDistance function (Sandro Santilli)

ST_CoverageEdges, returns MultiLinestring of distinct shared edges in polygonal coverage (Paul Ramsey)

ST_MinimumSpanningTree, window function to calculate MST (Paul Ramsey)

#5993, [topology] Add max_edges parameter to TopoGeo_AddLinestring (Sandro Santilli)

#6001, support MultiLineString in ST_MakeLine (Paul Ramsey)

#2858, ST_MMin and ST_MMax (Paul Ramsey)

#5941, [raster] Support GDT_Float16 pixel type (Darafei Praliaskouski)

#5992, Optimize GiST index for repeated edge subdivision in topology (Darafei Praliaskouski)

#5702, Allow the compiler to detect the parallelism -flto=auto (Darafei Praliaskouski)

#4798, ST_AsGeoJSON warns about duplicate property keys (Darafei Praliaskouski)

#5950, Document POSTGIS_REGRESS_DB_OWNER for sandboxed regression roles (Darafei Praliaskouski)

#4332, Clarify the scope of several Function Reference categories (Darafei Praliaskouski)

#4222, [raster] ST_DumpAsPolygons honours PostgreSQL interrupts (Darafei Praliaskouski)

#5109, Document the meaning of topology.next_left_edge and topology.next_right_edge links (Darafei Praliaskouski)

#5889, [topology] Include representative locations in topology build errors (Darafei Praliaskouski)

#2614, Use GEOSPreparedDistance and caching to accelerate ST_DWithin (Paul Ramsey)

#6022, Document ST_AsMVTGeom tile-coordinate simplification (Darafei Praliaskouski)

GH-839, ST_Multi support for TIN and surfaces (Loïc Bartoletti)

#4398, Add ST_CatmullSmoothing smoothes but retains original vertices (Paul Ramsey)

#4560, ST_3DInterpolatePoint for M interpolation from XYZ inputs (Paul Ramsey)

#1291, ST_MakePolygon support for curved rings (Darafei Praliaskouski)

GT-270, Add NURBSCurve (Loïc Bartoletti)

#2863, Add ST_XSize, ST_YSize, ST_ZSize, and ST_MSize dimension helpers (Darafei Praliaskouski)

[fuzzers] Centralize OSS-Fuzz build/dependency logic in PostGIS scripts, prefer requested CXX over pg_config --cc for internal C++ checks, and make fuzzer linking/runtime dependency packaging portable (Darafei Praliaskouski)

A.1.3 Améliorations

#2045, pgsql2shp query dumps no longer require temporary table privileges (Darafei Praliaskouski)

#1577, [loader] Report unsupported .zip archive input before trying to open shapefile sidecar files (Darafei Praliaskouski)

#3158, Move geometry and geography typmod bit helpers out of the public liblwgeom header (Darafei Praliaskouski)

#4678, [raster] Move st_sum4ma neighborhood callback to C (Darafei Praliaskouski)

#2116, [raster] Add ST_Value nearest-neighbor boundary options (Darafei Praliaskouski)

#2804, #4315, [raster] Support two-argument ST_MapAlgebra callbacks and pass callback call data as actual arguments (Darafei Praliaskouski)

#2898, Document SQL function cost tiers for contributors

#6062, [topology] Stop using recursive snapping, for improved robustness (Sandro Santilli)

#6065, Improved winding order computation robustness for rings having collapsed elements (Sandro Santilli)

#3743, #4385, [raster loader] Document raster2pgsql -s FROM_SRID:SRID reprojection support and clarify index creation timing for append loads (Darafei Praliaskouski)

#2137, [raster] Reuse matching source raster overviews in raster2pgsql -l (Darafei Praliaskouski)

#4749, Use point-in-polygon predicate fast paths for point-only GeometryCollections (Darafei Praliaskouski)

#5532, Validate the manual against DocBook XMLSchema in check-xml (Darafei Praliaskouski)

#6087, Add a repository-owned CI status checker and static dashboard generator (Darafei Praliaskouski)

#2807, [raster] Preserve missing NODATA values in ST_MapAlgebra outputs (Darafei Praliaskouski)

#2832, [raster] Avoid padding single-tile raster2pgsql overviews when padding is not requested (Darafei Praliaskouski)

#2623, Generate a pgsql2shp GID field when exporting geometry-only shapefiles (Darafei Praliaskouski)

#2850, Preserve pgsql2shp numeric precision and scale in DBF metadata (Darafei Praliaskouski)

#2386, Add raster2pgsql man page (Darafei Praliaskouski)

Build PostgreSQL extension modules with `-fno-semantic-interposition` when available so LTO can optimize same-DSO calls to exported PostGIS entry points directly instead of routing through the module PLT; in a local PostgreSQL 18 `pgbench` microbenchmark over 100k four-point groups this reduced `ST_Collect` aggregate latency by about 4% (Darafei Praliaskouski)

#5973, Install PostGIS utilities under `pg_config --bindir` by default and document `pg_config`-derived install paths more accurately (Darafei Praliaskouski)

#2808, Clarify postgis_full_version() output when installed component scripts are newer than core scripts (Darafei Praliaskouski)

#5593, Docs: render manual images with GraphicsMagick without temporary files, enabling parallel builds, keeping ImageMagick fallbacks working, and documenting the generator workflow for contributors (Darafei Praliaskouski)

#5645, Docs: keep code operators ("=>") intact in translated manuals by enforcing verbatim CSS (Darafei Praliaskouski)

#1408, Docs: flatten transparent manual images onto white backgrounds for PDF builds while keeping HTML images transparent (Darafei Praliaskouski)

#2838, Emit valid X3D Coordinate nodes for 2D polygon output (Darafei Praliaskouski)

GH-899, [raster] Honor PostgreSQL interrupts in long-running GDAL progress callbacks (Darafei Praliaskouski)

#2583, Preserve Z/ZM dimensional qualifiers in ST_AsEWKT output for TIN (Darafei Praliaskouski)