

PostGIS 3.5.0alpha1 ☒☒☒☒☒☒

Contents

1	簡介	1
1.1	安裝與升級	1
1.2	安裝 - 簡介	2
1.3	安裝 - 簡介	2
1.4	安裝	3
2	PostGIS 簡介	6
2.1	簡介	6
2.2	安裝與升級	6
2.2.1	安裝	7
2.2.2	安裝	7
2.2.3	簡介	8
2.2.4	簡介	10
2.2.5	PostGIS Extensions 安裝	10
2.2.6	簡介	12
2.2.7	簡介	15
2.3	安裝與升級	16
2.4	安裝, 升級 Tiger Geocoder, 和 加載數據	16
2.4.1	Tiger Geocoder 啟用你的 PostGIS 數據庫	17
2.4.2	安裝 TIGER 數據	19
2.4.3	加載 tiger 數據所需的工具	19
2.4.4	升級你的 Tiger Geocoder 安裝和數據	20
2.5	安裝與升級	20
3	PostGIS 管理	22
3.1	性能調優	22
3.1.1	啟動	22
3.1.2	運行	23
3.2	配置光柵支持	23
3.3	安裝與升級	24

3.3.1	Spatially enable database using EXTENSION	24
3.3.2	Spatially enable database without using EXTENSION (discouraged)	24
3.4	Upgrading spatial databases	25
3.4.1	Soft upgrade	25
3.4.1.1	Soft Upgrade 9.1+ using extensions	25
3.4.1.2	Soft Upgrade Pre 9.1+ or without extensions	26
3.4.2	Hard upgrade	27
4	Data Management	29
4.1	GIS (OGC) Geometry	29
4.1.1	OGC Geometry	29
4.1.1.1	Point	30
4.1.1.2	LineString	30
4.1.1.3	LinearRing	30
4.1.1.4	Polygon	30
4.1.1.5	MultiPoint	30
4.1.1.6	MultiLineString	31
4.1.1.7	MultiPolygon	31
4.1.1.8	GeometryCollection	31
4.1.1.9	PolyhedralSurface	31
4.1.1.10	Triangle	31
4.1.1.11	TIN	31
4.1.2	SQL-MM Part 3	32
4.1.2.1	CircularString	32
4.1.2.2	CompoundCurve	32
4.1.2.3	CurvePolygon	32
4.1.2.4	MultiCurve	33
4.1.2.5	MultiSurface	33
4.1.3	OpenGIS WKB & WKT	33
4.2	Geometry Data Type	34
4.2.1	OpenGIS WKB & WKT	34
4.3	PostGIS Geometry	36
4.3.1	Geometry	37
4.3.2	PostGIS Geometry	38
4.3.3	Geometry Functions	39
4.3.4	Geometry FAQ	39
4.4	Geometry Validation	39
4.4.1	Simple Geometry	40
4.4.2	Valid Geometry	42

4.4.3	Managing Validity	44
4.5	SPATIAL_REF_SYS 简体中文	45
4.5.1	SPATIAL_REF_SYS Table	45
4.5.2	SPATIAL_REF_SYS 简体中文	46
4.6	简体中文	47
4.6.1	简体中文	47
4.6.2	The GEOMETRY_COLUMNS VIEW	48
4.6.3	geometry_columns 简体中文	48
4.7	GIS (GIS) 简体中文	50
4.7.1	SQL 简体中文	51
4.7.2	shp2pgsql: ESRI shapefile 简体中文	51
4.8	简体中文	53
4.8.1	SQL 简体中文	53
4.8.2	简体中文	54
4.9	简体中文	54
4.9.1	GiST 简体中文	55
4.9.2	GiST 简体中文	55
4.9.3	GiST 简体中文	57
4.9.4	简体中文	58
5	Spatial Queries	59
5.1	Determining Spatial Relationships	59
5.1.1	Dimensionally Extended 9-Intersection Model	59
5.1.2	Named Spatial Relationships	61
5.1.3	General Spatial Relationships	62
5.2	Using Spatial Indexes	64
5.3	Examples of Spatial SQL	65
6	简体中文	68
6.1	简体中文	68
6.1.1	简体中文	68
6.1.2	简体中文	68
6.2	简体中文	69
6.3	简体中文	69

7 PostGIS Reference	70
7.1 PostgreSQL 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	73
7.2.1 AddGeometryColumn	73
7.2.2 DropGeometryColumn	75
7.2.3 DropGeometryTable	75
7.2.4 Find_SRID	76
7.2.5 Populate_Geometry_Columns	77
7.2.6 UpdateGeometrySRID	78
7.3 (constructor)	79
7.3.1 ST_Collect	79
7.3.2 ST_LineFromMultiPoint	81
7.3.3 ST_MakeEnvelope	82
7.3.4 ST_MakeLine	82
7.3.5 ST_MakePoint	84
7.3.6 ST_MakePointM	85
7.3.7 ST_MakePolygon	87
7.3.8 ST_Point	88
7.3.9 ST_PointZ	90
7.3.10 ST_PointM	90
7.3.11 ST_PointZM	91
7.3.12 ST_Polygon	91
7.3.13 ST_TileEnvelope	92
7.3.14 ST_HexagonGrid	93
7.3.15 ST_Hexagon	96
7.3.16 ST_SquareGrid	97
7.3.17 ST_Square	98
7.3.18 ST_Letters	99
7.4 (accessor)	100
7.4.1 GeometryType	100
7.4.2 ST_Boundary	102
7.4.3 ST_BoundingDiagonal	104
7.4.4 ST_CoordDim	105
7.4.5 ST_Dimension	105

7.4.6 ST_Dump	106
7.4.7 ST_DumpPoints	108
7.4.8 ST_DumpSegments	112
7.4.9 ST_DumpRings	114
7.4.10 ST_EndPoint	115
7.4.11 ST_Envelope	116
7.4.12 ST_ExteriorRing	118
7.4.13 ST_GeometryN	119
7.4.14 ST_GeometryType	121
7.4.15 ST_HasArc	122
7.4.16 ST_InteriorRingN	123
7.4.17 ST_NumCurves	124
7.4.18 ST_CurveN	124
7.4.19 ST_IsClosed	125
7.4.20 ST_IsCollection	127
7.4.21 ST_IsEmpty	128
7.4.22 ST_IsPolygonCCW	129
7.4.23 ST_IsPolygonCW	130
7.4.24 ST_IsRing	131
7.4.25 ST_IsSimple	131
7.4.26 ST_M	132
7.4.27 ST_MemSize	133
7.4.28 ST_NDims	134
7.4.29 ST_NPoints	135
7.4.30 ST_NRings	136
7.4.31 ST_NumGeometries	136
7.4.32 ST_NumInteriorRings	137
7.4.33 ST_NumInteriorRing	138
7.4.34 ST_NumPatches	138
7.4.35 ST_NumPoints	139
7.4.36 ST_PatchN	140
7.4.37 ST_PointN	141
7.4.38 ST_Points	142
7.4.39 ST_StartPoint	143
7.4.40 ST_Summary	144
7.4.41 ST_X	145
7.4.42 ST_Y	146
7.4.43 ST_Z	147
7.4.44 ST_Zmflag	148

7.4.45	ST_HasZ	148
7.4.46	ST_HasM	149
7.5	编辑器	150
7.5.1	ST_AddPoint	150
7.5.2	ST_CollectionExtract	151
7.5.3	ST_CollectionHomogenize	152
7.5.4	ST_CurveToLine	154
7.5.5	ST_Scroll	156
7.5.6	ST_FlipCoordinates	157
7.5.7	ST_Force2D	158
7.5.8	ST_Force3D	159
7.5.9	ST_Force3DZ	159
7.5.10	ST_Force3DM	160
7.5.11	ST_Force4D	161
7.5.12	ST_ForceCollection	162
7.5.13	ST_ForceCurve	163
7.5.14	ST_ForcePolygonCCW	164
7.5.15	ST_ForcePolygonCW	164
7.5.16	ST_ForceSFS	165
7.5.17	ST_ForceRHR	165
7.5.18	ST_LineExtend	166
7.5.19	ST_LineToCurve	167
7.5.20	ST_Multi	168
7.5.21	ST_Normalize	169
7.5.22	ST_Project	170
7.5.23	ST_QuantizeCoordinates	170
7.5.24	ST_RemovePoint	173
7.5.25	ST_RemoveRepeatedPoints	173
7.5.26	ST_RemoveIrrelevantPointsForView	174
7.5.27	ST_RemoveSmallParts	176
7.5.28	ST_Reverse	178
7.5.29	ST_Segmentize	179
7.5.30	ST_SetPoint	180
7.5.31	ST_ShiftLongitude	181
7.5.32	ST_WrapX	182
7.5.33	ST_SnapToGrid	183
7.5.34	ST_Snap	185
7.5.35	ST_SwapOrdinates	187
7.6	Geometry Validation	188

7.6.1	ST_IsValid	188
7.6.2	ST_IsValidDetail	189
7.6.3	ST_IsValidReason	191
7.6.4	ST_MakeValid	192
7.7	Spatial Reference System Functions	197
7.7.1	ST_InverseTransformPipeline	197
7.7.2	ST_SetSRID	198
7.7.3	ST_SRID	199
7.7.4	ST_Transform	200
7.7.5	ST_TransformPipeline	202
7.7.6	postgis_srs_codes	204
7.7.7	postgis_srs	205
7.7.8	postgis_srs_all	205
7.7.9	postgis_srs_search	206
7.8	Geometry Input	207
7.8.1	Well-Known Text (WKT)	207
7.8.1.1	ST_BdPolyFromText	207
7.8.1.2	ST_BdMPolyFromText	208
7.8.1.3	ST_GeogFromText	208
7.8.1.4	ST_GeographyFromText	209
7.8.1.5	ST_GeomCollFromText	209
7.8.1.6	ST_GeomFromEWKT	210
7.8.1.7	ST_GeomFromMARC21	212
7.8.1.8	ST_GeometryFromText	214
7.8.1.9	ST_GeomFromText	215
7.8.1.10	ST_LineFromText	216
7.8.1.11	ST_MLineFromText	217
7.8.1.12	ST_MPointFromText	218
7.8.1.13	ST_MPolyFromText	218
7.8.1.14	ST_PointFromText	219
7.8.1.15	ST_PolygonFromText	220
7.8.1.16	ST_WKTToSQL	221
7.8.2	Well-Known Binary (WKB)	222
7.8.2.1	ST_GeogFromWKB	222
7.8.2.2	ST_GeomFromEWKB	222
7.8.2.3	ST_GeomFromWKB	224
7.8.2.4	ST_LineFromWKB	225
7.8.2.5	ST_LinestringFromWKB	225
7.8.2.6	ST_PointFromWKB	226

7.8.2.7	ST_WKBToSQL	227
7.8.3	Other Formats	228
7.8.3.1	ST_Box2dFromGeoHash	228
7.8.3.2	ST_GeomFromGeoHash	229
7.8.3.3	ST_GeomFromGML	230
7.8.3.4	ST_GeomFromGeoJSON	232
7.8.3.5	ST_GeomFromKML	233
7.8.3.6	ST_GeomFromTWKB	234
7.8.3.7	ST_GMLToSQL	235
7.8.3.8	ST_LineFromEncodedPolyline	235
7.8.3.9	ST_PointFromGeoHash	236
7.8.3.10	ST_FromFlatGeobufToTable	237
7.8.3.11	ST_FromFlatGeobuf	237
7.9	Geometry Output	238
7.9.1	Well-Known Text (WKT)	238
7.9.1.1	ST_AsEWKT	238
7.9.1.2	ST_AsText	239
7.9.2	Well-Known Binary (WKB)	240
7.9.2.1	ST_AsBinary	240
7.9.2.2	ST_AsEWKB	242
7.9.2.3	ST_AsHEXEWKB	243
7.9.3	Other Formats	244
7.9.3.1	ST_AsEncodedPolyline	244
7.9.3.2	ST_AsFlatGeobuf	245
7.9.3.3	ST_AsGeobuf	245
7.9.3.4	ST_AsGeoJSON	246
7.9.3.5	ST_AsGML	248
7.9.3.6	ST_AsKML	252
7.9.3.7	ST_AsLatLonText	253
7.9.3.8	ST_AsMARC21	254
7.9.3.9	ST_AsMVTGeom	257
7.9.3.10	ST_AsMVT	258
7.9.3.11	ST_AsSVG	259
7.9.3.12	ST_AsTWKB	261
7.9.3.13	ST_AsX3D	262
7.9.3.14	ST_GeoHash	265
7.10	⊞⊞⊞ (operator)	267
7.10.1	Bounding Box Operators	267
7.10.1.1	&&	267

7.10.1.2☒☒(geometry,box2df)	267
7.10.1.3☒☒(box2df,geometry)	268
7.10.1.4☒☒(box2df,box2df)	269
7.10.1.5☒☒	270
7.10.1.6☒☒☒(geometry,gidx)	271
7.10.1.7☒☒☒(gidx,geometry)	272
7.10.1.8☒☒☒(gidx,gidx)	273
7.10.1.9☒<	274
7.10.1.10☒<	275
7.10.1.11☒>	275
7.10.1.12☒<	276
7.10.1.13☒<	277
7.10.1.14☒	278
7.10.1.15☒>	279
7.10.1.16☒	280
7.10.1.17☒(geometry,box2df)	281
7.10.1.18☒(box2df,geometry)	282
7.10.1.19☒(box2df,box2df)	282
7.10.1.20☒☒>	283
7.10.1.21☒☒>	284
7.10.1.22☒☒	285
7.10.1.23☒☒(geometry,box2df)	286
7.10.1.24☒☒(box2df,geometry)	286
7.10.1.25☒☒(box2df,box2df)	287
7.10.1.26☒☒=	288
7.10.2☒☒☒ (operator)	289
7.10.2.1☒->	289
7.10.2.2☒ =	291
7.10.2.3☒<#>	292
7.10.2.4☒<<->>	293
7.11 Spatial Relationships	294
7.11.1 Topological Relationships	294
7.11.1.1 ST_3DIntersects	294
7.11.1.2 ST_Contains	295
7.11.1.3 ST_ContainsProperly	299
7.11.1.4 ST_CoveredBy	300
7.11.1.5 ST_Covers	301
7.11.1.6 ST_Crosses	303
7.11.1.7 ST_Disjoint	305

7.11.1.8	ST_Equals	306
7.11.1.9	ST_Intersects	307
7.11.1.10	ST_LineCrossingDirection	309
7.11.1.11	ST_OrderingEquals	312
7.11.1.12	ST_Overlaps	313
7.11.1.13	ST_Relate	316
7.11.1.14	ST_RelateMatch	318
7.11.1.15	ST_Touches	319
7.11.1.16	ST_Within	321
7.11.2	Distance Relationships	323
7.11.2.1	ST_3DDWithin	323
7.11.2.2	ST_3DDFullyWithin	324
7.11.2.3	ST_DFullyWithin	325
7.11.2.4	ST_DWithin	326
7.11.2.5	ST_PointInsideCircle	327
7.12	Measurement Functions	328
7.12.1	ST_Area	328
7.12.2	ST_Azimuth	330
7.12.3	ST_Angle	331
7.12.4	ST_ClosestPoint	332
7.12.5	ST_3DClosestPoint	334
7.12.6	ST_Distance	335
7.12.7	ST_3DDistance	337
7.12.8	ST_DistanceSphere	338
7.12.9	ST_DistanceSpheroid	339
7.12.10	ST_FrechetDistance	340
7.12.11	ST_HausdorffDistance	341
7.12.12	ST_Length	343
7.12.13	ST_Length2D	344
7.12.14	ST_3DLength	345
7.12.15	ST_LengthSpheroid	345
7.12.16	ST_LongestLine	347
7.12.17	ST_3DLongestLine	349
7.12.18	ST_MaxDistance	350
7.12.19	ST_3DMaxDistance	351
7.12.20	ST_MinimumClearance	352
7.12.21	ST_MinimumClearanceLine	353
7.12.22	ST_Perimeter	353
7.12.23	ST_Perimeter2D	355

7.12.2	ST_3DPerimeter	355
7.12.2	ST_ShortestLine	356
7.12.2	ST_3DShortestLine	358
7.13	Overlay Functions	359
7.13.1	ST_ClipByBox2D	359
7.13.2	ST_Difference	360
7.13.3	ST_Intersection	361
7.13.4	ST_MemUnion	364
7.13.5	ST_Node	364
7.13.6	ST_Split	365
7.13.7	ST_Subdivide	368
7.13.8	ST_SymDifference	371
7.13.9	ST_UnaryUnion	372
7.13.1	ST_Union	373
7.14	ST_	376
7.14.1	ST_Buffer	376
7.14.2	ST_BuildArea	381
7.14.3	ST_Centroid	382
7.14.4	ST_ChaikinSmoothing	384
7.14.5	ST_ConcaveHull	386
7.14.6	ST_ConvexHull	389
7.14.7	ST_DelaunayTriangles	391
7.14.8	ST_FilterByM	396
7.14.9	ST_GeneratePoints	397
7.14.1	ST_GeometricMedian	398
7.14.1	ST_LineMerge	400
7.14.1	ST_MaximumInscribedCircle	402
7.14.1	ST_LargestEmptyCircle	404
7.14.1	ST_MinimumBoundingCircle	406
7.14.1	ST_MinimumBoundingRadius	408
7.14.1	ST_OrientedEnvelope	408
7.14.1	ST_OffsetCurve	409
7.14.1	ST_PointOnSurface	413
7.14.1	ST_Polygonize	416
7.14.2	ST_ReducePrecision	418
7.14.2	ST_SharedPaths	419
7.14.2	ST_Simplify	421
7.14.2	ST_SimplifyPreserveTopology	423
7.14.2	ST_SimplifyPolygonHull	425

7.14.2	ST_SimplifyVW	428
7.14.2	ST_SetEffectiveArea	429
7.14.2	ST_TriangulatePolygon	431
7.14.2	ST_VoronoiLines	433
7.14.2	ST_VoronoiPolygons	434
7.15	Coverages	436
7.15.1	ST_CoverageInvalidEdges	436
7.15.2	ST_CoverageSimplify	437
7.15.3	ST_CoverageUnion	439
7.16	Affine Transformations	440
7.16.1	ST_Affine	440
7.16.2	ST_Rotate	442
7.16.3	ST_RotateX	443
7.16.4	ST_RotateY	444
7.16.5	ST_RotateZ	445
7.16.6	ST_Scale	446
7.16.7	ST_Translate	447
7.16.8	ST_TransScale	448
7.17	Clustering Functions	450
7.17.1	ST_ClusterDBSCAN	450
7.17.2	ST_ClusterIntersecting	452
7.17.3	ST_ClusterIntersectingWin	452
7.17.4	ST_ClusterKMeans	453
7.17.5	ST_ClusterWithin	455
7.17.6	ST_ClusterWithinWin	456
7.18	Bounding Box Functions	457
7.18.1	Box2D	457
7.18.2	Box3D	458
7.18.3	ST_EstimatedExtent	459
7.18.4	ST_Expand	460
7.18.5	ST_Extent	461
7.18.6	ST_3DExtent	462
7.18.7	ST_MakeBox2D	463
7.18.8	ST_3DMakeBox	464
7.18.9	ST_XMax	465
7.18.1	ST_XMin	466
7.18.1	ST_YMax	467
7.18.1	ST_YMin	468
7.18.1	ST_ZMax	469

7.18.1	ST_ZMin	470
7.19	ST_LineInterpolatePoint (Linear Referencing)	471
7.19.1	ST_LineInterpolatePoint	471
7.19.2	ST_3DLineInterpolatePoint	472
7.19.3	ST_LineInterpolatePoints	473
7.19.4	ST_LineLocatePoint	474
7.19.5	ST_LineSubstring	475
7.19.6	ST_LocateAlong	477
7.19.7	ST_LocateBetween	478
7.19.8	ST_LocateBetweenElevations	480
7.19.9	ST_InterpolatePoint	481
7.19.10	ST_AddMeasure	481
7.20	Trajectory Functions	482
7.20.1	ST_IsValidTrajectory	482
7.20.2	ST_ClosestPointOfApproach	483
7.20.3	ST_DistanceCPA	484
7.20.4	ST_CPAWithin	485
7.21	Version Functions	486
7.21.1	PostGIS_Extensions_Upgrade	486
7.21.2	PostGIS_Full_Version	487
7.21.3	PostGIS_GEOS_Version	487
7.21.4	PostGIS_GEOS_Compiled_Version	488
7.21.5	PostGIS_Liblwgeom_Version	488
7.21.6	PostGIS_LibXML_Version	489
7.21.7	PostGIS_Lib_Build_Date	489
7.21.8	PostGIS_Lib_Version	490
7.21.9	PostGIS_PROJ_Version	490
7.21.10	PostGIS_PROJ_Compiled_Version	491
7.21.11	PostGIS_Wagyu_Version	492
7.21.12	PostGIS_Scripts_Build_Date	492
7.21.13	PostGIS_Scripts_Installed	493
7.21.14	PostGIS_Scripts_Released	493
7.21.15	PostGIS_Version	494
7.22	PostGIS GUC(Grand Unified Custom Variable)	495
7.22.1	postgis.backend	495
7.22.2	postgis.gdal_datapath	495
7.22.3	postgis.gdal_enabled_drivers	496
7.22.4	postgis.enable_outdb_rasters	497
7.22.5	postgis.gdal_vsi_options	498
7.23	Troubleshooting Functions	499
7.23.1	PostGIS_AddBBox	499
7.23.2	PostGIS_DropBBox	499
7.23.3	PostGIS_HasBBox	500

8 SFCGAL Functions Reference	502
8.1 SFCGAL Management Functions	502
8.1.1 postgis_sfcgal_version	502
8.1.2 postgis_sfcgal_full_version	502
8.2 SFCGAL Accessors and Setters	503
8.2.1 CG_ForceLHR	503
8.2.2 CG_IsPlanar	503
8.2.3 CG_IsSolid	504
8.2.4 CG_MakeSolid	504
8.2.5 CG_Orientation	505
8.2.6 CG_Area	505
8.2.7 CG_3DArea	506
8.2.8 CG_Volume	506
8.2.9 ST_ForceLHR	507
8.2.10ST_IsPlanar	508
8.2.11ST_IsSolid	508
8.2.12ST_MakeSolid	509
8.2.13ST_Orientation	509
8.2.14ST_3DArea	510
8.2.15ST_Volume	511
8.3 SFCGAL Processing and Relationship Functions	512
8.3.1 CG_Intersection	512
8.3.2 CG_Intersects	513
8.3.3 CG_3DIntersects	513
8.3.4 CG_Difference	514
8.3.5 ST_3DDifference	515
8.3.6 CG_3DDifference	516
8.3.7 CG_Distance	517
8.3.8 CG_3DDistance	518
8.3.9 ST_3DConvexHull	519
8.3.10CG_3DConvexHull	519
8.3.11ST_3DIntersection	520
8.3.12CG_3DIntersection	521
8.3.13CG_Union	523
8.3.14ST_3DUnion	524
8.3.15CG_3DUnion	524
8.3.16ST_AlphaShape	526
8.3.17CG_AlphaShape	526
8.3.18CG_ApproxConvexPartition	529

8.3.19	ST_ApproximateMedialAxis	530
8.3.20	CG_ApproximateMedialAxis	531
8.3.21	ST_ConstrainedDelaunayTriangles	532
8.3.22	CG_ConstrainedDelaunayTriangles	533
8.3.23	ST_Extrude	534
8.3.24	CG_Extrude	535
8.3.25	CG_ExtrudeStraightSkeleton	537
8.3.26	CG_GreeneApproxConvexPartition	538
8.3.27	ST_MinkowskiSum	539
8.3.28	CG_MinkowskiSum	540
8.3.29	ST_OptimalAlphaShape	542
8.3.30	CG_OptimalAlphaShape	543
8.3.31	CG_OptimalConvexPartition	545
8.3.32	CG_StraightSkeleton	546
8.3.33	ST_StraightSkeleton	547
8.3.34	ST_Tesselate	549
8.3.35	CG_Tesselate	549
8.3.36	CG_Triangulate	552
8.3.37	CG_Visibility	553
8.3.38	CG_YMonotonePartition	554
9	Topology	556
9.1	Topology	556
9.1.1	getfaceedges_returntype	556
9.1.2	TopoGeometry	557
9.1.3	validatetopology_returntype	557
9.2	TopoElement	558
9.2.1	TopoElement	558
9.2.2	TopoElementArray	558
9.3	TopoGeometry	559
9.3.1	AddTopoGeometryColumn	559
9.3.2	RenameTopoGeometryColumn	560
9.3.3	DropTopology	561
9.3.4	RenameTopology	561
9.3.5	DropTopoGeometryColumn	562
9.3.6	Populate_Topology_Layer	562
9.3.7	TopologySummary	563
9.3.8	ValidateTopology	564
9.3.9	ValidateTopologyRelation	567

9.3.10	FindTopology	567
9.3.11	FindLayer	568
9.4	Topology Statistics Management	568
9.5	□□□□□	568
9.5.1	CreateTopology	568
9.5.2	CopyTopology	569
9.5.3	ST_InitTopoGeo	570
9.5.4	ST_CreateTopoGeo	571
9.5.5	TopoGeo_AddPoint	572
9.5.6	TopoGeo_AddLineString	572
9.5.7	TopoGeo_AddPolygon	573
9.5.8	TopoGeo_LoadGeometry	573
9.6	□□□□□	574
9.6.1	ST_AddIsoNode	574
9.6.2	ST_AddIsoEdge	574
9.6.3	ST_AddEdgeNewFaces	575
9.6.4	ST_AddEdgeModFace	576
9.6.5	ST_RemEdgeNewFace	576
9.6.6	ST_RemEdgeModFace	577
9.6.7	ST_ChangeEdgeGeom	578
9.6.8	ST_ModEdgeSplit	579
9.6.9	ST_ModEdgeHeal	579
9.6.10	ST_NewEdgeHeal	580
9.6.11	ST_MoveIsoNode	580
9.6.12	ST_NewEdgesSplit	581
9.6.13	ST_RemoveIsoNode	582
9.6.14	ST_RemoveIsoEdge	583
9.7	□□□□□	583
9.7.1	GetEdgeByPoint	583
9.7.2	GetFaceByPoint	584
9.7.3	GetFaceContainingPoint	585
9.7.4	GetNodeByPoint	585
9.7.5	GetTopologyID	586
9.7.6	GetTopologySRID	587
9.7.7	GetTopologyName	587
9.7.8	ST_GetFaceEdges	588
9.7.9	ST_GetFaceGeometry	589
9.7.10	GetRingEdges	590
9.7.11	GetNodeEdges	590

9.8		591
9.8.1	Polygonize	591
9.8.2	AddNode	591
9.8.3	AddEdge	592
9.8.4	AddFace	593
9.8.5	ST_Simplify	595
9.8.6	RemoveUnusedPrimitives	596
9.9	TopoGeometry	596
9.9.1	CreateTopoGeom	596
9.9.2	toTopoGeom	598
9.9.3	TopoElementArray_Agg	599
9.9.4	TopoElement	600
9.10	TopoGeometry	600
9.10.1	clearTopoGeom	600
9.10.2	TopoGeom_addElement	601
9.10.3	TopoGeom_remElement	601
9.10.4	TopoGeom_addTopoGeom	602
9.10.5	toTopoGeom	603
9.11	TopoGeometry	603
9.11.1	GetTopoGeomElementArray	603
9.11.2	GetTopoGeomElements	603
9.11.3	ST_SRID	604
9.12	TopoGeometry	605
9.12.1	AsGML	605
9.12.2	AsTopoJSON	607
9.13		608
9.13.1	Equals	608
9.13.2	Intersects	609
9.14	Importing and exporting Topologies	610
9.14.1	Using the Topology exporter	610
9.14.2	Using the Topology importer	610
10		612
10.1		612
10.1.1	raster2pgsql	612
10.1.1.1	Example Usage	612
10.1.1.2	raster2pgsql options	613
10.1.2	PostGIS	614
10.1.3	Using "out db" cloud rasters	615

10.2 地理函数	616
10.2.1 地理函数	616
10.2.2 地理函数	617
10.3 PostGIS 地理函数	618
10.3.1 地理函数 ST_AsPNG 使用 PHP 地理函数	618
10.3.2 地理函数 ST_AsPNG 使用 ASP.NET C# 地理函数	619
10.3.3 地理函数 ST_AsPNG 使用 Java 地理函数	620
10.3.4 PLPython 地理函数 SQL 地理函数	622
10.3.5 PSQL 地理函数	622
11 地理函数	624
11.1 地理函数	625
11.1.1 geomval	625
11.1.2 addbandarg	625
11.1.3 rastbandarg	625
11.1.4 raster	626
11.1.5 reclassarg	626
11.1.6 summarystats	627
11.1.7 unionarg	627
11.2 地理函数	628
11.2.1 AddRasterConstraints	628
11.2.2 DropRasterConstraints	630
11.2.3 AddOverviewConstraints	631
11.2.4 DropOverviewConstraints	632
11.2.5 PostGIS_GDAL_Version	632
11.2.6 PostGIS_Raster_Lib_Build_Date	633
11.2.7 PostGIS_Raster_Lib_Version	633
11.2.8 ST_GDALDrivers	634
11.2.9 ST_Contour	639
11.2.10 ST_InterpolateRaster	640
11.2.11 UpdateRasterSRID	640
11.2.12 ST_CreateOverview	641
11.3 地理函数 (constructor)	642
11.3.1 ST_AddBand	642
11.3.2 ST_AsRaster	644
11.3.3 ST_Band	646
11.3.4 ST_MakeEmptyCoverage	648
11.3.5 ST_MakeEmptyRaster	649
11.3.6 ST_Tile	650

11.3.7	ST_Retile	653
11.3.8	ST_FromGDALRaster	653
11.4	(accessor)	654
11.4.1	ST_GeoReference	654
11.4.2	ST_Height	655
11.4.3	ST_IsEmpty	656
11.4.4	ST_MemSize	656
11.4.5	ST_MetaData	657
11.4.6	ST_NumBands	658
11.4.7	ST_PixelHeight	658
11.4.8	ST_PixelWidth	659
11.4.9	ST_ScaleX	661
11.4.10	ST_ScaleY	661
11.4.11	ST_RasterToWorldCoord	662
11.4.12	ST_RasterToWorldCoordX	663
11.4.13	ST_RasterToWorldCoordY	664
11.4.14	ST_Rotation	665
11.4.15	ST_SkewX	665
11.4.16	ST_SkewY	666
11.4.17	ST_SRID	667
11.4.18	ST_Summary	667
11.4.19	ST_UpperLeftX	668
11.4.20	ST_UpperLeftY	669
11.4.21	ST_Width	669
11.4.22	ST_WorldToRasterCoord	670
11.4.23	ST_WorldToRasterCoordX	671
11.4.24	ST_WorldToRasterCoordY	671
11.5		672
11.5.1	ST_BandMetaData	672
11.5.2	ST_BandNoDataValue	674
11.5.3	ST_BandIsNoData	674
11.5.4	ST_BandPath	676
11.5.5	ST_BandFileSize	676
11.5.6	ST_BandFileTimestamp	677
11.5.7	ST_BandPixelType	677
11.5.8	ST_MinPossibleValue	678
11.5.9	ST_HasNoBand	679
11.6	(setter)	679
11.6.1	ST_PixelAsPolygon	679

11.6.2	ST_PixelAsPolygons	680
11.6.3	ST_PixelAsPoint	681
11.6.4	ST_PixelAsPoints	682
11.6.5	ST_PixelAsCentroid	683
11.6.6	ST_PixelAsCentroids	683
11.6.7	ST_Value	685
11.6.8	ST_NearestValue	688
11.6.9	ST_SetZ	689
11.6.10	ST_SetM	690
11.6.11	ST_Neighborhood	692
11.6.12	ST_SetValue	694
11.6.13	ST_SetValues	695
11.6.14	ST_DumpValues	703
11.6.15	ST_PixelOfValue	704
11.7	ST_Geometry	706
11.7.1	ST_SetGeoReference	706
11.7.2	ST_SetRotation	707
11.7.3	ST_SetScale	708
11.7.4	ST_SetSkew	709
11.7.5	ST_SetSRID	710
11.7.6	ST_SetUpperLeft	710
11.7.7	ST_Resample	711
11.7.8	ST_Rescale	712
11.7.9	ST_Reskew	714
11.7.10	ST_SnapToGrid	715
11.7.11	ST_Resize	716
11.7.12	ST_Transform	717
11.8	ST_Band	720
11.8.1	ST_SetBandNoDataValue	720
11.8.2	ST_SetBandIsNoData	721
11.8.3	ST_SetBandPath	723
11.8.4	ST_SetBandIndex	724
11.9	ST_Stats	726
11.9.1	ST_Count	726
11.9.2	ST_CountAgg	726
11.9.3	ST_Histogram	728
11.9.4	ST_Quantile	729
11.9.5	ST_SummaryStats	731
11.9.6	ST_SummaryStatsAgg	733

11.9.7	<code>ST_ValueCount</code>	735
11.10	Raster Inputs	737
11.10.1	<code>ST_RastFromWKB</code>	737
11.10.2	<code>ST_RastFromHexWKB</code>	738
11.11	Geometric Functions	739
11.11.1	<code>ST_AsBinary/ST_AsWKB</code>	739
11.11.2	<code>ST_AsHexWKB</code>	739
11.11.3	<code>ST_AsGDALRaster</code>	740
11.11.4	<code>ST_AsJPEG</code>	741
11.11.5	<code>ST_AsPNG</code>	742
11.11.6	<code>ST_AsTIFF</code>	743
11.12	Map Algebra	744
11.12.1	<code>ST_Clip</code>	744
11.12.2	<code>ST_ColorMap</code>	748
11.12.3	<code>ST_Grayscale</code>	751
11.12.4	<code>ST_Intersection</code>	753
11.12.5	<code>ST_MapAlgebra</code> (callback function version)	755
11.12.6	<code>ST_MapAlgebra</code> (expression version)	761
11.12.7	<code>ST_MapAlgebraExpr</code>	764
11.12.8	<code>ST_MapAlgebraExpr</code>	766
11.12.9	<code>ST_MapAlgebraFct</code>	771
11.12.10	<code>ST_MapAlgebraFct</code>	775
11.12.11	<code>ST_MapAlgebraFctNgb</code>	779
11.12.12	<code>ST_Reclass</code>	781
11.12.13	<code>ST_Union</code>	783
11.13	Neighborhood Functions	784
11.13.1	<code>ST_Distinct4ma</code>	784
11.13.2	<code>ST_InvDistWeight4ma</code>	785
11.13.3	<code>ST_Max4ma</code>	786
11.13.4	<code>ST_Mean4ma</code>	787
11.13.5	<code>ST_Min4ma</code>	789
11.13.6	<code>ST_MinDist4ma</code>	790
11.13.7	<code>ST_Range4ma</code>	791
11.13.8	<code>ST_StdDev4ma</code>	792
11.13.9	<code>ST_Sum4ma</code>	793
11.14	Topographic Functions	794
11.14.1	<code>ST_Aspect</code>	794
11.14.2	<code>ST_HillShade</code>	795
11.14.3	<code>ST_Roughness</code>	797

11.14.	\$T_Slope	798
11.14.	ST_TPI	800
11.14.	6T_TRI	800
11.15.	Box3D	801
11.15.	ST_ConvexHull	802
11.15.	ST_DumpAsPolygons	803
11.15.	ST_Envelope	804
11.15.	ST_MinConvexHull	804
11.15.	6T_Polygon	806
11.16.	&&	807
11.16.	&<	808
11.16.	&>	808
11.16.	4	809
11.16.	@	810
11.16.	6=	810
11.16.	7	811
11.17.	\$T_Contains	811
11.17.	ST_ContainsProperly	812
11.17.	ST_Covers	813
11.17.	ST_CoveredBy	814
11.17.	ST_Disjoint	815
11.17.	6T_Intersects	816
11.17.	ST_Overlaps	817
11.17.	8T_Touches	818
11.17.	9T_SameAlignment	819
11.17.	\$0_NotSameAlignmentReason	820
11.17.	ST_Within	821
11.17.	\$Z_DWithin	822
11.17.	\$F_DFullyWithin	823
11.18.	Master Tips	824
11.18.	Out-DB Rasters	824
11.18.1.	Directory containing many files	824
11.18.1.	Maximum Number of Open Files	824
11.18.1.2.	Maximum number of open files for the entire system	825
11.18.1.2.	Maximum number of open files per process	825

12 PostGIS Extras	827
12.1	827
12.1.1	827
12.1.2	828
12.1.2.1stdaddr	828
12.1.3	828
12.1.3.1rules table	828
12.1.3.2lex table	831
12.1.3.3gaz table	832
12.1.4	832
12.1.4.1debug_standardize_address	832
12.1.4.2parse_address	834
12.1.4.3standardize_address	835
12.2 TIGER	836
12.2.1 Drop_Indexes_Generate_Script	837
12.2.2 Drop_Nation_Tables_Generate_Script	838
12.2.3 Drop_State_Tables_Generate_Script	838
12.2.4 Geocode	839
12.2.5 Geocode_Intersection	842
12.2.6 Get_Geocode_Setting	843
12.2.7 Get_Tract	844
12.2.8 Install_Missing_Indexes	845
12.2.9 Loader_Generate_Census_Script	845
12.2.10 Loader_Generate_Script	847
12.2.11 Loader_Generate_Nation_Script	849
12.2.12 Missing_Indexes_Generate_Script	850
12.2.13 Normalize_Address	851
12.2.14 Page_Normalize_Address	852
12.2.15 Print_Addy	854
12.2.16 Reverse_Geocode	855
12.2.17 Topology_Load_Tiger	857
12.2.18 Set_Geocode_Setting	859
13 PostGIS Special Functions Index	861
13.1 PostGIS Aggregate Functions	861
13.2 PostGIS Window Functions	862
13.3 PostGIS SQL-MM Compliant Functions	862
13.4 PostGIS Geography Support Functions	866
13.5 PostGIS Raster Support Functions	868

13.6	PostGIS Geometry / Geography / Raster Dump Functions	873
13.7	PostGIS Box Functions	874
13.8	PostGIS Functions that support 3D	875
13.9	PostGIS Curved Geometry Support Functions	881
13.10	PostGIS Polyhedral Surface Support Functions	884
13.11	PostGIS Function Support Matrix	887
13.11	New, Enhanced or changed PostGIS Functions	897
13.12	PostGIS Functions new or enhanced in 3.5	897
13.12	PostGIS Functions new or enhanced in 3.4	898
13.12	PostGIS Functions new or enhanced in 3.3	899
13.12	PostGIS Functions new or enhanced in 3.2	900
13.12	PostGIS Functions new or enhanced in 3.1	901
13.12	PostGIS Functions new or enhanced in 3.0	902
13.12	PostGIS Functions new or enhanced in 2.5	903
13.12	PostGIS Functions new or enhanced in 2.4	904
13.12	PostGIS Functions new or enhanced in 2.3	905
13.12	PostGIS Functions new or enhanced in 2.2	907
13.12	PostGIS Functions new or enhanced in 2.1	910
13.12	PostGIS Functions new or enhanced in 2.0	911
13.12	PostGIS Functions new or enhanced in 1.5	917
13.12	PostGIS Functions new or enhanced in 1.4	918
13.12	PostGIS Functions new or enhanced in 1.3	919
14	Reporting Problems	920
14.1	Reporting Software Bugs	920
14.2	Reporting Documentation Issues	920
A	Appendix	922
A.1	PostGIS 3.4.0	922
A.1.1	New features	922
A.1.2	Enhancements	923
A.1.3	Breaking Changes	923

Abstract

PostGIS 是 PostgreSQL 数据库的 GIS 扩展，它提供了 GIS 功能。PostGIS 使用 GiST 和 R-Tree 索引，GIS 应用广泛。

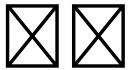


PostGIS 3.5.0alpha1 发布。



PostGIS 3.5.0alpha1 发布。PostGIS 是 PostgreSQL 数据库的 GIS 扩展，它提供了 GIS 功能。PostGIS 使用 GiST 和 R-Tree 索引，GIS 应用广泛。
<https://postgis.net>

Chapter 1



PostGIS is a spatial extension for the PostgreSQL relational database that was created by Refractions Research Inc, as a spatial database technology research project. Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom software development.

PostGIS is now a project of the OSGeo Foundation and is developed and funded by many FOSS4G developers and organizations all over the world that gain great benefit from its functionality and versatility.

The PostGIS project development group plans on supporting and enhancing PostGIS to better support a range of important GIS functionality in the areas of OGC and SQL/MM spatial standards, advanced topological constructs (coverages, surfaces, networks), data source for desktop user interface tools for viewing and editing GIS data, and web-based access tools.

1.1 Contributors

PostGIS (Project Steering Committee; PSC) PostGIS, CI, documentation. PSC, PostGIS, CI, documentation, PSC API PostGIS

Raúl Marín Rodríguez MVT support, Bug fixing, Performance and stability improvements, GitHub curation, alignment of PostGIS with PostgreSQL releases

Regina Obe CI and website maintenance, Windows production and experimental builds, documentation, alignment of PostGIS with PostgreSQL releases, X3D support, TIGER geocoder support, management functions.

Darafei Praliaskouski Index improvements, bug fixing and geometry/geography function improvements, SFCGAL, raster, GitHub curation, and ci maintenance.


Paul Ramsey 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.

1.2 -

 (**Nicklas Avén**)  (3D ) , TWKB(Tiny WKB) 
 ()

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

1.3 -






 (**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**) , GDAL , 

 (**Olivier Courtin**) XML(KML, GML)/GeoJSON , 3D 

 (**Chris Hodgson**)  PSC . , , OSGeo 



 (**Mateusz Loskot**) PostGIS  CMake , ,  API


 (**Kevin Neufeld**)  PSC . , , PostGIS 


 (**Dave Blasby**) PostGIS . , 


 (**Jeff Lounsbury**) shapefile /PostGIS 

 (**Mark Leslie**) . , shapefile GUI 

 (**Pierre Racine**) Architect of PostGIS raster implementation. Raster overall architecture, prototyping, programming support

 (**David Zwarg**)  () 

1.4

Alex Bodnaru	Gino Lucrezi	Matthias Bay
Alex Mayrhofer	Greg Troxel	Maxime Guillaud
Andrea Peri	Guillaume Lelarge	Maxime van Noppen
Andreas Forø Tollefsen	Giuseppe Broccolo	Maxime Schoemans
Andreas Neumann	Han Wang	Michael Fuhr
Andrew Gierth	Hans Lemuet	Mike Toews
Anne Ghisla	Haribabu Kommi	Nathan Wagner
Antoine Bajolet	Havard Tveite	Nathaniel Clay
Arthur Lesuisse	IIDA Tetsushi	Nikita Shulga
Artur Zakirov	Ingvild Nystuen	Norman Vine
Barbara Phillipot	Jackie Leng	Patricia Tozer
Ben Jubb	James Addison	Rafal Magda
Bernhard Reiter	James Marca	Ralph Mason
Björn Esser	Jan Katins	Rémi Cura
Brian Hamlin	Jan Tojnar	Richard Greenwood
Bruce Rindahl	Jason Smith	Robert Coup
Bruno Wolff III	Jeff Adams	Roger Crew
Bryce L. Nordgren	Jelte Fennema	Ron Mayer
Carl Anderson	Jim Jones	Sam Peters
Charlie Savage	Joe Conway	Sebastiaan Couwenberg
Chris Mayo	Jonne Savolainen	Sergei Shoulbakov
Christian Schroeder	Jose Carlos Martinez Llari	Sergey Fedoseev
Christoph Berg	Jörg Habenicht	Shinichi Sugiyama
Christoph Moench-Tegeder	Julien Rouhaud	Shoab Burq
Dane Springmeyer	Kashif Rasul	Silvio Grosso
Dapeng Wang	Klaus Foerster	Stefan Corneliu Petrea
Daryl Herzmann	Kris Jurka	Steffen Macke
Dave Fuhry	Laurenz Albe	Stepan Kuzmin
(David Zwarg)	Lars Roessiger	Stephen Frost
(David Zwarg)	Leo Hsu	Steven Ottens
(David Zwarg)	Loic Dachary	Talha Rizwan
Dmitry Vasilyev	Luca S. Percich	Teramoto Ikuhiro
Eduin Carrillo	Lucas C. Villa Real	Tom Glancy
Esteban Zimanyi	Maria Arias de Reyna	Tom van Tilburg
Eugene Antimirov	Marc Ducobu	Victor Collod
Even Rouault	Mark Sondheim	Vincent Bre
Florian Weimer	Markus Schaber	Vincent Mora
Frank Warmerdam	Markus Wanner	Vincent Picavet
George Silva	Matt Amos	Volf Tomáš
Gerald Fenoy	Matt Bretl	Zuo Chenwei

PostGIS, ,

- [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](#)
- [HighGo](#)
- [Hunter Systems Group](#)
- [INIA-CSIC](#)
- [ISciences, LLC](#)
- [Kontur](#)
- [Lidwala Consulting Engineers](#)
- [LISAssoft](#)
- [Logical Tracking & Tracing International AG](#)
- [Maponics](#)
- [Michigan Tech Research Institute](#)
- [Natural Resources Canada](#)
- [Norwegian Forest and Landscape Institue](#)
- [Norwegian Institute of Bioeconomy Research \(NIBIO\)](#)
- [OSGeo](#)
- [Oslandia](#)
- [Palantir Technologies](#)
- [Paragon Corporation](#)
- [R3 GIS](#)
- [Refractions Research](#)
- [Regione Toscana - SITA](#)
- [Safe Software](#)
- [Sirius Corporation plc](#)
- [Stadt Uster](#)
- [UC Davis Center for Vectorborne Diseases](#)
- [Université Laval](#)
- [U.S. Department of State \(HIU\)](#)
- [Zonar Systems](#)

PostGIS 3.5.0alpha1 is a pre-release version of the PostGIS project. It is intended for testing and feedback. The release schedule is not guaranteed. The PostGIS project is a community effort and we appreciate your contributions. For more information, please visit the PostGIS website.

PostGIS 2.0.0 is a stable release of the PostGIS project. It is supported by PledgBank. For more information, please visit the PostGIS website.

postgistopology - 10 toTopGeometry
250

postgis64windows - 20 64 PostGIS 100
PostgreSQL PostGIS 2.0.1 64

The **GEOS** geometry operations 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.

The **PROJ** cartographic projection library

PostgreSQL DBMS - PostGIS
PostgreSQL, GiST, SQL

Chapter 2

PostGIS

PostGIS

2.1

```
tar -xvzf postgis-3.5.0alpha1.tar.gz
cd postgis-3.5.0alpha1
./configure
make
make install
```

PostGIS, PostGIS (Section 3.3) (Section 3.4)

2.2

Note



This section includes general compilation instructions, if you are compiling for Windows etc or another OS, you may find additional more detailed help at [PostGIS User contributed compile guides](#) and [PostGIS Dev Wiki](#).
 Pre-Built Packages for various OS are listed in [PostGIS Pre-built Packages](#)
[Stackbuilder](#) [PostGIS Windows download site](#)
 1~2 [very bleeding-edge windows experimental builds](#). [PostGIS](#)

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 3.5.0alpha1 requires full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 12 - 17. Earlier versions of PostgreSQL are not supported.

Refer to the PostgreSQL installation guides if you haven't already installed PostgreSQL. <https://www.postgre>

Note

GEOS requires PostgreSQL to be installed before building PostGIS. The build process will fail if PostgreSQL is not installed.



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

The build process will fail if PostgreSQL is not installed. The build process will fail if PostgreSQL is not installed. The build process will fail if PostgreSQL is not installed.

PostGIS requires PostgreSQL to be installed before building PostGIS. The build process will fail if PostgreSQL is not installed.

2.2.1

PostGIS source code can be downloaded from <https://download.osgeo.org/postgis/source/postgis-3.5.0alpha1.tar.gz>.

```
wget https://download.osgeo.org/postgis/source/postgis-3.5.0alpha1.tar.gz
tar -xvzf postgis-3.5.0alpha1.tar.gz
cd postgis-3.5.0alpha1
```

Alternatively, you can clone the PostGIS source code from the repository.

Using `svn`, checkout the trunk: <http://svn.osgeo.org/postgis/trunk/>

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

Alternatively, you can clone the PostGIS source code from the repository.

```
./configure
```

2.2.2

PostGIS requires the following dependencies to be installed:

Dependencies:

- PostgreSQL 12 - 17. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from <https://www.postgresql.org>. For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to <https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS>
- GNU C compiler (gcc). PostGIS requires ANSI C compatible compiler gcc.
- GNU Make (make). GNU make or make is required. `make -v` should output GNU Makefile.
- Proj reprojection library. Proj 6.1 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from <https://proj.org/>.
- GEOS geometry library, version 3.8.0 or greater, but GEOS 3.12+ 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 or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGM and ST_GeomFromKML). LibXML2 is available for download from <https://gitlab.gnome.org/GNOME/libxml2/-/releases>.
- JSON-C 0.9. JSON-C ST_GeomFromGeoJson GeoJSON. JSON-C <https://github.com/json-c/json-c/releases/>.
- GDAL, version 2+ is required 3+ is preferred. This is required for raster support. <https://gdal.org/download.html>.
- PostgreSQL. PostgreSQL <http://trac.osgeo.org/postgis/ticket/635>.

2.2.2

- Section 2.1
- shapefile shp2pgsql-gui GTK(GTK+2.0, 2.8+). <http://www.gtk.org/>.
- SFCGAL, version 1.3.1 (or higher), 1.4.1 or higher is recommended and required to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS of Chapter 8. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST_Intersection or ST_Area, for instance). A PostgreSQL configuration variable postgis.backend allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: <https://sfcgal.org> <https://gitlab.com/sfcgal/SFCGAL/>).
- In order to build the Section 12.1 you will also need PCRE <http://www.pcre.org> (which generally is already installed on nix systems). Section 12.1 will automatically be built if it detects a PCRE library, or you pass in a valid --with-pcre-dir=/path/to/pcre during configure.
- To enable ST_AsMVT protobuf-c library 1.1.0 or higher (for usage) and the protoc-c compiler (for building) are required. Also, pkg-config is required to verify the correct minimum version of protobuf-c. See [protobuf-c](#). By default, Postgis will use Wagyu to validate MVT polygons faster which requires a c++11 compiler. It will use CXXFLAGS and the same compiler as the PostgreSQL installation. To disable this and use GEOS instead use the --without-wagyu during the configure step.
- CUnit(CUnit). <http://cunit.sourceforge.net/>
- DocBook(xsltproc) DocBook <http://www.docbook.org/>
- DBLatex(dblatex) PDF DBLatex <http://dblatex.sourceforge.net/>
- ImageMagick(convert) ImageMagick <http://www.imagemagick.org/>

2.2.3

Makefile

./configure

PostGIS

--help --help=short

- with-library-minor-version** Starting with PostGIS 3.0, the library files generated by default will no longer have the minor version as part of the file name. This means all PostGIS 3 libs will end in `postgis-3`. This was done to make `pg_upgrade` easier, with downside that you can only install one version PostGIS 3 series in your server. To get the old behavior of file including the minor version: e.g. `postgis-3.0` add this switch to your configure statement.
- prefix=PREFIX** PostGIS SQL PostgreSQL. PostgreSQL.



Caution

PostgreSQL. PostgreSQL. <http://trac.osgeo.org/postgis/ticket/635>.

- with-pgconfig=FILE** PostgreSQL PostGIS PostgreSQL `pg_config`. PostGIS PostgreSQL `--with-pgconfig=/path/to/pg_config`.
- with-gdalconfig=FILE** GDAL GDAL `gdal-config`. PostGIS GDAL `--with-gdalconfig=/path/to/gdal-config`.
- with-geosconfig=FILE** GEOS GEOS `geos-config`. PostGIS GEOS `--with-geosconfig=/path/to/geos-config`.
- with-xml2config=FILE** LibXML is the library required for doing GeomFromKML/GML processes. It normally is found if you have libxml installed, but if not or you want a specific version used, you'll need to point PostGIS at a specific xml2 - config confi file to enable software installations to locate the LibXML installation directory. Use this parameter (`>--with-xml2config=/path/to/xml2-config`) to manually specify a particular LibXML installation that PostGIS will build against.
- with-projdir=DIR** Proj4 PostGIS Proj4 `--with-projdir=/path/to/projdir`.
- with-libiconv=DIR** iconv
- with-jsondir=DIR** JSON-C MIT JSON, PostGIS ST_GeomFromJSON `--with-jsondir=/path/to/jsondir`.
- with-pcredir=DIR** PCRE BSD, address_standardizer `--with-pcredir=/path/to/pcredir`.
- with-gui** GUI (GTK+2.0). `shp2pgsql-gui` `shp2pgsql`.
- without-raster**
- without-topology** Disable topology support. There is no corresponding library as all logic needed for topology is in `postgis-3.5.0alpha1` library.
- with-gettext=no** PostGIS `gettext`. `gettext`. <http://trac.osgeo.org/postgis/ticket/748>. `gettext` GUI.

--with-sfcgal=PATH Path to PostGIS sfcgal. PATH sfcgal-config

--without-phony-revision Disable updating postgis_revision.h to match current HEAD of the git repository.



Note

PostGIS SVN, configure

./autogen.sh

configure PostGIS

tar PostGIS configure ./autogen.sh

2.2.4

Makefile PostGIS

make

"PostGIS was built successfully. Ready to install."

As of PostGIS v1.4.0, 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 comments files raster_comments.sql, topology_comments.sql 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.

make comments

PostGIS 2.0. xsltproc, doc topology_cheatsheet.html, tiger_geocoder_cheatsheet.html, raster_cheatsheet.html, postgis_cheatsheet.html 4

html pdf [PostGIS / PostgreSQL Study Guides](#)

make cheatsheets

2.2.5 PostGIS Extensions

PostgreSQL 9.1 PostGIS extentions

function descriptions. docbook

make comments

tar tar comments

PostgreSQL 9.1 extensions

```
cd extensions
cd postgis
make clean
```

```

make
export PGUSER=postgres #overwrite psql variables
make check #to test before install
make install
# to test extensions
make check RUNTESTFLAGS=-extension

```



Note

make check uses psql to run tests and as such can use psql environment variables. Common ones useful to override are PGUSER,PGPORT, and PHOST. Refer to [psql environment variables](#)

extension files OS specific PostGIS binaries. PostGIS binaries are installed to the same location as PostgreSQL binaries.

extension files PostgreSQL share/extension/PostgreSQL share/extension/postgis extensions PostGIS binaries.

- extension files postgis.control, postgis_topology.control.
- extension files /sql/postgis/sql/*.sql, extensions/postgis/sql/*.sql, extensions/postgis_topology/sql/*.sql

Once you do that, you should see postgis, postgis_topology as available extensions in PgAdmin -> extensions.

psql command to list available extensions.

```

SELECT name, default_version,installed_version
FROM pg_available_extensions WHERE name LIKE 'postgis%' or name LIKE 'address%';

```

name	default_version	installed_version
address_standardizer	3.5.0alpha1	3.5.0alpha1
address_standardizer_data_us	3.5.0alpha1	3.5.0alpha1
postgis	3.5.0alpha1	3.5.0alpha1
postgis_raster	3.5.0alpha1	3.5.0alpha1
postgis_sfcgal	3.5.0alpha1	
postgis_tiger_geocoder	3.5.0alpha1	3.5.0alpha1
postgis_topology	3.5.0alpha1	

(6 rows)

extension files, installed_version files. PostGIS extension files. PgAdmin III 1.14 extensions files.

extension files pgAdmin extension files sql files postgis extension files:

```

CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster;
CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION fuzzystmatch; --needed for postgis_tiger_geocoder
--optional used by postgis_tiger_geocoder, or can be used standalone
CREATE EXTENSION address_standardizer;

```

```
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION postgis_topology;
```

PSQL

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

```
List of installed extensions
-[ RECORD 1 ]-----
Name          | postgis
Version       | 3.5.0alpha1
Schema        | public
Description   | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----
Name          | postgis_raster
Version       | 3.0.0dev
Schema        | public
Description   | PostGIS raster types and functions
-[ RECORD 3 ]-----
Name          | postgis_tiger_geocoder
Version       | 3.5.0alpha1
Schema        | tiger
Description   | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 4 ]-----
Name          | postgis_topology
Version       | 3.5.0alpha1
Schema        | topology
Description   | PostGIS topology spatial types and functions
```

Warning



spatial_ref_sys, layer, topology extensions. The postgis and postgis_topology extension are installed. PostgreSQL 12.1. PostGIS 2.0.1 uses the srid extension. The trac extension is not installed. CREATE EXTENSION is used to install PostgreSQL extensions.

PostGIS 3.5.0alpha1 installation, raster_upgrade_22_minor.sql, topology_upgrade_22_minor.sql.

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

2.2.6

PostGIS installation, make check.

make check

PostgreSQL installation, make check.

**Note**

PostgreSQL, GEOS, Proj4, LD_LIBRARY_PATH.

**Caution**

make check PATH PGPORT. PostgreSQL **--with-pgconfig** PATH.

If successful, make check will produce the output of almost 500 tests. The results will look similar to the following (numerous lines omitted below):

```
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

```

postgis_tiger_geocoder address_standardizer PostgreSQL (installcheck) make install

address_standardizer:

```

cd extensions/address_standardizer
make install
make installcheck

```

:


```

===== dropping database "contrib_regression" =====
DROP DATABASE
===== creating database "contrib_regression" =====
CREATE DATABASE
ALTER DATABASE
===== running regression test queries =====
test test-init-extensions      ... ok
test test-parseaddress         ... ok
test test-standardize_address_1 ... ok
test test-standardize_address_2 ... ok

=====
All 4 tests passed.
=====

```

TIGER, PostgreSQL, PostGIS, fuzziystrmatch, address_standardizer, address_standardizer.

```

cd extensions/postgis_tiger_geocoder
make install
make installcheck

```

:

```

===== dropping database "contrib_regression" =====
DROP DATABASE
===== creating database "contrib_regression" =====
CREATE DATABASE
ALTER DATABASE
===== installing fuzziystrmatch =====
CREATE EXTENSION
===== installing postgis =====
CREATE EXTENSION
===== installing postgis_tiger_geocoder =====
CREATE EXTENSION
===== installing address_standardizer =====
CREATE EXTENSION
===== running regression test queries =====
test test-normalize_address    ... ok
test test-pagc_normalize_address ... ok

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

```

2.2.7

PostGIS.

make install

--prefix PostgreSQL.

- (loader) [prefix]/bin.
- postgis.sql SQL [prefix]/share/contrib.
- PostGIS [prefix]/lib.

Run `postgis_comments.sql`, `raster_comments.sql` and `make comments` to install the comments, and `sql` to install the SQL.

make comments-install



Note

xsitproc `postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql`.

2.3 Installing the address_standardizer extension

`address_standardizer` is included in PostGIS 2.2 and later. See Section 12.1 for details.

Normalize Address is a PostGIS extension that uses the TIGER geocoder (geocoder) to standardize addresses. See Section 2.4.2 for details. It also includes a building block extension.

PCRE is required. PCRE is available at <http://www.pcre.org>. See Section 2.2.3 for details. PCRE can be installed using `--with-pcre=/path/to/pcre` or `PCRE include lib`.

PostGIS 2.1 includes `address_standardizer` extension. `CREATE EXTENSION` is used to install it.

The following SQL creates the extension:

```
CREATE EXTENSION address_standardizer;
```

Rules, gaz, and lex are also installed.

```
SELECT num, street, city, state, zip
FROM parse_address('1 Devonshire Place PH301, Boston, MA 02109');
```

The output is:

num	street	city	state	zip
1	Devonshire Place PH301	Boston	MA	02109

2.4 Installing, Upgrading Tiger Geocoder, and loading data

Extras like Tiger geocoder may not be packaged in your PostGIS distribution. If you are missing the tiger geocoder extension or want a newer version than what your install comes with, then use the `share/extension/postgis_tiger_geocoder.*` files from the packages in **Windows Unreleased Versions** section for your version of PostgreSQL. Although these packages are for windows, the `postgis_tiger_geocoder` extension files will work on any OS since the extension is an SQL/plpgsql only extension.

2.4.1 Tiger Geocoder Enabling your PostGIS database

1. These directions assume your PostgreSQL installation already has the `postgis_tiger_geocoder` extension installed.
2. PSQL, pgAdmin or SQL client. `fuzzystrmatch`.

```
CREATE EXTENSION postgis;
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION postgis_tiger_geocoder;
--this one is optional if you want to use the rules based standardizer ( ←
    pagc_normalize_address)
CREATE EXTENSION address_standardizer;
```

`postgis_tiger_geocoder`:

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

`tiger.loader_platform` `tiger.loader_variables`

3. SQL:

```
SELECT na.address, na.streetname,na.streotypeabbrev, na.zip
      FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
```

Result:

address	streetname	streotypeabbrev	zip
1	Devonshire	Pl	02109

4. `tiger.loader_platform` `sh` (convention) `debbie`.

```
INSERT INTO tiger.loader_platform(os, declare_sect, pgbin, wget, unzip_command, psql,
    path_sep,
    loader, environ_set_command, county_process_command)
SELECT 'debbie', declare_sect, pgbin, wget, unzip_command, psql, path_sep,
    loader, environ_set_command, county_process_command
FROM tiger.loader_platform
WHERE os = 'sh';
```

`debbie` `pg`, `unzip`, `shp2pgsql`, `PSQL` `declare_sect`

`loader_platform` (common case)

5. As of PostGIS 2.4.1 the Zip code-5 digit tabulation area `zcta5` load step was revised to load current `zcta5` data and is part of the **Loader_Generate_Nation_Script** when enabled. It is turned off by default because it takes quite a bit of time to load (20 to 60 minutes), takes up quite a bit of disk space, and is not used that often.

To enable it, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta520';
```

If present the **Geocode** function can use it if a boundary filter is added to limit to just zips in that boundary. The **Reverse_Geocode** function uses it if the returned address is missing a zip, which often happens with highway reverse geocoding.

6. On the PC, create a directory `gisdata` in the `staging_fold` directory. Copy the `TIGER` files from the `temp` directory to the `gisdata` directory. Copy the `tiger.loader_variables` file to the `staging_fold` directory.

7. Copy the `staging_fold` directory to the `temp` directory. Copy the `TIGER` files from the `temp` directory to the `gisdata` directory.

8. Then run the `Loader_Generate_Nation_Script` SQL function make sure to use the name of your custom profile and copy the script to a `.sh` or `.bat` file. So for example to build the nation load:

```
psql -c "SELECT Loader_Generate_Nation_Script('debbie');" -d geocoder -tA > /gisdata/nation_script_load.sh
```

9. Run the generated nation load commandline scripts.

```
cd /gisdata
sh nation_script_load.sh
```

10. After you are done running the nation script, you should have three tables in your `tiger_data` schema and they should be filled with data. Confirm you do by doing the following queries from `psql` or `pgAdmin`

```
SELECT count(*) FROM tiger_data.county_all;
```

```
count
-----
  3235
(1 row)
```

```
SELECT count(*) FROM tiger_data.state_all;
```

```
count
-----
    56
(1 row)
```

This will only have data if you marked `zcta5` to be loaded

```
SELECT count(*) FROM tiger_data.zcta5_all;
```

```
count
-----
 33931
(1 row)
```

11. By default the tables corresponding to `bg`, `tract`, `tabblock20` are not loaded. These tables are not used by the geocoder but are used by folks for population statistics. If you wish to load them as part of your state loads, run the following statement to enable them.

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name IN ('tract', 'bg', 'tabblock20');
```

Alternatively you can load just these tables after loading state data using the `Loader_Generate_Census`

12. For each state you want to load data for, generate a state script `Loader_Generate_Script`.



Warning

DO NOT Generate the state script until you have already loaded the nation data, because the state script utilizes county list loaded by nation script.

```
13. psql -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'debbie')" -d geocoder -tA > /
gisdata/ma_load.sh
```

```
14. cd /gisdata
sh ma_load.sh
```

```
15. SELECT install_missing_indexes();
vacuum (analyze, verbose) tiger.addr;
vacuum (analyze, verbose) tiger.edges;
vacuum (analyze, verbose) tiger.faces;
vacuum (analyze, verbose) tiger.featnames;
vacuum (analyze, verbose) tiger.place;
vacuum (analyze, verbose) tiger.cousub;
vacuum (analyze, verbose) tiger.county;
vacuum (analyze, verbose) tiger.state;
vacuum (analyze, verbose) tiger.zcta5;
vacuum (analyze, verbose) tiger.zip_lookup_base;
vacuum (analyze, verbose) tiger.zip_state;
vacuum (analyze, verbose) tiger.zip_state_loc;
```

2.4.2 TIGER Data Loading

The `Normalize Address` script is used to standardize the address data. It uses the `address_standardizer` tool, described in Section 2.3, to standardize the addresses.

The `postgis_tiger_geocoder` script uses the `Normalize Address` script to generate the `Pagc Normalize Address` script. This script is used to load the TIGER data into the database. The `TIGER` data is loaded into the `rules table` (`tiger.pagc_rules`), `gaz table` (`tiger.pagc_gaz`), and `lex table` (`tiger.pagc_lex`).

2.4.3 Required tools for tiger data loading

The `Drop_State_Tables_Generate_Script` script is used to drop the state tables and generate the state script.

The following tools are required for loading the TIGER data:

- `unzip` (Unix) or `7-zip` (Windows).
<http://www.7-zip.org/>

- PostGIS shp2pgsql
- wget Unix/Linux <http://gnuwin32.sourceforge.net/packages/wget.htm>.

If you are upgrading from tiger_2010, you'll need to first generate and run **Drop_Nation_Tables_Generate_Sc**. Before you load any state data, you need to load the nation wide data which you do with **Loader_Generate_Na**. Which will generate a loader script for you. **Loader_Generate_Nation_Script** is a one-time step that should be done for upgrading (from a prior year tiger census data) and for new installs.

Loader_Generate_Script. **Drop_Nation_Tables_Generate_Script**. **Loader_Generate_Nation_Script**.

Install_Missing_Indexes:

```
SELECT install_missing_indexes();
```

Geocode.

2.4.4 Upgrading your Tiger Geocoder Install and Data

First upgrade your postgis_tiger_geocoder extension as follows:

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

Drop_Nation_Tables_Generate_Script

```
SELECT drop_nation_tables_generate_script();
```

drop SQL

SELECT nation load **Loader_Generate_Nation_Script**

```
SELECT loader_generate_nation_script('windows');
```

unix/linux

```
SELECT loader_generate_nation_script('sh');
```

Refer to Section 2.4.1 for instructions on how to run the generate script. This only needs to be done once.



Note

You can have a mix of different year state tables and can upgrade each state separately. Before you upgrade a state you first need to drop the prior year state tables for that state using **Drop_State_Tables_Generate_Script**.

2.5

1. PostgreSQL 12. PostgreSQL (Linux) PostgreSQL PostgreSQL 12 PostgreSQL psql:

```
SELECT version();
```

RPM rpm: **rpm -qa | grep postgresql**

2. PostGIS

```
SELECT postgis_full_version();
```

PostgreSQL, Proj4 GEOS

1. postgis_config.hh. POSTGIS_PGSQL_VERSION, POSTGIS_PROJ_VERSION and POSTGIS_GEOS_VERSION

Chapter 3

PostGIS Administration

3.1 Performance Tuning

Tuning for PostGIS performance is much like tuning for any PostgreSQL workload. The only additional consideration is that geometries and rasters are usually large, so memory-related optimizations generally have more of an impact on PostGIS than other types of PostgreSQL queries.

For general details about optimizing PostgreSQL, refer to [Tuning your PostgreSQL Server](#).

For PostgreSQL 9.4+ configuration can be set at the server level without touching `postgresql.conf` or `postgresql.auto.conf` by using the `ALTER SYSTEM` command.

```
ALTER SYSTEM SET work_mem = '256MB';
-- this forces non-startup configs to take effect for new connections
SELECT pg_reload_conf();
-- show current setting value
-- use SHOW ALL to see all settings
SHOW work_mem;
```

In addition to the Postgres settings, PostGIS has some custom settings which are listed in [Section 7.22](#).

3.1.1 Startup

These settings are configured in `postgresql.conf`:

`constraint_exclusion`

- Default: `partition`
- This is generally used for table partitioning. The default for this is set to “`partition`” which is ideal for PostgreSQL 8.4 and above since it will force the planner to only analyze tables for constraint consideration if they are in an inherited hierarchy and not pay the planner penalty otherwise.

`shared_buffers`

- Default: ~128MB in PostgreSQL 9.6
- Set to about 25% to 40% of available RAM. On windows you may not be able to set as high.

`max_worker_processes` This setting is only available for PostgreSQL 9.4+. For PostgreSQL 9.6+ this setting has additional importance in that it controls the max number of processes you can have for parallel queries.

- Default: 8
- Sets the maximum number of background processes that the system can support. This parameter can only be set at server start.

3.1.2 Runtime

work_mem - sets the size of memory used for sort operations and complex queries

- Default: 1-4MB
- Adjust up for large dbs, complex queries, lots of RAM
- Adjust down for many concurrent users or low RAM.
- If you have lots of RAM and few developers:

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

maintenance_work_mem - the memory size used for VACUUM, CREATE INDEX, etc.

- Default: 16-64MB
- Generally too low - ties up I/O, locks objects while swapping memory
- Recommend 32MB to 1GB on production servers w/lots of RAM, but depends on the # of concurrent users. If you have lots of RAM and few developers:

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

max_parallel_workers_per_gather

This setting is only available for PostgreSQL 9.6+ and will only affect PostGIS 2.3+, since only PostGIS 2.3+ supports parallel queries. 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.

- Default: 0
- Sets the maximum number of workers that can be started by a single Gather node. Parallel workers are taken from the pool of processes established by `max_worker_processes`. Note that the requested number of workers may not actually be available at run time. If this occurs, the plan will run with fewer workers than expected, which may be inefficient. Setting this value to 0, which is the default, disables parallel query execution.

3.2 Configuring raster support

If you enabled raster support you may want to read below how to properly configure it.

As of PostGIS 2.1.3, out-of-db rasters and all raster drivers are disabled by default. In order to re-enable these, you need to set the following environment variables `POSTGIS_GDAL_ENABLED_DRIVERS` and `POSTGIS_ENABLE_OUTDB_RASTERS` in the server environment. For PostGIS 2.2, you can use the more cross-platform approach of setting the corresponding Section [7.22](#).

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

Any other setting or no setting at all will disable out of db rasters.

In order to enable all GDAL drivers available in your GDAL install, set this environment variable as follows

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

If you want to only enable specific drivers, set your environment variable as follows:

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



Note

If you are on windows, do not quote the driver list

Setting environment variables varies depending on OS. For PostgreSQL installed on Ubuntu or Debian via apt-postgresql, the preferred way is to edit `/etc/postgresql/10/main/environment` where 10 refers to version of PostgreSQL and main refers to the cluster.

On windows, if you are running as a service, you can set via System variables which for Windows 7 you can get to by right-clicking on Computer->Properties Advanced System Settings or in explorer navigating to Control Panel\All Control Panel Items\System. Then clicking *Advanced System Settings ->Advanced->Environment Variables* and adding new system variables.

After you set the environment variables, you'll need to restart your PostgreSQL service for the changes to take effect.

3.3 ☒☒☒☒☒☒☒☒☒☒

3.3.1 Spatially enable database using EXTENSION

If you are using PostgreSQL 9.1+ and have compiled and installed the extensions/postgis modules, you can turn a database into a spatial one using the EXTENSION mechanism.

Core postgis extension includes geometry, geography, spatial_ref_sys and all the functions and comments. Raster and topology are packaged as a separate extension.

Run the following SQL snippet in the database you want to enable spatially:

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

3.3.2 Spatially enable database without using EXTENSION (discouraged)



Note

This is generally only needed if you cannot or don't want to get PostGIS installed in the PostgreSQL extension directory (for example during testing, development or in a restricted environment).

Adding PostGIS objects and function definitions into your database is done by loading the various sql files located in [prefix]/share/contrib as specified during the build phase.

The core PostGIS objects (geometry and geography types, and their support functions) are in the `postgis.sql` script. Raster objects are in the `rtpostgis.sql` script. Topology objects are in the `topology.sql` script.

For a complete set of EPSG coordinate system definition identifiers, you can also load the `spatial_ref_sys.sql` definitions file and populate the `spatial_ref_sys` table. This will permit you to perform `ST_Transform()` operations on geometries.

If you wish to add comments to the PostGIS functions, you can find them in the `postgis_comments.sql` script. Comments can be viewed by simply typing `\dd [function_name]` from a **psql** terminal window.

Run the following Shell commands in your terminal:

```
DB=[yourdatabase]
SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.4/

# 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 Upgrading spatial databases

Upgrading existing spatial databases can be tricky as it requires replacement or introduction of new PostGIS object definitions.

Unfortunately not all definitions can be easily replaced in a live database, so sometimes your best bet is a dump/reload process.

PostGIS provides a SOFT UPGRADE procedure for minor or bugfix releases, and a HARD UPGRADE procedure for major releases.

Before attempting to upgrade PostGIS, it is always worth to backup your data. If you use the `-Fc` flag to `pg_dump` you will always be able to restore the dump with a HARD UPGRADE.

3.4.1 Soft upgrade

If you installed your database using extensions, you'll need to upgrade using the extension model as well. If you installed using the old sql script way, you are advised to switch your install to extensions because the script way is no longer supported.

3.4.1.1 Soft Upgrade 9.1+ using extensions

If you originally installed PostGIS with extensions, then you need to upgrade using extensions as well. Doing a minor upgrade with extensions, is fairly painless.

If you are running PostGIS 3 or above, then you should use the [PostGIS_Extensions_Upgrade](#) function to upgrade to the latest version you have installed.

```
SELECT postgis_extensions_upgrade();
```

If you are running PostGIS 2.5 or lower, then do the following:

```
ALTER EXTENSION postgis UPDATE;
SELECT postgis_extensions_upgrade();
-- This second call is needed to rebundle postgis_raster extension
SELECT postgis_extensions_upgrade();
```

If you have multiple versions of PostGIS installed, and you don't want to upgrade to the latest, you can explicitly specify the version as follows:

```
ALTER EXTENSION postgis UPDATE TO "3.5.0alpha1";
ALTER EXTENSION postgis_topology UPDATE TO "3.5.0alpha1";
```

If you get an error notice something like:

```
No migration path defined for b'...' to 3.5.0alpha1
```

Then you'll need to backup your database, create a fresh one as described in Section 3.3.1 and then restore your backup on top of this new database.

If you get a notice message like:

```
Version "3.5.0alpha1" of extension "postgis" is already installed
```

Then everything is already up to date and you can safely ignore it. **UNLESS** you're attempting to upgrade from an development version to the next (which doesn't get a new version number); in that case you can append "next" to the version string, and next time you'll need to drop the "next" suffix again:

```
ALTER EXTENSION postgis UPDATE TO "3.5.0alpha1next";
ALTER EXTENSION postgis_topology UPDATE TO "3.5.0alpha1next";
```



Note

If you installed PostGIS originally without a version specified, you can often skip the reinstallation of postgis extension before restoring since the backup just has CREATE EXTENSION postgis and thus picks up the newest latest version during restore.



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. You can drop as follows:

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions

This section applies only to those who installed PostGIS not using extensions. If you have extensions and try to upgrade with this approach you'll get messages like:

```
can't drop b'...' because postgis extension depends on it
```

NOTE: if you are moving from PostGIS 1.* to PostGIS 2.* or from PostGIS 2.* prior to r7409, you cannot use this procedure but would rather need to do a **HARD UPGRADE**.

After compiling and installing (make install) you should find a set of *_upgrade.sql files in the installation folders. You can list them all with:

```
ls `pg_config --sharedir`/contrib/postgis-3.5.0alpha1/*_upgrade.sql
```

Load them all in turn, starting from postgis_upgrade.sql.

```
psql -f postgis_upgrade.sql -d your_spatial_database
```

The same procedure applies to raster, topology and sfcgal extensions, with upgrade files named rtpostgis_upgrade.sql, topology_upgrade.sql and sfcgal_upgrade.sql respectively. If you need them:

```
psql -f rtpostgis_upgrade.sql -d your_spatial_database
```

```
psql -f topology_upgrade.sql -d your_spatial_database
```

```
psql -f sfcgal_upgrade.sql -d your_spatial_database
```

You are advised to switch to an extension based install by running

```
psql -c "SELECT postgis_extensions_upgrade();"
```

**Note**

If you can't find the postgis_upgrade.sql specific for upgrading your version you are using a version too early for a soft upgrade and need to do a **HARD UPGRADE**.

The **PostGIS_Full_Version** function should inform you about the need to run this kind of upgrade using a "procs need upgrade" message.

3.4.2 Hard upgrade

By HARD UPGRADE we mean full dump/reload of postgis-enabled databases. You need a HARD UPGRADE when PostGIS objects' internal storage changes or when SOFT UPGRADE is not possible. The **Release Notes** appendix reports for each version whether you need a dump/reload (HARD UPGRADE) to upgrade.

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 at **Windows Hard upgrade**.

The Procedure is as follows:

1. Create a "custom-format" dump of the database you want to upgrade (let's call it olddb) include binary blobs (-b) and verbose (-v) output. The user can be the owner of the db, need not be postgres super account.

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

2. Do a fresh install of PostGIS in a new database -- we'll refer to this database as newdb. Please refer to Section 3.3.2 and Section 3.3.1 for instructions on how to do this.

The `spatial_ref_sys` entries found in your dump will be restored, but they will not override existing ones in `spatial_ref_sys`. This is to ensure that fixes in the official set will be properly propagated to restored databases. If for any reason you really want your own overrides of standard entries just don't load the `spatial_ref_sys.sql` file when creating the new db.

If your database is really old or you know you've been using long deprecated functions in your views and functions, you might need to load `legacy.sql` for all your functions and views etc. to properly come back. Only do this if `really_needed`. Consider upgrading your views and functions before dumping instead, if possible. The deprecated functions can be later removed by loading `uninstall_legacy.sql`.

3. Restore your backup into your fresh newdb database using `postgis_restore`. Unexpected errors, if any, will be printed to the standard error stream by `psql`. Keep a log of those.

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

Errors may arise in the following cases:

1. Some of your views or functions make use of deprecated PostGIS objects. In order to fix this you may try loading `legacy.sql` script prior to restore or you'll have to restore to a version of PostGIS which still contains those objects and try a migration again after porting your code. If the `legacy.sql` way works for you, don't forget to fix your code to stop using deprecated functions and drop them loading `uninstall_legacy.sql`.
2. Some custom records of `spatial_ref_sys` in dump file have an invalid SRID value. Valid SRID values are bigger than 0 and smaller than 999000. Values in the 999000.999999 range are reserved for internal use while values > 999999 can't be used at all. All your custom records with invalid SRIDs will be retained, with those > 999999 moved into the reserved range, but the `spatial_ref_sys` table would lose a check constraint guarding for that invariant to hold and possibly also its primary key (when multiple invalid SRIDS get converted to the same reserved SRID value).

In order to fix this you should copy your custom SRS to a SRID with a valid value (maybe in the 910000..910999 range), convert all your tables to the new srid (see [UpdateGeometrySRID](#)), delete the invalid entry from `spatial_ref_sys` and re-construct the check(s) with:

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

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

If you are upgrading an old database containing french **IGN** cartography, you will have probably SRIDs out of range and you will see, when importing your database, issues like this :

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

In this case, you can try following steps : first throw out completely the IGN from the sql which is resulting from `postgis_restore`. So, after having run :

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

run this command :

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

Create then your newdb, activate the required Postgis extensions, and insert properly the french system IGN with : [this script](#) After these operations, import your data :

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

Chapter 4

Data Management

4.1 GIS (OGC) Geometry

4.1.1 OGC Geometry

The Open Geospatial Consortium (OGC) developed the *Simple Features Access* standard (SFA) to provide a model for geospatial data. It defines the fundamental spatial type of **Geometry**, along with operations which manipulate and transform geometry values to perform spatial analysis tasks. PostGIS implements the OGC Geometry model as the PostgreSQL data types **geometry** and **geography**.

Geometry is an *abstract* type. Geometry values belong to one of its *concrete* subtypes which represent various kinds and dimensions of geometric shapes. These include the **atomic** types **Point**, **LineString**, **LinearRing** and **Polygon**, and the **collection** types **MultiPoint**, **MultiLineString**, **MultiPolygon** and **GeometryCollection**. The *Simple Features Access - Part 1: Common architecture v1.2.1* adds subtypes for the structures **PolyhedralSurface**, **Triangle** and **TIN**.

Geometry models shapes in the 2-dimensional Cartesian plane. The **PolyhedralSurface**, **Triangle**, and **TIN** types can also represent shapes in 3-dimensional space. The size and location of shapes are specified by their **coordinates**. Each coordinate has a X and Y **ordinate** value determining its location in the plane. Shapes are constructed from points or line segments, with points specified by a single coordinate, and line segments by two coordinates.

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.

Geometry values are associated with a **spatial reference system** indicating the coordinate system in which it is embedded. The spatial reference system is identified by the geometry SRID number. The units of the X and Y axes are determined by the spatial reference system. In **planar** reference systems the X and Y coordinates typically represent easting and northing, while in **geodetic** systems they represent longitude and latitude. SRID 0 represents an infinite Cartesian plane with no units assigned to its axes. See Section 4.5.

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

A geometry value may be **empty**. Empty values contain no vertices (for atomic geometry types) or no elements (for collections).

An important property of geometry values is their spatial **extent** or **bounding box**, which the OGC model calls **envelope**. This is the 2 or 3-dimensional box which encloses the coordinates of a geometry.

It is an efficient way to represent a geometry's extent in coordinate space and to check whether two geometries interact.

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.

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 the 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

A Point is a 0-dimensional geometry that represents a single location in coordinate space.

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

4.1.1.2 LineString

A LineString is a 1-dimensional line formed by a contiguous sequence of line segments. Each line segment is defined by two points, with the end point of one segment forming the start point of the next segment. An OGC-valid LineString has either zero or two or more points, but PostGIS also allows single-point LineStrings. LineStrings may cross themselves (self-intersect). A LineString is **closed** if the start and end points are the same. A LineString is **simple** if it does not self-intersect.

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

4.1.1.3 LinearRing

A LinearRing is a LineString which is both closed and simple. The first and last points must be equal, and the line must not self-intersect.

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

4.1.1.4 Polygon

A Polygon is a 2-dimensional planar region, delimited by an exterior boundary (the shell) and zero or more interior boundaries (holes). Each boundary is a [LinearRing](#).

```
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

A MultiPoint is a collection of Points.

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


4.1.1.6 MultiLineString

A MultiLineString is a collection of LineStrings. A MultiLineString is closed if each of its elements is closed.

```
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.

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

4.1.1.8 GeometryCollection

A GeometryCollection is a heterogeneous (mixed) collection of geometries.

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

4.1.1.9 PolyhedralSurface

A PolyhedralSurface is a contiguous collection of patches or facets which share some edges. Each patch is a planar Polygon. If the Polygon coordinates have Z ordinates then the surface is 3-dimensional.

```
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

A Triangle is a polygon defined by three distinct non-collinear vertices. Because a Triangle is a polygon it is specified by four coordinates, with the first and fourth being equal.

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

4.1.1.11 TIN

A TIN is a collection of non-overlapping **Triangles** representing a **Triangulated Irregular Network**.

```
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

The *ISO/IEC 13249-3 SQL Multimedia - Spatial* standard (SQL/MM) extends the OGC SFA to define Geometry subtypes containing curves with circular arcs. The SQL/MM types support 3DM, 3DZ and 4D coordinates.



Note

SQL-MM `CIRCULARSTRING` uses a tolerance of `1E-8`.

4.1.2.1 CircularString

`CIRCULARSTRING` is a subtype of `LINESTRING`. It is defined as a sequence of circular arcs. Each arc is defined by three points: the start point, the end point, and a third point that defines the curvature. The third point is the midpoint of the arc. The arcs are connected end-to-end. The first point of the first arc is the start point of the `CIRCULARSTRING`. The last point of the last arc is the end point of the `CIRCULARSTRING`.

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

4.1.2.2 CompoundCurve

`COMPOUNDCURVE` (compound curve) is a subtype of `CURVE`. It is defined as a sequence of `CIRCULARSTRING` and `LINESTRING` objects. The objects are connected end-to-end. The first point of the first object is the start point of the `COMPOUNDCURVE`. The last point of the last object is the end point of the `COMPOUNDCURVE`.

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

4.1.2.3 CurvePolygon

`CURVEPOLYGON` is a subtype of `POLYGON`. It is defined as a `COMPOUNDCURVE` that forms a closed loop. The start and end points of the `COMPOUNDCURVE` are the same. The interior of the loop is the area enclosed by the `COMPOUNDCURVE`.

PostGIS 1.4 `CURVEPOLYGON` uses a tolerance of `1E-8`.

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

Example: A `CurvePolygon` with the shell defined by a `CompoundCurve` containing a `CircularString` and a `LineString`, and a hole defined by a `CircularString`

```
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

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

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

4.1.2.5 MultiSurface

MULTISURFACE `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)))`.

```
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 OpenGIS WKB & WKT

OpenGIS `WKB` (Well-Known Binary) and `WKT` (Well-Known Text) are standard formats for representing spatial data. WKT is a text-based format, while WKB is a binary format. WKT is often used for data exchange and storage, while WKB is used for efficient storage and retrieval.

WKT (Well-Known Text) is a text-based format for representing spatial data. WKT SRS (Spatial Reference System) is used to define the coordinate system for the data.

- POINT(0 0)
- POINT(0 0)
- POINT(0 0)
- POINT EMPTY
- LINESTRING(0 0,1 1,1 2)
- LINESTRING
- 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((0 0),(1 2))
- MULTIPOINT
- 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

Input and output of WKT is provided by the functions `ST_AsText` and `ST_GeomFromText`:

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

OGC (Open Geospatial Consortium) is the organization that defines the standards for spatial data formats and services.

Because of these limitations PostGIS defined extended EWKB and EWKT formats. They provide 3D (XYZ and XYM) and 4D (XYZM) coordinate support and include SRID information. Including all geometry information allows PostGIS to use EWKB as the format of record (e.g. in DUMP files).

EWKB and EWKT are used for the "canonical forms" of PostGIS data objects. For input, the canonical form for binary data is EWKB, and for text data either EWKB or EWKT is accepted. This allows geometry values to be created by casting a text value in either HEXEWKB or EWKT to a geometry value using ::geometry. For output, the canonical form for binary is EWKB, and for text it is HEXEWKB (hex-encoded EWKB).

For example this statement creates a geometry by casting from an EWKT text value, and outputs it using the canonical form of HEXEWKB:

```
SELECT 'SRID=4;POINT(0 0)>:::geometry;
geometry
-----
0101000020040000000000000000000000000000000000000000000000000000
```

PostGIS EWKT output has a few differences to OGC WKT:

- For 3DZ geometries the Z qualifier is omitted:
POINT(0 0)
POINT(0 0)
- For 3DM geometries the M qualifier is included:
POINT(0 0)
POINT(0 0)
- For 4D geometries the ZM qualifier is omitted:
POINT(0 0)
POINT(0 0)

EWKT avoids over-specifying dimensionality and the inconsistencies that can occur with the OGC/ISO format, such as:

- POINT(0 0)
- POINT(0 0)
- POINT(0 0)



Caution

PostGIS does not support OGC WKB/WKT (or EWKB/EWKT) for 3D geometries. OGC WKB/WKT does not support SRID. OGC WKB/WKT does not support PostGIS geometry types. OGC WKB/WKT does not support PostGIS geometry types!

PostGIS EWKT (WKT) output examples:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- SRID XY
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- SRID XYM

4.3.1 Geography

You can create a table to store geography data using the **CREATE TABLE** SQL statement with a column of type **geography**. The following example creates a table with a geography column storing 2D LineStrings in the WGS84 geodetic coordinate system (SRID 4326):

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

The geography type supports two optional type modifiers:

- POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON.** **Z, M** or **ZM**. **SRID**. **LINESTRINGM** **SRID** **3**, **POINTZM**.
- the SRID modifier restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.

Examples of creating tables with geography columns:

- POINT: 2D:

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

- POINT: 2D:

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

- Create a table with 3D (XYZ) POINTs and an explicit SRID of 4326:

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

- Create a table with 2D LINESTRING geography with the default SRID 4326:

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

- POINT: 2D:

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

Geography fields are registered in the `geography_columns` system view. You can query the `geography_columns` view and see that the table is listed:

```
SELECT * FROM geography_columns;
```

```
PostGIS
```

```
-- Index the test table with a spherical index
CREATE INDEX global_points_gix ON global_points USING GIST ( location );
```

4.3.2 PostGIS

You can insert data into geography tables in the same way as geometry. Geometry data will autocast to the geography type if it has SRID 4326. The **EWKT** and **EWKB** formats can also be used to specify geography values.

```
-- Add some data into the test table
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)');
```

Any geodetic (long/lat) spatial reference system listed in `spatial_ref_sys` table may be specified as a geography SRID. Non-geodetic coordinate systems raise an error if used.

```
-- NAD 83 lon/lat
SELECT 'SRID=4269;POINT(-123 34)::geography;
       geography
-----
0101000020AD100000000000000000C05EC00000000000004140
```

```
-- NAD27 lon/lat
SELECT 'SRID=4267;POINT(-123 34)::geography;
       geography
-----
0101000020AB100000000000000000C05EC00000000000004140
```

```
-- NAD83 UTM zone meters - gives an error since it is a meter-based planar projection
SELECT 'SRID=26910;POINT(-123 34)::geography;

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

Geography uses a spherical Earth model. Geometry uses a flat Earth model. The difference is significant for large distances or areas.

```
-- A distance query using a 1000km tolerance
SELECT name FROM global_points WHERE ST_DWithin(location, 'SRID=4326;POINT(-110 29)::
       geography, 1000000);
```

Distance calculation using GEOGRAPHY. The difference between the two methods is significant for large distances or areas.

```
-- Distance calculation using GEOGRAPHY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)::geography, 'POINT(-21.96 64.15)
       '::geography);
       st_distance
-----
122235.23815667
```

Great Circle mapper uses a spherical Earth model. Geometry uses a flat Earth model. The difference is significant for large distances or areas.

```
-- Distance calculation using GEOMETRY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)::geometry, 'POINT(-21.96 64.15)
       '::geometry);
       st_distance
-----
13.342271221453624
```


Simple Features geometry model to represent spatial objects in a consistent and unambiguous way that supports efficient computation. (Note: the OGC SF and SQL/MM have the same definitions for simple and valid.)

4.4.1 Simple Geometry

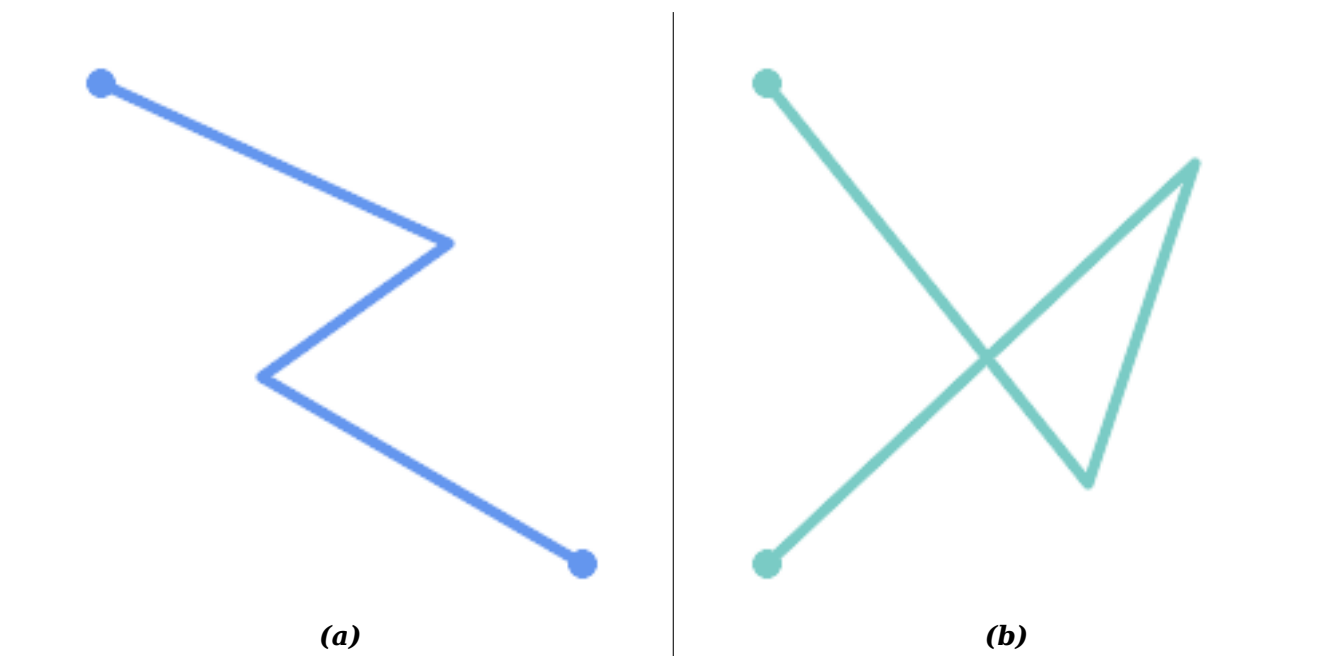
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

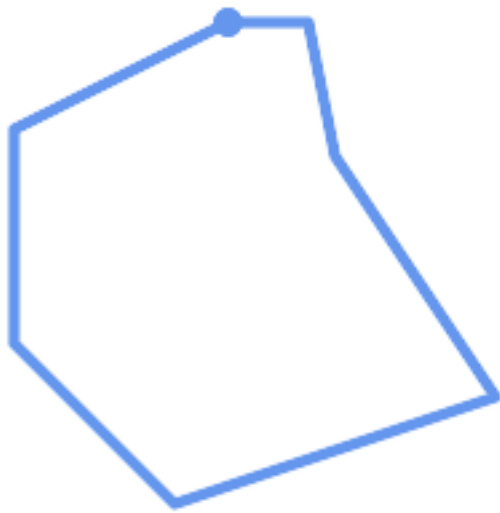
POINT 0

MULTIPOINT (POINT)

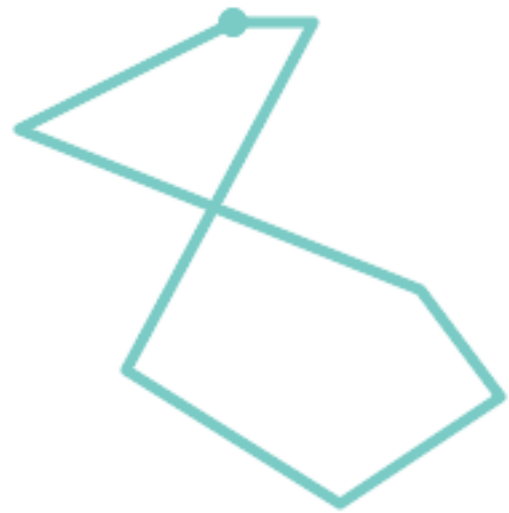
A LINESTRING is *simple* if it does not pass through the same point twice, except for the endpoints. If the endpoints of a simple LineString are identical it is called *closed* and referred to as a Linear Ring.

(a) and **(c)** are simple LINESTRINGs. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.





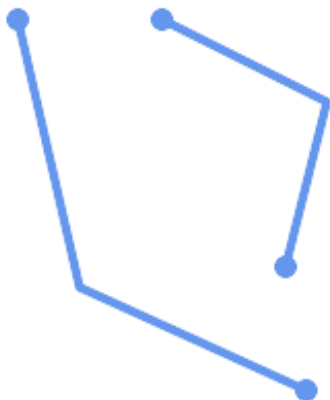
(c)



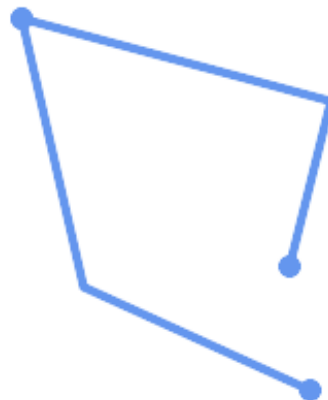
(d)

A MULTILINESTRING is *simple* only if all of its elements are simple and the only intersection between any two elements occurs at points that are on the boundaries of both elements.

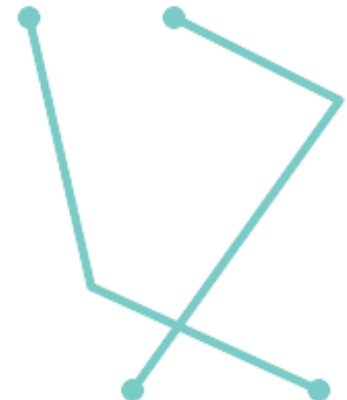
(e) and **(f)** are simple MULTILINESTRINGs. **(g)** is not simple.



(e)



(f)



(g)

POLYGONS are formed from linear rings, so valid polygonal geometry is always *simple*.

To test if a geometry is simple use the **ST_IsSimple** function:

```
SELECT
  ST_IsSimple('LINESTRING(0 0, 100 100)') AS straight,
  ST_IsSimple('LINESTRING(0 0, 100 100, 100 0, 0 100)') AS crossing;

straight | crossing
-----+-----
t        | f
```

Generally, PostGIS functions do not require geometric arguments to be simple. Simplicity is primarily used as a basis for defining geometric validity. It is also a requirement for some kinds of spatial data models (for example, linear networks often disallow lines that cross). Multipoint and linear geometry can be made simple using [ST_UnaryUnion](#).

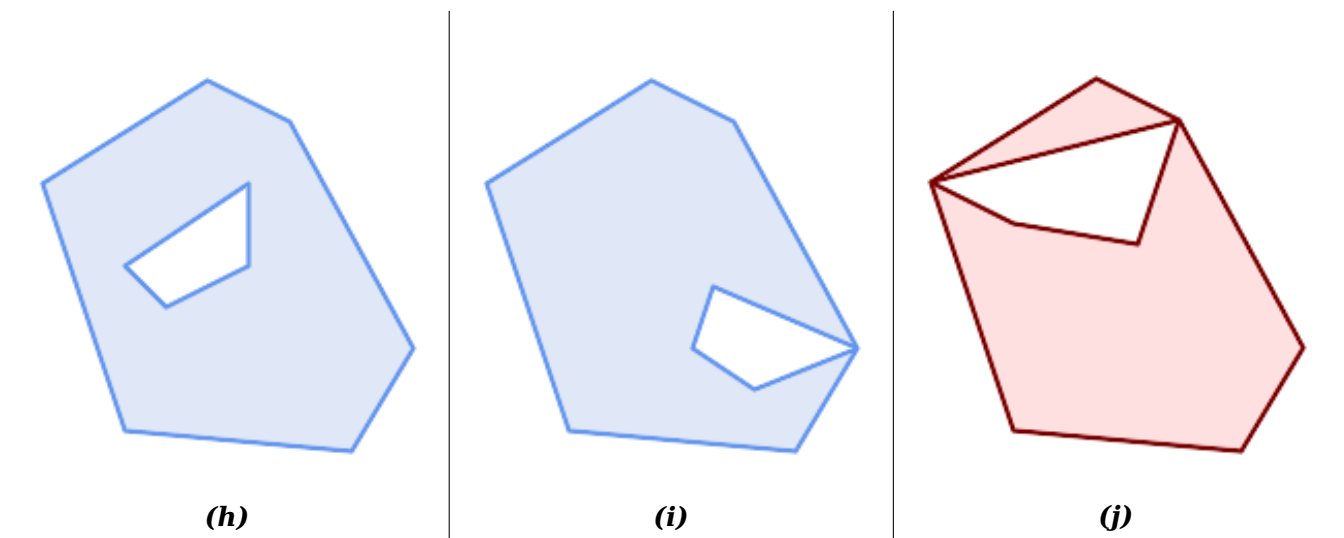
4.4.2 Valid Geometry

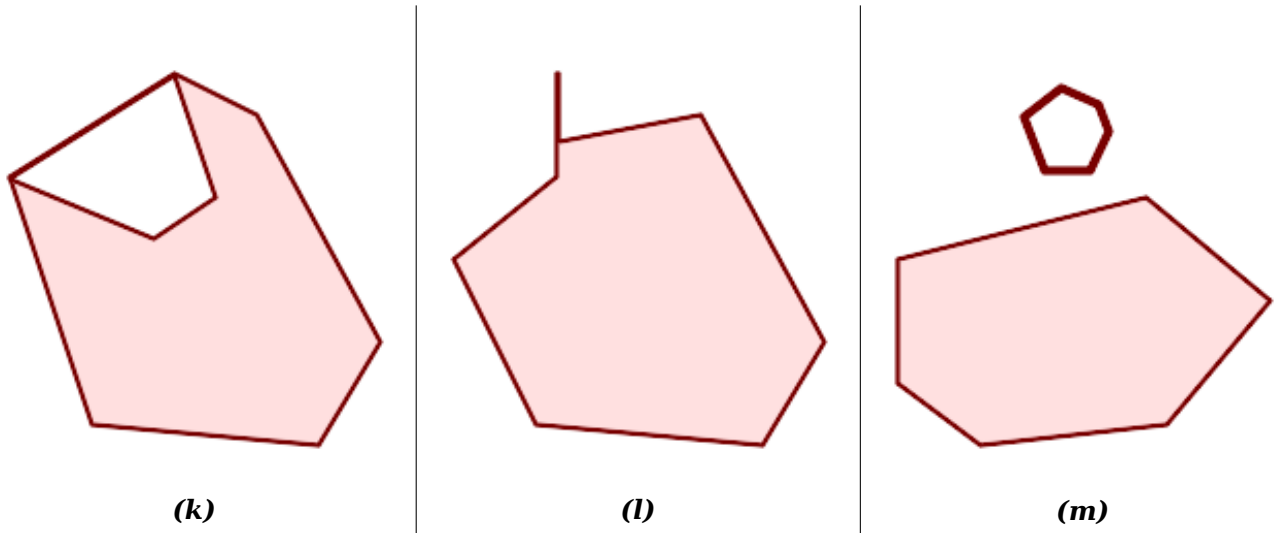
Geometry validity primarily applies to 2-dimensional geometries (POLYGONS and MULTIPOLYGONS). Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

A POLYGON is *valid* if:

1. the polygon boundary rings (the exterior shell ring and interior hole rings) are *simple* (do not cross or self-touch). Because of this a polygon cannot have cut lines, spikes or loops. This implies that polygon holes must be represented as interior rings, rather than by the exterior ring self-touching (a so-called "inverted hole").
2. boundary rings do not cross
3. boundary rings may touch at points but only as a tangent (i.e. not in a line)
4. interior rings are contained in the exterior ring
5. the polygon interior is simply connected (i.e. the rings must not touch in a way that splits the polygon into more than one part)

(h) and **(i)** are valid POLYGONS. **(j-m)** are invalid. **(j)** can be represented as a valid MULTIPOLYGON.

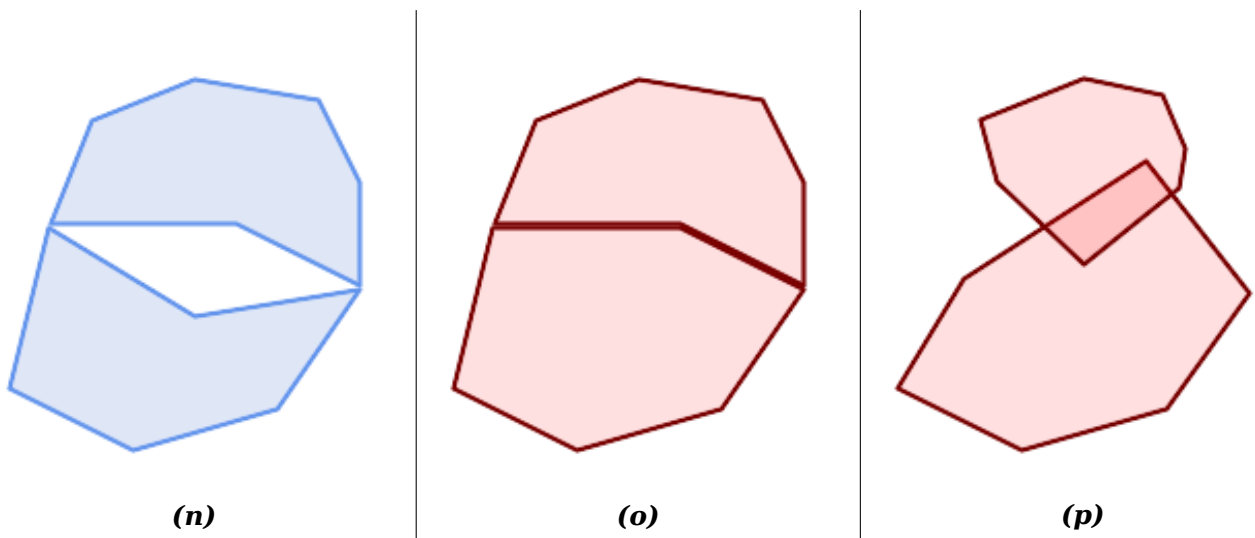




A MULTIPOLYGON is *valid* if:

1. its element POLYGONS are valid
2. elements do not overlap (i.e. their interiors must not intersect)
3. elements touch only at points (i.e. not along a line)

(n) is a valid MULTIPOLYGON. **(o)** and **(p)** are invalid.



These rules mean that valid polygonal geometry is also *simple*.

For linear geometry the only validity rule is that LINESTRINGS must have at least two points and have non-zero length (or equivalently, have at least two distinct points.) Note that non-simple (self-intersecting) lines are valid.

```
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;
```

```

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

```

POINT and MULTIPOINT geometries have no validity rules.

4.4.3 Managing Validity

PostGIS allows creating and storing both valid and invalid Geometry. This allows invalid geometry to be detected and flagged or fixed. There are also situations where the OGC validity rules are stricter than desired (examples of this are zero-length linestrings and polygons with inverted holes.)

Many of the functions provided by PostGIS rely on the assumption that geometry arguments are valid. For example, it does not make sense to calculate the area of a polygon that has a hole defined outside of the polygon, or to construct a polygon from a non-simple boundary line. Assuming valid geometric inputs allows functions to operate more efficiently, since they do not need to check for topological correctness. (Notable exceptions are that zero-length lines and polygons with inversions are generally handled correctly.) Also, most PostGIS functions produce valid geometry output if the inputs are valid. This allows PostGIS functions to be chained together safely.

If you encounter unexpected error messages when calling PostGIS functions (such as "GEOS Intersection() threw an error!"), you should first confirm that the function arguments are valid. If they are not, then consider using one of the techniques below to ensure the data you are processing is valid.



Note

If a function reports an error with valid inputs, then you may have found an error in either PostGIS or one of the libraries it uses, and you should report this to the PostGIS project. The same is true if a PostGIS function returns an invalid geometry for valid input.

To test if a geometry is valid use the [ST_IsValid](#) function:

```

SELECT ST_IsValid('POLYGON ((20 180, 180 180, 180 20, 20 20, 20 180))');
-----
t

```

Information about the nature and location of an geometry invalidity are provided by the [ST_IsValidDetail](#) function:

```

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

```

valid	reason	location
f	Self-intersection	POINT(91.51162790697674 141.56976744186045)

In some situations it is desirable to correct invalid geometry automatically. Use the [ST_MakeValid](#) function to do this. ([ST_MakeValid](#) is a case of a spatial function that *does* allow invalid input!)

By default, PostGIS does not check for validity when loading geometry, because validity testing can take a lot of CPU time for complex geometries. If you do not trust your data sources, you can enforce a validity check on your tables by adding a check constraint:

```

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

```

4.5 SPATIAL_REF_SYS

A **Spatial Reference System** (SRS) (also called a Coordinate Reference System (CRS)) defines how geometry is referenced to locations on the Earth’s surface. There are three types of SRS:

- A **geodetic** SRS uses angular coordinates (longitude and latitude) which map directly to the surface of the earth.
- A **projected** SRS uses a mathematical projection transformation to “flatten” the surface of the spheroidal earth onto a plane. It assigns location coordinates in a way that allows direct measurement of quantities such as distance, area, and angle. The coordinate system is Cartesian, which means it has a defined origin point and two perpendicular axes (usually oriented North and East). Each projected SRS uses a stated length unit (usually metres or feet). A projected SRS may be limited in its area of applicability to avoid distortion and fit within the defined coordinate bounds.
- A **local** SRS is a Cartesian coordinate system which is not referenced to the earth’s surface. In PostGIS this is specified by a SRID value of 0.

There are many different spatial reference systems in use. Common SRSes are standardized in the European Petroleum Survey Group **EPSG database**. For convenience PostGIS (and many other spatial systems) refers to SRS definitions using an integer identifier called a SRID.

A geometry is associated with a Spatial Reference System by its SRID value, which is accessed by **ST_SRID**. The SRID for a geometry can be assigned using **ST_SetSRID**. Some geometry constructor functions allow supplying a SRID (such as **ST_Point** and **ST_MakeEnvelope**). The **EWKT** format supports SRIDs with the SRID=n; prefix.

Spatial functions processing pairs of geometries (such as **overlay** and **relationship** functions) require that the input geometries are in the same spatial reference system (have the same SRID). Geometry data can be transformed into a different spatial reference system using **ST_Transform** and **ST_TransformPipe**. Geometry returned from functions has the same SRS as the input geometries.

4.5.1 SPATIAL_REF_SYS Table

The SPATIAL_REF_SYS table used by PostGIS is an OGC-compliant database table that defines the available spatial reference systems. It holds the numeric SRIDs and textual descriptions of the coordinate systems.

SPATIAL_REF_SYS:

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

Columns:

srid (SRS) ID.

auth_name Authority name. “EPSG” AUTH_NAME.

auth_srid The ID of the Spatial Reference System as defined by the Authority cited in the auth_name. In the case of EPSG, this is the EPSG code.

srtext WKT(Well-Known Text) SRS:

```

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]
]

```

For a discussion of SRS WKT, see the OGC standard [Well-known text representation of coordinate reference systems](#).

proj4text PostGIS `proj4text` stores Proj4 text. `PROJ4TEXT` stores SRID `proj4text` Proj4 text.

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

`spatial_ref_sys.sql` contains EPSG SRTEXT and PROJ4TEXT.

When retrieving spatial reference system definitions for use in transformations, PostGIS uses the following strategy:

- If `auth_name` and `auth_srid` are present (non-NULL) use the PROJ SRS based on those entries (if one exists).
- If `srttext` is present create a SRS using it, if possible.
- If `proj4text` is present create a SRS using it, if possible.

4.5.2 SPATIAL_REF_SYS

PostGIS `SPATIAL_REF_SYS` stores Proj text, Proj4 text, and SRTEXT. `PROJ4TEXT` stores SRID `proj4text` Proj4 text.

`SPATIAL_REF_SYS` contains <http://spatialreference.org/> SRTEXT.

4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, NAD 83 WGS 84 UTM (帶; zone) UTM, 6

() ESRI spatialreference.org

You can even define non-Earth-based coordinate systems, such as **Mars 2000** This Mars coordinate system is non-planar (it's in degrees spheroidal), but you can use it with the `geography` type to obtain length and proximity measurements in meters instead of degrees.

Here is an example of loading a custom coordinate system using an unassigned SRID and the PROJ definition for a US-centric Lambert Conformal projection:

```
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 Geometry

4.6.1 Creating Geometry Tables

You can create a table to store geometry data using the **CREATE TABLE** SQL statement with a column of type geometry. The following example creates a table with a geometry column storing 2D (XY) LineStrings in the BC-Albers coordinate system (SRID 3005):

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

The geometry type supports two optional **type modifiers**:

- **Dimensional modifiers**. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON. Z, M ZM. POINTZ, POINTM, POINTZM, POINTMZ, LINESTRINGM, LINESTRINGZ, LINESTRINGZM, LINESTRINGMZ, POLYGONZ, POLYGONM, POLYGONZM, POLYGONMZ. POINTZM, POINTMZ, LINESTRINGZM, LINESTRINGMZ, POLYGONZM, POLYGONMZ.
- the **SRID modifier** restricts the **spatial reference system** SRID to a particular number. If omitted, the SRID defaults to 0.

Examples of creating tables with geometry columns:

- Create a table holding any kind of geometry with the default SRID:

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

- Create a table with 2D POINT geometry with the default SRID:

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

- Create a table with 3D (XYZ) POINTs and an explicit SRID of 3005:

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

- Create a table with 4D (XYZM) LINESTRING geometry with the default SRID:

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

- Create a table with 2D POLYGON geometry with the SRID 4267 (NAD 1927 long lat):

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

It is possible to have more than one geometry column in a table. This can be specified when the table is created, or a column can be added using the **ALTER TABLE** SQL statement. This example adds a column that can hold 3D LineStrings:

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

4.6.2 The GEOMETRY_COLUMNS VIEW

OpenGIS "SQL (Simple Features Specification for SQL)" GIS, ... OpenGIS.

\d geometry_columns

```
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)  |
```

:

f_table_catalog, f_table_schema, f_table_name ... PostgreSQL (public).

f_geometry_column ...

coord_dimension (2, 3, 4) ...

srid ... ID, SPATIAL_REF_SYS ... (foreign key).

type ... POINT, LINestring, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION XYM POINTM, LINestringM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. "GEOMETRY".

4.6.3 geometry_columns

AddGeometryColumn() ... SQL (bulk insert) ... geometry_columns ... PostGIS 2.0 ... typmod ...

```
-- Lets say you have a view created like this
CREATE VIEW public.vwmytablemercator AS
  SELECT gid, ST_Transform(geom, 3395) As geom, f_name
  FROM public.mytable;

-- For it to register correctly
-- You need to cast the geometry
--
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 you know the geometry type for sure is a 2D POLYGON then you could do
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;
```

```
-- Lets say you created a derivative table by doing a bulk insert
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 2D index on new table
CREATE INDEX idx_myschema_myspecialpois_geom_gist
    ON myschema.my_special_pois USING gist(geom);
```

```
-- If your points are 3D points or 3M points,
-- then you might want to create an nd index instead of a 2D index
CREATE INDEX my_special_pois_geom_gist_nd
    ON my_special_pois USING gist(geom gist_geometry_ops_nd);
```

```
-- To manually register this new table's geometry column in geometry_columns.
-- Note it will also change the underlying structure of the table to
-- to make the column typmod based.
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass);
```

```
-- If you are using PostGIS 2.0 and for whatever reason, you
-- you need the constraint based definition behavior
-- (such as case of inherited tables where all children do not have the same type and srid)
-- set optional use_typmod argument to false
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass, false);
```

CREATE TABLE pois_ny(gid SERIAL PRIMARY KEY, poi_name text, cat text, geom geometry(Point, 4326));

```
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);
```

PSQL

```
\d pois_ny;
```

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)

geometry_columns

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

f_table_name	f_geometry_column	srid	type
pois_ny	geom	4326	POINT
pois_ny	geom_2160	2160	POINT

--

```
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';
```

typmod

f_table_name	f_geometry_column	srid	type
vw_pois_ny_parks	geom	4326	POINT
vw_pois_ny_parks	geom_2160	0	GEOMETRY

PostGIS

```
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';
```

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 GIS ()

SQL shapefile / PostGIS/PostgreSQL

4.7.1 SQL

PostGIS (formatted) SQL. Oracle SQL, SQL "INSERT" (piping)

(roads.sql)

```
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;
```

"psql" SQL PostgreSQL

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

4.7.2 shp2pgsql: ESRI shapefile

shp2pgsql ESRI shapefile, PostGIS/PostgreSQL SQL (command line)

shp2pgsql, PostGIS shp2pgsql-gui pgAdmin III

- c|a|d|p --** shapefile
- c** shapefile
- a** shapefile
- d** (drop) shapefile
- p** SQL
- ?**
- D** PostgreSQL " (dump)"
- s** [<FROM_SRID>:]<SRID> SRID
- k** (,) shapefile

- i DBF 数据类型 64 位 bigint 数据类型, 数据类型 32 位 数据类型 64 位 bigint 数据类型。
 - I 启用 GiST 索引。
 - m "-m 选项" 选项 (选项) 选项 10 位 DBF 数据类型。 数据类型 数据类型 数据类型, 数据类型 数据类型。 数据类型 数据类型:
- ```

COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2

```
- S 选项 (multi) 选项。 选项 (选项: 选项 选项 选项 选项 选项 选项) 选项。
  - t <dimensionality> 选项。 选项。 选项: 2D, 3DZ, 3DM, 4D 选项, 选项 0 选项。 选项 选项, 选项。
  - w WKB 选项 WKT 选项。 选项。
  - e 选项。 选项。 "选项" 选项 -D 选项。
  - W <encoding> 选项 (DBF 选项) 选项。 选项, DBF 选项 选项 UTF8 选项。 选项 SQL 选项 SET CLIENT\_ENCODING to UTF8 选项, 选项 UTF8 选项。
  - N <policy> NULL 选项 -- insert\*(选项), skip(选项), abort(选项)
  - n DBF 选项。 选项 shapefile 选项, 选项 DBF 选项。 选项 shapefile 选项。
  - G (选项/选项) 选项 WGS84 选项 (SRID=4326) 选项。
  - T <tablespace> 选项。 -X 选项 PostgreSQL 选项。
  - X <tablespace> 选项。 选项 (primary key) 选项, -I 选项 GiST 选项。
  - Z When used, this flag will prevent the generation of ANALYZE statements. Without the -Z flag (default behavior), the ANALYZE statements will be generated.

选项:

```

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

```

UNIX 选项 (pipe) 选项:

```

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

```

### 4.8

SQL shapefile /... SQL ...

#### 4.8.1 SQL

SQL (select) ...

```
db=# SELECT road_id, ST_AsText(road_geom) AS geom, road_name FROM roads;
```

| 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)

SQL ...

**ST\_Intersects** This function tells whether two geometries share any space.

```
= ST_Intersects('POLYGON((0 0,1 1,1 0,0 0))', 'POLYGON((0 0,1 1,1 0,0 0))')
```

SQL "ST\_GeomFromText()" ...

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

"ROADS\_GEOM" ...

To check whether some of the roads passes in the area defined by a polygon:

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

"(map frame)" ... (frame-based)"

"&&" BOX3D ...

Using a "BOX3D" object for the frame, such a query looks like this:

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

SRID 312

### 4.8.2 使用 pgsql2shp

pgsql2shp 是一个命令行工具，用于将 PostgreSQL 数据库中的表导出为 shapefile。用法如下：

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

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

选项如下：

- f <filename> 输出文件的名称。
- h <host> 数据库所在的主机名。
- p <port> 数据库的端口号。
- P <password> 数据库用户的密码。
- u <user> 数据库用户的名称。
- g <geometry column> 指定 shapefile 中的几何列名。
- b 指定几何列的数据类型。默认为 GEOMETRY。如果指定为 POINT、POLYGON 等，则会自动进行类型转换 (cast)。
- r 指定输出格式为 raw。gid 列将包含几何对象的 ID。
- m filename 指定 remap 文件。该文件用于指定非常长的符号 (VERYLONGSYMBOL) 和非常短的符号 (SHORTONE) 的映射。

### 4.9 索引

PostgreSQL 支持多种索引方法。对于空间数据，B-Tree、R-Tree 和 GiST 索引是常用的。B-Tree 索引适用于一维数据，而 R-Tree 和 GiST 索引适用于多维空间数据。

The B-tree index method commonly used for attribute data is not very useful for spatial data, since it only supports storing and querying data in a single dimension. Data such as geometry (which has 2 or more dimensions) requires an index method that supports range query across all the data dimensions. One of the key advantages of PostgreSQL for spatial data handling is that it offers several kinds of index methods which work well for multi-dimensional data: GiST, BRIN and SP-GiST indexes.

- GiST (Generalized Search Tree) 索引适用于多维空间数据。PostGIS 使用 GiST 索引来加速 GIS 查询。与 R-Tree 索引相比，GiST 索引通常具有更好的性能。
- **BRIN (Block Range Index)** 索引通过总结表的范围来操作。搜索是通过扫描范围来完成的。BRIN 仅适用于某些类型的数据 (空间排序、无更新或低频更新)。但它提供了更快的索引创建时间和更小的索引大小。



- **SP-GiST (Space-Partitioned Generalized Search Tree)** is a generic index method that supports partitioned search trees such as quad-trees, k-d trees, and radix trees (tries).

Spatial indexes store only the bounding box of geometries. Spatial queries use the index as a **primary filter** to quickly determine a set of geometries potentially matching the query condition. Most spatial queries require a **secondary filter** that uses a spatial predicate function to test a more specific spatial condition. For more information on queying with spatial predicates see Section 5.2.

See also the [PostGIS Workshop section on spatial indexes](#), and the [PostgreSQL manual](#).

### 4.9.1 GiST

GiST is a general-purpose index method introduced in PostgreSQL 9.5. It 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](#).

BRIN stands for "Block Range Index". It is a general-purpose index method introduced in PostgreSQL 9.5. 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](#).

"GiST" is a general-purpose index method introduced in PostgreSQL 9.5. It 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](#).

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

BRIN stands for "Block Range Index". It is a general-purpose index method introduced in PostgreSQL 9.5. 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](#).

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

Building a spatial index is a computationally intensive exercise. It also blocks write access to your table for the time it creates, so on a production system you may want to do in in a slower CONCURRENTLY-aware way:

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

After building an index, it is sometimes helpful to force PostgreSQL to collect table statistics, which are used to optimize query plans:

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

### 4.9.2 GiST

BRIN stands for "Block Range Index". It is a general-purpose index method introduced in PostgreSQL 9.5. 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](#).

BRIN stands for "Block Range Index". It is a general-purpose index method introduced in PostgreSQL 9.5. 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](#).

A BRIN index stores the bounding box enclosing all the geometries contained in the rows in a contiguous set of table blocks, called a *block range*. When executing a query using the index the block ranges are scanned to find the ones that intersect the query extent. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are





An SP-GiST index can accelerate queries involving the following operators:

- <<, &<, &>, >>, <<|, &<|, |&>, |>>, &&, @>, <@, and ~=:, for 2-dimensional indexes,
- &/&, ~==, @>>, and <<@, for 3-dimensional indexes.

There is no support for kNN searches at the moment.

#### 4.9.4

PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes.

PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes.

- Examine the query plan and check your query actually computes the thing you need. An erroneous JOIN, either forgotten or to the wrong table, can unexpectedly retrieve table records multiple times. To get the query plan, execute with EXPLAIN in front of the query.
- Make sure statistics are gathered about the number and distributions of values in a table, to provide the query planner with better information to make decisions around index usage. VACUUM ANALYZE will compute both.

You should regularly vacuum your databases anyways. Many PostgreSQL DBAs run VACUUM as an off-peak cron job on a regular basis.

- SET ENABLE\_SEQSCAN=OFF. PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes.
- (cost) postgresql.conf random\_page\_cost. PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes. PostgreSQL 9.5 introduced support for SP-GiST indexes.
- If SET ENABLE\_SEQSCAN TO OFF; does not help your query, the query may be using a SQL construct that the Postgres planner is not yet able to optimize. It may be possible to rewrite the query in a way that the planner is able to handle. For example, a subquery with an inline SELECT may not produce an efficient plan, but could possibly be rewritten using a LATERAL JOIN.

For more information see the Postgres manual section on Query Planning.

# Chapter 5

## Spatial Queries

The *raison d'être* of spatial databases is to perform queries inside the database which would ordinarily require desktop GIS functionality. Using PostGIS effectively requires knowing what spatial functions are available, how to use them in queries, and ensuring that appropriate indexes are in place to provide good performance.

### 5.1 Determining Spatial Relationships

Spatial relationships indicate how two geometries interact with one another. They are a fundamental capability for querying geometry.

#### 5.1.1 Dimensionally Extended 9-Intersection Model

According to the [OpenGIS Simple Features Implementation Specification for SQL](#), "the basic approach to comparing two geometries is to make pair-wise tests of the intersections between the Interiors, Boundaries and Exteriors of the two geometries and to classify the relationship between the two geometries based on the entries in the resulting 'intersection' matrix."

In the theory of point-set topology, the points in a geometry embedded in 2-dimensional space are categorized into three sets:

##### Boundary

The boundary of a geometry is the set of geometries of the next lower dimension. For POINTs, which have a dimension of 0, the boundary is the empty set. The boundary of a LINESTRING is the two endpoints. For POLYGONS, the boundary is the linework of the exterior and interior rings.

##### Interior

The interior of a geometry are those points of a geometry that are not in the boundary. For POINTs, the interior is the point itself. The interior of a LINESTRING is the set of points between the endpoints. For POLYGONS, the interior is the areal surface inside the polygon.

##### Exterior

The exterior of a geometry is the rest of the space in which the geometry is embedded; in other words, all points not in the interior or on the boundary of the geometry. It is a 2-dimensional non-closed surface.

---

The **Dimensionally Extended 9-Intersection Model** (DE-9IM) describes the spatial relationship between two geometries by specifying the dimensions of the 9 intersections between the above sets for each geometry. The intersection dimensions can be formally represented in a 3x3 **intersection matrix**.

For a geometry  $g$  the *Interior*, *Boundary*, and *Exterior* are denoted using the notation  $I(g)$ ,  $B(g)$ , and  $E(g)$ . Also,  $dim(s)$  denotes the dimension of a set  $s$  with the domain of  $\{0, 1, 2, F\}$ :



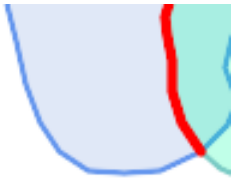

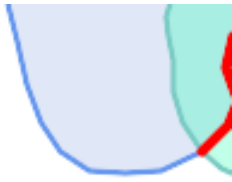
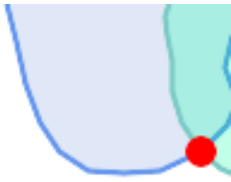
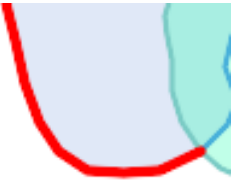
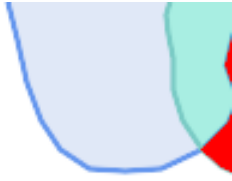
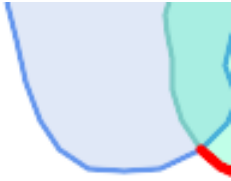
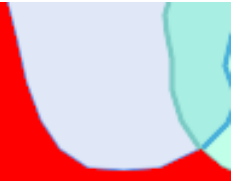
- 0 => point
- 1 => line
- 2 => area
- F => empty set

Using this notation, the intersection matrix for two geometries  $a$  and  $b$  is:

|                 | <b>Interior</b>         | <b>Boundary</b>         | <b>Exterior</b>         |
|-----------------|-------------------------|-------------------------|-------------------------|
| <b>Interior</b> | $dim( I(a) \cap I(b) )$ | $dim( I(a) \cap B(b) )$ | $dim( I(a) \cap E(b) )$ |
| <b>Boundary</b> | $dim( B(a) \cap I(b) )$ | $dim( B(a) \cap B(b) )$ | $dim( B(a) \cap E(b) )$ |
| <b>Exterior</b> | $dim( E(a) \cap I(b) )$ | $dim( E(a) \cap B(b) )$ | $dim( E(a) \cap E(b) )$ |

Visually, for two overlapping polygonal geometries, this looks like:



|                                                                                   |                 | Interior                                                                                                           | Boundary                                                                                                            | Exterior                                                                                                             |
|-----------------------------------------------------------------------------------|-----------------|--------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
|  | <b>Interior</b> | <br>$dim( I(a) \cap I(b) ) = 2$   | <br>$dim( I(a) \cap B(b) ) = 1$   | <br>$dim( I(a) \cap E(b) ) = 2$   |
|                                                                                   | <b>Boundary</b> | <br>$dim( B(a) \cap I(b) ) = 1$   | <br>$dim( B(a) \cap B(b) ) = 0$   | <br>$dim( B(a) \cap E(b) ) = 1$   |
|                                                                                   | <b>Exterior</b> | <br>$dim( E(a) \cap I(b) ) = 2$ | <br>$dim( E(a) \cap B(b) ) = 1$ | <br>$dim( E(a) \cap E(b) ) = 2$ |

Reading from left to right and top to bottom, the intersection matrix is represented as the text string '212101212'.

For more information, refer to:

- [OpenGIS Simple Features Implementation Specification for SQL](#) (version 1.1, section 2.1.13.2)
- [Wikipedia: Dimensionally Extended Nine-Intersection Model \(DE-9IM\)](#)
- [GeoTools: Point Set Theory and the DE-9IM Matrix](#)

### 5.1.2 Named Spatial Relationships

To make it easy to determine common spatial relationships, the OGC SFS defines a set of *named spatial relationship predicates*. PostGIS provides these as the functions `ST_Contains`, `ST_Crosses`, `ST_Disjoint`, `ST_Equals`, `ST_Intersects`, `ST_Overlaps`, `ST_Touches`, `ST_Within`. It also defines the non-standard relationship predicates `ST_Covers`, `ST_CoveredBy`, and `ST_ContainsProperly`.

Spatial predicates are usually used as conditions in SQL WHERE or JOIN clauses. The named spatial predicates automatically use a spatial index if one is available, so there is no need to use the bounding box operator && as well. For example:

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

For more details and illustrations, see the [PostGIS Workshop](#).

### 5.1.3 General Spatial Relationships

In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.

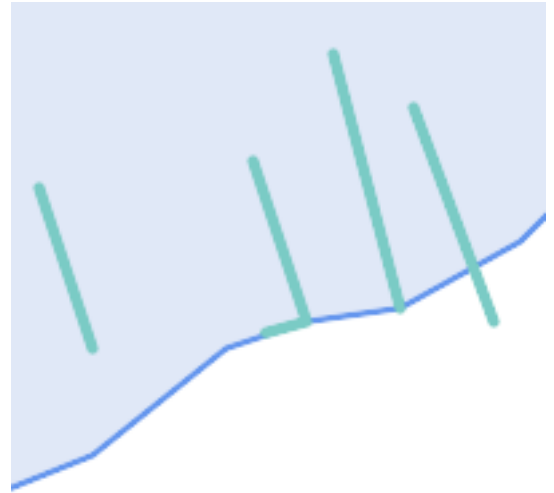


For example, consider a linear dataset representing a road network. It may be required to identify all road segments that cross each other, not at a point, but in a line (perhaps to validate some business rule). In this case **ST\_Crosses** does not provide the necessary spatial filter, since for linear features it returns `true` only where they cross at a point.

A two-step solution would be to first compute the actual intersection (**ST\_Intersection**) of pairs of road lines that spatially intersect (**ST\_Intersects**), and then check if the intersection's **ST\_GeometryType** is 'LINESTRING' (properly dealing with cases that return **GEOMETRYCOLLECTIONs** of [MULTI]POINTs, [MULTI]LINESTRINGs, etc.).

Clearly, a simpler and faster solution is desirable.





A second example is locating wharves that intersect a lake's boundary on a line and where one end of the wharf is up on shore. In other words, where a wharf is within but not completely contained by a lake, intersects the boundary of a lake on a line, and where exactly one of the wharf's endpoints is within or on the boundary of the lake. It is possible to use a combination of spatial predicates to find the required features:

- `ST_Contains`(lake, wharf) = TRUE
  - `ST_ContainsProperly`(lake, wharf) = FALSE
  - `ST_GeometryType`(`ST_Intersection`(wharf, lake)) = 'LINESTRING'
  - `ST_NumGeometries`(`ST_Multi`(`ST_Intersection`(`ST_Boundary`(wharf), `ST_Boundary`(lake)))) = 1
- ... but needless to say, this is quite complicated.

These requirements can be met by computing the full DE-9IM intersection matrix. PostGIS provides the `ST_Relate` function to do this:

```
SELECT ST_Relate('LINESTRING (1 1, 5 5)',
 'POLYGON ((3 3, 3 7, 7 7, 7 3, 3 3))');
st_relate

1010F0212
```

To test a particular spatial relationship, an **intersection matrix pattern** is used. This is the matrix representation augmented with the additional symbols {T,\*}:

- T => intersection dimension is non-empty; i.e. is in {0,1,2}
- \* => don't care

Using intersection matrix patterns, specific spatial relationships can be evaluated in a more succinct way. The `ST_Relate` and the `ST_RelateMatch` functions can be used to test intersection matrix patterns. For the first example above, the intersection matrix pattern specifying two lines intersecting in a line is `'1*1***1**'`:

```
-- Find road segments that intersect in a line
SELECT a.id
```

```
FROM roads a, roads b
WHERE a.id != b.id
 AND a.geom && b.geom
 AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

For the second example, the intersection matrix pattern specifying a line partly inside and partly outside a polygon is **'102101FF2'**:

```
-- Find wharves partly on a lake's shoreline
SELECT a.lake_id, b.wharf_id
FROM lakes a, wharfs b
WHERE a.geom && b.geom
 AND ST_Relate(a.geom, b.geom, '102101FF2');
```

## 5.2 Using Spatial Indexes

When constructing queries using spatial conditions, for best performance it is important to ensure that a spatial index is used, if one exists (see Section 4.9). To do this, a spatial operator or index-aware function must be used in a `WHERE` or `ON` clause of the query.

Spatial operators include the bounding box operators (of which the most commonly used is `&&`; see Section 7.10.1 for the full list) and the distance operators used in nearest-neighbor queries (the most common being `<->`; see Section 7.10.2 for the full list.)

Index-aware functions automatically add a bounding box operator to the spatial condition. Index-aware functions include the named spatial relationship predicates `ST_Contains`, `ST_ContainsProperly`, `ST_CoveredBy`, `ST_Covers`, `ST_Crosses`, `ST_Intersects`, `ST_Overlaps`, `ST_Touches`, `ST_Within`, `ST_Within`, and `ST_3DIntersects`, and the distance predicates `ST_DWithin`, `ST_DFullyWithin`, `ST_3DDFullyWithin`, and `ST_3DDWithin` .)

Functions such as `ST_Distance` do *not* use indexes to optimize their operation. For example, the following query would be quite slow on a large table:

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

This query selects all the geometries in `geom_table` which are within 100 units of the point (100000, 200000). It will be slow because it is calculating the distance between each point in the table and the specified point, ie. one `ST_Distance()` calculation is computed for **every** row in the table.

The number of rows processed can be reduced substantially by using the index-aware function `ST_DWithin`:

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

This query selects the same geometries, but it does it in a more efficient way. This is enabled by `ST_DWithin()` using the `&&` operator internally on an expanded bounding box of the query geometry. If there is a spatial index on `geom`, the query planner will recognize that it can use the index to reduce the number of rows scanned before calculating the distance. The spatial index allows retrieving only records with geometries whose bounding boxes overlap the expanded extent and hence which *might* be within the required distance. The actual distance is then computed to confirm whether to include the record in the result set.

For more information and examples see the [PostGIS Workshop](#).

## 5.3 Examples of Spatial SQL

The examples in this section make use of a table of linear roads, and a table of polygonal municipality boundaries. The definition of the `bc_roads` table is:

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

The definition of the `bc_municipality` table is:

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

1. *What is the total length of all roads, expressed in kilometers?*

You can answer this question with a very simple piece of SQL:

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

```
km_roads

70842.1243039643
```

2. *How large is the city of Prince George, in hectares?*

This query combines an attribute condition (on the municipality name) with a spatial calculation (of the polygon area):

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

```
hectares

32657.9103824927
```

3. *What is the largest municipality in the province, by area?*

This query uses a spatial measurement as an ordering value. There are several ways of approaching this problem, but the most efficient is below:

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

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

Note that in order to answer this query we have to calculate the area of every polygon. If we were doing this a lot it would make sense to add an area column to the table that could be indexed for performance. By ordering the results in a descending direction, and then using the PostgreSQL "LIMIT" command we can easily select just the largest value without using an aggregate function like MAX().

4. *What is the length of roads fully contained within each municipality?*

This is an example of a "spatial join", which brings together data from two tables (with a join) using a spatial interaction ("contained") as the join condition (rather than the usual relational approach of joining on a common key):

```
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;
```

| name             | roads_km         |
|------------------|------------------|
| SURREY           | 1539.47553551242 |
| VANCOUVER        | 1450.33093486576 |
| LANGLEY DISTRICT | 833.793392535662 |
| BURNABY          | 773.769091404338 |
| PRINCE GEORGE    | 694.37554369147  |
| ...              |                  |

This query takes a while, because every road in the table is summarized into the final result (about 250K roads for the example table). For smaller datasets (several thousand records on several hundred) the response can be very fast.

5. *Create a new table with all the roads within the city of Prince George.*

This is an example of an "overlay", which takes in two tables and outputs a new table that consists of spatially clipped or cut resultants. Unlike the "spatial join" demonstrated above, this query creates new geometries. An overlay is like a turbo-charged spatial join, and is useful for more exact analysis work:

```
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. *What is the length in kilometers of "Douglas St" in Victoria?*

```
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';
```

```
kilometers

4.89151904172838
```

7. *What is the largest municipality polygon that has a hole?*

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

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



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

geom\_column bbox && ST\_SetSRID('BOX3D(0 0,1 1)::box3d,4326):

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

mytable, bbox " " . (trigger) . UPDATE

### 6.2

PostgreSQL CLUSTER .

PostgreSQL PostGIS GiST . GiST NULL

```
lwgeom=# CLUSTER my_geom_index ON my_table;
ERROR: cannot cluster when index access method does not handle null values
HINT: You may be able to work around this by marking column "geom" NOT NULL.
```

HINT "not null"

```
lwgeom=# ALTER TABLE my_table ALTER COLUMN geom SET not null;
ALTER TABLE
```

NULL . "ALTER TABLE blubb ADD CHECK (geometry is not null);" CHECK

### 6.3

3D 4D 2D OpenGIS ST\_AsText() ST\_AsBinary() ST\_Force2D()

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

AddGeometryColumn() geometry\_columns

WHERE "VACUUM;" "WHERE dimension(the\_geom)>2" 2D

# Chapter 7

## PostGIS Reference

PostGIS is a spatial database, PostGIS is a PostGIS extension.

**Note**

PostGIS is a SQL-MM-compliant Spatial Type (ST) extension. It provides a set of functions for working with spatial data. The ST\_ prefix is used for all spatial functions.

### 7.1 PostgreSQL PostGIS Geometry/Geography/Box

#### 7.1.1 box2d

box2d — The type representing a 2-dimensional bounding box.

box3d is a PostGIS extension. ST\_3DExtent is a PostGIS extension.

The representation contains the values xmin, ymin, xmax, ymax. These are the minimum and maximum values of the X and Y extents.

box2d objects have a text representation which looks like BOX(1 2,5 6).

|          |          |
|----------|----------|
| box3d    | box3d    |
| geometry | geometry |





|           |  |
|-----------|--|
| geography |  |
| text      |  |

Section 4.1, Section 4.3

### 7.1.4 geometry\_dump

geometry\_dump — A composite type used to describe the parts of complex geometry.

geometry\_dump is a composite data type containing the fields:

- geom - a geometry representing a component of the dumped geometry. The geometry type depends on the originating function.
- path[] - an integer array that defines the navigation path within the dumped geometry to the geom component. The path array is 1-based (i.e. path[1] is the first element.)

It is used by the ST\_Dump\* family of functions as an output type to explode a complex geometry into its constituent parts.

Section 13.6

### 7.1.5 geography

geography — The type representing spatial features with geodetic (ellipsoidal) coordinate systems.

geography

Spatial operations on the geography type provide more accurate results by taking the ellipsoidal model into account.

|          |  |
|----------|--|
|          |  |
| geometry |  |



☒☒

```
-- Create schema to hold data
CREATE SCHEMA my_schema;
-- Create a new simple PostgreSQL table
CREATE TABLE my_schema.my_spatial_table (id serial);

-- Describing the table shows a simple table with a single "id" column.
postgis=# \d my_schema.my_spatial_table
 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
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom',4326,'POINT',2);

-- Add a point using the old constraint based behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geom_c',4326,'POINT',2, false);

--Add a curvepolygon using old constraint behavior
SELECT AddGeometryColumn ('my_schema','my_spatial_table','geomcp_c',4326,'CURVEPOLYGON',2, ←
 false);

-- Describe the table again reveals the addition of a new geometry columns.
\d my_schema.my_spatial_table
 addgeometrycolumn
-----+-----+-----
my_schema.my_spatial_table.geomcp_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2
(1 row)

 Table "my_schema.my_spatial_table"
Column | Type | Modifiers
-----+-----+-----
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)

-- geometry_columns view also registers the new columns --
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';

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



**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);

`geometry_columns` (schema-aware) pgSQL `current_schema()`



**Note**

2.0.0. `geometry_columns` `DROP TABLE`

```
SELECT DropGeometryTable ('my_schema','my_spatial_table');
----RESULT output ---
my_schema.my_spatial_table dropped.

-- The above is now equivalent to --
DROP TABLE my_schema.my_spatial_table;
```

[AddGeometryColumn](#), [DropGeometryColumn](#), Section 4.6.2

**7.2.4 Find\_SRID**

Find\_SRID — Returns the SRID defined for a geometry column.

**Synopsis**

integer **Find\_SRID**(varchar a\_schema\_name, varchar a\_table\_name, varchar a\_geomfield\_name);

Returns the integer SRID of the specified geometry column by searching through the `GEOMETRY_COLUMNS` table. If the geometry column has not been properly added (e.g. with the [AddGeometryColumn](#) function), this function will not work.

```
SELECT Find_SRID('public', 'tiger_us_state_2007', 'geom_4269');
find_srid

4269
```















### Synopsis

```
geometry ST_MakeLine(geometry geom1, geometry geom2);
geometry ST_MakeLine(geometry[] geoms_array);
geometry ST_MakeLine(geometry set geoms);
```

**Geometry**


Creates a LineString containing the points of Point, MultiPoint, or LineString geometries. Other geometry types cause an error.

**Variation 1:** accepts two input geometries

**Variation 2:** accepts an array of geometries

**Variation 3:** aggregate function accepting a rowset of geometries. To ensure the order of the input geometries use ORDER BY in the function call, or a subquery with an ORDER BY clause.

Repeated nodes at the beginning of input LineStrings are collapsed to a single point. Repeated points in Point and MultiPoint inputs are not collapsed. **ST\_RemoveRepeatedPoints** can be used to collapse repeated points from the output LineString.

 This function supports 3d and will not drop the z-index.

2.0.0 **ST\_MakeLine**(geometry geom1, geometry geom2);

2.0.0 **ST\_MakeLine**(geometry[] geoms\_array);

1.4.0 **ST\_MakeLine**(geometry set geoms);. **ST\_MakeLine**(geometry geom1, geometry geom2). **ST\_MakeLine**(geometry[] geoms\_array). **ST\_MakeLine**(geometry set geoms);

**Examples:**

Create a line composed of two points.

```
SELECT ST_AsText(ST_MakeLine(ST_Point(1,2), ST_Point(3,4)));
```

```
 st_astext

LINESTRING(1 2,3 4)
```

**3D** **ST\_MakeLine**(geometry geom1, geometry geom2);

```
SELECT ST_AsEWKT(ST_MakeLine(ST_MakePoint(1,2,3), ST_MakePoint(3,4,5)));
```

```
 st_asewkt

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

**Concatenation**, **ST\_MakeLine**(geometry geom1, geometry geom2);

```
select ST_AsText(ST_MakeLine('LINESTRING(0 0, 1 1)', 'LINESTRING(2 2, 3 3)'));
```

```
 st_astext

LINESTRING(0 0,1 1,2 2,3 3)
```

**ST\_MakeLine**

Create a line from an array formed by a subquery with ordering.

```
SELECT ST_MakeLine(ARRAY(SELECT ST_Centroid(geom) FROM visit_locations ORDER BY
 visit_time));
```

Create a 3D line from an array of 3D points

```
SELECT ST_AsEWKT(ST_MakeLine(
 ARRAY[ST_MakePoint(1,2,3), ST_MakePoint(3,4,5), ST_MakePoint(6,6,6)]));

st_asewkt

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

**ST\_MakeLine**

Using aggregate `ORDER BY` provides a correctly-ordered LineString.

Using aggregate `ORDER BY` provides a correctly-ordered LineString.

```
SELECT gps.track_id, ST_MakeLine(gps.geom ORDER BY gps_time) As geom
 FROM gps_points As gps
 GROUP BY track_id;
```

Prior to PostgreSQL 9, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

```
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 track_id;
```

**ST\_RemoveRepeatedPoints**

[ST\\_RemoveRepeatedPoints](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_MakePoint](#)

### 7.3.5 ST\_MakePoint

`ST_MakePoint` — Creates a 2D, 3DZ or 4D Point.

#### 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);







### 7.3.7 ST\_MakePolygon

ST\_MakePolygon — Creates a Polygon from a shell and optional list of holes.

#### Synopsis

```
geometry ST_MakePolygon(geometry linestring);
geometry ST_MakePolygon(geometry outerlinestring, geometry[] interiorlinestrings);
```

⚠

⚠ (shell) ⚠. ⚠.

**Variant 1:** Accepts one shell LineString.

**Variant 2:** Accepts a shell LineString and an array of inner (hole) LineStrings. A geometry array can be constructed using the PostgreSQL array\_agg(), ARRAY[] or ARRAY() constructs.



**Note**

⚠. ⚠ **ST\_LineMerge** ⚠ **ST\_Dump** ⚠.



This function supports 3d and will not drop the z-index.

⚠: ⚠

⚠.

```
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)'));
```

Create a Polygon from an open LineString, using **ST\_StartPoint** and **ST\_AddPoint** to close it.

```
SELECT ST_MakePolygon(ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)))
FROM (
 SELECT ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)') As open_line) As foo;
```

⚠.

```
SELECT ST_AsEWKT(ST_MakePolygon('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15
 29.53 1)'));
```

```
st_asewkt

POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))
```

Create a Polygon from a LineString with measures

```
SELECT ST_AsEWKT(ST_MakePolygon('LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15
 29.53 2)')));
```

```
st_asewkt

POLYGONM((75.15 29.53 1,77 29 1,77.6 29.5 2,75.15 29.53 2))
```

**Example:** Creating a polygon with a hole.

The following SQL query creates a polygon with a hole.

```
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;
```

Create a set of province boundaries with holes representing lakes. The input is a table of province Polygons/MultiPolygons and a table of water linestrings. Lines forming lakes are determined by using [ST\\_IsClosed](#). The province linework is extracted by using [ST\\_Boundary](#). As required by [ST\\_MakePolygon](#), the boundary is forced to be a single LineString by using [ST\\_LineMerge](#). (However, note that if a province has more than one region or has islands this will produce an invalid polygon.) Using a LEFT JOIN ensures all provinces are included even if they have no lakes.



#### Note

NULL values in the `array_agg` function will result in a NULL value for the `ST_MakePolygon` function. Use `CASE` to handle NULL values.

```
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
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;
```

Another technique is to utilize a correlated subquery and the `ARRAY()` constructor that converts a row set to an array.

```
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;
```

**Example:**

[ST\\_BuildArea](#) [ST\\_Polygon](#)

### 7.3.8 ST\_Point

`ST_Point` — Creates a Point with X, Y and SRID values.

## Synopsis

geometry **ST\_Point**(float x, float y);

geometry **ST\_Point**(float x, float y, integer srid=unknown);

☒☒

Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for **ST\_MakePoint** that takes just X and Y.



### Note

For geodetic coordinates, X is longitude and Y is latitude

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with **ST\_SetSRID** to mark the srid on the geometry.



This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

☒☒: ☒☒

```
SELECT ST_Point(-71.104, 42.315);
```

Creating a point with SRID specified:

```
SELECT ST_Point(-71.104, 42.315, 4326);
```

Alternative way of specifying SRID:

```
SELECT ST_SetSRID(ST_Point(-71.104, 42.315), 4326);
```

☒☒: ☒☒☒

Create **geography** points using the `::` cast syntax:

```
SELECT ST_Point(-71.104, 42.315, 4326)::geography;
```

Pre-PostGIS 3.2 code, using `CAST`:

```
SELECT CAST(ST_SetSRID(ST_Point(-71.104, 42.315), 4326) AS geography);
```

If the point coordinates are not in a geodetic coordinate system (such as WGS84), then they must be reprojected before casting to a geography. In this example a point in Pennsylvania State Plane feet (SRID 2273) is projected to WGS84 (SRID 4326).

```
SELECT ST_Transform(ST_Point(3637510, 3014852, 2273), 4326)::geography;
```

☒☒

**ST\_MakePoint**, **ST\_PointZ**, **ST\_PointM**, **ST\_PointZM**, **ST\_SetSRID**, **ST\_Transform**

### 7.3.9 ST\_PointZ

ST\_PointZ — Creates a Point with X, Y, Z and SRID values.

#### Synopsis

geometry **ST\_PointZ**(float x, float y, float z, integer srid=unknown);

☒☒

☒☒☒☒☒☒☒☒ ST\_Point ☒☒☒☒☒☒. ST\_MakePoint ☒☒☒☒ OGC ☒☒☒☒☒.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry.

☒☒

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, srid => 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4)
```

☒☒

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.10 ST\_PointM

ST\_PointM — Creates a Point with X, Y, M and SRID values.

#### Synopsis

geometry **ST\_PointM**(float x, float y, float m, integer srid=unknown);

☒☒

☒☒☒☒☒☒☒☒ ST\_Point ☒☒☒☒☒☒. ST\_MakePoint ☒☒☒☒ OGC ☒☒☒☒☒.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry.

☒☒

```
SELECT ST_PointM(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4, srid => 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4)
```

☒☒

[ST\\_MakePoint](#), [ST\\_PointFromText](#), [ST\\_SetSRID](#), [ST\\_MakePointM](#)

### 7.3.11 ST\_PointZM

`ST_PointZM` — Creates a Point with X, Y, Z, M and SRID values.

#### Synopsis

geometry **ST\_PointZM**(float x, float y, float z, float m, integer srid=unknown);

☒☒

☒☒☒☒☒☒☒☒ `ST_Point` ☒☒☒☒☒☒☒. `ST_MakePoint` ☒☒☒☒ OGC ☒☒☒☒☒☒.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with `ST_SetSRID` to mark the srid on the geometry.

☒☒

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, srid => 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5)
```

☒☒

[ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_PointM](#), [ST\\_PointZ](#), [ST\\_SetSRID](#)

### 7.3.12 ST\_Polygon

`ST_Polygon` — Creates a Polygon from a LineString with a specified SRID.

#### Synopsis

geometry **ST\_Polygon**(geometry lineString, integer srid);

☒☒

Returns a polygon built from the given LineString and sets the spatial reference system from the `srid`.

`ST_Polygon` is similar to [ST\\_MakePolygon](#) Variant 1 with the addition of setting the SRID.

, [ST\\_MakePoint](#), [ST\\_SetSRID](#)

**Note**

This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). This method implements the [SQL/MM specification](#). [SQL-MM 3: 8.3.2](#). This function supports 3d and will not drop the z-index.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the [SQL/MM specification](#). [SQL-MM 3: 8.3.2](#)
- ✔ This function supports 3d and will not drop the z-index.

☒☒

Create a 2D polygon.

```
SELECT ST_AsText(ST_Polygon('LINESTRING(75 29, 77 29, 77 29, 75 29)::geometry, 4326));
-- result --
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Create a 3D polygon.

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 1)'), 4326));
-- result --
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

☒☒

[ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromEWKT](#), [ST\\_GeomFromText](#), [ST\\_LineMerge](#), [ST\\_MakePolygon](#)

### 7.3.13 ST\_TileEnvelope

**ST\_TileEnvelope** — Creates a rectangular Polygon in [Web Mercator](#) (SRID:3857) using the [XYZ tile system](#).

#### Synopsis

```
geometry ST_TileEnvelope(integer tileZoom, integer tileX, integer tileY, geometry bounds=SRID=3857;LINESTRING(20037508.342789 -20037508.342789,20037508.342789 20037508.342789), float margin=0.0);
```

☒☒

Creates a rectangular Polygon giving the extent of a tile in the [XYZ tile system](#). The tile is specified by the zoom level Z and the XY index of the tile in the grid at that level. Can be used to define the tile bounds required by [ST\\_AsMVTGeom](#) to convert geometry into the MVT tile coordinate space.

By default, the tile envelope is in the [Web Mercator](#) coordinate system (SRID:3857) using the standard range of the Web Mercator system (-20037508.342789, 20037508.342789). This is the most common coordinate system used for MVT tiles. The optional bounds parameter can be used to generate tiles in

any coordinate system. It is a geometry that has the SRID and extent of the "Zoom Level zero" square within which the XYZ tile system is inscribed.

The optional `margin` parameter can be used to expand a tile by the given percentage. E.g. `margin=0.125` expands the tile by 12.5%, which is equivalent to `buffer=512` when the tile extent size is 4096, as used in [ST\\_AsMVTGeom](#). This is useful to create a tile buffer to include data lying outside of the tile's visible area, but whose existence affects the tile rendering. For example, a city name (a point) could be near an edge of a tile, so its label should be rendered on two tiles, even though the point is located in the visible area of just one tile. Using expanded tiles in a query will include the city point in both tiles. Use a negative value to shrink the tile instead. Values less than -0.5 are prohibited because that would eliminate the tile completely. Do not specify a margin when using with `ST_AsMVTGeom`. See the example for [ST\\_AsMVT](#).

`ST_AsText( ST_TileEnvelope(2, 1, 1) )` SRID `4326`.

`2.1.0` `ST_AsText( ST_TileEnvelope(2, 1, 1) )`.

`ST_AsText( ST_TileEnvelope(2, 1, 1) )`

```
SELECT ST_AsText(ST_TileEnvelope(2, 1, 1));
```

```

st_astext

POLYGON((-10018754.1713945 0,-10018754.1713945 10018754.1713945,0 10018754.1713945,0 ←
 0,-10018754.1713945 0))
```

```
SELECT ST_AsText(ST_TileEnvelope(3, 1, 1, ST_MakeEnvelope(-180, -90, 180, 90, 4326)));
```

```

 st_astext

POLYGON((-135 45,-135 67.5,-90 67.5,-90 45,-135 45))
```

`ST`

[ST\\_MakeEnvelope](#)

### 7.3.14 ST\_HexagonGrid

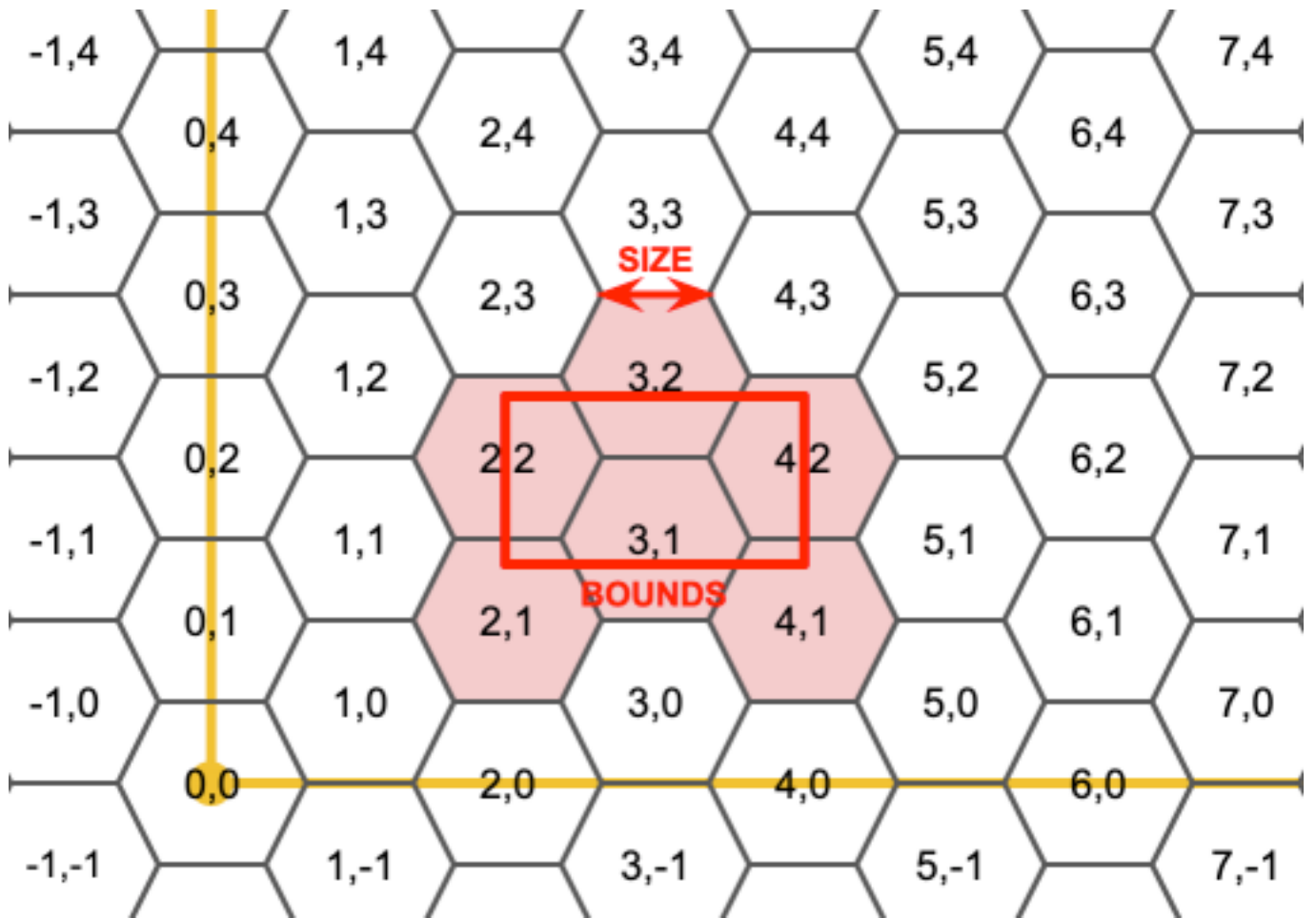
`ST_HexagonGrid` — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

#### Synopsis

setof record `ST_HexagonGrid`(float8 size, geometry bounds);

`ST`

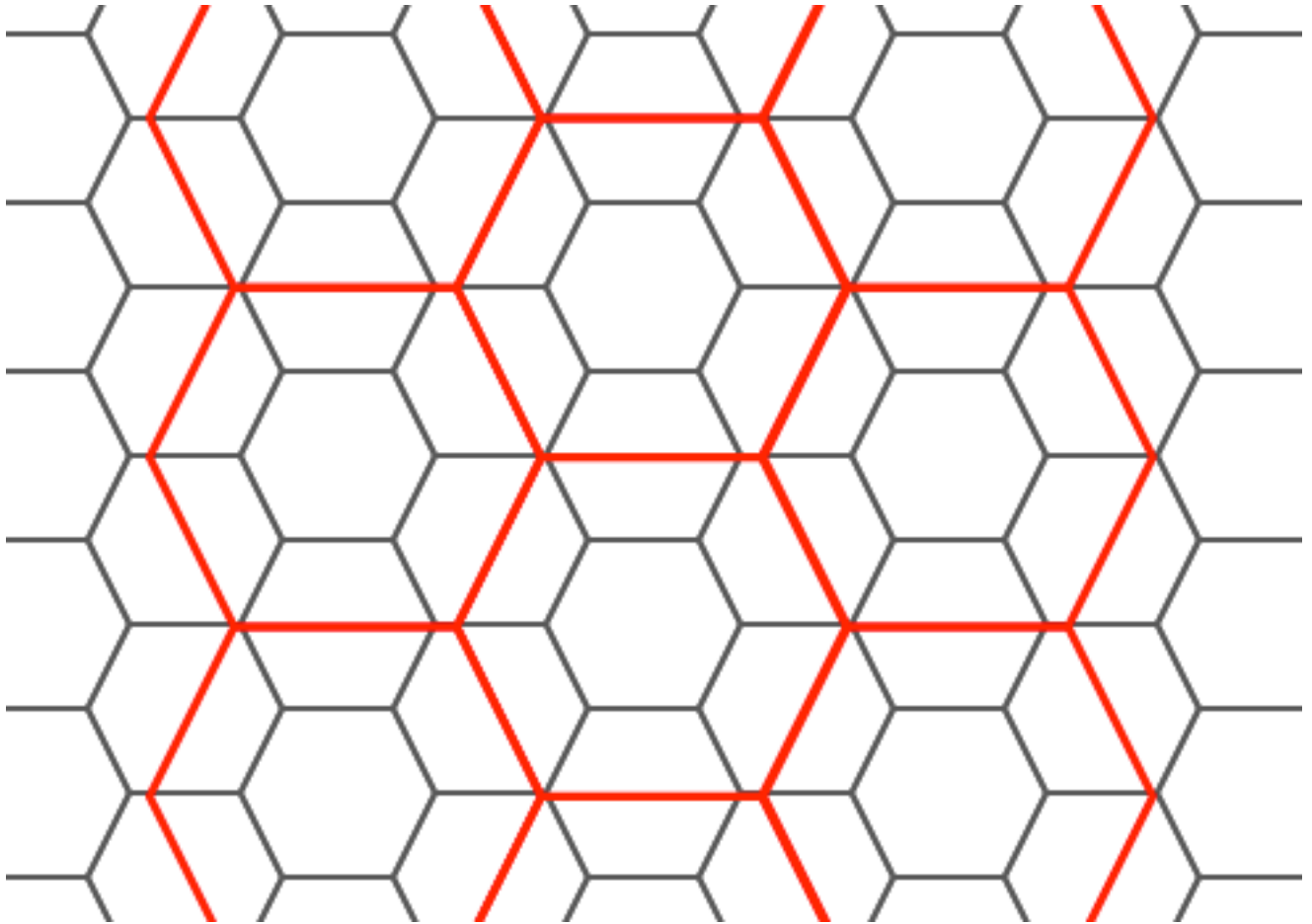
Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe, this is not the [H3](#) tiling scheme.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, `Tiling(SRS, Size)`. This function answers the question: what hexagons in a given `Tiling(SRS, Size)` overlap with a given bounds.



The SRS for the output hexagons is the SRS provided by the bounds geometry.

Doubling or tripling the edge size of the hexagon generates a new parent tiling that fits with the origin tiling. Unfortunately, it is not possible to generate parent hexagon tilings that the child tiles perfectly fit inside.





2.1.0 ☒☒☒☒☒☒☒☒☒☒.

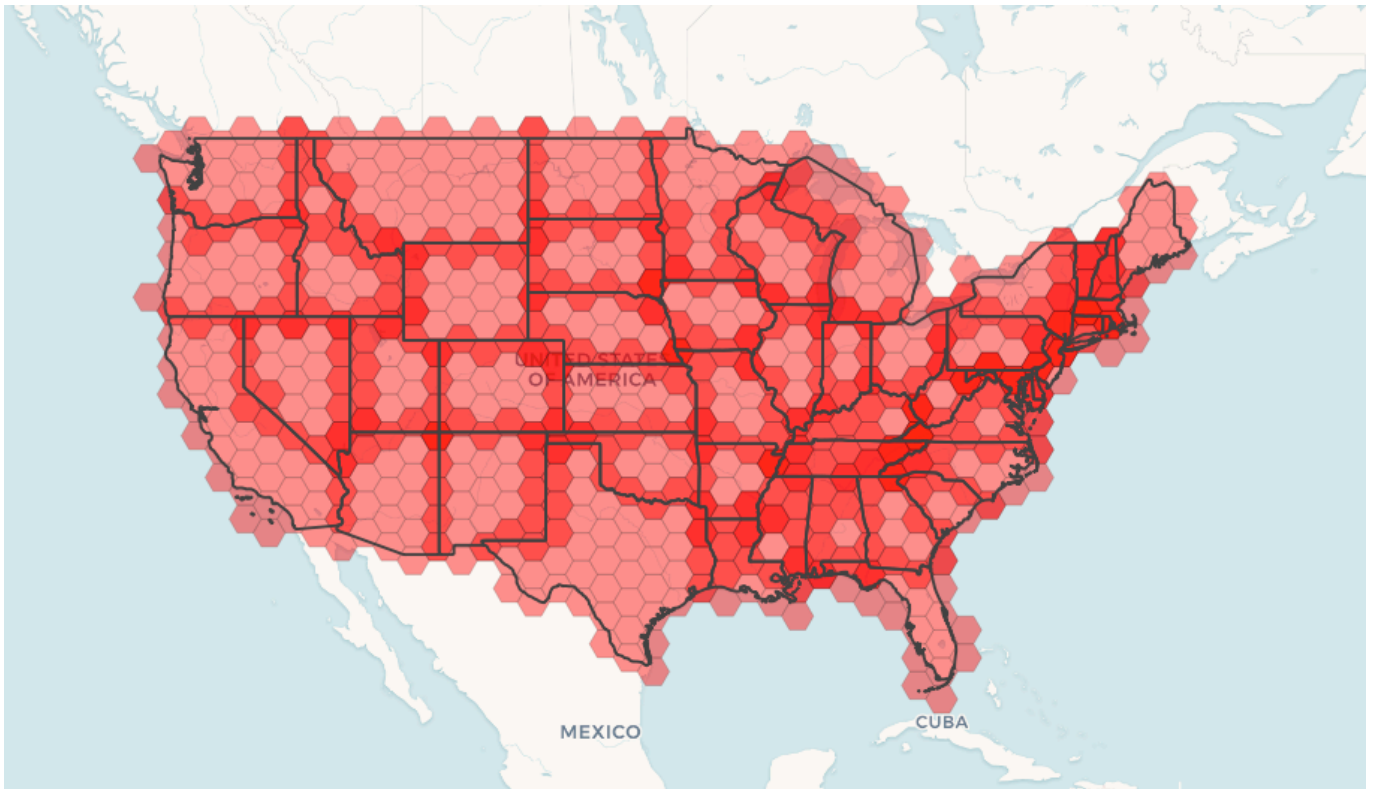
☒☒: ☒☒☒☒☒☒☒☒

To do a point summary against a hexagonal tiling, generate a hexagon grid using the extent of the points as the bounds, then spatially join to that grid.

```
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;
```

☒☒: ☒☒☒☒☒☒☒☒☒☒

If we generate a set of hexagons for each polygon boundary and filter out those that do not intersect their hexagons, we end up with a tiling for each polygon.



Tiling states results in a hexagon coverage of each state, and multiple hexagons overlapping at the borders between states.

**Note**

The LATERAL keyword is implied for set-returning functions when referring to a prior table in the FROM list. So CROSS JOIN LATERAL, CROSS JOIN, or just plain , are equivalent constructs for this example.

```
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)
```



[ST\\_EstimatedExtent](#), [ST\\_MakePoint](#), [ST\\_Point](#), [ST\\_SRID](#)

### 7.3.15 ST\_Hexagon

**ST\_Hexagon** — Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.

## Synopsis

geometry **ST\_Hexagon**(float8 size, integer cell\_i, integer cell\_j, geometry origin);

☒☒

Uses the same hexagon tiling concept as [ST\\_HexagonGrid](#), but generates just one hexagon at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Hexagons are generated with no SRID set, so use [ST\\_SetSRID](#) to set the SRID to the one you expect.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

### Example: Creating a hexagon at the origin

```
SELECT ST_AsText(ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857));

POLYGON((-1 0,-0.5
 -0.866025403784439,0.5
 -0.866025403784439,1
 0,0.5
 0.866025403784439,-0.5
 0.866025403784439,-1 0))
```

☒☒

[ST\\_TileEnvelope](#), [ST\\_MakePoint](#), [ST\\_SetSRID](#)

## 7.3.16 ST\_SquareGrid

**ST\_SquareGrid** — Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

### Synopsis

setof record **ST\_SquareGrid**(float8 size, geometry bounds);

☒☒

Starts with the concept of a square tiling of the plane. For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique square tiling of the plane, `Tiling(SRS, Size)`. This function answers the question: what grids in a given `Tiling(SRS, Size)` overlap with a given bounds.

The SRS for the output squares is the SRS provided by the bounds geometry.

Doubling or edge size of the square generates a new parent tiling that perfectly fits with the original tiling. Standard web map tilings in mercator are just powers-of-two square grids in the mercator plane.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



## Synopsis

geometry **ST\_Square**(float8 size, integer cell\_i, integer cell\_j, geometry origin);

☒☒

Uses the same square tiling concept as [ST\\_SquareGrid](#), but generates just one square at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Squares are generated with no SRID set, so use [ST\\_SetSRID](#) to set the SRID to the one you expect.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

### Example: Creating a square at the origin

```
SELECT ST_AsText(ST_SetSRID(ST_Square(1.0, 0, 0), 3857));

POLYGON((0 0,0 1,1 1,1 0,0 0))
```

☒☒

[ST\\_TileEnvelope](#), [ST\\_MakeLine](#), [ST\\_MakePolygon](#)

## 7.3.18 ST\_Letters

**ST\_Letters** — Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.

### Synopsis

geometry **ST\_Letters**(text letters, json font);

☒☒

Uses a built-in font to render out a string as a multipolygon geometry. The default text height is 100.0, the distance from the bottom of a descender to the top of a capital. The default start position places the start of the baseline at the origin. Over-riding the font involves passing in a json map, with a character as the key, and base64 encoded TWKB for the font shape, with the fonts having a height of 1000 units from the bottom of the descenders to the tops of the capitals.

The text is generated at the origin by default, so to reposition and resize the text, first apply the [ST\\_Scale](#) function and then apply the [ST\\_Translate](#) function.

Availability: 3.3.0

---

例: 文字を生成する

```
SELECT ST_AsText(ST_Letters('Yo'), 1);
```



*Letters generated by ST\_Letters*

#### Example: Scaling and moving words

```
SELECT ST_Translate(ST_Scale(ST_Letters('Yo'), 10, 10), 100,100);
```

例

[ST\\_AsTWKB](#), [ST\\_Scale](#), [ST\\_Translate](#)

## 7.4 ST\_GeometryType (accessor)

### 7.4.1 GeometryType

GeometryType — ST\_Geometry 文字列を返す。

#### Synopsis

```
text GeometryType(geometry geomA);
```

例

ST\_GeometryType(ST\_Geometry('LINESTRING(0 0,1 1)')). 例: 'LINESTRING', 'POLYGON', 'MULTIPOINT' 等。

OGC 簡易 s2.1.1.1 - 文字列を返す。例: 'LINESTRING', 'POLYGON', 'MULTIPOINT' 等。



**Note**

'POINTM' 支持 3D 点。

PostGIS 2.0.0 支持 TIN 几何类型。

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

SQL

```
SELECT GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
geometrytype

LINESTRING
```

```
SELECT ST_GeometryType(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))
))'));
--result
POLYHEDRALSURFACE
```

```
SELECT GeometryType(geom) as result
FROM
 (SELECT
 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 geom
) AS g;
result

TIN
```









### 7.4.4 ST\_CoordDim

ST\_CoordDim — ST\_Geometry

#### Synopsis

```
integer ST_CoordDim(geometry geomA);
```

ST\_Geometry

MM, **ST\_NDims**

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.3
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

```
SELECT ST_CoordDim('CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)');
 ---result--
 3

 SELECT ST_CoordDim(ST_Point(1,2));
 --result--
 2
```

**ST\_NDims**

### 7.4.5 ST\_Dimension

ST\_Dimension — ST\_Geometry

#### Synopsis

```
integer ST_Dimension(geometry g);
```





**Note**

1.3.4 (curve) 1.3.4

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.

Examples

```
SELECT sometable.field1, sometable.field1,
 (ST_Dump(sometable.geom)).geom AS geom
FROM sometable;

-- Break a compound curve into its constituent linestrings and circularstrings
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
FROM (SELECT (ST_Dump(p_geom)).geom AS geom
 FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0
1))') AS p_geom) AS b
) AS a;
 st_asewkt | st_hasarc
-----+-----
CIRCULARSTRING(0 0,1 1,1 0) | t
LINESTRING(1 0,0 1) | f
(2 rows)
```

Examples, TIN

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT (a.p_geom).path[1] As path, ST_AsEWKT((a.p_geom).geom) As geom_ewkt
FROM (SELECT ST_Dump(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 p_geom) AS a;

path | geom_ewkt
-----+-----
1 | POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0))
2 | POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
3 | POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))
4 | POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0))
5 | POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0))
6 | POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
```



- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.

**Classic Explode a Table of LineStrings into nodes**

```
SELECT edge_id, (dp).path[1] As index, ST_AsText((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;
edge_id | index | wktnode
-----+-----+-----
1 | 1 | POINT(1 2)
1 | 2 | POINT(3 4)
1 | 3 | POINT(10 10)
2 | 1 | POINT(3 5)
2 | 2 | POINT(5 6)
2 | 3 | POINT(9 10)
```

☒☒☒☒



```
SELECT path, ST_AsText(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;

```

| 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)

**TIN**

```

-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_DumpPoints(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 gdump
) AS g;
-- result --

```

| 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)

```

```

-- Triangle --
SELECT (g.gdump).path, ST_AsText((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_DumpPoints(ST_GeomFromEWKT('TRIANGLE ((
 0 0,
 0 9,
 9 0,
 0 0
))')) AS gdump
) AS g;
-- result --
path | wkt
-----+-----
{1} | POINT(0 0)
{2} | POINT(0 9)
{3} | POINT(9 0)
{4} | POINT(0 0)

```

```

-- TIN --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
FROM
 (SELECT
 ST_DumpPoints(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 gdump

```



```

SELECT path, ST_AsText(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;

```

| path    | b'' b'' | st_astext           |
|---------|---------|---------------------|
| {1,1}   | b'' b'' | LINESTRING(1 1,3 3) |
| {1,2}   | b'' b'' | LINESTRING(3 3,4 4) |
| {2,1,1} | b'' b'' | LINESTRING(5 5,6 6) |
| {2,1,2} | b'' b'' | LINESTRING(6 6,7 7) |
| {2,1,3} | b'' b'' | LINESTRING(7 7,5 5) |

(5 rows)

### ☒☒☒☒, TIN ☒☒☒☒☒☒

```

-- Triangle --
SELECT path, ST_AsText(geom)
FROM (
 SELECT (ST_DumpSegments(g.geom)).*
 FROM (SELECT 'TRIANGLE((
 0 0,
 0 9,
 9 0,
 0 0
))'::geometry AS geom
) AS g
) j;

```

| path  | b'' b'' | st_astext           |
|-------|---------|---------------------|
| {1,1} | b'' b'' | LINESTRING(0 0,0 9) |
| {1,2} | b'' b'' | LINESTRING(0 9,9 0) |
| {1,3} | b'' b'' | LINESTRING(9 0,0 0) |

(3 rows)

```

-- TIN --
SELECT path, ST_AsEWKT(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;

```



```

SELECT path, ST_AsEWKT(geom) As geom
FROM ST_DumpRings(
 ST_GeomFromEWKT('POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 ←
 5132839 1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 ←
 1,-8148924 5132394 1,
 -8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 ←
 1,-8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,
 -8150305 5132788 1,-8149064 5133092 1),
 (-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 ←
 1,-8149362 5132394 1)))')
) as foo;

```

| path | geom                                                                                                                                                                                                                                                                                                                                                           |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {0}  | POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767  ←<br>1,-8148958 5132508 1,<br>  -8148941 5132466 1,-8148924 5132394 1,<br>  -8148903 5132210 1,-8148930 5131967 1,<br>  -8148992 5131978 1,-8149237 5132093 1,<br>  -8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305  ←<br>5132788 1,-8149064 5133092 1)) |
| {1}  | POLYGON((-8149362 5132394 1,-8149446 5132501 1,<br>  -8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))                                                                                                                                                                                                                                                |



geometry\_dump, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries

### 7.4.10 ST\_EndPoint

ST\_EndPoint — ST\_LineString ST\_CircularString

#### Synopsis

geometry ST\_EndPoint(geometry g);



LINSTRING ST\_CIRCULARLINSTRING POINT. LINSTRING ST\_CIRCULARLINSTRING NULL.

- ✓ This method implements the SQL/MM specification. SQL-MM 3: 7.1.4
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This method supports Circular Strings and Curves.

**Note**  
 2.0.0. PostGIS 2.0.0 NULL. 2.0.0 NULL.



☒☒

```

SELECT ST_AsText(ST_Envelope('POINT(1 3)::geometry'));
 st_astext

 POINT(1 3)
(1 row)

SELECT ST_AsText(ST_Envelope('LINESTRING(0 0, 1 3)::geometry'));
 st_astext

 POLYGON((0 0,0 3,1 3,1 0,0 0))
(1 row)

SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))::geometry ←
));
 st_astext

 POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)
SELECT ST_AsText(ST_Envelope('POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0))':: ←
geometry));
 st_astext

 POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))
(1 row)

SELECT Box3D(geom), Box2D(geom), ST_AsText(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.*

```

SELECT ST_AsText(ST_Envelope(
 ST_Collect(
 ST_GeomFromText('LINESTRING(55 75,125 150)'),
 ST_Point(20, 80)
)
));

```





```
);
st_asewkt

LINESTRING(0 0 1,1 1 1,1 2 1,1 1 1,0 0 1)
```

☐☐

[ST\\_InteriorRingN](#), [ST\\_Boundary](#), [ST\\_NumInteriorRings](#)

### 7.4.13 ST\_GeometryN

ST\_GeometryN — ST\_Geometry 子字符串。

#### Synopsis

geometry **ST\_GeometryN**(geometry geomA, integer n);

☐☐

几何对象，(X,Y) 点，(X,Y,Z) 点，多线 (multicurve) 或 (X,Y) 多线。1-N 子字符串，NULL 返回 NULL。



**Note**  
0.8.0 版本实现 OGC 子字符串 1-字符串。返回 0-字符串。



**Note**  
子字符串 ST\_Dump 返回结果，子字符串。

版本: 2.0.0 版本，返回 TIN 子字符串。

版本: 2.0.0 版本返回 NULL 子字符串。2.0.0 版本 ST\_GeometryN(..,1) 子字符串。

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 9.1.5
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```
--Extracting a subset of points from a 3d multipoint
SELECT n, ST_AsEWKT(ST_GeometryN(geom, n)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('MULTIPOINT((1 2 7), (3 4 7), (5 6 7), (8 9 10))')),
(ST_GeomFromEWKT('MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))'))
)As foo(geom)
CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

| n | geomewkt                                |
|---|-----------------------------------------|
| 1 | POINT(1 2 7)                            |
| 2 | POINT(3 4 7)                            |
| 3 | POINT(5 6 7)                            |
| 4 | POINT(8 9 10)                           |
| 1 | CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5) |
| 2 | LINestring(10 11,12 11)                 |

```
--Extracting all geometries (useful when you want to assign an id)
SELECT gid, n, ST_GeometryN(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

☒☒☒☒☒, TIN ☒☒☒☒☒☒

```
-- Polyhedral surface example
-- Break a Polyhedral surface into its faces
SELECT ST_AsEWKT(ST_GeometryN(p_geom,3)) As geom_ewkt
FROM (SELECT 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 p_geom) AS a;
```

| geom_ewkt                                |
|------------------------------------------|
| POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)) |

```
-- TIN --
SELECT ST_AsEWKT(ST_GeometryN(geom,2)) as wkt
FROM
(SELECT
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
)))
```



```
SELECT ST_GeometryType(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) ←
))'));
--result
ST_PolyhedralSurface
```

```
SELECT ST_GeometryType(geom) as result
FROM
 (SELECT
 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 g;
result

ST_Tin
```

☒☒

**GeometryType**

**7.4.15 ST\_HasArc**

ST\_HasArc — Tests if a geometry contains a circular arc

**Synopsis**

boolean **ST\_HasArc**(geometry geomA);

☒☒

☒☒☒☒☒☒☒☒☒☒, ☒☒☒, ☒☒☒☒☒☒☒ TRUE ☒☒☒☒☒☒.

1.2.2 ☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_HasArc(ST_Collect('LINESTRING(1 2, 3 4, 5 6)', 'CIRCULARSTRING(1 1, 2 3, 4 5, 6 6, 7, 5 6)'));
 st_hasarc

 t
```

☒☒

[ST\\_CurveToLine](#), [ST\\_PointN](#)

## 7.4.16 ST\_InteriorRingN

ST\_InteriorRingN — Returns the *n*th interior ring of a polygon.

### Synopsis

geometry **ST\_InteriorRingN**(geometry a\_polygon, integer n);

☒☒

*n* must be a positive integer. If *n* is greater than the number of interior rings, the function returns NULL.

Note!

### Note

This function is implemented in the `postgis` extension. It is available in the `ST_Dump` function.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(ST_InteriorRingN(geom, 1)) As geom
FROM (SELECT ST_BuildArea(
 ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
 ST_Buffer(ST_Point(1, 2), 10,3))) As geom
) as foo;
```

☒☒

[ST\\_ExteriorRing](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.4.17 ST\_NumCurves

ST\_NumCurves — Return the number of component curves in a CompoundCurve.

#### Synopsis

```
integer ST_NumCurves(geometry a_compoundcurve);
```

☒☒

Return the number of component curves in a CompoundCurve, zero for an empty CompoundCurve, or NULL for a non-CompoundCurve input.



This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5



This function supports 3d and will not drop the z-index.

☒☒

```
-- Returns 3
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)
)');

-- Returns 0
SELECT ST_NumCurves('COMPOUNDCURVE EMPTY');
```

☒☒

[ST\\_CurveN](#), [ST\\_Dump](#), [ST\\_ExteriorRing](#), [ST\\_NumInteriorRings](#), [ST\\_NumGeometries](#)

### 7.4.18 ST\_CurveN

ST\_CurveN — Returns the Nth component curve geometry of a CompoundCurve.

#### Synopsis

```
geometry ST_CurveN(geometry a_compoundcurve, integer index);
```

☒☒

Returns the Nth component curve geometry of a CompoundCurve. The index starts at 1. Returns NULL if the geometry is not a CompoundCurve or the index is out of range.



This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5



This function supports 3d and will not drop the z-index.



```

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 1 1)::geometry');
 st_isclosed

 f
(1 row)

postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1, 0 0)::geometry');
 st_isclosed

 t
(1 row)

postgis=# SELECT ST_IsClosed('MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))::geometry');
 st_isclosed

 f
(1 row)

postgis=# SELECT ST_IsClosed('POINT(0 0)::geometry');
 st_isclosed

 t
(1 row)

postgis=# SELECT ST_IsClosed('MULTIPOINT((0 0), (1 1))::geometry');
 st_isclosed

 t
(1 row)

```

☒☒☒☒☒☒

```

-- A cube --
SELECT ST_IsClosed(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) ←
))'));

 st_isclosed

 t

-- Same as cube but missing a side --
SELECT ST_IsClosed(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)))'));

 st_isclosed

 f

```







☒☒

```

SELECT ST_IsEmpty(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY'));
 st_isempty

t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON EMPTY'));
 st_isempty

t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));

 st_isempty

f
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2)))') = false;
?column?

t
(1 row)

SELECT ST_IsEmpty(ST_GeomFromText('CIRCULARSTRING EMPTY'));
 st_isempty

t
(1 row)

```

### 7.4.22 ST\_IsPolygonCCW

`ST_IsPolygonCCW` — Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.

#### Synopsis

boolean **ST\_IsPolygonCCW** ( geometry geom );

☒☒

Returns true if all polygonal components of the input geometry use a counter-clockwise orientation for their exterior ring, and a clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.



#### Note

Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

**Note**

If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both `ST_IsPolygonCW` and `ST_IsPolygonCCW` will return false.

2.2.0



This function supports 3d and will not drop the z-index.



This function supports M coordinates.

[ST\\_ForcePolygonCW](#) , [ST\\_ForcePolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.4.23 ST\_IsPolygonCW

`ST_IsPolygonCW` — Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

#### Synopsis

boolean **ST\_IsPolygonCW** ( geometry geom );

Returns true if all polygonal components of the input geometry use a clockwise orientation for their exterior ring, and a counter-clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.

**Note**

Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

**Note**

If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both `ST_IsPolygonCW` and `ST_IsPolygonCCW` will return false.

2.2.0



This function supports 3d and will not drop the z-index.



This function supports M coordinates.





¶¶

¶¶¶ M ¶¶¶¶¶¶¶¶. M ¶¶¶¶¶¶¶ NULL ¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶.



**Note**

¶¶¶ (¶¶) OGC ¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶ (extractor) ¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification.
- ✔ This function supports 3d and will not drop the z-index.

¶¶

```
SELECT ST_M(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_m

 4
(1 row)
```

¶¶

[ST\\_GeomFromEWKT](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_Z](#)

### 7.4.27 ST\_MemSize

ST\_MemSize — ST\_Geometry ¶¶¶¶¶¶¶¶¶¶¶¶.

**Synopsis**

integer **ST\_MemSize**(geometry geomA);

¶¶

ST\_Geometry ¶¶¶¶¶¶¶¶¶¶¶¶.

This complements the PostgreSQL built-in [database object functions](#) `pg_column_size`, `pg_size_pretty`, `pg_relation_size`, `pg_total_relation_size`.



**Note**

`pg_relation_size` which gives the byte size of a table may return byte size lower than `ST_MemSize`. This is because `pg_relation_size` does not add toasted table contribution and large geometries are stored in TOAST tables.

`pg_total_relation_size` ¶¶¶¶¶¶, TOAST ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

`pg_column_size` returns how much space a geometry would take in a column considering compression, so may be lower than `ST_MemSize`

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Changed: 2.2.0 name changed to ST\_MemSize to follow naming convention.

☒

```
--Return how much byte space Boston takes up in our Mass data set
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)*1.00 /
 SUM(ST_MemSize(geom))*100 As numeric(10,2)) As perbos
FROM towns;
```

| totgeomsum | bossum | perbos |
|------------|--------|--------|
| 1522 kB    | 30 kB  | 1.99   |

```
SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));

```

73

```
--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_MemSize(geom)) As geomsize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsize pergeom

262144 96238 36.71188354492187500000
```

### 7.4.28 ST\_NDims

ST\_NDims — ST\_Geometry

#### Synopsis

integer **ST\_NDims**(geometry g1);

☒

PostGIS 2 - 2 (x,y), 3 - 3 (x,y,z), 3 - 2 (x,y,m), 4 - 3 (x,y,z,m)

- ✔ This function supports 3d and will not drop the z-index.









☒☒

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
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;
```

☒☒

[ST\\_NumInteriorRing](#), [ST\\_PointN](#)

### 7.4.33 ST\_NumInteriorRing

**ST\_NumInteriorRing** — Returns the number of interior rings in a polygon. **ST\_NumInteriorRings** is the preferred name.

#### Synopsis

integer **ST\_NumInteriorRing**(geometry a\_polygon);

☒☒

[ST\\_NumInteriorRings](#), [ST\\_PointN](#)

### 7.4.34 ST\_NumPatches

**ST\_NumPatches** — Returns the number of patches in a geometry. Returns NULL if the geometry is NULL.

#### Synopsis

integer **ST\_NumPatches**(geometry g1);

☒☒

Available since PostGIS 2.0.0. **ST\_NumGeometries** is the preferred name. **ST\_NumGeometries** is the preferred name.

2.0.0

-  This function supports 3d and will not drop the z-index.
-  This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



### 7.4.36 ST\_PatchN

ST\_PatchN — ST\_Geometry

#### Synopsis

geometry **ST\_PatchN**(geometry geomA, integer n);

POLYHEDRALSURFACE, POLYHEDRALSURFACEM 1- N ( ) . NULL . ST\_GeometryN . ST\_GeometryN .



**Note**

1-



**Note**

ST\_Dump

2.0.0

- ✓ This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5
- ✓ This function supports 3d and will not drop the z-index.
- ✓ This function supports Polyhedral surfaces.

```
--Extract the 2nd face of the polyhedral surface
SELECT ST_AsEWKT(ST_PatchN(geom, 2)) As geomewkt
FROM (
VALUES (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 foo(geom);

geomewkt
-----+-----
POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))
```

[ST\\_AsEWKT](#), [ST\\_GeomFromEWKT](#), [ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

### 7.4.37 ST\_PointN

ST\_PointN — ST\_LineString ST\_CircularString

#### Synopsis

geometry **ST\_PointN**(geometry a\_linestring, integer n);

ST\_PointN returns the Nth point of the linestring. If N is 0, it returns the start point. If N is -1, it returns the end point. If N is NULL, it returns NULL.



**Note**

0.8.0 OGC Simple Features Implementation Specification for SQL 1.1. OGC Simple Features Implementation Specification (SQL/MM) 2.0.0. 0-Indexing.



**Note**

ST\_PointN returns the Nth point of the linestring, ST\_Dump returns the points of the linestring.

- This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5
- This function supports 3d and will not drop the z-index.
- This method supports Circular Strings and Curves.



**Note**

PostGIS 2.0.0: ST\_PointN returns the Nth point of the linestring. PostGIS 2.0.0: ST\_PointN returns the Nth point of the linestring. 2.0.0: ST\_PointN returns the Nth point of the linestring. NULL. 2.3.0: ST\_PointN returns the Nth point of the linestring (-1 indexing).

```
-- Extract all POINTs from a LINESTRING
SELECT ST_AsText(
 ST_PointN(
 column1,
 generate_series(1, ST_NPoints(column1))
))
FROM (VALUES ('LINESTRING(0 0, 1 1, 2 2)::geometry') AS foo;

st_astext

POINT(0 0)
```







ST\_LineString ST\_CircularString

```
SELECT ST_AsText(ST_StartPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)')::geometry ←
);
 st_astext

POINT(5 2)
```

ST\_EndPoint, ST\_PointN

### 7.4.40 ST\_Summary

ST\_Summary —

#### Synopsis

```
text ST_Summary(geometry g);
text ST_Summary(geography g);
```

Parameters:  
 geometry g: geometry object.

- M: M value.
- Z: Z value.
- B: bounding box.
- G: geometry type (name) code.
- S: surface type code.

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

1.2.2

2.0.0

2.1.0: S

2.2.0 TIN (curve)

☒☒

```

=# SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) as geom,
 ST_Summary(ST_GeogFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) geog;
-----+-----
geom | geog
-----+-----
LineString[B] with 2 points | Polygon[BGS] with 1 rings
 | ring 0 has 5 points
 :
(1 row)

=# SELECT ST_Summary(ST_GeogFromText('LINESTRING(0 0 1, 1 1 1)')) As geog_line,
 ST_Summary(ST_GeomFromText('SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←
 ') As geom_poly;
;
-----+-----
geog_line | geom_poly
-----+-----
LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings
 : ring 0 has 5 points
 :
(1 row)

```

☒☒

[PostGIS\\_DropBBox](#), [PostGIS\\_AddBBox](#), [ST\\_Force3DM](#), [ST\\_Force3DZ](#), [ST\\_Force2D](#), [geography](#)  
[ST\\_IsValid](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#)

### 7.4.41 ST\_X

ST\_X — Returns the X coordinate of a Point.

#### Synopsis

float **ST\_X**(geometry a\_point);

☒☒

☒☒☒☒ X ☒☒☒☒☒☒☒☒. X ☒☒☒☒☒☒☒☒ NULL ☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



**Note**

To get the minimum and maximum X value of geometry coordinates use the functions [ST\\_XMin](#) and [ST\\_XMax](#).



This method implements the SQL/MM specification. SQL-MM 3: 6.1.3



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_X(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_x

 1
(1 row)

SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y

 1.5
(1 row)
```

☒☒

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_XMax](#), [ST\\_XMin](#), [ST\\_Y](#), [ST\\_Z](#)

### 7.4.42 ST\_Y

`ST_Y` — Returns the Y coordinate of a Point.

#### Synopsis

float **ST\_Y**(geometry a\_point);

☒☒

☒☒☒☒ Y ☒☒☒☒☒☒☒☒. Y ☒☒☒☒☒☒☒☒ NULL ☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



#### Note

To get the minimum and maximum Y value of geometry coordinates use the functions [ST\\_YMin](#) and [ST\\_YMax](#).



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 6.1.4



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_Y(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_y

 2
(1 row)
```

```
SELECT ST_Y(ST_Centroid(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)')));
 st_y

 1.5
(1 row)
```

☒☒

[ST\\_Centroid](#), [ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_Z](#)

### 7.4.43 ST\_Z

**ST\_Z** — Returns the Z coordinate of a Point.

#### Synopsis

float **ST\_Z**(geometry a\_point);

☒☒

☒☒☒☒ Z ☒☒☒☒☒☒☒☒. Z ☒☒☒☒☒☒☒☒ NULL ☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



#### Note

To get the minimum and maximum Z value of geometry coordinates use the functions [ST\\_ZMin](#) and [ST\\_ZMax](#).



This method implements the SQL/MM specification.



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_Z(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_z

 3
(1 row)
```

☒☒

[ST\\_GeomFromEWKT](#), [ST\\_M](#), [ST\\_X](#), [ST\\_Y](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.4.44 ST\_Zmflag

ST\_Zmflag — ST\_Geometry

#### Synopsis

smallint **ST\_Zmflag**(geometry geomA);

ST\_Geometry

Values are: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
st_zmflag

 0

SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRINGM(1 2 3, 3 4 3)'));
st_zmflag

 1

SELECT ST_Zmflag(ST_GeomFromEWKT('CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)'));
st_zmflag

 2

SELECT ST_Zmflag(ST_GeomFromEWKT('POINT(1 2 3 4)'));
st_zmflag

 3
```

[ST\\_CoordDim](#), [ST\\_NDims](#), [ST\\_Dimension](#)

### 7.4.45 ST\_HasZ

ST\_HasZ — Checks if a geometry has a Z dimension.

#### Synopsis

boolean **ST\_HasZ**(geometry geom);



Checks if the input geometry has a Z dimension and returns a boolean value. If the geometry has a Z dimension, it returns true; otherwise, it returns false.

Geometry objects with a Z dimension typically represent three-dimensional (3D) geometries, while those without it are two-dimensional (2D) geometries.

This function is useful for determining if a geometry has elevation or height information.

Availability: 3.5.0



This function supports 3d and will not drop the z-index.



This function supports M coordinates.



```
SELECT ST_HasZ(ST_GeomFromText('POINT(1 2 3)'));
-- result
true
```

```
SELECT ST_HasZ(ST_GeomFromText('LINESTRING(0 0, 1 1)'));
-- result
false
```



[ST\\_Zmflag](#)

[ST\\_HasM](#)

#### 7.4.46 ST\_HasM

**ST\_HasM** — Checks if a geometry has an M (measure) dimension.

##### Synopsis

boolean **ST\_HasM**(geometry geom);



Checks if the input geometry has an M (measure) dimension and returns a boolean value. If the geometry has an M dimension, it returns true; otherwise, it returns false.

Geometry objects with an M dimension typically represent measurements or additional data associated with spatial features.

This function is useful for determining if a geometry includes measure information.

Availability: 3.5.0



This function supports 3d and will not drop the z-index.



This function supports M coordinates.





---

☒☒

[ST\\_RemovePoint](#), [ST\\_SetPoint](#)

## 7.5.2 ST\_CollectionExtract

`ST_CollectionExtract` — Given a geometry collection, returns a multi-geometry containing only elements of a specified type.

### Synopsis

geometry **ST\_CollectionExtract**(geometry collection);

geometry **ST\_CollectionExtract**(geometry collection, integer type);

☒☒

Given a geometry collection, returns a homogeneous multi-geometry.

If the *type* is not specified, returns a multi-geometry containing only geometries of the highest dimension. So polygons are preferred over lines, which are preferred over points.

If the *type* is specified, returns a multi-geometry containing only that type. If there are no sub-geometries of the right type, an EMPTY geometry is returned. Only points, lines and polygons are supported. The type numbers are:

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

For atomic geometry inputs, the geometry is returned unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using [ST\\_Multi](#).



### Warning

MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an [ST\\_Split](#) result.) This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

1.5.0 ☒☒☒☒☒☒☒☒☒☒.



### Note

Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.

---



**Warning**

This function does not ensure that the result is valid. In particular, a collection containing adjacent or overlapping Polygons will create an invalid MultiPolygon. This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

Single-element collection converted to an atomic geometry

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));

st_astext

POINT(0 0)
```

Nested single-element collection converted to an atomic geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));

st_astext

POINT(0 0)
```

Collection converted to a multi-geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));

st_astext

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

Nested heterogeneous collection flattened to a GeometryCollection:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ←
(LINESTRING(1 1, 2 2))')));

st_astext

GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

Collection of Polygons converted to an (invalid) MultiPolygon:

```
SELECT ST_AsText(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))')));

st_astext

MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

☒☒

[ST\\_CollectionExtract](#), [ST\\_Multi](#), [ST\\_IsValid](#), [ST\\_MakeValid](#)

## 7.5.4 ST\_CurveToLine

ST\_CurveToLine — Converts a geometry containing curves to a linear geometry.

### Synopsis

geometry **ST\_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance\_type, integer flags);

☒☒

Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types

Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given `tolerance` and options (32 segments per quadrant and no options by default).

The `tolerance\_type` argument determines interpretation of the `tolerance` argument. It can take the following values:

- 0 (default): Tolerance is max segments per quadrant.
- 1: Tolerance is max-deviation of line from curve, in source units.
- 2: Tolerance is max-angle, in radians, between generating radii.

The `flags` argument is a bitfield. 0 by default. Supported bits are:

- 1: Symmetric (orientation independent) output.
- 2: Retain angle, avoids reducing angles (segment lengths) when producing symmetric output. Has no effect when Symmetric flag is off.

Availability: 1.3.0

Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output.

Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
  - ✔ This method implements the SQL/MM specification. SQL-MM 3: 7.1.7
  - ✔ This function supports 3d and will not drop the z-index.
  - ✔ This method supports Circular Strings and Curves.
-

☒☒

```
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)')));

--Result --
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)

--3d example
SELECT ST_AsEWKT(ST_CurveToLine(ST_GeomFromEWKT('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
SELECT ST_AsText(ST_CurveToLine(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 ←
 150505,220227 150406)'),2));
st_astext

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)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
 'CIRCULARSTRING(0 0,100 -100,200 0)::geometry,
 20, -- Tolerance
 1, -- Above is max distance between curve and line
 1 -- Symmetric flag
));
st_astext

LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)

```

☒☒

[ST\\_LineToCurve](#)

## 7.5.5 ST\_Scroll

ST\_Scroll — Change start point of a closed LineString.

### Synopsis

geometry **ST\_Scroll**(geometry linestring, geometry point);



Changes the start/end point of a closed LineString to the given vertex *point*.

Availability: 3.2.0



This function supports 3d and will not drop the z-index.



This function supports M coordinates.



Make e closed line start at its 3rd vertex

```
SELECT ST_AsEWKT(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)'));
```

```
st_asewkt
```

```

```

```
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```



[ST\\_Normalize](#)

## 7.5.6 ST\_FlipCoordinates

`ST_FlipCoordinates` — Returns a version of a geometry with X and Y axis flipped.

### Synopsis

geometry **ST\_FlipCoordinates**(geometry geom);



Returns a version of the given geometry with X and Y axis flipped. Useful for fixing geometries which contain coordinates expressed as latitude/longitude (Y,X).

2.0.0         .



This method supports Circular Strings and Curves.



This function supports 3d and will not drop the z-index.



This function supports M coordinates.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).









☒☒

Forces the geometries into XYM mode. If a geometry has no M component, then a *Mvalue* M coordinate is tacked on. If it has a Z component, then Z is removed

☒☒☒☒: 2.1.0 ☒☒☒☒, ☒ 2.0.x ☒☒☒☒☒☒☒☒☒☒ ST\_Force\_3DM ☒☒☒☒☒.

Changed: 3.1.0. Added support for supplying a non-zero M value.



This method supports Circular Strings and Curves.

☒☒

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromEWKT('CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 2)')));
 st_asewkt

CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

SELECT ST_AsEWKT(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))'));
 st_asewkt

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))
```

☒☒

[ST\\_AsEWKT](#), [ST\\_Force2D](#), [ST\\_Force3DM](#), [ST\\_Force3D](#), [ST\\_GeomFromEWKT](#)

### 7.5.11 ST\_Force4D

ST\_Force4D — ☒☒☒ XYZM ☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

☒☒

Forces the geometries into XYZM mode. *Zvalue* and *Mvalue* is tacked on for missing Z and M dimensions, respectively.

☒☒☒☒: 2.1.0 ☒☒☒☒, ☒ 2.0.x ☒☒☒☒☒☒☒☒☒☒ ST\_Force\_4D ☒☒☒☒.

Changed: 3.1.0. Added support for supplying non-zero Z and M values.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.





- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_AsText(
 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
)
);
 st_astext

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))
(1 row)
```

☒☒

[ST\\_LineToCurve](#)

### 7.5.14 ST\_ForcePolygonCCW

`ST_ForcePolygonCCW` — Orients all exterior rings counter-clockwise and all interior rings clockwise.

#### Synopsis

geometry **ST\_ForcePolygonCCW** ( geometry geom );

☒☒

Forces (Multi)Polygons to use a counter-clockwise orientation for their exterior ring, and a clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports M coordinates.

☒☒

[ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#)

### 7.5.15 ST\_ForcePolygonCW

`ST_ForcePolygonCW` — Orients all exterior rings clockwise and all interior rings counter-clockwise.



**Synopsis**

geometry **ST\_ForcePolygonCW** ( geometry geom );



Forces (Multi)Polygons to use a clockwise orientation for their exterior ring, and a counter-clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.



2.2.0 .

-  This function supports 3d and will not drop the z-index.
-  This function supports M coordinates.



*ST\_ForcePolygonCCW* , *ST\_IsPolygonCCW* , *ST\_IsPolygonCW*





**7.5.16 ST\_ForceSFS**

ST\_ForceSFS —  SFS 1.1 .





**Synopsis**

geometry **ST\_ForceSFS**(geometry geomA);  
 geometry **ST\_ForceSFS**(geometry geomA, text version);



-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
-  This method supports Circular Strings and Curves.
-  This function supports 3d and will not drop the z-index.

**7.5.17 ST\_ForceRHR**

ST\_ForceRHR —  (orientation)  (Right-Hand Rule)   
.

**Synopsis**

geometry **ST\_ForceRHR**(geometry g);

☒☒

Forces the orientation of the vertices in a polygon to follow a Right-Hand-Rule, in which the area that is bounded by the polygon is to the right of the boundary. In particular, the exterior ring is orientated in a clockwise direction and the interior rings in a counter-clockwise direction. This function is a synonym for [ST\\_ForcePolygonCW](#)

Note!

#### Note

The above definition of the Right-Hand-Rule conflicts with definitions used in other contexts. To avoid confusion, it is recommended to use [ST\\_ForcePolygonCW](#).

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒☒ (polyhedral surface) ☒☒☒☒☒☒☒.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_AsEWKT(
 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))'
)
);
```

|                                                              | st_asewkt |
|--------------------------------------------------------------|-----------|
| -----                                                        |           |
| 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)) |           |
| (1 row)                                                      |           |

☒☒

[ST\\_ForcePolygonCCW](#) , [ST\\_ForcePolygonCW](#) , [ST\\_IsPolygonCCW](#) , [ST\\_IsPolygonCW](#) , [ST\\_BuildArea](#), [ST\\_Polygonize](#), [ST\\_Reverse](#)

## 7.5.18 ST\_LineExtend

[ST\\_LineExtend](#) — Returns a line extended forwards and backwards by specified distances.

### Synopsis

geometry **ST\_LineExtend**(geometry line, float distance\_forward, float distance\_backward=0.0);

☒☒

Returns a line extended forwards and backwards by adding new start (and end) points at the given distance(s). A distance of zero does not add a point. Only non-negative distances are allowed. The direction(s) of the added point(s) is determined by the first (and last) two distinct points of the line. Duplicate points are ignored.

Availability: 3.4.0



**Example: Extends a line 5 units forward and 6 units backward**

```
SELECT ST_AsText(ST_LineExtend('LINESTRING(0 0, 0 10)::geometry, 5, 6));

LINESTRING(0 -6,0 0,0 10,0 15)
```

☒☒

[ST\\_LineSubstring](#), [ST\\_LocateAlong](#), [ST\\_Project](#)

**7.5.19 ST\_LineToCurve**

ST\_LineToCurve — Converts a linear geometry to a curved geometry.

**Synopsis**

geometry **ST\_LineToCurve**(geometry geomANoncircular);

☒☒

Converts plain LINESTRING/POLYGON to CIRCULAR STRINGs and Curved Polygons. Note much fewer points are needed to describe the curved equivalent.



**Note**

If the input LINESTRING/POLYGON is not curved enough to clearly represent a curve, the function will return the same input geometry.

Availability: 1.3.0

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.

☒☒

```
-- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext,ST_AsText(foo.geom) As ↔
 non_curvedastext
 FROM (SELECT ST_Buffer('POINT(1 3)::geometry, 3) As geom) As foo;

curvedastext non_curvedastext
-----|-----
CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4 ↔
 3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,
1 0,-1.12132034355965 5.12132034355963,4 3)) | 3.49440883690764 ↔
 1.33328930094119,3.12132034355964 0.878679656440359, |
 | 2.66671069905881 ↔
 | 0.505591163092366,2.14805029
 | 0.228361402466141,
```





## 7.5.22 ST\_Project

ST\_Project — Returns a point projected from a start point by a distance and bearing (azimuth).

### 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);
```

☒☒

Returns a point projected from a point along a geodesic using a given distance and azimuth (bearing). This is known as the direct geodesic problem.

The two-point version uses the path from the first to the second point to implicitly define the azimuth and uses the distance as before.

The distance is given in meters. Negative values are supported.

The azimuth (also known as heading or bearing) is given in radians. It is measured clockwise from true north.

- North is azimuth zero (0 degrees)
- East is azimuth  $\pi/2$  (90 degrees)
- South is azimuth  $\pi$  (180 degrees)
- West is azimuth  $3\pi/2$  (270 degrees)

Negative azimuth values and values greater than  $2\pi$  (360 degrees) are supported.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.

Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth.

### Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)::geography, 100000, radians(45.0)));

POINT(0.635231029125537 0.639472334729198)
```

☒☒

[ST\\_Azimuth](#), [ST\\_Distance](#), [PostgreSQL function radians\(\)](#)

## 7.5.23 ST\_QuantizeCoordinates

ST\_QuantizeCoordinates — Sets least significant bits of coordinates to zero

## Synopsis

```
geometry ST_QuantizeCoordinates (geometry g , int prec_x , int prec_y , int prec_z , int prec_m);
```

☒☒

`ST_QuantizeCoordinates` determines the number of bits (N) required to represent a coordinate value with a specified number of digits after the decimal point, and then sets all but the N most significant bits to zero. The resulting coordinate value will still round to the original value, but will have improved compressibility. This can result in a significant disk usage reduction provided that the geometry column is using a **compressible storage type**. The function allows specification of a different number of digits after the decimal point in each dimension; unspecified dimensions are assumed to have the precision of the x dimension. Negative digits are interpreted to refer digits to the left of the decimal point, (i.e., `prec_x=-2` will preserve coordinate values to the nearest 100.

The coordinates produced by `ST_QuantizeCoordinates` are independent of the geometry that contains those coordinates and the relative position of those coordinates within the geometry. As a result, existing topological relationships between geometries are unaffected by use of this function. The function may produce invalid geometry when it is called with a number of digits lower than the intrinsic precision of the geometry.

Availability: 2.5.0

## Technical Background

PostGIS stores all coordinate values as double-precision floating point integers, which can reliably represent 15 significant digits. However, PostGIS may be used to manage data that intrinsically has fewer than 15 significant digits. An example is TIGER data, which is provided as geographic coordinates with six digits of precision after the decimal point (thus requiring only nine significant digits of longitude and eight significant digits of latitude.)

When 15 significant digits are available, there are many possible representations of a number with 9 significant digits. A double precision floating point number uses 52 explicit bits to represent the significand (mantissa) of the coordinate. Only 30 bits are needed to represent a mantissa with 9 significant digits, leaving 22 insignificant bits; we can set their value to anything we like and still end up with a number that rounds to our input value. For example, the value 100.123456 can be represented by the floating point numbers closest to 100.123456000000, 100.123456000001, and 100.123456432199. All are equally valid, in that `ST_AsText(geom, 6)` will return the same result with any of these inputs. As we can set these bits to any value, `ST_QuantizeCoordinates` sets the 22 insignificant bits to zero. For a long coordinate sequence this creates a pattern of blocks of consecutive zeros that is compressed by PostgreSQL more efficiently.



### Note

Only the on-disk size of the geometry is potentially affected by `ST_QuantizeCoordinates`. `ST_MemSize`, which reports the in-memory usage of the geometry, will return the the same value regardless of the disk space used by a geometry.

☒☒

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)::geometry, 4));
st_astext

POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)::geometry AS geom)
SELECT
 digits,
 encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
 ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;
```

| digits | encode                                       | st_astext                                  |
|--------|----------------------------------------------|--------------------------------------------|
| 15     | 010100000005f9a72083cdd5e405f9a72083cdd5e40  | POINT(123.456789123456 123.456789123456) ↔ |
| 14     | 010100000005f9a72083cdd5e405f9a72083cdd5e40  | POINT(123.456789123456 123.456789123456) ↔ |
| 13     | 010100000005f9a72083cdd5e405f9a72083cdd5e40  | POINT(123.456789123456 123.456789123456) ↔ |
| 12     | 010100000005c9a72083cdd5e405c9a72083cdd5e40  | POINT(123.456789123456 123.456789123456) ↔ |
| 11     | 01010000000409a72083cdd5e40409a72083cdd5e40  | POINT(123.456789123456 123.456789123456) ↔ |
| 10     | 0101000000009a72083cdd5e40009a72083cdd5e40   | POINT(123.456789123455 123.456789123455) ↔ |
| 9      | 0101000000009072083cdd5e40009072083cdd5e40   | POINT(123.456789123418 123.456789123418) ↔ |
| 8      | 0101000000008072083cdd5e40008072083cdd5e40   | POINT(123.45678912336 123.45678912336) ↔   |
| 7      | 0101000000000070083cdd5e40000070083cdd5e40   | POINT(123.456789121032 123.456789121032) ↔ |
| 6      | 0101000000000040083cdd5e40000040083cdd5e40   | POINT(123.456789076328 123.456789076328) ↔ |
| 5      | 010100000000000083cdd5e400000000083cdd5e40   | POINT(123.456789016724 123.456789016724) ↔ |
| 4      | 010100000000000003cdd5e400000000003cdd5e40   | POINT(123.456787109375 123.456787109375) ↔ |
| 3      | 01010000000000000003cdd5e400000000003cdd5e40 | POINT(123.456787109375 123.456787109375) ↔ |
| 2      | 010100000000000000038dd5e4000000000038dd5e40 | POINT(123.45654296875 123.45654296875) ↔   |
| 1      | 01010000000000000000dd5e400000000000dd5e40   | POINT(123.453125 123.453125) ↔             |
| 0      | 01010000000000000000dc5e400000000000dc5e40   | POINT(123.4375 123.4375) ↔                 |
| -1     | 01010000000000000000c05e400000000000c05e40   | POINT(123 123) ↔                           |
| -2     | 0101000000000000000005e400000000000005e40    | POINT(120 120) ↔                           |
| -3     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -4     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -5     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -6     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -7     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -8     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -9     | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -10    | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -11    | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -12    | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -13    | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -14    | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |
| -15    | 01010000000000000000058400000000000005840    | POINT(96 96) ↔                             |



ST\_SnapToGrid

## 7.5.24 ST\_RemovePoint

ST\_RemovePoint — Remove a point from a linestring.

### Synopsis

geometry **ST\_RemovePoint**(geometry linestring, integer offset);



Removes a point from a LineString, given its index (0-based). Useful for turning a closed line (ring) into an open linestring.

Enhanced: 3.2.0

1.1.0     .



This function supports 3d and will not drop the z-index.



Guarantees no lines are closed by removing the end point of closed lines (rings). Assumes geom is of type LINESTRING

```
UPDATE sometable
 SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
 FROM sometable
 WHERE ST_IsClosed(geom);
```



[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#)

## 7.5.25 ST\_RemoveRepeatedPoints

ST\_RemoveRepeatedPoints — Returns a version of a geometry with duplicate points removed.

### Synopsis

geometry **ST\_RemoveRepeatedPoints**(geometry geom, float8 tolerance);



Returns a version of the given geometry with duplicate consecutive points removed. The function processes only (Multi)LineStrings, (Multi)Polygons and MultiPoints but it can be called with any kind of geometry. Elements of GeometryCollections are processed individually. The endpoints of LineStrings are preserved.

If the *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates.

Enhanced: 3.2.0

## 2.2.0 `ST_RemoveRepeatedPoints()`

- ✓ This function supports Polyhedral surfaces.
- ✓ This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('MULTIPOINT ((1 1), (2 2), (3 3), (2 2))'));

MULTIPOINT(1 1,2 2,3 3)
```

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('LINESTRING (0 0, 0 0, 1 1, 0 0, 1 1, 2 2)'));

LINESTRING(0 0,1 1,0 0,1 1,2 2)
```

**Example:** Collection elements are processed individually.

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2, 2 2, 3 3), POINT (4 4), POINT (4 4), POINT (5 5))'));

GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3),POINT(4 4),POINT(4 4),POINT(5 5))
```

**Example:** Repeated point removal with a distance tolerance.

```
SELECT ST_AsText(ST_RemoveRepeatedPoints('LINESTRING (0 0, 0 0, 1 1, 5 5, 1 1, 2 2)', 2));

LINESTRING(0 0,5 5,2 2)
```

☒☒

## `ST_Simplify`

### 7.5.26 `ST_RemoveIrrelevantPointsForView`

`ST_RemoveIrrelevantPointsForView` — Removes points that are irrelevant for rendering a specific rectangular view of a geometry.

#### Synopsis

geometry **`ST_RemoveIrrelevantPointsForView`**(geometry geom, box2d bounds);

☒☒

Returns a **geometry** without points being irrelevant for rendering the geometry within a given rectangular view.

This function can be used to quickly preprocess geometries that should be rendered only within certain bounds.

Only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING are evaluated. Other geometries keep unchanged.

In contrast to `ST_ClipByBox2D()` this function



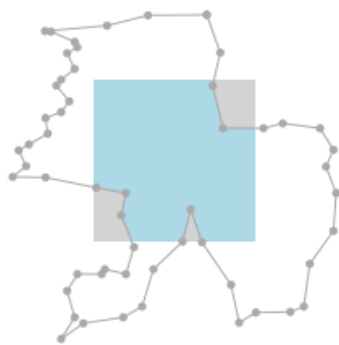
- sorts out points without computing new intersection points which avoids rounding errors and usually increases performance,
- returns a geometry with equal or similar point number,
- leads to the same rendering result within the specified view, and
- may introduce self-intersections which would make the resulting geometry invalid (see example below).



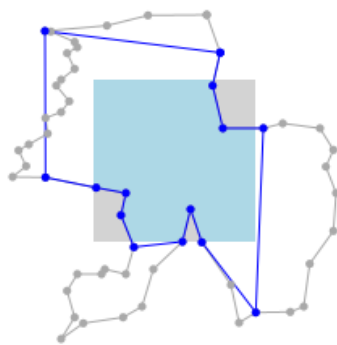
**Warning**

For polygons, this function does currently not ensure that the result is valid. This situation can be checked with [ST\\_IsValid](#) and repaired with [ST\\_MakeValid](#).

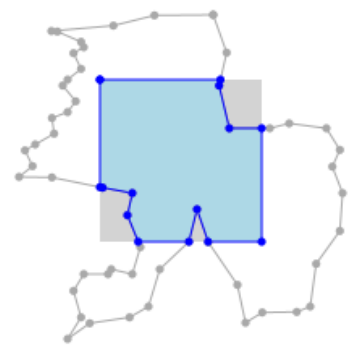
original  
55 points



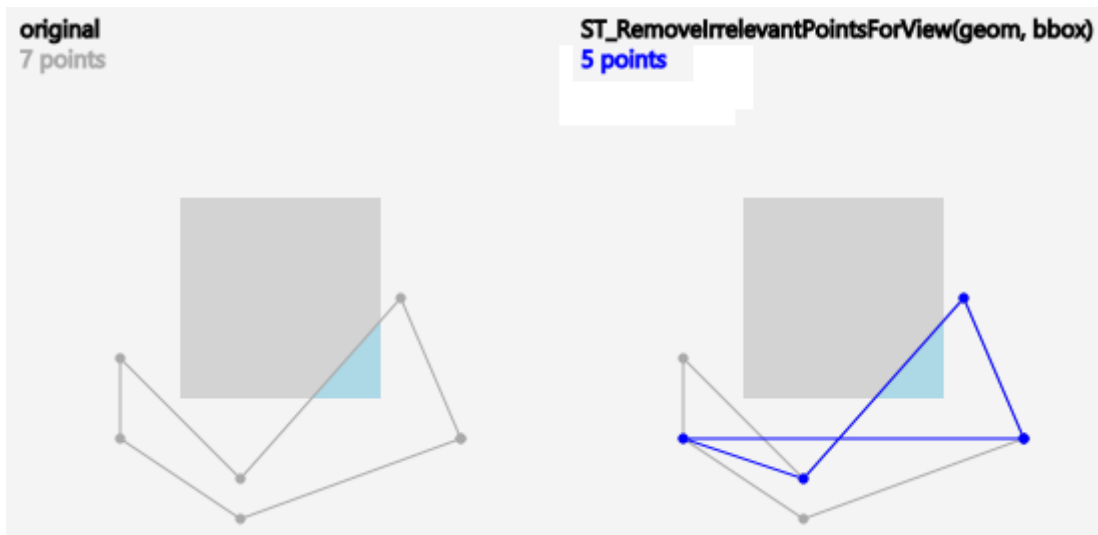
ST\_RemoveIrrelevantPointsForView(geom, bbox)  
15 points



ST\_ClipByBox2D(geom, bbox)  
15 points



Example: *ST\_RemoveIrrelevantPointsForView()* applied to a polygon. Blue points remain, the rendering result (light-blue area) within the grey view box remains as well.



Example: *Due to the fact that points are just sorted out and no new points are computed, the result of ST\_RemoveIrrelevantPointsForView() may contain self-intersections.*

Availability: 3.5.0

☒☒

```
SELECT ST_AsText(
 ST_RemoveIrrelevantPointsForView(
 ST_GeomFromText('MULTIPOLYGON(((10 10, 20 10, 30 10, 40 10, 20 20, ←
 10 20, 10 10)),((10 10, 20 10, 20 20, 10 20, 10 10)))'),
 ST_MakeEnvelope(12,12,18,18));

 st_astext

 MULTIPOLYGON(((10 10,40 10,20 20,10 20,10 10)),((10 10,20 10,20 20,10 ←
 20,10 10)))
```

```
SELECT ST_AsText(
 ST_RemoveIrrelevantPointsForView(
 ST_GeomFromText('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))'),
 ST_MakeEnvelope(12,12,18,18));

 st_astext

 MULTILINESTRING((10 15,15 15,20 15),(13 13,15 15,17 17))
```

```
SELECT ST_AsText(
 ST_RemoveIrrelevantPointsForView(
 ST_GeomFromText('LINESTRING(0 0, 10 0,20 0,30 0)'),
 ST_MakeEnvelope(12,12,18,18));

 st_astext

 LINESTRING EMPTY
```

☒☒

[ST\\_ClipByBox2D](#), [ST\\_Intersection](#)

## 7.5.27 ST\_RemoveSmallParts

`ST_RemoveSmallParts` — Removes small parts (polygon rings or linestrings) of a geometry.

### Synopsis

`geometry` **ST\_RemoveSmallParts**(`geometry geom`, double precision `minSizeX`, double precision `minSizeY`);

☒☒

Returns a [geometry](#) without small parts (exterior or interior polygon rings, or linestrings).

This function can be used as preprocessing step for creating simplified maps, e. g. to remove small islands or holes.

It evaluates only geometries of type (MULTI)POLYGON and (MULTI)LINESTRING. Other geometries remain unchanged.

If *minSizeX* is greater than 0, parts are sorted out if their width is smaller than *minSizeX*.

If *minSizeY* is greater than 0, parts are sorted out if their height is smaller than *minSizeY*.

Both *minSizeX* and *minSizeY* are measured in coordinate system units of the geometry.

For polygon types, evaluation is done separately for each ring which can lead to one of the following results:

- the original geometry,
- a POLYGON with all rings with less vertices,
- a POLYGON with a reduced number of interior rings (having possibly less vertices),
- a POLYGON EMPTY, or
- a MULTIPOLYGON with a reduced number of polygons (having possibly less interior rings or vertices), or
- a MULTIPOLYGON EMPTY.

For linestring types, evaluation is done for each linestring which can lead to one of the following results:

- the original geometry,
- a LINESTRING with a reduced number of vertices,
- a LINESTRING EMPTY,
- a MULTILINESTRING with a reduced number of linestrings (having possibly less vertices), or
- a MULTILINESTRING EMPTY.

**original**  
20 points, 4 parts



**ST\_RemoveSmallParts(geom, 30,30)**  
15 points, 3 parts



**ST\_RemoveSmallParts(geom, 50,50)**  
10 points, 2 parts



*Example: ST\_RemoveSmallParts() applied to a multi-polygon. Blue parts remain.*

Availability: 3.5.0








1.1.0

2.3.0

 This function supports 3d and will not drop the z-index.



```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
 st_astext

LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT ↵
(-1 1 3)'))))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
 st_asewkt

LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GeomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
 , ST_PointN(g,1) as p;
 st_astext

LINESTRING(0 0,1 1,0 0,3 3,4 4)
```



[ST\\_AddPoint](#), [ST\\_NPoints](#), [ST\\_NumPoints](#), [ST\\_PointN](#), [ST\\_RemovePoint](#)

### 7.5.31 ST\_ShiftLongitude

`ST_ShiftLongitude` — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

#### Synopsis

geometry **ST\_ShiftLongitude**(geometry geom);



Reads every point/vertex in a geometry, and shifts its longitude coordinate from -180..0 to 180..360 and vice versa if between these ranges. This function is symmetrical so the result is a 0..360 representation of a -180..180 data and a -180..180 representation of a 0..360 data.



#### Note

This is only useful for data with coordinates in longitude/latitude; e.g. SRID 4326 (WGS 84 geographic)



**Warning**

1.3.4 `ST_ShiftLongitude` (polyhedral surface) `TIN`. 1.3.4 `ST_ShiftLongitude` `ST_ShiftLongitude`.



This function supports 3d and will not drop the z-index.

2.0.0 `ST_ShiftLongitude` (polyhedral surface) `TIN`.

2.2.0 `ST_ShiftLongitude`, `ST_ShiftLongitude` "ST\_Shift\_Longitude" `ST_ShiftLongitude`.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

`ST_ShiftLongitude`

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)::geometry))

st_astext

POINT(-90 0)

--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)::geometry))

st_astext

POINT(270 0)

--for linestrings the functions affects only to the sufficient coordinates
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)::geometry))

st_astext

LINESTRING(174 12, -178 13)
```

`ST_WrapX`

**ST\_WrapX**

**7.5.32 ST\_WrapX**

`ST_WrapX` — X `ST_WrapX`.

**Synopsis**

geometry **ST\_WrapX**(geometry geom, float8 wrap, float8 move);







**Note**

ST\_IsSimple (ST\_IsSimple).



**Note**

1.1.0 2 1.1.0, , , .

1.0.0RC1

1.1.0 Z M



This function supports 3d and will not drop the z-index.

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
 SET geom = ST_SnapToGrid(geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
 ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897,
 4.11112 3.23748667)'),
 0.001)
);
 st_astext

LINESTRING(1.112 2.123,4.111 3.237)
--Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
 ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
 4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.111112)'),
 ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
 0.1, 0.1, 0.1, 0.01));
 st_asewkt

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)

--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m
 and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
 4.111111 3.2374897 3.1234 1.1111)'),
 0.01));
 st_asewkt

LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

[ST\\_Snap](#), [ST\\_AsEWKT](#), [ST\\_AsText](#), [ST\\_GeomFromText](#), [ST\\_GeomFromEWKT](#), [ST\\_Simplify](#)

### 7.5.34 ST\_Snap

ST\_Snap — 将几何体中的顶点和段对齐到另一个几何体的顶点。

#### Synopsis

geometry **ST\_Snap**(geometry input, geometry reference, float tolerance);

返回

Snaps the vertices and segments of a geometry to another Geometry's vertices. A snap distance tolerance is used to control where snapping is performed. The result geometry is the input geometry with the vertices snapped. If no snapping occurs then the input geometry is returned unchanged.

返回的几何体是输入几何体，其顶点（和段）已对齐到参考几何体的顶点（和段）。如果未发生对齐，则返回输入几何体。

对齐距离用于控制在哪里进行对齐。结果几何体是输入几何体，其顶点（和段）已对齐到参考几何体的顶点（和段）。如果未发生对齐，则返回输入几何体。



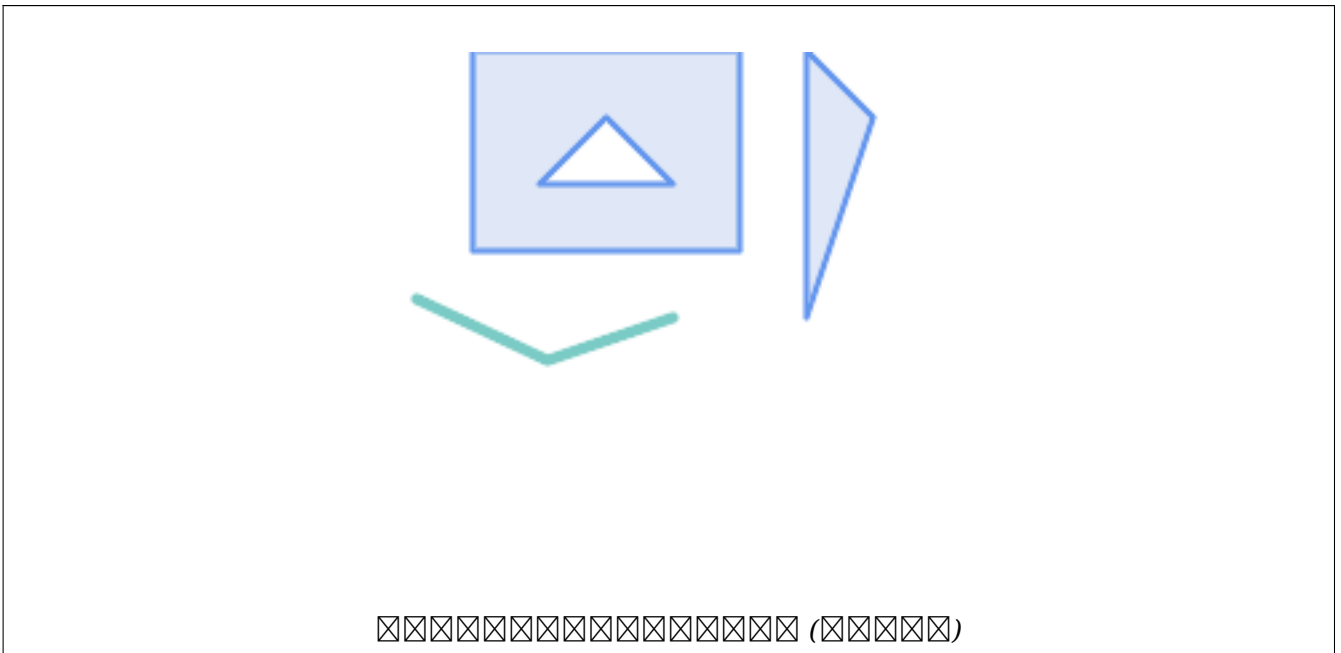
#### Note

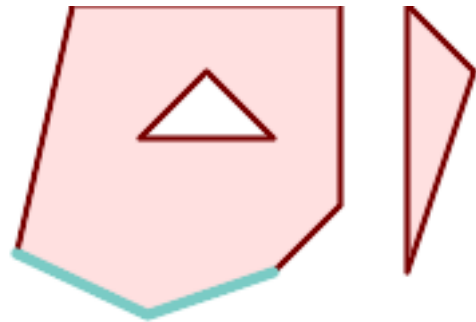
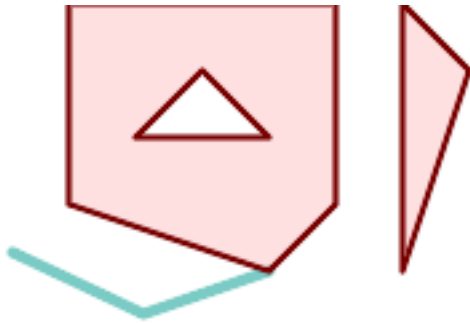
如果输入几何体不是简单几何体 (ST\_IsSimple 返回 false) 或无效几何体 (ST\_IsValid 返回 false)，则返回 null。

GEOS 2.0.0 及以上版本。

2.0.0 及以上版本。

返回





Distance 1.01. The cyan line segment is not snapped to the polygon.

Distance 1.25. The cyan line segment is snapped to the polygon.

```
SELECT ST_AsText(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;
```

```
SELECT ST_AsText(
 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;
```

polysnapped

polysnapped

```
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)))
```

```
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)))
```





☒☒

Tests if an `ST_Geometry` value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid.

For the version with the `flags` parameter, supported values are documented in [ST\\_IsValidDetail](#). This version does not print a NOTICE explaining invalidity.

For more information on the definition of geometry validity, refer to [Section 4.4](#)

Note!

**Note**

SQL-MM defines the result of `ST_IsValid(NULL)` to be 0, while PostGIS returns NULL.

GEOS ☒☒☒☒☒

The version accepting flags is available starting with 2.0.0.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method implements the SQL/MM specification. SQL-MM 3: 5.1.9

Note!

**Note**

Neither OGC-SFS nor SQL-MM specifications include a flag argument for `ST_IsValid`. The flag is a PostGIS extension.

☒☒

```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
 ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
NOTICE: Self-intersection at or near point 0 0
good_line | bad_poly
-----+-----
t | f
```

☒☒

[ST\\_IsSimple](#), [ST\\_IsValidReason](#), [ST\\_IsValidDetail](#),

## 7.6.2 ST\_IsValidDetail

`ST_IsValidDetail` — Returns a `valid_detail` row stating if a geometry is valid or if not a reason and a location.

### Synopsis

`valid_detail` **ST\_IsValidDetail**(geometry geom, integer flags);

☒☒

Returns a `valid_detail` row, containing a boolean (`valid`) stating if a geometry is valid, a varchar (`reason`) stating a reason why it is invalid and a geometry (`location`) pointing out where it is invalid. Useful to improve on the combination of `ST_IsValid` and `ST_IsValidReason` to generate a detailed report of invalid geometries.

The optional `flags` parameter is a bitfield. It can have the following values:

- 0: Use usual OGC SFS validity semantics.
- 1: Consider certain kinds of self-touching rings (inverted shells and exverted holes) as valid. This is also known as "the ESRI flag", since this is the validity model used by those tools. Note that this is invalid under the OGC model.

GEOS ☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, reason(ST_IsValidDetail(geom)), ST_AsText(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;
```

| gid  | reason            | location    |
|------|-------------------|-------------|
| 5330 | Self-intersection | POINT(32 5) |
| 5340 | Self-intersection | POINT(42 5) |
| 5350 | Self-intersection | POINT(52 5) |

```
--simple example
SELECT * FROM ST_IsValidDetail('LINESTRING(220227 150406,2220227 150407,222020 150410)');
```

| valid | reason | location |
|-------|--------|----------|
| t     |        |          |



☒☒

[ST\\_IsValid](#), [ST\\_IsValidReason](#)

### 7.6.3 ST\_IsValidReason

`ST_IsValidReason` — Returns text stating if a geometry is valid, or a reason for invalidity.

#### Synopsis

```
text ST_IsValidReason(geometry geomA);
text ST_IsValidReason(geometry geomA, integer flags);
```

☒☒

Returns text stating if a geometry is valid, or if invalid a reason why.

Useful in combination with [ST\\_IsValid](#) to generate a detailed report of invalid geometries and reasons.

Allowed flags are documented in [ST\\_IsValidDetail](#).

GEOS ☒☒☒☒☒

Availability: 1.4

Availability: 2.0 version taking flags.

☒☒

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
 'POLYGON ((100 200, 100 100, 200 200,
 200 100, 100 200))'::geometry) as validity_info;
validity_info

Self-intersection[150 150]
```

```
--First 3 Rejects from a successful quintuplet experiment
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;

gid | validity_info
-----+-----
5330 | Self-intersection [32 5]
5340 | Self-intersection [42 5]
5350 | Self-intersection [52 5]

--simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');

st_isvalidreason

Valid Geometry

```

☒☒

[ST\\_IsValid, ST\\_Summary](#)

## 7.6.4 ST\_MakeValid

ST\_MakeValid — Attempts to make an invalid geometry valid without losing vertices.

### Synopsis

```

geometry ST_MakeValid(geometry input);
geometry ST_MakeValid(geometry input, text params);

```

☒☒

The function attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Valid geometries are returned unchanged.

Supported inputs are: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS and GEOMETRYCOLLECTIONS containing any mix of them.

In case of full or partial dimensional collapses, the output geometry may be a collection of lower-to-equal dimension geometries, or a geometry of lower dimension.

Single polygons may become multi-geometries in case of self-intersections.

The params argument can be used to supply an options string to select the method to use for building valid geometry. The options string is in the format "method=linework|structure keepcollapsed=true|false". If no "params" argument is provided, the "linework" algorithm will be used as the default.

The "method" key has two values.

- "linework" is the original algorithm, and builds valid geometries by first extracting all lines, nodding that linework together, then building a value output from the linework.
- "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 "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false". When set to "false", geometry components that collapse to a lower dimensionality, for example a one-point linestring would be dropped.

GEOS 

2.0.0 

Enhanced: 2.0.1, speed improvements

Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

Enhanced: 3.1.0, added removal of Coordinates with NaN values.

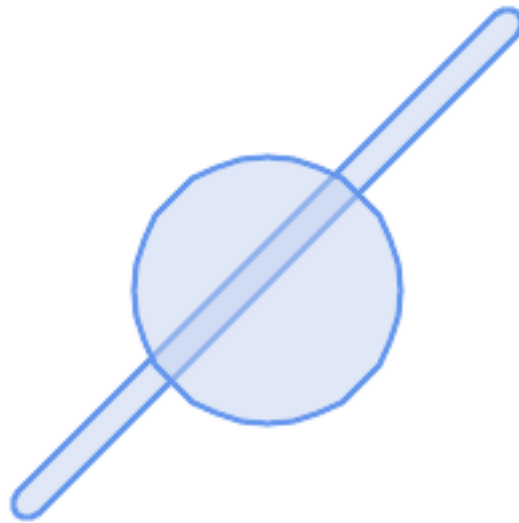
Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS  $\geq$  3.10.0.



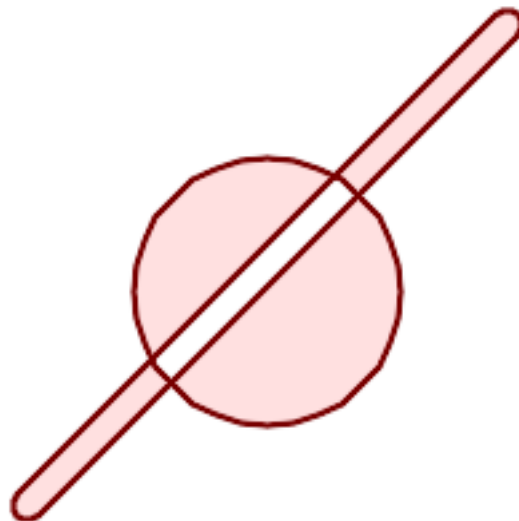
This function supports 3d and will not drop the z-index.



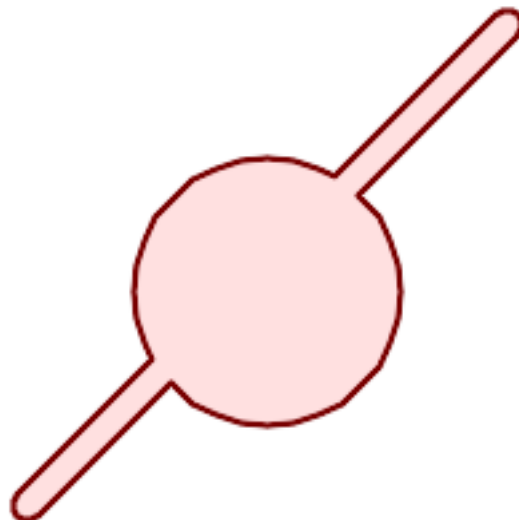
---



*before\_geom: MULTIPOLYGON of 2 overlapping polygons*



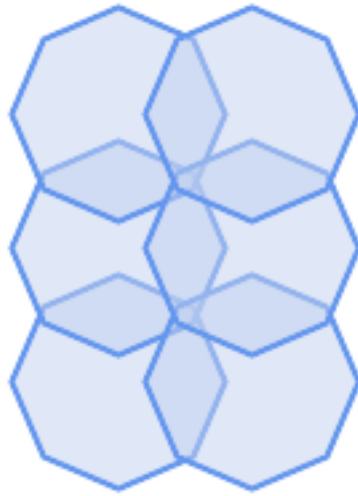
*after\_geom: MULTIPOLYGON of 4 non-overlapping polygons*



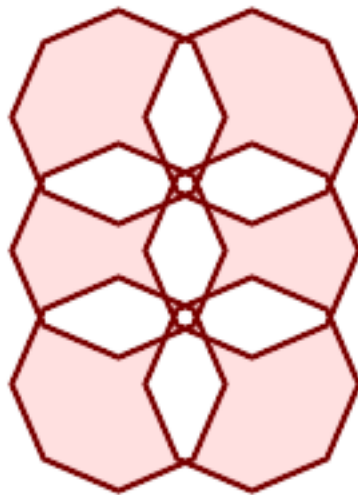
*after\_geom\_structure: MULTIPOLYGON of 1 non-overlapping polygon*

```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom, ↵
 'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
```

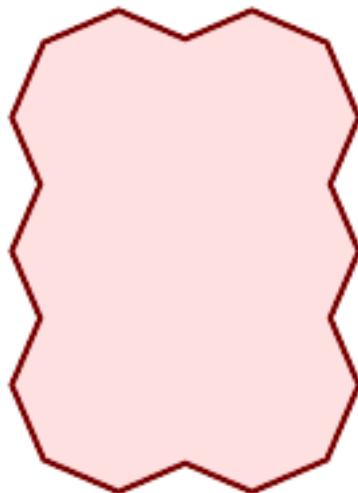




*before\_geom: MULTIPOLYGON of 6 overlapping polygons*



*after\_geom: MULTIPOLYGON of 14 Non-overlapping polygons*



*after\_geom\_structure: MULTIPOLYGON of 1 Non-overlapping polygon*

```
SELECT c.geom AS before_geom,
 ST_MakeValid(c.geom) AS after_geom,
 ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((91 50.79 22.51 10.23 22.11 50.23 78.51 90.79 78.91 ↵
```

☒☒

```
SELECT ST_AsText(ST_MakeValid(
 'LINESTRING(0 0, 0 0)',
 'method=structure keepcollapsed=true'
));
```

```
st_astext

POINT(0 0)
```

```
SELECT ST_AsText(ST_MakeValid(
 'LINESTRING(0 0, 0 0)',
 'method=structure keepcollapsed=false'
));
```

```
st_astext

LINESTRING EMPTY
```

☒☒

[ST\\_IsValid](#), [ST\\_Collect](#), [ST\\_CollectionExtract](#)

## 7.7 Spatial Reference System Functions

### 7.7.1 ST\_InverseTransformPipeline

`ST_InverseTransformPipeline` — Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.

#### Synopsis

geometry **ST\_InverseTransformPipeline**(geometry geom, text pipeline, integer to\_srid);

☒☒

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline to go in the inverse direction.

Refer to [ST\\_TransformPipeline](#) for details on writing a transformation pipeline.

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using [ST\\_TransformPipeline](#) the pipeline is executed in a forward direction. Using `ST_InverseTransformPipeline()` the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of [ST\\_Transform](#). In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

☒☒

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)':: geometry,
'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

 wgs_geom

POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS gda2020_auto;

 gda2020_auto

POINT(143.00000635638918 -36.999986706128176)
(1 row)
```

☒☒

[ST\\_Transform](#), [ST\\_TransformPipeline](#)

## 7.7.2 ST\_SetSRID

ST\_SetSRID — Set the SRID on a geometry.

### Synopsis

geometry **ST\_SetSRID**(geometry geom, integer srid);

☒☒

Sets the SRID on a geometry to a particular integer value. Useful in constructing bounding boxes for queries.



#### Note

This function does not transform the geometry coordinates in any way - it simply sets the meta data defining the spatial reference system the geometry is assumed to be in. Use [ST\\_Transform](#) if you want to transform the geometry into a new projection.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).



This method supports Circular Strings and Curves.



☒☒

-- Mark a point as WGS 84 long lat --

```
SELECT ST_SetSRID(ST_Point(-123.365556, 48.428611),4326) As wgs84long_lat;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=4326;POINT(-123.365556 48.428611)
```

-- Mark a point as WGS 84 long lat and then transform to web mercator (Spherical Mercator) --

```
SELECT ST_Transform(ST_SetSRID(ST_Point(-123.365556, 48.428611),4326),3785) As spere_merc;
-- the ewkt representation (wrap with ST_AsEWKT) -
SRID=3785;POINT(-13732990.8753491 6178458.96425423)
```

☒☒

Section [4.5](#), [ST\\_SRID](#), [ST\\_Transform](#), [UpdateGeometrySRID](#)

### 7.7.3 ST\_SRID

ST\_SRID — Returns the spatial reference identifier for a geometry.

#### Synopsis

integer **ST\_SRID**(geometry g1);

☒☒

Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. Section [4.5](#)



#### Note

spatial\_ref\_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s2.1.1.1



This method implements the SQL/MM specification. SQL-MM 3: 5.1.5



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
-- result
4326
```

☒☒

Section 4.5, [ST\\_SetSRID](#), [ST\\_Transform](#), [ST\\_SRID](#), [ST\\_SRID](#)

### 7.7.4 ST\_Transform

`ST_Transform` — Return a new geometry with coordinates transformed to a different spatial reference system.

#### 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);
```

☒☒

Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference `to_srid` may be identified by a valid SRID integer parameter (i.e. it must exist in the `spatial_ref_sys` table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for `to_proj` and/or `from_proj`, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with `from_proj`, input geometries must have a defined SRID.

`ST_Transform` is often confused with [ST\\_SetSRID](#). `ST_Transform` actually changes the coordinates of a geometry from one spatial reference system to another, while `ST_SetSRID()` simply changes the SRID identifier of the geometry.

`ST_Transform` automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use [ST\\_TransformPipeline](#).



**Note**

Requires PostGIS be compiled with PROJ support. Use [PostGIS\\_Full\\_Version](#) to confirm you have PROJ support compiled in.



**Note**

If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage.



**Note**

1.3.4 [ST\\_Transform](#) (curve) [ST\\_Transform](#). 1.3.4 [ST\\_Transform](#).

☒☒☒☒: 2.0.0 [ST\\_Transform](#) (polyhedral surface) [ST\\_Transform](#).

Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.

- ☑ This method implements the SQL/MM specification. SQL-MM 3: 5.1.6
- ☑ This method supports Circular Strings and Curves.
- ☑ This function supports Polyhedral surfaces.

☒☒

Change Massachusetts state plane US feet geometry to WGS 84 long lat

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

wgs_geom

POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416 ↵
1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

st_asewkt

SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326 ↵
42.3903829478009 2,
-71.1775844305465 42.3903826677917 3,
-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

Example of creating a partial functional index. For tables where you are not sure all the geometries will be filled in, its best to use a partial index that leaves out null geometries which will both conserve space and make your index smaller and more efficient.

```
CREATE INDEX idx_geom_26986_parcel
ON parcels
USING gist
(ST_Transform(geom, 26986))
WHERE geom IS NOT NULL;
```

Examples of using PROJ.4 text to transform with custom spatial references.

```
-- Find intersection of two polygons near the North pole, using a custom Gnomonic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
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_AsText(
 ST_Transform(
 ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
 gnom, 4326))
FROM data;

st_astext

```

```
POLYGON((-170 74.053793645338, -141 73.4268621378904, -141 68, -170 68, -170 74.053793645338) ↵
)
```

## Configuring transformation behavior

Sometimes coordinate transformation involving a grid-shift can fail, for example if PROJ.4 has not been built with grid-shift files or the coordinate does not lie within the range for which the grid shift is defined. By default, PostGIS will throw an error if a grid shift file is not present, but this behavior can be configured on a per-SRID basis either by testing different `to_proj` values of PROJ.4 text, or altering the `proj4text` value within the `spatial_ref_sys` table.

For example, the `proj4text` parameter `+datum=NAD87` is a shorthand form for the following `+nadgrids` parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

The `@` prefix means no error is reported if the files are not present, but if the end of the list is reached with no file having been appropriate (ie. found and overlapping) then an error is issued.

If, conversely, you wanted to ensure that at least the standard files were present, but that if all files were scanned without a hit a null transformation is applied you could use:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

The null grid shift file is a valid grid shift file covering the whole world and applying no shift. So for a complete example, if you wanted to alter PostGIS so that transformations to SRID 4267 that didn't lie within the correct range did not throw an ERROR, you would use the following:

```
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;
```

☒☒

Section [4.5](#), [ST\\_SetSRID](#), [ST\\_SRID](#), [UpdateGeometrySRID](#), [ST\\_TransformPipeline](#)

### 7.7.5 ST\_TransformPipeline

`ST_TransformPipeline` — Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

#### Synopsis

```
geometry ST_TransformPipeline(geometry g1, text pipeline, integer to_srid);
```

☒☒

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Transformation pipelines are defined using any of the following string formats:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Note that a simple `EPSG:CODE` string does not uniquely identify a coordinate operation: the same EPSG code can be used for a CRS definition.

- A PROJ pipeline string of the form: `+proj=pipeline . . .`. Automatic axis normalisation will not be applied, and if necessary the caller will need to add an additional pipeline step, or remove axis swap steps.
- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG`

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using `ST_TransformPipeline()` the pipeline is executed in a forward direction. Using `ST_InverseTransformPipeline` the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of `ST_Transform`. In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

☒☒

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

```
-- Forward direction
SELECT ST_AsText(ST_TransformPipeline('SRID=4326;POINT(2 49)::geometry,
 'urn:ogc:def:coordinateOperation:EPSG::16031')) AS utm_geom;

 utm_geom

POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293):: ←
 geometry,
 'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

 wgs_geom

POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)::geometry, 7844)) AS ←
 gda2020_auto;

 gda2020_auto

POINT(143.00000635638918 -36.999986706128176)
(1 row)

-- using a defined conversion (EPSG:8447)
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)::geometry,
 'urn:ogc:def:coordinateOperation:EPSG::8447')) AS gda2020_code;

 gda2020_code

POINT(143.0000063280214 -36.999986718287545)
(1 row)

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
```

```
-- NOTE: any 'axiswap' steps must be removed.
SELECT ST_AsText(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;

 gda2020_pipeline

POINT(143.0000063280214 -36.999986718287545)
(1 row)
```

☒☒

[ST\\_Transform](#), [ST\\_InverseTransformPipeline](#)

## 7.7.6 postgis\_srs\_codes

`postgis_srs_codes` — Return the list of SRS codes associated with the given authority.

### Synopsis

setof text **postgis\_srs\_codes**(text auth\_name);

☒☒

Returns a set of all auth\_srid for the given auth\_name.

Availability: 3.4.0

Proj version 6+

☒☒

List the first ten codes associated with the EPSG authority.

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;
```

```
postgis_srs_codes

2000
20004
20005
20006
20007
20008
20009
2001
20010
20011
```

☒☒

[postgis\\_srs](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

### 7.7.7 postgis\_srs

postgis\_srs — Return a metadata record for the requested authority and srid.

#### Synopsis

setof record **postgis\_srs**(text auth\_name, text auth\_srid);

☒☒

Returns a metadata record for the requested auth\_srid for the given auth\_name. The record will have the auth\_name, auth\_srid, sname, srtext, proj4text, and the corners of the area of usage, point\_sw and point\_ne.

Availability: 3.4.0

Proj version 6+

☒☒

Get the metadata for EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');
```

```

auth_name | EPSG
auth_srid | 3005
sname | NAD83 / BC Albers
srtext | PROJCS["NAD83 / BC Albers", ...]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 +
 datum=NAD83 +units=m +no_defs +type=crs
point_sw | 0101000020E6100000E17A14AE476161C00000000000204840
point_ne | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40

```

☒☒

[postgis\\_srs\\_codes](#), [postgis\\_srs\\_all](#), [postgis\\_srs\\_search](#)

### 7.7.8 postgis\_srs\_all

postgis\_srs\_all — Return metadata records for every spatial reference system in the underlying Proj database.

#### Synopsis

setof record **postgis\_srs\_all**(void);

☒☒

Returns a set of all metadata records in the underlying Proj database. The records will have the auth\_name, auth\_srid, sname, srtext, proj4text, and the corners of the area of usage, point\_sw and point\_ne.

Availability: 3.4.0

Proj version 6+

☒☒

Get the first 10 metadata records from the Proj database.

```
SELECT auth_name, auth_srid, sname FROM postgis_srs_all() LIMIT 10;
```

| auth_name | auth_srid | sname                                    |
|-----------|-----------|------------------------------------------|
| EPSG      | 2000      | Anguilla 1957 / British West Indies Grid |
| EPSG      | 20004     | Pulkovo 1995 / Gauss-Kruger zone 4       |
| EPSG      | 20005     | Pulkovo 1995 / Gauss-Kruger zone 5       |
| EPSG      | 20006     | Pulkovo 1995 / Gauss-Kruger zone 6       |
| EPSG      | 20007     | Pulkovo 1995 / Gauss-Kruger zone 7       |
| EPSG      | 20008     | Pulkovo 1995 / Gauss-Kruger zone 8       |
| EPSG      | 20009     | Pulkovo 1995 / Gauss-Kruger zone 9       |
| EPSG      | 2001      | Antigua 1943 / British West Indies Grid  |
| EPSG      | 20010     | Pulkovo 1995 / Gauss-Kruger zone 10      |
| EPSG      | 20011     | Pulkovo 1995 / Gauss-Kruger zone 11      |

☒☒

[postgis\\_srs\\_codes](#), [postgis\\_srs](#), [postgis\\_srs\\_search](#)

### 7.7.9 postgis\_srs\_search

`postgis_srs_search` — Return metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter.

#### Synopsis

setof record **postgis\_srs\_search**(geometry bounds, text auth\_name=EPSG);

☒☒

Return a set of metadata records for projected coordinate systems that have areas of usage that fully contain the bounds parameter. Each record will have the `auth_name`, `auth_srid`, `sname`, `srtxt`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

The search only looks for projected coordinate systems, and is intended for users to explore the possible systems that work for the extent of their data.

Availability: 3.4.0

Proj version 6+

☒☒

Search for projected coordinate systems in Louisiana.

```
SELECT auth_name, auth_srid, sname,
 ST_AsText(point_sw) AS point_sw,
 ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;
```





 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

GEOS

1.1.0

[ST\\_BuildArea](#), [ST\\_BdMPolyFromText](#)

### 7.8.1.2 ST\_BdMPolyFromText

ST\_BdMPolyFromText — WKT

#### Synopsis

geometry **ST\_BdMPolyFromText**(text WKT, integer srid);

WKT



#### Note

WKT. [ST\\_BdPolyFromText](#), [ST\\_BuildArea\(\)](#)

 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

GEOS

1.1.0

[ST\\_BuildArea](#), [ST\\_BdPolyFromText](#)

### 7.8.1.3 ST\_GeogFromText

ST\_GeogFromText — WKT (EWKT)

#### Synopsis

geography **ST\_GeogFromText**(text EWKT);



☒☒

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID (conformance suite).

WKT (GEOMETRYCOLLECTION) null.



#### Note

WKT, SRID. ST\_GeomFromText.

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

✔ This method implements the SQL/MM specification.

☒☒

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

☒☒

[ST\\_GeomFromText](#), [ST\\_SRID](#)

### 7.8.1.6 ST\_GeomFromEWKT

ST\_GeomFromEWKT — EWKT(Extended Well-Known Text) ST\_Geometry.

#### Synopsis

geometry **ST\_GeomFromEWKT**(text EWKT);

☒☒

OGC EWKT(Extended Well-Known Text) PostGIS ST\_Geometry.



#### Note

EWKT OGC, SRID PostGIS.

2.0.0 (polyhedral surface) TIN.

✔ This function supports 3d and will not drop the z-index.

✔ This method supports Circular Strings and Curves.

✔ This function supports Polyhedral surfaces.

✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```

SELECT ST_GeomFromEWKT('SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837 ↵
 42.259113,-71.161144 42.25932)');
SELECT ST_GeomFromEWKT('SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837 ↵
 42.259113,-71.161144 42.25932)');

SELECT ST_GeomFromEWKT('SRID=4269;POINT(-71.064544 42.28787)');

SELECT 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)))');

SELECT 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)))');

```

--3d circular string

```
SELECT ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)');
```

--Polyhedral Surface example

```

SELECT 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))
)');

```

☒☒

**ST\_AsEWKT, ST\_GeomFromText**

### 7.8.1.7 ST\_GeomFromMARC21

ST\_GeomFromMARC21 — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

#### Synopsis

geometry **ST\_GeomFromMARC21** ( text marcxml );

☒☒

This function creates a PostGIS geometry from a MARC21/XML record, which can contain a POINT or a POLYGON. In case of multiple geographic data entries in the same MARC21/XML record, a MULTIPOINT or MULTIPOLYGON will be returned. If the record contains mixed geometry types, a GEOMETRYCOLLECTION will be returned. It returns NULL if the MARC21/XML record does not contain any geographic data (datafield:034).

LOC MARC21/XML versions supported:

- [MARC21/XML 1.1](#)

Availability: 3.3.0, requires libxml2 2.6+



#### Note

The MARC21/XML Coded Cartographic Mathematical Data currently does not provide any means to describe the Spatial Reference System of the encoded coordinates, so this function will always return a geometry with SRID 0.



#### Note

Returned POLYGON geometries will always be clockwise oriented.

☒☒

Converting MARC21/XML geographic data containing a single POINT encoded as hddd.ddddd

```
SELECT
 ST_AsText(
 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>
>'));

 st_astext

POINT(-4.5 54.25)
(1 row)

```

Converting MARC21/XML geographic data containing a single POLYGON encoded as hdddmmss

```

 SELECT
 ST_AsText(
 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>
 >'));

 st_astext

POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667 ←
 52.38333333333333,13.1 52.38333333333333,13.1 52.65))
(1 row)

```

Converting MARC21/XML geographic data containing a POLYGON and a POINT:

```

 SELECT
 ST_AsText(
 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>
>');
 st_astext ←

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))
(1 row)

```

☒☒

**ST\_AsMARC21**

**7.8.1.8 ST\_GeometryFromText**

ST\_GeometryFromText — WKT(Well-Known Text) ☒☒☒☒☒☒ ST\_Geometry ☒☒☒☒☒☒☒. ☒☒☒☒☒ ST\_GeomFromText ☒☒☒☒☒☒☒☒☒.

**Synopsis**

```

geometry ST_GeometryFromText(text WKT);
geometry ST_GeometryFromText(text WKT, integer srid);

```

☒☒

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

☒☒

**ST\_GeomFromText**









**Note**

ST\_GeomFromText, ST\_GeomFromText. ST\_GeomFromText.



This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

SQL

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS null_return;
aline	null_return
01020000000200000000000000000000F ... | t
```

SQL

**ST\_GeomFromText**

**7.8.1.11 ST\_MLineFromText**

ST\_MLineFromText — WKT ST\_MultiLineString.

**Synopsis**

geometry ST\_MLineFromText(text WKT, integer srid);
geometry ST\_MLineFromText(text WKT);

SQL

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID (conformance suite).

WKT null.



**Note**

WKT, ST\_GeomFromText.



This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2



This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

☒☒

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

☒☒

**ST\_GeomFromText**

### 7.8.1.12 ST\_MPointFromText

**ST\_MPointFromText** — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

#### Synopsis

```
geometry ST_MPointFromText(text WKT, integer srid);
geometry ST_MPointFromText(text WKT);
```

☒☒

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC 3.2.6.2 - SRID (conformance suite).

WKT null.



#### Note

WKT, SRID. ST\_GeomFromText.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1.3.2.6.2](#)



This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

☒☒

```
SELECT ST_MPointFromText('MULTIPOINT((1 2),(3 4))');
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180),(-70.9611 42.1223))', 4326);
```

☒☒

**ST\_GeomFromText**

### 7.8.1.13 ST\_MPolyFromText

**ST\_MPolyFromText** — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

### Synopsis

geometry **ST\_MPolyFromText**(text WKT, integer srid);  
 geometry **ST\_MPolyFromText**(text WKT);

☒☒

Makes a MultiPolygon from WKT with the given SRID. If SRID is not given, it defaults to 0.  
 OGC 3.2.6.2 - SRID (conformance suite).  
 WKT.



#### Note

WKT, ST\_GeomFromText.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

☒☒

```
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)))');
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);
```

☒☒

### ST\_GeomFromText, ST\_SRID

#### 7.8.1.14 ST\_PointFromText

ST\_PointFromText — SRID WKT. SRID, 0.

### Synopsis

geometry **ST\_PointFromText**(text WKT);  
 geometry **ST\_PointFromText**(text WKT, integer srid);

☒☒

Constructs a PostGIS ST\_Geometry point object from the OGC Well-Known text representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.



**Note**

ST\_PointFromText **2** **SRID** **ST\_Geometry** **spatial\_ref\_sys** **SRID**



**Note**

WKT **ST\_GeomFromText** **ST\_MakePoint** **OGC** **ST\_Point**



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**. 3.2.6.2 - **SRID** (conformance suite)



This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

☒☒

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

☒☒

**ST\_GeomFromText, ST\_MakePoint, ST\_Point, ST\_SRID**

**7.8.1.15 ST\_PolygonFromText**

ST\_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

**Synopsis**

```
geometry ST_PolygonFromText(text WKT);
geometry ST_PolygonFromText(text WKT, integer srid);
```

☒☒

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Returns null if WKT is not a polygon.

OGC 3.2.6.2 - **SRID** (conformance suite)



**Note**

WKT POLYGON((-71.1776585052917 42.3902909739571, -71.1776820268866 42.3903701743239, -71.1776063012595 42.3903825660754, -71.1775826583081 42.3903033653531, -71.1776585052917 42.3902909739571)); ST\_GeomFromText

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

✔ This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

☒

```
SELECT ST_PolygonFromText('POLYGON((-71.1776585052917 42.3902909739571, -71.1776820268866 42.3903701743239, -71.1776063012595 42.3903825660754, -71.1775826583081 42.3903033653531, -71.1776585052917 42.3902909739571))');
st_polygonfromtext

010300000001000000050000006...

SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;
point_is_not_poly

t
```

☒

**ST\_GeomFromText**

**7.8.1.16 ST\_WKTTToSQL**

ST\_WKTTToSQL — WKT(Well-Known Text) ST\_Geometry. ST\_GeomFromText

**Synopsis**

geometry **ST\_WKTTToSQL**(text WKT);

☒

✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

☒

**ST\_GeomFromText**

## 7.8.2 Well-Known Binary (WKB)

### 7.8.2.1 ST\_GeogFromWKB

ST\_GeogFromWKB — WKB EWKB(WKB) ST\_Geometry.

#### Synopsis

geometry **ST\_GeogFromWKB**(bytea wkb);

☒

ST\_GeogFromWKB WKB PostGIS WKB ST\_Geometry. SQL (Geometry Factory).

SRID, 4326(WGS84) ☒.



This method supports Circular Strings and Curves.

☒

```
--Although bytea rep contains single \, these need to be escaped when inserting into a table
SELECT ST_AsText(
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@')
);

 st_astext

LINESTRING(-113.98 39.198, -113.981 39.195)
(1 row)
```

☒

[ST\\_GeogFromText](#), [ST\\_AsBinary](#)

### 7.8.2.2 ST\_GeomFromEWKB

ST\_GeomFromEWKB — EWKB(Extended Well-Known Binary) ST\_Geometry.

#### Synopsis

geometry **ST\_GeomFromEWKB**(bytea EWKB);





### 7.8.2.3 ST\_GeomFromWKB

ST\_GeomFromWKB — WKB(Well-Known Binary) SRID.

#### Synopsis

geometry **ST\_GeomFromWKB**(bytea geom);  
 geometry **ST\_GeomFromWKB**(bytea geom, integer srid);

ST\_GeomFromWKB WKB SRID( $\text{SRID ID}$ )  $\text{SRID}$ .  $\text{SQL}$  (Geometry Factory).  $\text{ST\_WKBTtoSQL}$ .

SRID,  $\text{SRID}$  0(unknown).

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2](#) -  $\text{SRID}$  (conformance suite).

✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.41

✔ This method supports Circular Strings and Curves.

```
--Although bytea rep contains single \, these need to be escaped when inserting into a table
-- unless standard_conforming_strings is set to on.
SELECT ST_AsEWKT(
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)
);
 st_asewkt

SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)

SELECT
 ST_AsText(
 ST_GeomFromWKB(
 ST_AsEWKB('POINT(2 5)::geometry)
)
);
 st_astext

POINT(2 5)
(1 row)
```

[ST\\_WKBToSQL](#), [ST\\_AsBinary](#), [ST\\_GeomFromEWKB](#)

### 7.8.2.4 ST\_LineFromWKB

ST\_LineFromWKB — SRID WKB LINestring.

#### Synopsis

geometry **ST\_LineFromWKB**(bytea WKB);  
 geometry **ST\_LineFromWKB**(bytea WKB, integer srid);

ST\_LineFromWKB WKB SRID(SRID ID) -  
 LINestring - . SQL (Geometry Factory) .  
 SRID, 0 . bytea, NULL .



**Note**

OGC 3.2.6.2 - SRID (conformance suite) .



**Note**

LINestring, ST\_GeomFromWKB . ST\_GeomFromWKB, .

✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2](#)

✔ This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

```
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;
aline	null_return
01020000000200000000000000000000F ... | t
```

[ST\\_GeomFromWKB](#), [ST\\_LinestringFromWKB](#)

### 7.8.2.5 ST\_LinestringFromWKB

ST\_LinestringFromWKB — SRID WKB.

## Synopsis

geometry **ST\_LineStringFromWKB**(bytea WKB);  
 geometry **ST\_LineStringFromWKB**(bytea WKB, integer srid);

**ST\_LineStringFromWKB** WKB SRID([SRID ID](#)) [Geometry Factory](#) [LINESTRING](#) - [SQL](#) (Geometry Factory).

SRID, 0. bytea [LINESTRING](#), NULL.



**Note**  
 OGC 3.2.6.2 - SRID (conformance suite).



**Note**  
[LINESTRING](#), [ST\\_GeomFromWKB](#) [ST\\_GeomFromWKB](#), [LINESTRING](#).

This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#). s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

```
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;

01020000000200000000000000000000F ... | t
```

[ST\\_GeomFromWKB](#), [ST\\_LineFromWKB](#)

### 7.8.2.6 ST\_PointFromWKB

**ST\_PointFromWKB** — SRID WKB







### 7.8.3.3 ST\_GeomFromGML

ST\_GeomFromGML — GML PostGIS.

#### Synopsis

```
geometry ST_GeomFromGML(text geomgml);
geometry ST_GeomFromGML(text geomgml, integer srid);
```

OGC GML PostGIS ST\_Geometry.

ST\_GeomFromGML GML (geometry fragment). GML

OGC GML:

- GML 3.2.1
- GML 3.1.1 SF-2 (GML 3.1.0 3.0.0)
- GML 2.1.2

OGC GML: <http://www.opengeospatial.org/standards/gml>

1.5. LibXML2 1.6.

: 2.0.0 (polyhedral surface) TIN.

: 2.0.0 SRID.

✔ This function supports 3d and will not drop the z-index.

✔ This function supports Polyhedral surfaces.

✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

GML (MultiGeometry) 2D 3D. PostGIS Z ST\_GeomFromGML 2D.

GML SRS. PostGIS, ST\_GeomFromGML SRS. GML srsName,.

ST\_GeomFromGML GML. GML XLink.



#### Note

ST\_GeomFromGML SQL/MM.



☒☒: **srsName** ☒☒☒☒☒☒☒

```
SELECT ST_GeomFromGML($$
 <gml:LineString xmlns:gml="http://www.opengis.net/gml"
 srsName="EPSG:4269">
 <gml:coordinates>
 -71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932
 </gml:coordinates>
 </gml:LineString>
$$);
```

☒☒: **XLink** ☒☒

```
SELECT ST_GeomFromGML($$
 <gml:LineString xmlns:gml="http://www.opengis.net/gml"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 srsName="urn:ogc:def:crs:EPSG::4269">
 <gml:pointProperty>
 <gml:Point gml:id="p1"
 ><gml:pos
 >42.258729 -71.16028</gml:pos
 ></gml:Point>
 </gml:pointProperty>
 <gml:pos
 >42.259112 -71.160837</gml:pos>
 <gml:pointProperty>
 <gml:Point xlink:type="simple" xlink:href="#p1"/>
 </gml:pointProperty>
 </gml:LineString>
$$);
```

## ☒☒: ☒☒☒☒☒☒

```
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 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</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</gml:posList
 ></gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
 </gml:PolygonPatch>
 </gml:polygonPatches>
 </gml:PolyhedralSurface>
 ');
```

```

>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:polygons>
</gml:PolyhedralSurface
>');

-- result --
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)))

```

☒☒

Section [2.2.3](#), [ST\\_AsGML](#), [ST\\_GMLToSQL](#)

### 7.8.3.4 ST\_GeomFromGeoJSON

ST\_GeomFromGeoJSON — GeoJSON ☒☒☒☒☒☒☒ PostGIS ☒☒☒☒☒☒☒☒☒.

#### Synopsis

```

geometry ST_GeomFromGeoJSON(text geomjson);
geometry ST_GeomFromGeoJSON(json geomjson);
geometry ST_GeomFromGeoJSON(jsonb geomjson);

```

¶¶

GeoJSON ¶¶¶¶¶¶ PostGIS ¶¶¶¶¶¶¶¶¶¶.

ST\_GeomFromGML ¶ JSON ¶¶¶¶ (geometry fragment) ¶¶¶¶¶¶¶¶¶¶. ¶¶¶ JSON ¶¶¶¶ ¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise.

Enhanced: 2.5.0 can now accept json and jsonb as inputs.

2.0.0 ¶¶¶¶¶¶¶¶¶¶¶¶. JSON-C 0.9 ¶¶¶¶¶¶¶¶¶¶¶¶.



**Note**

JSON-C ¶¶¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. JSON-C ¶¶¶¶¶¶ ¶¶, "--with-jsondir=/path/to/json-c" ¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶ Section 2.2.3 ¶¶¶¶ ¶¶¶¶.



This function supports 3d and will not drop the z-index.

¶¶

```
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[-48.23456,20.12345]}')) ←
 As wkt;
wkt

POINT(-48.23456 20.12345)
```

```
-- a 3D linestring
SELECT ST_AsText(ST_GeomFromGeoJSON('{"type":"LineString","coordinates ←
 ":[1,2,3],[4,5,6],[7,8,9]}')) As wkt;
wkt

LINESTRING(1 2,4 5,7 8)
```

¶¶

[ST\\_AsText](#), [ST\\_AsGeoJSON](#), Section 2.2.3

**7.8.3.5 ST\_GeomFromKML**

ST\_GeomFromKML — ¶¶¶ KML ¶¶¶¶¶¶¶¶ PostGIS ¶¶¶¶¶¶¶¶¶¶.

**Synopsis**

geometry **ST\_GeomFromKML**(text geomkml);

¶¶

OGC KML ¶¶¶¶¶¶ PostGIS ST\_Geometry ¶¶¶¶¶¶¶¶.

ST\_GeomFromKML ¶ KML ¶¶¶¶ (geometry fragment) ¶¶¶¶¶¶¶¶¶¶. ¶¶¶ KML ¶¶¶¶¶¶  
¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶¶¶ OGC KML ¶¶¶¶¶¶¶¶¶¶:

- KML 2.2.0 ¶¶¶¶¶¶

OGC KML ¶¶: <http://www.opengespatial.org/standards/kml>

Availability: 1.5, requires libxml2 2.6+



This function supports 3d and will not drop the z-index.



**Note**

ST\_GeomFromKML ¶¶¶ SQL/MM ¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

¶¶: **srsName** ¶¶¶¶¶¶¶¶

```
SELECT ST_GeomFromKML($$
 <LineString>
 <coordinates>
>-71.1663,42.2614
 -71.1667,42.2616</coordinates>
 </LineString>
$$);
```

¶¶

Section [2.2.3, ST\\_AsKML](#)

**7.8.3.6 ST\_GeomFromTWKB**

ST\_GeomFromTWKB — TWKB("Tiny Well-Known Binary") ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶.

**Synopsis**

geometry **ST\_GeomFromTWKB**(bytea twkb);

¶¶

ST\_GeomFromTWKB ¶¶¶ TWKB("Tiny Well-Known Binary") ¶¶¶¶¶¶ (WKB) ¶¶¶¶¶¶¶¶¶¶¶¶¶¶  
¶¶¶¶¶¶¶¶¶¶.





```

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcgyy4d0dbxqz0'));
 st_astext

POINT(-115.172816 36.114646)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcgyy4d0dbxqz0', 4));
 st_astext

POINT(-115.13671875 36.123046875)

SELECT ST_AsText(ST_PointFromGeoHash('9qqj7nmxcgyy4d0dbxqz0', 10));
 st_astext

POINT(-115.172815918922 36.1146435141563)

```

☒☒

[ST\\_GeoHash](#), [ST\\_Box2dFromGeoHash](#), [ST\\_GeomFromGeoHash](#)

### 7.8.3.10 ST\_FromFlatGeobufToTable

`ST_FromFlatGeobufToTable` — Creates a table based on the structure of FlatGeobuf data.

#### Synopsis

`void ST_FromFlatGeobufToTable`(text schemaname, text tablename, bytea FlatGeobuf input data);

☒☒

Creates a table based on the structure of FlatGeobuf data. (<http://flatgeobuf.org>).

`schema` Schema name.

`table` Table name.

`data` Input FlatGeobuf data.

Availability: 3.2.0

### 7.8.3.11 ST\_FromFlatGeobuf

`ST_FromFlatGeobuf` — Reads FlatGeobuf data.

#### Synopsis

setof anyelement `ST_FromFlatGeobuf`(anyelement Table reference, bytea FlatGeobuf input data);

☒☒

Reads FlatGeobuf data (<http://flatgeobuf.org>). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

## 7.9 Geometry Output

### 7.9.1 Well-Known Text (WKT)

#### 7.9.1.1 ST\_AsEWKT

ST\_AsEWKT — WKT(Well-Known Text) 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);
```

Returns the Well-Known Text representation of the geometry prefixed with the SRID. The optional *maxdecimaldigits* argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

To perform the inverse conversion of EWKT representation to PostGIS geometry use [ST\\_GeomFromEWKT](#).



#### Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use [ST\\_ReducePrecision](#) with a suitable gridsize first.



#### Note

The WKT spec does not include the SRID. To get the OGC WKT format use [ST\\_AsText](#).



#### Warning

WKT format does not maintain precision so to prevent floating truncation, use [ST\\_AsBinary](#) or [ST\\_AsEWKB](#) format for transport.

Enhanced: 3.1.0 support for optional precision parameter.

2.0.0, TIN.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).













```
SELECT ST_AsHEXEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
 which gives same answer as

 SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326)::text;

 st_ashexewkb

 0103000020E6100000010000000500
 000000000000000000000000000000
 0000000000000000000000000000F03F
 000000000000F03F000000000000F03F000000000000F03
 F000
```

### 7.9.3 Other Formats

#### 7.9.3.1 ST\_AsEncodedPolyline

ST\_AsEncodedPolyline —

#### Synopsis

text **ST\_AsEncodedPolyline**(geometry geom, integer precision=5);

Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

Optional precision specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

2.2.0

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95
40.7,-126.453 43.252)'));
-- result--
|_p~iF~ps|U_uLLnnqC_mqNvxq`@
```

(segmentize) ,

```
-- the SQL for Boston to San Francisco, segments every 100 KM
SELECT ST_AsEncodedPolyline(
 ST_Segmentize(
 ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
 100000)::geometry) As encodedFlightPath;
```

\$

```

<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries= ↵
 geometry"
></script>
<script type="text/javascript">
 flightPath = new google.maps.Polyline({
 path: google.maps.geometry.encoding.decodePath("$encodedFlightPath ↵
 "),
 map: map,
 strokeColor: '#0000CC',
 strokeOpacity: 1.0,
 strokeWeight: 4
 });
</script>

```

☒☒

[ST\\_LineFromEncodedPolyline](#), [ST\\_Segmentize](#)

### 7.9.3.2 ST\_AsFlatGeobuf

`ST_AsFlatGeobuf` — Return a FlatGeobuf representation of a set of rows.

#### Synopsis

```

bytea ST_AsFlatGeobuf(anyelement set row);
bytea ST_AsFlatGeobuf(anyelement row, bool index);
bytea ST_AsFlatGeobuf(anyelement row, bool index, text geom_name);

```

☒☒

Return a FlatGeobuf representation (<http://flatgeobuf.org>) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

`row` row data with at least a geometry column.

`index` toggle spatial index creation. Default is false.

`geom_name` is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

### 7.9.3.3 ST\_AsGeobuf

`ST_AsGeobuf` — Return a Geobuf representation of a set of rows.

#### Synopsis

```

bytea ST_AsGeobuf(anyelement set row);
bytea ST_AsGeobuf(anyelement row, text geom_name);

```

☒☒

Return a Geobuf representation (<https://github.com/mapbox/geobuf>) of a set of rows corresponding to a FeatureCollection. Every input geometry is analyzed to determine maximum precision for optimal storage. Note that Geobuf in its current form cannot be streamed so the full output will be assembled in memory.

row row data with at least a geometry column.

geom\_name is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
 FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
 st_asgeobuf

GAAiEAo0CgwIBBoIAAAAAgIAAAE=
```

### 7.9.3.4 ST\_AsGeoJSON

ST\_AsGeoJSON — Return a geometry or feature in GeoJSON format.

#### 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);
```

☒☒

Returns a geometry as a GeoJSON "geometry" object, or a row as a GeoJSON "feature" object.

The resulting GeoJSON geometry and feature representations conform with the [GeoJSON specifications RFC 7946](#), except when the parsed geometries are referenced with a CRS other than WGS84 longitude and latitude ([EPSG:4326](#), [urn:ogc:def:crs:OGC::CRS84](#)); the GeoJSON geometry object will then have a short CRS SRID identifier attached by default. 2D and 3D Geometries are both supported. GeoJSON only supports SFS 1.1 geometry types (no curve support for example).

The geom\_column parameter is used to distinguish between multiple geometry columns. If omitted, the first geometry column in the record will be determined. Conversely, passing the parameter will save column type lookups.

The maxdecimaldigits argument may be used to reduce the maximum number of decimal places used in output (defaults to 9). If you are using EPSG:4326 and are outputting the geometry only for display, maxdecimaldigits=6 can be a good choice for many maps.



#### Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use [ST\\_ReducePrecision](#) with a suitable gridsize first.





```
{ "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" } }] }
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*, id_column =
> 'id')
FROM (VALUES (1, 'one', 'POINT(1 1)::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
```

```
{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [1,1] }, "id": 1, "properties": { "name": "one" } }
```

Don't forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
st_asgeojson
```

```
{ "type": "MultiLineString", "coordinates": [[[-89.734634999999997, 31.492072000000000], [-89.734959999999997, 31.492237999999997]]] }
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{ "type": "LineString", "coordinates": [[[1, 2, 3], [4, 5, 6]]] }
```

☒☒

[ST\\_GeomFromGeoJSON](#), [ST\\_ForcePolygonCCW](#), [ST\\_Transform](#)

### 7.9.3.5 ST\_AsGML

ST\_AsGML — ☒☒☒ GML 2 ☒☒ GML 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);
```



## ☒☒: ☒☒ 2

```
SELECT ST_AsGML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
 st_asgml

 <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>
```

## ☒☒: ☒☒ 3

```
-- Flip coordinates and output extended EPSG (16 | 1)--
SELECT ST_AsGML(3, ST_GeomFromText('POINT(5.234234233242 6.34534534534)',4326), 5, 17);
 st_asgml

 <gml:Point srsName="urn:ogc:def:crs:EPSG::4326"
><gml:pos
>6.34535 5.23423</gml:pos
></gml:Point>
```

```
-- Output the envelope (32) --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 32);
 st_asgml

 <gml:Envelope srsName="EPSG:4326">
 <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 --
SELECT ST_AsGML(3, ST_GeomFromText('LINESTRING(1 2, 3 4, 10 20)',4326), 5, 49);
 st_asgml

<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
 <gml:lowerCorner
>2 1</gml:lowerCorner>
 <gml:upperCorner
>20 10</gml:upperCorner>
</gml:Envelope>
```

```
-- Polyhedral Example --
SELECT ST_AsGML(3, 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)))'));
 st_asgml
```

```

<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 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 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 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 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 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 0</gml:posList>
 </gml:LinearRing>
 </gml:exterior>
 </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface>

```

☒☒

[ST\\_GeomFromGML](#)

### 7.9.3.6 ST\_AsKML

ST\_AsKML — GML 2 GML 3

#### Synopsis

```
text ST_AsKML(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text ST_AsKML(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);
```

KML(Keyhole Markup Language) . . . (15), 2 . . .



#### Warning

Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use **ST\_ReducePrecision** with a suitable gridsize first.



#### Note

PostGIS Proj . . . Proj . . . **PostGIS\_Full\_Version** . . .



#### Note

1.2.2 . . . 1.3.2 . . .



#### Note

2.0.0 . . . . .



#### Note

Changed: 3.0.0 - Removed the "versioned" variant signature



#### Note

ST\_AsKML SRID . . .



This function supports 3d and will not drop the z-index.









This function returns a MARC21/XML record with **Coded Cartographic Mathematical Data** representing the bounding box of a given geometry. The format parameter allows to encode the coordinates in subfields \$d,\$e,\$f and \$g in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): hdddmmss
- decimal degrees with cardinal direction: hddd.ddddd
- decimal degrees without cardinal direction: ddd.ddddd
- decimal minutes with cardinal direction: hdddmm.mmmm
- decimal minutes without cardinal direction: dddmm.mmmm
- decimal seconds with cardinal direction: hdddmmss.sss

The decimal sign may be also a comma, e.g. hdddmm,mmm.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. hdddmm.mm for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- **MARC21/XML 1.1**

Availability: 3.3.0



#### Note

This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).



#### Note

The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.



Converting a POINT to MARC21/XML formatted as hdddmmss (default)

```
SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312) '::geometry);

 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>

```

Converting a POLYGON to MARC21/XML formatted in decimal degrees

```

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>

```

Converting a GEOMETRYCOLLECTION to MARC21/XML formatted in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```

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');

 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>

```

☒☒

## ST\_GeomFromMARC21

### 7.9.3.9 ST\_AsMVTGeom

ST\_AsMVTGeom — Transforms a geometry into the coordinate space of a MVT tile.

#### Synopsis

```
geometry ST_AsMVTGeom(geometry geom, box2d bounds, integer extent=4096, integer buffer=256,
boolean clip_geom=true);
```

☒☒

Transforms a geometry into the coordinate space of a MVT ([Mapbox Vector Tile](#)) tile, clipping it to the tile bounds if required. The geometry must be in the coordinate system of the target map (using [ST\\_Transform](#) if needed). Commonly this is [Web Mercator](#) (SRID:3857).

The function attempts to preserve geometry validity, and corrects it if needed. This may cause the result geometry to collapse to a lower dimension.

The rectangular bounds of the tile in the target map coordinate space must be provided, so the geometry can be transformed, and clipped if required. The bounds can be generated using [ST\\_TileEnvelope](#).

This function is used to convert geometry into the tile coordinate space required by [ST\\_AsMVT](#).

geom is the geometry to transform, in the coordinate system of the target map.

bounds is the rectangular bounds of the tile in map coordinate space, with no buffer.

extent is the tile extent size in tile coordinate space as defined by the [MVT specification](#). Defaults to 4096.

buffer is the buffer size in tile coordinate space for geometry clipping. Defaults to 256.

clip\_geom is a boolean to control if geometries are clipped or encoded as-is. Defaults to true.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

**Note**

From 3.0, Wagyu can be chosen at configure time to clip and validate MVT polygons. This library is faster and produces more correct results than the GEOS default, but it might drop small polygons.

☒☒

```
SELECT ST_AsText(ST_AsMVTGeom(
 ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
 ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
 4096, 0, false));
 st_astext

MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry.

```
SELECT ST_AsMVTGeom(
 ST_Transform(geom, 3857),
 ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
FROM data
WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
```

☒☒

[ST\\_AsMVT](#), [ST\\_TileEnvelope](#), [PostGIS\\_Wagyu\\_Version](#)

### 7.9.3.10 ST\_AsMVT

ST\_AsMVT — Aggregate function returning a MVT representation of a set of rows.

#### 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);
```

☒☒

An aggregate function which returns a binary [Mapbox Vector Tile](#) representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the [MVT specification](#). [ST\\_AsMVTGeom](#) can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

The **Mapbox Vector Tile** format can store features with varying sets of attributes. To use this capability supply a JSONB column in the row data containing Json objects one level deep. The keys and values in the JSONB values will be encoded as feature attributes.

Tiles with multiple layers can be created by concatenating multiple calls to this function using `||` or `STRING_AGG`.



### Important

Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use `ST_AsMVTGeom` to prepare a geometry collection for inclusion.

row row data with at least a geometry column.

name is the name of the layer. Default is the string "default".

extent is the tile extent in screen space as defined by the specification. Default is 4096.

geom\_name is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically **folds unquoted identifiers to lower case**, which means that unless the geometry column is quoted, e.g. "MyMVTGeom", this parameter must be provided as lowercase.

feature\_id\_name is the name of the Feature ID column in the row data. If NULL or negative the Feature ID is not set. The first column matching name and valid type (smallint, integer, bigint) will be used as Feature ID, and any subsequent column will be added as a property. JSON properties are not supported.

Enhanced: 3.0 - added support for Feature ID.

Enhanced: 2.5.0 - added support parallel query.

2.2.0

```
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;
```

**ST\_AsMVTGeom, ST\_TileEnvelope**

### 7.9.3.11 ST\_AsSVG

**ST\_AsSVG** — Returns SVG path data for a geometry.

### Synopsis

text **ST\_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);  
 text **ST\_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

SVG

SVG (Scalar Vector Graphics) is a vector graphics format. **rel** (relative move) is 0, **rel** (absolute move) is 1. **maxdecimaldigits** is 0-15. **rel** is 0 or 1. **rel** is 0: **cx/cy** is the center of the circle, **rel** is 1: **x/y** is the center of the circle. **rel** is 0: **cx/cy** is the center of the circle, **rel** is 1: **x/y** is the center of the circle. **rel** is 0: **cx/cy** is the center of the circle, **rel** is 1: **x/y** is the center of the circle.

For working with PostGIS SVG graphics, checkout [pg\\_svg](#) library which provides plpgsql functions for working with outputs from ST\_AsSVG.

Enhanced: 3.4.0 to support all curve types

rel: 2.0.0 (default arg) rel (named arg) rel.



**Note**

1.2.2 [http://www.w3.org/TR/SVG/paths.html#PathDataBNF](#) L [http://www.w3.org/TR/SVG/paths.html#PathDataBNF](#)

This method supports Circular Strings and Curves.

SVG

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,0 0))'::geometry);

st_assvg

M 0 0 L 0 -1 1 -1 1 0 Z
```

Circular string

```
SELECT ST_AsSVG(ST_GeomFromText('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)'));

st_assvg

M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

Multi-curve

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
 CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);

st_assvg

M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Multi-surface

```
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);
```

st\_assvg

```

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

```

### 7.9.3.12 ST\_AsTWKB

ST\_AsTWKB — TWKB(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);

TWKB(Tiny Well-Known Binary). TWKB .

. ., .

. . TWKB .

. TWKB . array\_agg .



#### Note

<https://github.com/TWKB/Specification>, <https://github.com/TWKB/twkb.js>

Enhanced: 2.4.0 memory and speed improvements.

2.2.0 .

```

SELECT ST_AsTWKB('LINESTRING(1 1,5 5)')::geometry);
 st_astwkb

```

```

\x02000202020808

```

TWKB , "array\_agg()" TWKB .





PostGIS	2D X3D	3D X3D
POINT		
(MULTI) POLYGON, POLYHEDRALSURFACE	X3D (markup)	IndexedFaceSet (faceset)
TIN	TriangleSet2D	IndexedTriangleSet

**Note!** 2 ...

Lots of advancements happening in 3D space particularly with **X3D Integration with HTML5** ... <http://freewrl.sourceforge.net/> ... **FreeWrl** ... **FreeWRL\_Launcher** ... Also check out **PostGIS minimalist X3D viewer** that utilizes this function and **x3dDom html/js open source toolkit**.

2.0.0 ISO-IEC-19776-1.2-X3DEncodings-XML ... 2.2.0 (x/y, z) ...

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

FreeWrl X3D ...

```
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>
 </Shape>
 </Transform>
 </Scene>
</X3D>
' As x3ddoc;

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

Copy and paste the output of this query to [x3d scene viewer](#) and click Show

```
SELECT string_agg('<Shape
>' || ST_AsX3D(ST_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);
```



*Buildings formed by subdividing PostGIS and extrusion*

☒☒: ☒☒☒☒☒☒ 6 ☒☒ 3 ☒☒☒☒☒☒☒☒☒

```
SELECT ST_AsX3D(
ST_Translate(
 ST_Force_3d(
 ST_Buffer(ST_Point(10,10),5, 'quad_segs=2')), 0,0,
 3)
,6) As x3dfrag;
```

```
x3dfrag

<IndexedFaceSet 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

```
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;

x3dfrag

<IndexedTriangleSet index='0 1 2 3 4 5'
><Coordinate point='0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0' /></IndexedTriangleSet>
```

## MULTILINESTRING (MULTILINESTRING)

```
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;

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 — GeoHash

#### Synopsis

text **ST\_GeoHash**(geometry geom, integer maxchars=full\_precision\_of\_point);

☒☒

Computes a **GeoHash** representation of a geometry. A GeoHash encodes a geographic Point into a text form that is sortable and searchable based on prefixing. A shorter GeoHash is a less precise representation of a point. It can be thought of as a box that contains the point.

Non-point geometry values with non-zero extent can also be mapped to GeoHash codes. The precision of the code depends on the geographic extent of the geometry.

If `maxchars` is not specified, the returned GeoHash code is for the smallest cell containing the input geometry. Points return a GeoHash with 20 characters of precision (about enough to hold the full double precision of the input). Other geometric types may return a GeoHash with less precision, depending on the extent of the geometry. Larger geometries are represented with less precision, smaller ones with more precision. The box determined by the GeoHash code always contains the input feature.

If `maxchars` is specified the returned GeoHash code has at most that many characters. It maps to a (possibly) lower precision representation of the input geometry. For non-points, the starting point of the calculation is the center of the bounding box of the geometry.

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



#### Note

ST\_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_GeoHash(ST_Point(-126,48));
```

```
 st_geohash
```

```

c0w3hf1s70w3hf1s70w3
```

```
SELECT ST_GeoHash(ST_Point(-126,48), 5);
```

```
 st_geohash
```

```

c0w3h
```

```
-- This line contains the point, so the GeoHash is a prefix of the point code
```

```
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)::geometry);
```

```
 st_geohash
```

```

c0w3
```

☒☒

**ST\_GeomFromGeoHash, ST\_PointFromGeoHash, ST\_Box2dFromGeoHash**

## 7.10 **&&** (operator)

### 7.10.1 Bounding Box Operators

#### 7.10.1.1 &&

**&&** — A 2D geometry B 2D geometry TRUE.

#### Synopsis

boolean **&&**( geometry A , geometry B );  
 boolean **&&**( geography A , geography B );

**&&** A 2D geometry B 2D geometry TRUE.



**Note**

(operand) geometry.

2.0.0 (polyhedral surface) geometry.

1.5.0 geometry.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

```
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;
```

column1	column1	overlaps
1	3	t
2	3	f

(2 rows)

**ST\_Intersects**, **ST\_Extent**, **|&>**, **&>**, **&<|**, **&<**, **~**, **@**

#### 7.10.1.2 **&&(geometry,box2df)**

**&&(geometry,box2df)** — Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

## Synopsis

boolean **&&**( geometry A , box2df B );

☒☒

The **&&** operator returns TRUE if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;
```

```
overlaps

t
(1 row)
```

☒☒

**&&**(box2df,geometry), **&&**(box2df,box2df), **~**(geometry,box2df), **~**(box2df,geometry), **~**(box2df,box2df), **@**(geometry,box2df), **@**(box2df,geometry), **@**(box2df,box2df)

### 7.10.1.3 **&&**(box2df,geometry)

**&&**(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.

## Synopsis

boolean **&&**( box2df A , geometry B );

☒☒

The && operator returns TRUE if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;
```

```
overlaps

t
(1 row)
```

☒☒

[&&\(geometry,box2df\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,geometry\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

**7.10.1.4 &&(box2df,box2df)**

[&&\(box2df,box2df\)](#) — Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.

**Synopsis**

boolean **&&**( box2df A , box2df B );

☒☒

The && operator returns TRUE if two 2D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.





**3**

```
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;
```

column1	column1	overlaps_3d	overlaps_2d
1	3	t	t
2	3	f	t

**3DM**

```
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;
```

column1	column1	overlaps_3zm	overlaps_2d
1	3	t	t
2	3	f	t

**&&**

**&&**

**7.10.1.6 &&&(geometry,gidx)**

**&&&(geometry,gidx)** — Returns TRUE if a geometry’s (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).

**Synopsis**

boolean **&&&**( geometry A , gidx B );

**&&**

The **&&** operator returns TRUE if the cached n-D bounding box of geometry A intersects the n-D bounding box B, using float precision. This means that if B is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports 3d and will not drop the z-index.



```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS overlaps; ↵
```

```
overlaps

t
(1 row)
```



**`&&&(gidx,geometry), &&&(gidx,gidx)`**

**7.10.1.7 &&&(gidx,geometry)**

`&&&(gidx,geometry)` — Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

**Synopsis**

boolean **`&&&`**( gidx A , geometry B );



The `&&&` operator returns TRUE if the n-D bounding box A intersects the cached n-D bounding box of geometry B, using float precision. This means that if A is a (double precision) `box3d`, it will be internally converted to a float precision 3D bounding box (GIDX)

**Note**

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS overlaps; ↵
```

```
overlaps

t
(1 row)
```

☒☒

**&&&(geometry,gidx), &&&(gidx,gidx)**

### 7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

#### Synopsis

boolean **&&&**( gidx A , gidx B );

☒☒

The &&& operator returns TRUE if two n-D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)



#### Note

This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This function supports 3d and will not drop the z-index.





**Synopsis**

boolean **&>**( geometry A , geometry B );

**&>** Returns true if geometry A intersects geometry B, false otherwise. TRUE if the intersection is non-empty.



**Note**

(operand) must be a geometry.

```
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;
```

column1	column1	overright
1	2	t
1	3	t
1	4	f

(3 rows)

**&&**, **|&>**, **&<**, **&<**

**7.10.1.12 <<**

**<<** — A contains B, TRUE if TRUE.

**Synopsis**

boolean **<<**( geometry A , geometry B );

**<<** Returns true if geometry A contains geometry B, false otherwise. TRUE if the intersection is non-empty.



**Note**

(operand) must be a geometry.



```

1 | 3 | f
1 | 4 | f
(3 rows)

```

☒☒

<<, >>, |>>

### 7.10.1.14 =

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

#### Synopsis

```

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

```

☒☒

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).



#### Note

Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use [ST\\_OrderingEquals](#) or [ST\\_Equals](#)



#### Caution

This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `~=` instead.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒







```

-----+-----+-----
 1 | 2 | t
 1 | 3 | f
 1 | 4 | t
(3 rows)

```

☒☒

~, &&

### 7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Returns TRUE if a geometry’s 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

#### Synopsis

boolean @( geometry A , box2df B );

☒☒

The @ operator returns TRUE if the A geometry’s 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```

SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point(
(5,5)) AS is_contained;

```

```

is_contained

t
(1 row)

```

☒☒

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)

### 7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.

#### Synopsis

```
boolean @(box2df A , geometry B);
```

☒☒

The @ operator returns TRUE if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)



#### Note

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)') ←
, 10) AS is_contained;
```

```
is_contained

t
(1 row)
```

☒☒

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry),  
~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

### 7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

#### Synopsis

```
boolean @(box2df A , box2df B);
```







**7.10.1.23 ~(**geometry,box2df**)**

~(**geometry,box2df**) — Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

**Synopsis**

boolean ~( geometry A , box2df B );

☒☒

The ~ operator returns TRUE if the 2D bounding box of a geometry A contains the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point(↵
(2,2)) AS contains;
```

```
contains

t
(1 row)
```

☒☒

**&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df), @ (geometry,box2df), @ (box2df,geometry), @ (box2df,box2df)**

**7.10.1.24 ~(**box2df,geometry**)**

~(**box2df,geometry**) — Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.

**Synopsis**

boolean ~( box2df A , geometry B );



☒☒

The `~` operator returns TRUE if the 2D bounding box A contains the B geometry's bounding box, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.

☒☒

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)') ←
, 1) AS contains;
```

```
contains

t
(1 row)
```

☒☒

[&&\(geometry,box2df\)](#), [&&\(box2df,geometry\)](#), [&&\(box2df,box2df\)](#), [~\(geometry,box2df\)](#), [~\(box2df,box2df\)](#), [@\(geometry,box2df\)](#), [@\(box2df,geometry\)](#), [@\(box2df,box2df\)](#)

**7.10.1.25 ~(box2df,box2df)**

`~(box2df,box2df)` — Returns TRUE if a 2D float precision bounding box (`BOX2DF`) contains another 2D float precision bounding box (`BOX2DF`).

**Synopsis**

```
boolean ~(box2df A , box2df B);
```

☒☒

The `~` operator returns TRUE if the 2D bounding box A contains the 2D bounding box B, using float precision. This means that if A is a (double precision) `box2d`, it will be internally converted to a float precision 2D bounding box (`BOX2DF`)

**Note**

This operand is intended to be used internally by BRIN indexes, more than by users.













☒☒

<->

## 7.11 Spatial Relationships

### 7.11.1 Topological Relationships

#### 7.11.1.1 ST\_3DIntersects

**ST\_3DIntersects** — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

#### Synopsis

boolean **ST\_3DIntersects**( geometry geomA , geometry geomB );

☒☒

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.

Note!

#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Note!

#### Note

Because of floating robustness failures, geometries don't always intersect as you'd expect them to after geometric processing. For example the closest point on a linestring to a geometry may not lie on the linestring. For these kind of issues where a distance of a centimeter you want to just consider as intersecting, use [ST\\_3DDWithin](#).

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1



☒☒☒☒

```
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;
st_3dintersects | st_intersects
-----+-----
f | t
(1 row)
```

## TIN Examples

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))::geometry, 'POINT(.1 .1 0):: ↵
 geometry);
st_3dintersects

t
```

☒☒

[ST\\_3DDWithin](#), [ST\\_Intersects](#)

### 7.11.1.2 ST\_Contains

**ST\_Contains** — Tests if every point of B lies in A, and their interiors have a point in common

#### Synopsis

boolean **ST\_Contains**(geometry geomA, geometry geomB);

☒☒

Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

In mathematical terms:  $ST\_Contains(A, B) \Leftrightarrow (A \sqsupseteq B = B) \wedge (Int(A) \sqcap Int(B) \neq \emptyset)$

The contains relationship is reflexive: every geometry contains itself. (In contrast, in the [ST\\_ContainsProperly](#) predicate a geometry does *not* properly contain itself.) The relationship is antisymmetric: if  $ST\_Contains(A, B) = true$  and  $ST\_Contains(B, A) = true$ , then the two geometries must be topologically equal ( $ST\_Equals(A, B) = true$ ).

**ST\_Contains** is the converse of [ST\\_Within](#). So,  $ST\_Contains(A, B) = ST\_Within(B, A)$ .



#### Note

Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see [Subtleties of OGC Covers, Contains, Within](#). The [ST\\_Covers](#) predicate provides a more inclusive relationship.



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

GEOS ☒☒☒☒☒

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.



**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



**Important**

Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



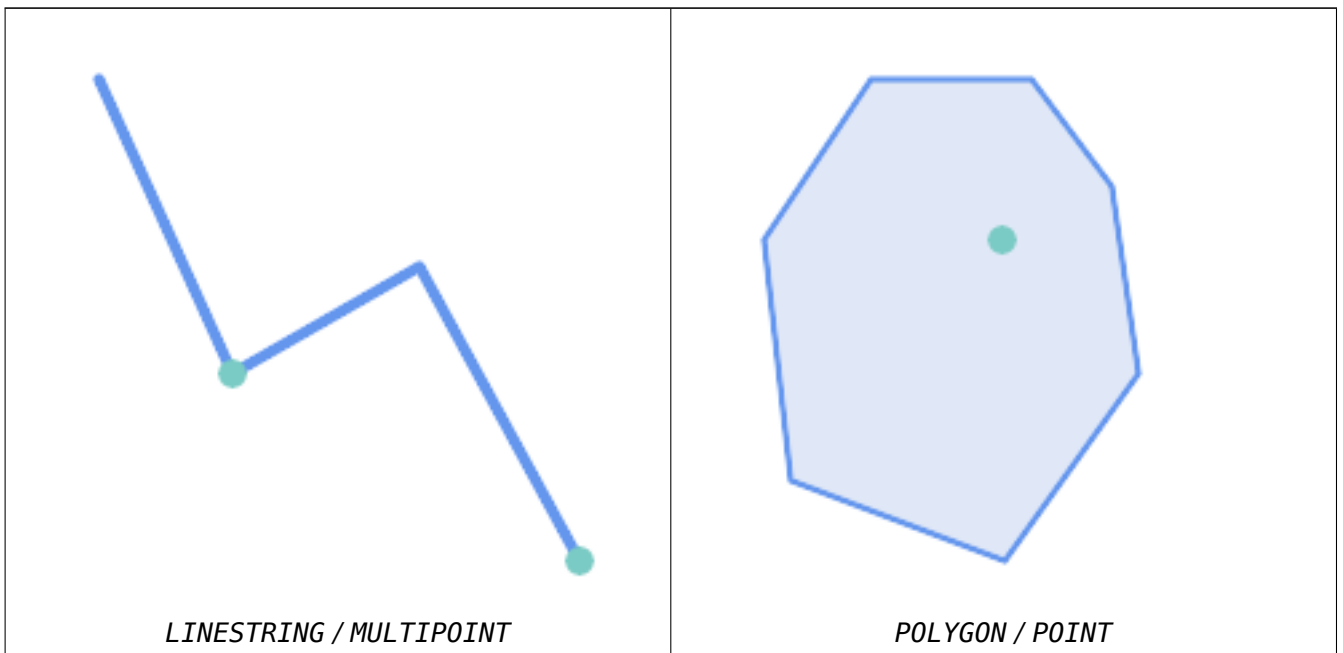
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#) - same as `within(geometry B, geometry A)`

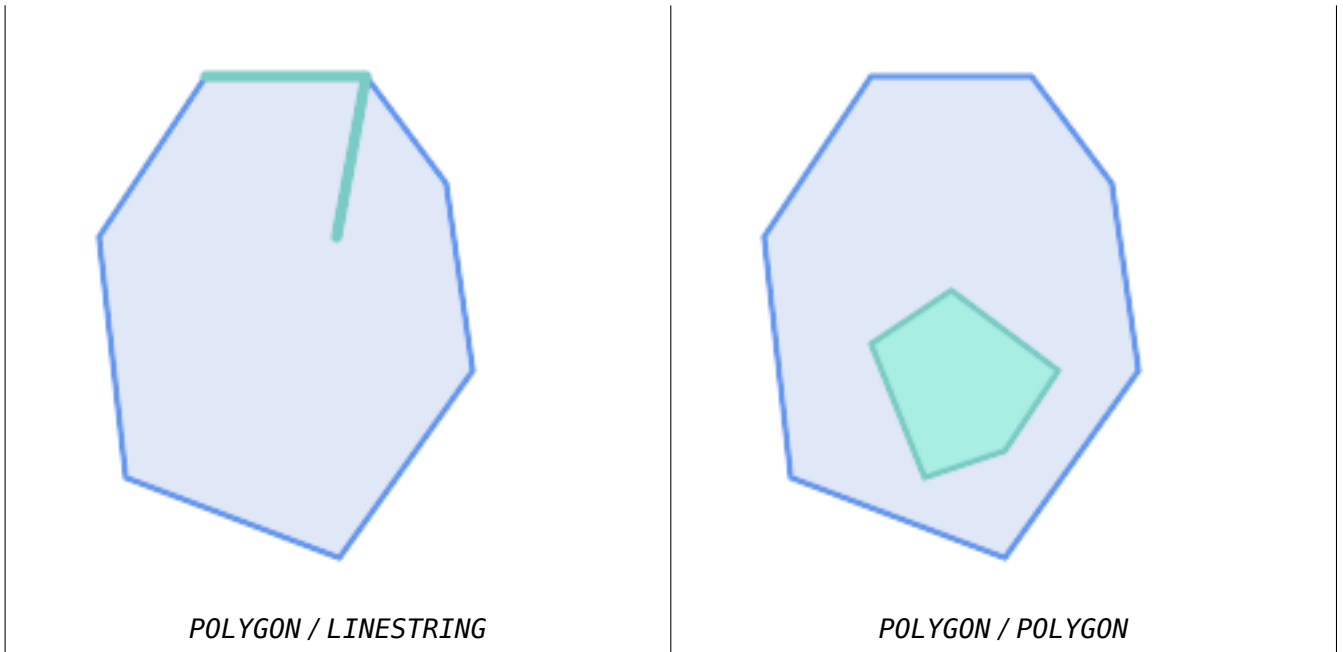


This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

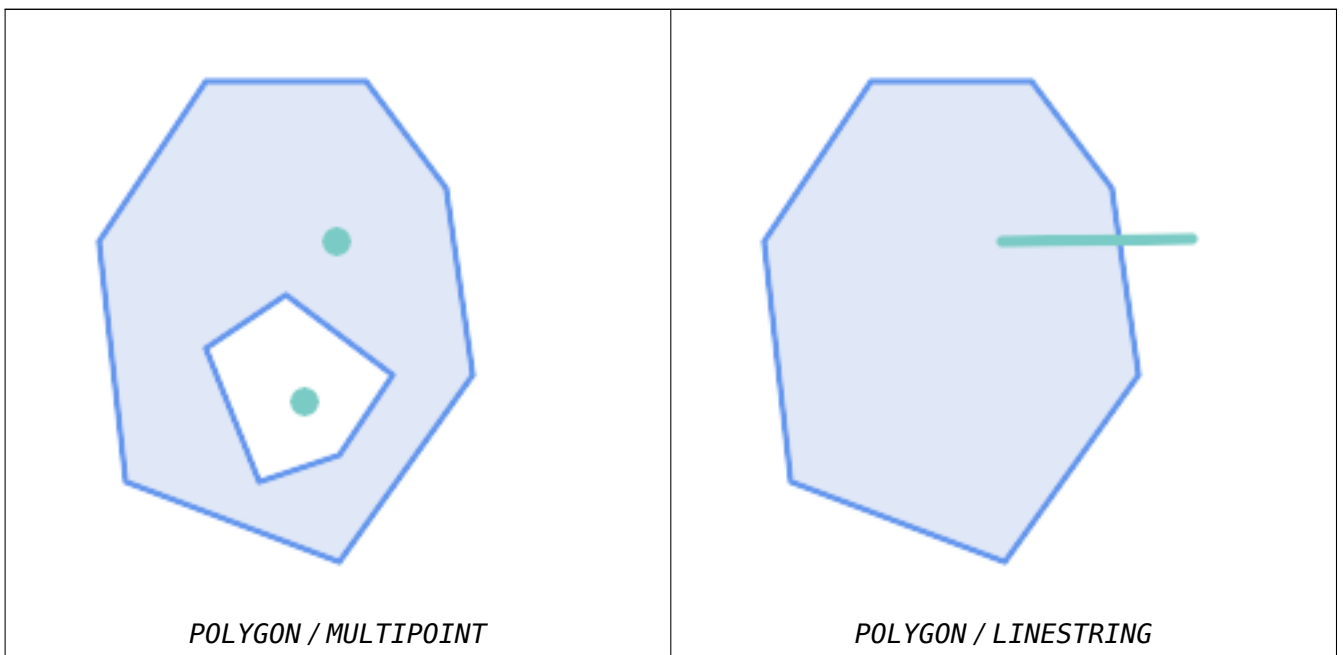
☒☒

`ST_Contains` returns TRUE in the following situations:



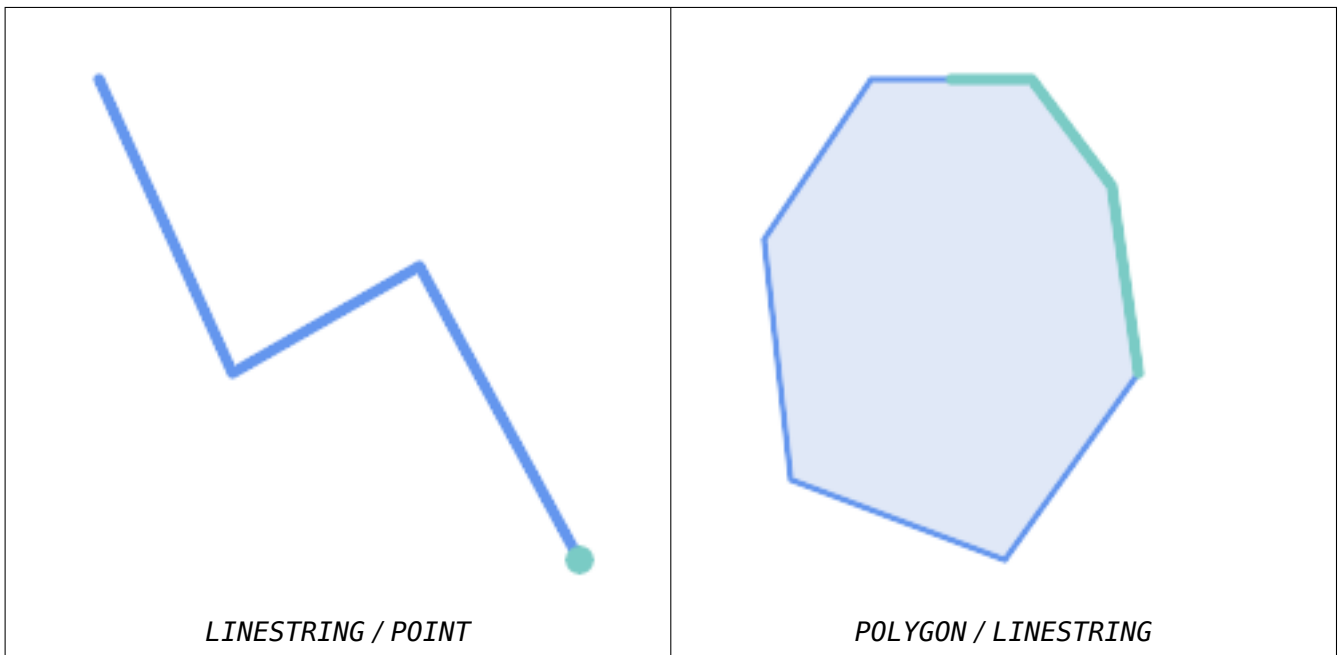


ST\_Contains returns FALSE in the following situations:



Due to the interior intersection condition ST\_Contains returns FALSE in the following situations (whereas ST\_Covers returns TRUE):

---



```
-- A circle within a circle
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 (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;

-- Result
smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | bigcontainsexterior |
-----+-----+-----+-----+-----+-----+
f | t | t | t | t | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,
 ST_ContainsProperly(geomA, geomA) AS acontainspropa,
 ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,
 ST_Boundary(geomA)) 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(geomA);

geomtype | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----+
ST_Polygon | t | f | f | f
ST_LineString | t | f | f | f
ST_Point | t | t | f | f
```

---

☒☒

[ST\\_Boundary](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Equals](#), [ST\\_Within](#)

### 7.11.1.3 ST\_ContainsProperly

`ST_ContainsProperly` — Tests if every point of B lies in the interior of A

#### Synopsis

boolean **ST\_ContainsProperly**(geometry geomA, geometry geomB);

☒☒

Returns true if every point of B lies in the interior of A (or equivalently, no point of B lies in the the boundary or exterior of A).

In mathematical terms:  $ST\_ContainsProperly(A, B) \Leftrightarrow Int(A) \supset B = B$

A contains B properly if the DE-9IM Intersection Matrix for the two geometries matches [T\*\*FF\*FF\*]

A does not properly contain itself, but does contain itself.

A use for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use `containsProperly` to filter out test geometries which lie fully inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.

---



#### Note

The advantage of this predicate over `ST_Contains` and `ST_Intersects` is that it can be computed more efficiently, with no need to compute topology at individual points.

---

GEOS ☒☒☒☒☒

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

---



#### Important

Do not use this function with invalid geometries. You will get unexpected results.

---

☒☒

```
--a circle within a circle
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 (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion | ↔
bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----+-----+
f | t | f | t | ↔
 | f | | |
--example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa, ↔
 ST_ContainsProperly(geomA, geomA) AS acontainspropa,
 ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA, ↔
 ST_Boundary(geomA)) 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(geomA);

geomtype | acontainsa | acontainspropa | acontainsba | acontainspropba
-----+-----+-----+-----+-----+
ST_Polygon | t | f | f | f
ST_LineString | t | f | f | f
ST_Point | t | t | f | f
```

☒☒

[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 if every point of A lies in B

#### Synopsis

boolean **ST\_CoveredBy**(geometry geomA, geometry geomB);  
 boolean **ST\_CoveredBy**(geography geogA, geography geogB);

☒☒

Returns true if every point in Geometry/Geography A lies inside (i.e. intersects the interior or boundary of) Geometry/Geography B. Equivalently, tests that no point of A lies outside (in the exterior of) B.

In mathematical terms:  $ST\_CoveredBy(A, B) \Leftrightarrow A \sqcap B = A$

ST\_CoveredBy is the converse of **ST\_Covers**. So, ST\_CoveredBy(A,B) = ST\_Covers(B,A).

Generally this function should be used instead of **ST\_Within**, since it has a simpler definition which does not have the quirk that "boundaries are not within their geometry".



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.



**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



**Important**

Do not use this function with invalid geometries. You will get unexpected results.

GEOS

1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

```
--a circle coveredby a circle
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 exeriorwithinbig
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig
-----+-----+-----+-----
t | t | t | f
(1 row)
```

**ST\_Contains, ST\_Covers, ST\_ExteriorRing, ST\_Within**

**7.11.1.5 ST\_Covers**

ST\_Covers — Tests if every point of B lies in A





```
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
--Result
smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----
t | f | t | f
(1 row)
```

Geography Example

```
-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer
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(ST_GeogFromText('SRID=4326;POINT(-99.327 31.4821)'), 300) As ↵
 geog_poly,
 ST_GeogFromText('SRID=4326;POINT(-99.33 31.483)') As geog_pt) As foo;

poly_covers_pt | buff_10m_covers_cent
-----+-----
f | t
```



[ST\\_Contains](#), [ST\\_CoveredBy](#), [ST\\_Within](#)

**7.11.1.6 ST\_Crosses**

ST\_Crosses — Tests if two geometries have some, but not all, interior points in common

**Synopsis**

boolean **ST\_Crosses**(geometry g1, geometry g2);



Compares two geometry objects and returns true if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns false. The crosses relation is symmetric and irreflexive.

In mathematical terms:  $ST\_Crosses(A, B) \Leftrightarrow (dim(Int(A) \cap Int(B)) < \max(dim(Int(A)), dim(Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$

Geometries cross if their DE-9IM Intersection Matrix matches:

- T\*T\*\*\*\*\* for Point/Line, Point/Area, and Line/Area situations
- T\*\*\*\*\*T\*\* for Line/Point, Area/Point, and Area/Line situations
- 0\*\*\*\*\* for Line/Line situations
- the result is false for Point/Point and Area/Area situations



**Note**

The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



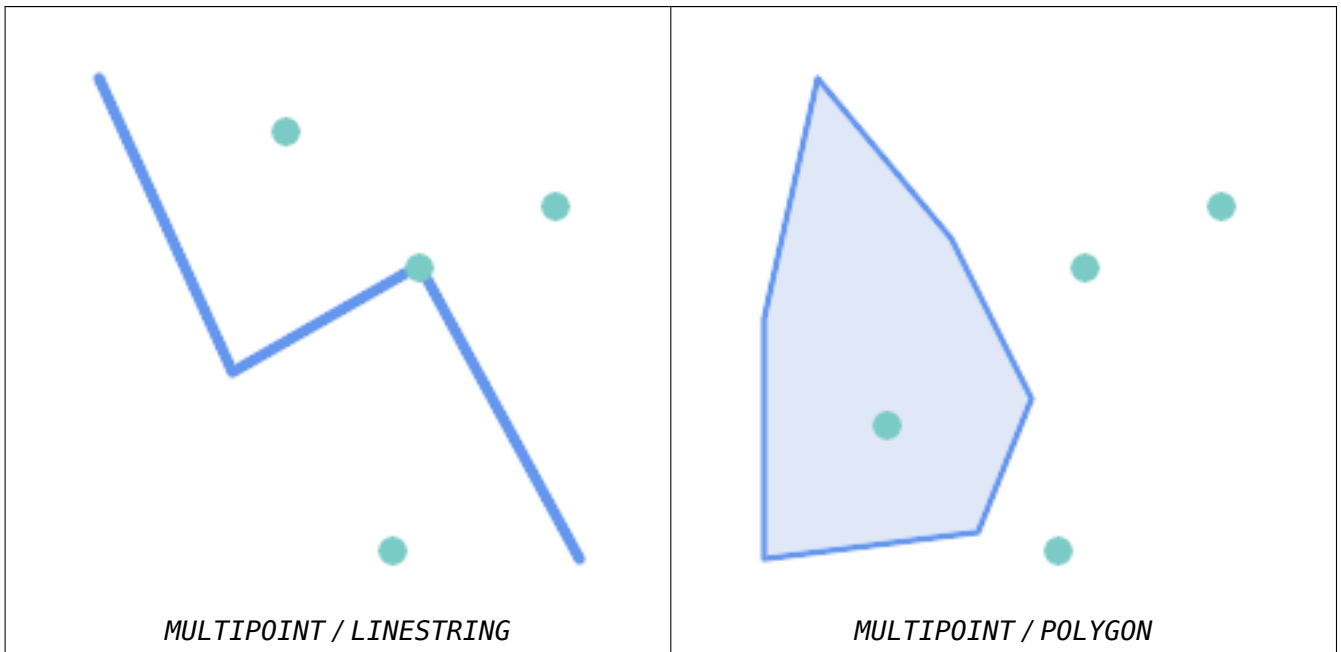
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3](#)

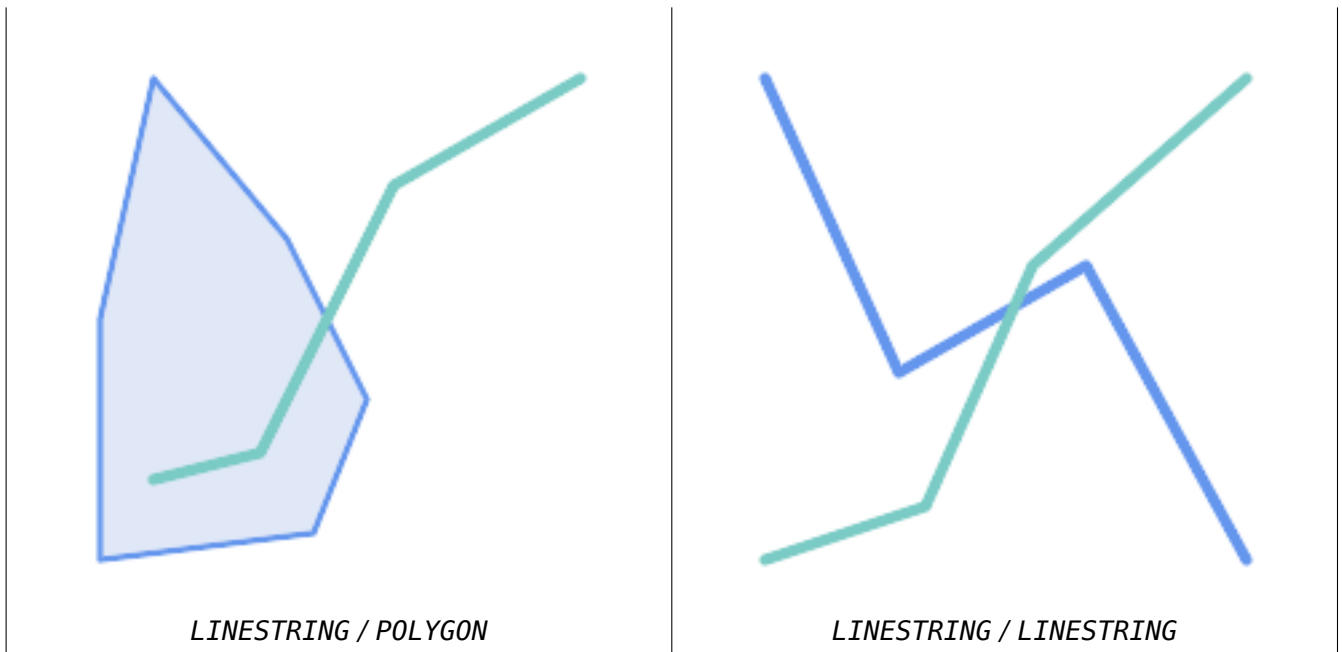


This method implements the SQL/MM specification. SQL-MM 3: 5.1.29



The following situations all return true.





Consider a situation where a user has two tables: a table of roads and a table of highways.

<pre>CREATE TABLE roads (   id serial NOT NULL,   geom geometry,   CONSTRAINT roads_pkey PRIMARY KEY ( ↵     road_id) );</pre>	<pre>CREATE TABLE highways (   id serial NOT NULL,   the_geom geometry,   CONSTRAINT roads_pkey PRIMARY KEY ( ↵     road_id) );</pre>
--------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------

To determine a list of roads that cross a highway, use a query similar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

☒☒

**ST\_Contains, ST\_Overlaps**

### 7.11.1.7 ST\_Disjoint

ST\_Disjoint — Tests if two geometries have no points in common

#### Synopsis

boolean **ST\_Disjoint**( geometry A , geometry B );

---

 ☒☒

Returns true if two geometries are disjoint. Geometries are disjoint if they have no point in common. If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that **ST\_Intersects** is false.

In mathematical terms:  $ST\_Disjoint(A, B) \Leftrightarrow A \cap B = \emptyset$



### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

---

GEOS ☒☒☒☒☒



### Note

This function call does not use indexes. A negated **ST\_Intersects** predicate can be used as a more performant alternative that uses indexes:  $ST\_Disjoint(A,B) = NOT\ ST\_Intersects(A,B)$

---



### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

---



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF\*FF\*\*\*\*')**



This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

☒☒

```
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2) '::geometry);
st_disjoint

t
(1 row)
SELECT ST_Disjoint('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2) '::geometry);
st_disjoint

f
(1 row)
```

☒☒

**ST\_Intersects**

### 7.11.1.8 ST\_Equals

**ST\_Equals** — Tests if two geometries include the same set of points

---

## Synopsis

boolean **ST\_Equals**(geometry A, geometry B);

☒☒

Returns true if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use **ST\_OrderingEquals** (it must be noted **ST\_OrderingEquals** is a little more stringent than simply verifying order of points are the same).

In mathematical terms:  $ST\_Equals(A, B) \Leftrightarrow A = B$

The following relation holds:  $ST\_Equals(A, B) \Leftrightarrow ST\_Within(A,B) \wedge ST\_Within(B,A)$



### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2**



This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

☒☒

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals

t
(1 row)
```

```
SELECT ST_Equals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_equals

t
(1 row)
```

☒☒

**ST\_IsValid**, **ST\_OrderingEquals**, **ST\_Reverse**, **ST\_Within**

### 7.11.1.9 ST\_Intersects

**ST\_Intersects** — Tests if two geometries intersect (they have at least one point in common)

## Synopsis

```
boolean ST_Intersects(geometry geomA , geometry geomB);
boolean ST_Intersects(geography geogA , geography geogB);
```

☒☒

Returns true if two geometries intersect. Geometries intersect if they have any point in common. For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect).

In mathematical terms:  $ST\_Intersects(A, B) \Leftrightarrow A \cap B \neq \emptyset$

Geometries intersect if their DE-9IM Intersection Matrix matches one of:

- T\*\*\*\*\*
- \*T\*\*\*\*\*
- \*\*\*T\*\*\*\*\*
- \*\*\*\*T\*\*\*\*\*

Spatial intersection is implied by all the other spatial relationship tests, except **ST\_Disjoint**, which tests that geometries do NOT intersect.

 Note!

### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL version removed and native support for 2D TINs added.

Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.

 Note!

### Note

For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.

 Note!

### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - ST\\_Intersects\(g1, g2 \) --> Not \(ST\\_Disjoint\(g1, g2 \)\)](#)

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.27

 This method supports Circular Strings and Curves.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```

SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2) '::geometry);
 st_intersects

 f
(1 row)
SELECT ST_Intersects('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2) '::geometry);
 st_intersects

 t
(1 row)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
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))');
 id | name
-----+-----
 2 | Minsk
(1 row)

```

☒☒☒☒☒

```

SELECT ST_Intersects(
 'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568) '::geography,
 'SRID=4326;POINT(-43.23456 72.4567772) '::geography
);

 st_intersects

 t

```

☒☒

**&&**, [ST\\_3DIntersects](#), [ST\\_Disjoint](#)

### 7.11.1.10 ST\_LineCrossingDirection

**ST\_LineCrossingDirection** — Returns a number indicating the crossing behavior of two LineStrings

#### Synopsis

integer **ST\_LineCrossingDirection**(geometry linestringA, geometry linestringB);

☒☒

Given two linestrings returns an integer between -3 and 3 indicating what kind of crossing behavior exists between them. 0 indicates no crossing. This is only supported for LINESTRINGs.

The crossing number has the following meaning:

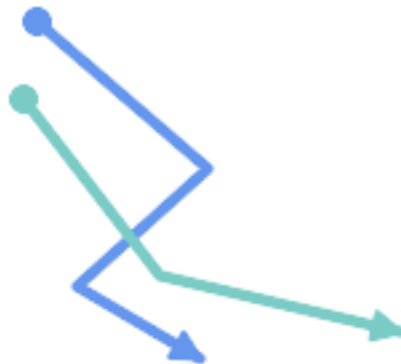
- 0: LINE NO CROSS
- -1: LINE CROSS LEFT

- 1: LINE CROSS RIGHT
- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Availability: 1.4

☒☒

**Example:** LINE CROSS LEFT and LINE CROSS RIGHT



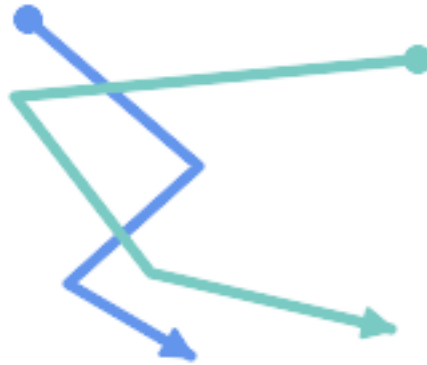
*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
 ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB
) As foo;
```

A_cross_B	B_cross_A
-1	1

**Example:** LINE MULTICROSS END SAME FIRST LEFT and LINE MULTICROSS END SAME FIRST RIGHT



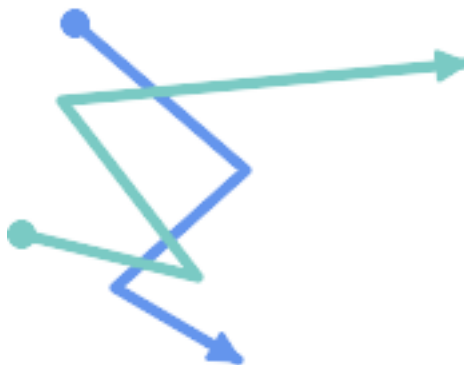


*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
 ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
) As foo;
```

A_cross_B	B_cross_A
3	-3

**Example:** LINE MULTICROSS END LEFT and LINE MULTICROSS END RIGHT



*Blue: Line A; Green: Line B*

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
 ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
) As foo;
```

```
A_cross_B | B_cross_A
-----+-----
 -2 | 2
```

**Example:** Finds all streets that cross

```
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;
```

☒☒

**ST\_Crosses**

### 7.11.1.11 ST\_OrderingEquals

**ST\_OrderingEquals** — Tests if two geometries represent the same geometry and have points in the same directional order

#### Synopsis

boolean **ST\_OrderingEquals**(geometry A, geometry B);

☒☒

**ST\_OrderingEquals** compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).



#### Note

This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. [http://edndoc.esri.com/arcscde/9.1/sql\\_api/sqlapi3.htm#ST\\_OrderingEquals](http://edndoc.esri.com/arcscde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

☒☒

```
SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'));
 st_orderingequals

 f
(1 row)
```

```
SELECT ST_OrderingEquals(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals
```

```

 t
(1 row)

SELECT ST_OrderingEquals(ST_Reverse(ST_GeomFromText('LINESTRING(0 0, 10 10)'),
 ST_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));
 st_orderingequals

 f
(1 row)

```

☒☒

&&, [ST\\_Equals](#), [ST\\_Reverse](#)

### 7.11.1.12 ST\_Overlaps

**ST\_Overlaps** — Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other

#### Synopsis

boolean **ST\_Overlaps**(geometry A, geometry B);

☒☒

Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive.

In mathematical terms:  $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \cap Int(B))) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

GEOS ☒☒☒☒☒



#### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

NOTE: this is the "allowable" version that returns a boolean, not an integer.



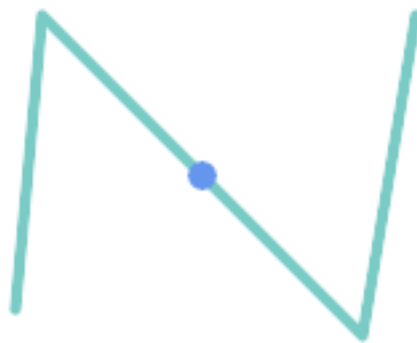
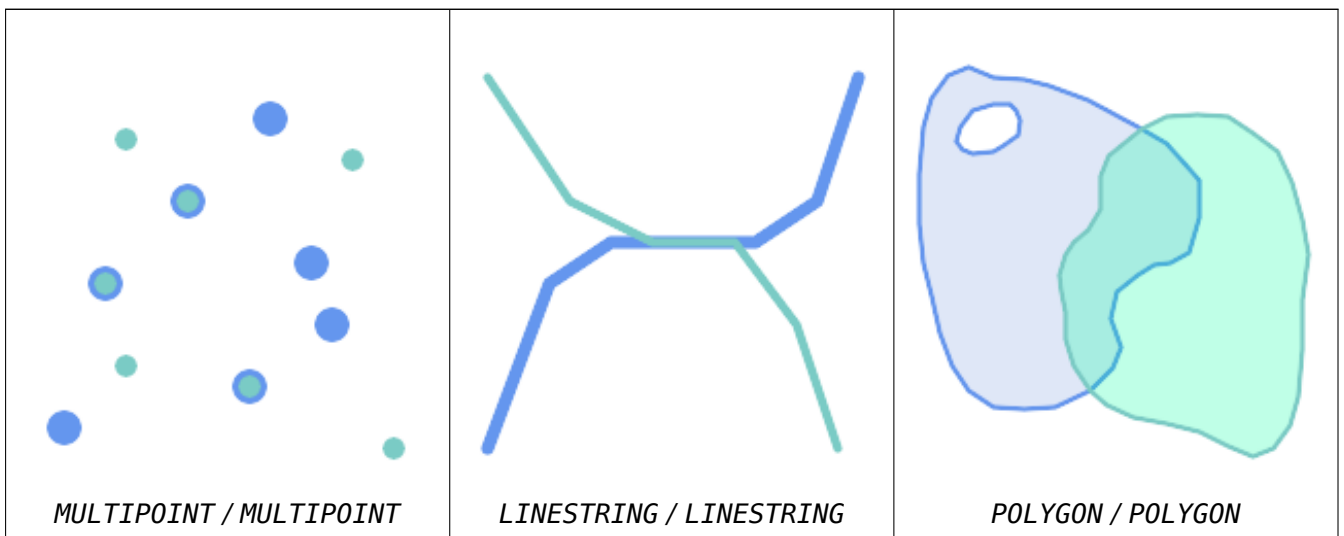
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

☒☒

ST\_Overlaps returns TRUE in the following situations:



A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
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 ST_GeomFromText('POINT (100 100)') As a,
 ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)') AS b) AS t
```

overlaps	crosses	intersects	b_contains_a
f	f	t	t



A LineString that partly covers a Polygon intersects and crosses, but does not overlap since it has different dimension.

```
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 ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
 ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;
```

overlap	crosses	intersects	contains
f	t	t	f



Two Polygons that intersect but with neither contained by the other overlap, but do not cross because their intersection has the same dimension.

```
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 ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
 ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) AS t;
```

overlaps	crosses	intersects	b_contains_a	dim_a	dim_b	dim_int
t	f	t	f	2	2	2

☒☒

[ST\\_Contains](#), [ST\\_Crosses](#), [ST\\_Dimension](#), [ST\\_Intersects](#)

### 7.11.1.13 ST\_Relate

**ST\_Relate** — Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

#### 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);
```

☒☒

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the [Dimensionally Extended 9-Intersection Model](#) (DE-9IM).

The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102').

A specific kind of spatial relationship can be tested by matching the intersection matrix to an *intersection matrix pattern*. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '\*' (meaning "any value"). Common spatial relationships are provided by the named functions [ST\\_Contains](#), [ST\\_ContainsProperly](#), [ST\\_Covers](#), [ST\\_CoveredBy](#), [ST\\_Crosses](#), [ST\\_Disjoint](#), [ST\\_Equals](#), [ST\\_Intersects](#), [ST\\_Overlaps](#), [ST\\_Touches](#), and [ST\\_Within](#). Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern T\*\*\*\*\*, which is not evaluated by any named predicate.

For more information refer to [Section 5.1](#).

**Variant 1:** Tests if two geometries are spatially related according to the given `intersectionMatrixPattern`

**Note**

Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the `&&` index call.

**Note**

It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evaluation.

**Variante 2:** Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using [ST\\_RelateMatch](#).

**Variante 3:** Like variante 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The boundaryNodeRule values are:

- 1: **OGC-Mod2** - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for ST\_Relate.
- 2: **Endpoint** - all endpoints are in the Boundary.
- 3: **MultivalentEndpoint** - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones).
- 4: **MonovalentEndpoint** - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints.

This function is not in the OGC spec, but is implied. see s2.1.13.2



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

GEOS ☒☒☒☒☒

Enhanced: 2.0.0 - added support for specifying boundary node rule.



### Important

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

☒☒

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '0FFFFF212');
st_relate

t
```

```
SELECT ST_Relate(POINT(1 2)', ST_Buffer('POINT(1 2)', 2), '*FF*FF212');
st_relate

t
```

Testing a custom spatial relationship pattern as a query condition, with && to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

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*****');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate('POINT(1 2)',
 ST_Buffer('POINT(1 2)', 2));

0FFFFFF212

SELECT ST_Relate('LINESTRING(1 2, 3 4)',
 'LINESTRING(5 6, 7 8)');

FF1FF0102
```

Using different Boundary Node Rules to compute the spatial relationship between a LineString and a MultiLineString with a duplicate endpoint (3 3):

- Using the **OGC-Mod2** rule (1) the duplicate endpoint is in the **interior** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is 0 and [aB:bB] is F.
- Using the **Endpoint** rule (2) the duplicate endpoint is in the **boundary** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is F and [aB:bB] is 0.

```
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;

bnr_mod2 | bnr_endpoint
-----+-----
FF10F0102 | FF1F00102
```



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 — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

#### Synopsis

boolean **ST\_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);



Tests if a **Dimensionally Extended 9-Intersection Model** (DE-9IM) intersectionMatrix value satisfies an intersectionMatrixPattern. Intersection matrix values can be computed by [ST\\_Relate](#).

For more information refer to Section 5.1.

GEOS

2.0.0



☒☒

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
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***'),
 ('Disjoint', 'FF**F***')) AS pat(name,val)
CROSS JOIN
(VVALUES ('non-intersecting', 'FF1FF0212'),
 ('overlapping', '1010F0212'),
 ('inside', '1FF0FF212')) AS mat(name,val);
```

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***	non-intersecting	FF1FF0212	f
Within	T**F***	overlapping	1010F0212	f
Within	T**F***	inside	1FF0FF212	t
Disjoint	FF**F***	non-intersecting	FF1FF0212	t
Disjoint	FF**F***	overlapping	1010F0212	f
Disjoint	FF**F***	inside	1FF0FF212	f

☒☒

Section 5.1, [ST\\_Relate](#)

**7.11.1.15 ST\_Touches**

ST\_Touches — Tests if two geometries have at least one point in common, but their interiors do not intersect

**Synopsis**

boolean **ST\_Touches**(geometry A, geometry B);

☒☒

Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary.

In mathematical terms:  $ST\_Touches(A, B) \Leftrightarrow (Int(A) \cap Int(B) = \emptyset) \wedge (A \cap B \neq \emptyset)$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT\*\*\*\*\*
- F\*\*T\*\*\*\*\*
- F\*\*\*T\*\*\*\*



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.



**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



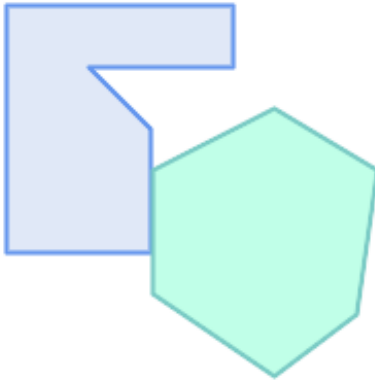
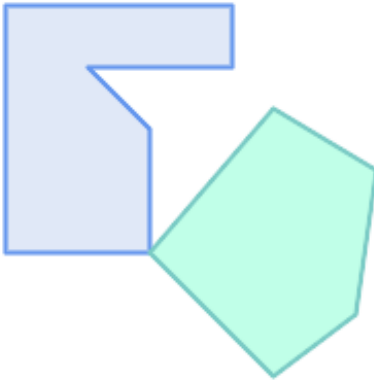
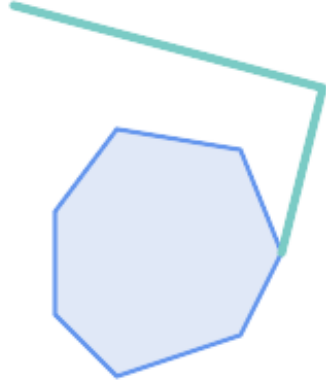
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3](#)

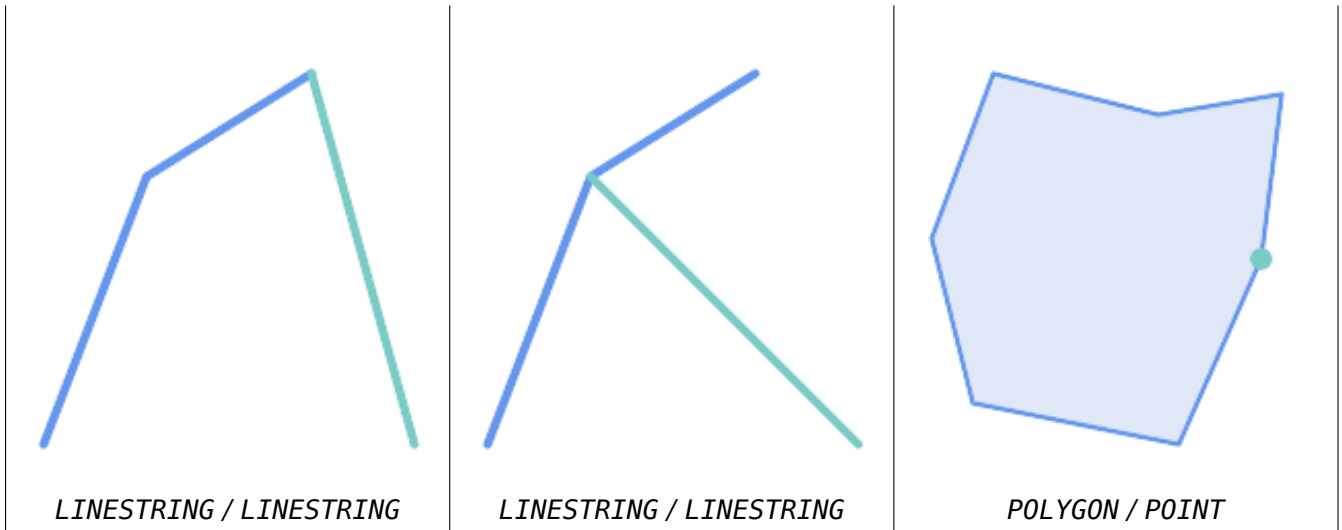


This method implements the SQL/MM specification. SQL-MM 3: 5.1.28



The `ST_Touches` predicate returns TRUE in the following examples.

 <p><i>POLYGON / POLYGON</i></p>	 <p><i>POLYGON / POLYGON</i></p>	 <p><i>POLYGON / LINESTRING</i></p>
---------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------



```

SELECT ST_Touches('LINestring(0 0, 1 1, 0 2)::geometry, 'POINT(1 1)::geometry);
st_touches

f
(1 row)

SELECT ST_Touches('LINestring(0 0, 1 1, 0 2)::geometry, 'POINT(0 2)::geometry);
st_touches

t
(1 row)

```

### 7.11.1.16 ST\_Within

ST\_Within — Tests if every point of A lies in B, and their interiors have a point in common

#### Synopsis

boolean **ST\_Within**(geometry A, geometry B);

☒☒

Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common.

For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

In mathematical terms:  $ST\_Within(A, B) \Leftrightarrow (A \sqcap B = A) \wedge (Int(A) \sqcap Int(B) \neq \square)$

The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if  $ST\_Within(A, B) = true$  and  $ST\_Within(B, A) = true$ , then the two geometries must be topologically equal ( $ST\_Equals(A, B) = true$ ).

ST\_Within is the converse of **ST\_Contains**. So,  $ST\_Within(A, B) = ST\_Contains(B, A)$ .



**Note**

Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are *not* within the geometry. For further details see [Subtleties of OGC Covers, Contains, Within](#). The `ST_CoveredBy` predicate provides a more inclusive relationship.



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Within`.

GEOS ☒☒☒☒☒

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.



**Important**

Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION



**Important**

Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate\(b, 'T\\*\\*F\\*\\*\\*'\)](#)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

☒☒

```
--a circle within a circle
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 beginunion,
 ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
 ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
smallinsmall | smallinbig | biginsmall | unioninbig | beginunion | bigisunion
-----+-----+-----+-----+-----+-----
t | t | f | t | t | t
(1 row)
```



☒☒

*ST\_Contains, ST\_CoveredBy, ST\_Equals, ST\_IsValid*

## 7.11.2 Distance Relationships

### 7.11.2.1 ST\_3DDWithin

ST\_3DDWithin — Tests if two 3D geometries are within a given 3D distance

#### Synopsis

boolean **ST\_3DDWithin**(geometry g1, geometry g2, double precision distance\_of\_srid);

☒☒

Returns true if the 3D distance between two geometry values is no larger than distance `distance_of_srid`. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID).



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This method implements the SQL/MM specification. SQL-MM ?

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point ↔
 and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↔
 units as final.
SELECT ST_3DDWithin(
 ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
 ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↔
 20)'),2163),
 126.8
) As within_dist_3d,
ST_DWithin(
 ST_Transform(ST_GeomFromEWKT('SRID=4326;POINT(-72.1235 42.3521 4)'),2163),
 ST_Transform(ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ↔
 20)'),2163),
 126.8
) As within_dist_2d;

within_dist_3d | within_dist_2d
-----+-----
f | t
```

☒☒

[ST\\_3DDFullyWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DDistance](#), [ST\\_Distance](#), [ST\\_3DMaxDistance](#), [ST\\_Transform](#)

### 7.11.2.2 ST\_3DDFullyWithin

ST\_3DDFullyWithin — Tests if two 3D geometries are entirely within a given 3D distance

#### Synopsis

boolean **ST\_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

☒☒

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.



**Note**

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.

☒☒

```
-- This compares the difference between fully within and distance within as well
-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully
-- within
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
(select ST_GeomFromEWKT('POINT(1 1 2)') as geom_a,
ST_GeomFromEWKT('LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)') as geom_b) t1;
d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20
-----+-----+-----+-----
f | t | t | f
```

☒☒

[ST\\_3DDWithin](#), [ST\\_DWithin](#), [ST\\_DFullyWithin](#), [ST\\_3DMaxDistance](#)

### 7.11.2.3 ST\_DFullyWithin

`ST_DFullyWithin` — Tests if a geometry is entirely inside a distance of another

#### Synopsis

boolean `ST_DFullyWithin`(geometry g1, geometry g2, double precision distance);

☒☒

Returns true if g2 is entirely within distance of g1. Visually, the condition is true if g2 is contained within a distance buffer of g1. The distance is specified in units defined by the spatial reference system of the geometries.



#### Note

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

1.5.0 ☒☒☒☒☒☒☒☒☒☒☒.

Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the `ST_MaxDistance` algorithm. Results will differ from prior versions, but should be closer to user expectations.

☒☒

```
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)', 'LINESTRING(1 5, 2 7, 1 9, 14 12)')
) AS v(geom_a, geom_b)
```

dfullywithin10		dwithin10		dfullywithin20
-----	+	-----	+	-----
f		t		t

☒☒

[ST\\_MaxDistance](#), [ST\\_DWithin](#), [ST\\_3DDWithin](#), [ST\\_3DDFullyWithin](#)

#### 7.11.2.4 ST\_DWithin

ST\_DWithin — Tests if two geometries are within a given distance

##### 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);

☒☒

Returns true if the geometries are within a given distance

For geometry: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must be in the same coordinate system (have the same SRID).

For geography: units are in meters and distance measurement defaults to `use_spheroid = true`. For faster evaluation use `use_spheroid = false` to measure on the sphere.



##### Note

Use [ST\\_3DDWithin](#) for 3D geometries.



##### Note

This function call includes a bounding box comparison that makes use of any indexes that are available on the geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

Availability: 1.5.0 support for geography was introduced

Enhanced: 2.1.0 improved speed for geography. See [Making Geography faster](#) for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Prior to 1.3, [ST\\_Expand](#) was commonly used in conjunction with `&&` and `ST_Distance` to test for distance, and in pre-1.3.4 this function used that logic. From 1.3.4, `ST_DWithin` uses a faster short-circuit distance function.



☒☒

```

-- 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
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 is in units of spatial ref (e.g. meters, feet, degrees)
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 broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
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);

```

☒☒

**ST\_Distance, ST\_3DDWithin**

### 7.11.2.5 ST\_PointInsideCircle

ST\_PointInsideCircle — Tests if a point geometry is inside a circle defined by a center and radius

#### Synopsis

boolean **ST\_PointInsideCircle**(geometry a\_point, float center\_x, float center\_y, float radius);

☒☒

Returns true if the geometry is a point and is inside the circle with center center\_x,center\_y and radius radius.



**Warning**

Does not use spatial indexes. Use **ST\_DWithin** instead.

Availability: 1.2

Changed: 2.2.0 In prior versions this was called ST\_Point\_Inside\_Circle

SQL

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle

t
```

SQL

**ST\_DWithin**

## 7.12 Measurement Functions

### 7.12.1 ST\_Area

ST\_Area — Returns the area of a geometry.

**Synopsis**

```
float ST_Area(geometry g1);
float ST_Area(geography geog, boolean use_spheroid = true);
```




SQL

ST\_Area(geometry g1) - ST\_Surface or ST\_MultiSurface - SRID 2 (curved surface) ST\_Area(geog,false)

2.0.0 (polyhedral surface)

2.2.0 GeographicLib Proj 4.9.0

Changed: 3.0.0 - does not depend on SFCGAL anymore.

-  This method implements the **OGC Simple Features Implementation Specification for SQL 1.1**.
-  This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3
-  This function supports Polyhedral surfaces.



```

select ST_Transform(
 'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265
 2967450,743265.625 2967416,743238 2967416))'::geometry,
 4326
) :: geography geog
) as subquery;

```

If your data is in geography already:

```

select ST_Area(geog) / 0.3048 ^ 2 sqft,
 ST_Area(the_geog) sqm
from somegeogtable;

```



[ST\\_3DArea](#), [ST\\_GeomFromEWKT](#), [ST\\_LengthSpheroid](#), [ST\\_Perimeter](#), [ST\\_Transform](#)

### 7.12.2 ST\_Azimuth

ST\_Azimuth — [Geography](#) 2 [Geometry](#).

#### Synopsis

```

float ST_Azimuth(geometry origin, geometry target);
float ST_Azimuth(geography origin, geography target);

```



Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast =  $\pi/4$ ; East =  $\pi/2$ ; Southeast =  $3\pi/4$ ; South =  $\pi$ ; Southwest  $5\pi/4$ ; West =  $3\pi/2$ ; Northwest =  $7\pi/4$ .

For the geography type, the azimuth solution is known as the **inverse geodesic problem**.

The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

ST\_Translate. [Plpgsqlfunctions PostGIS wiki section](#) upgis\_lineshift

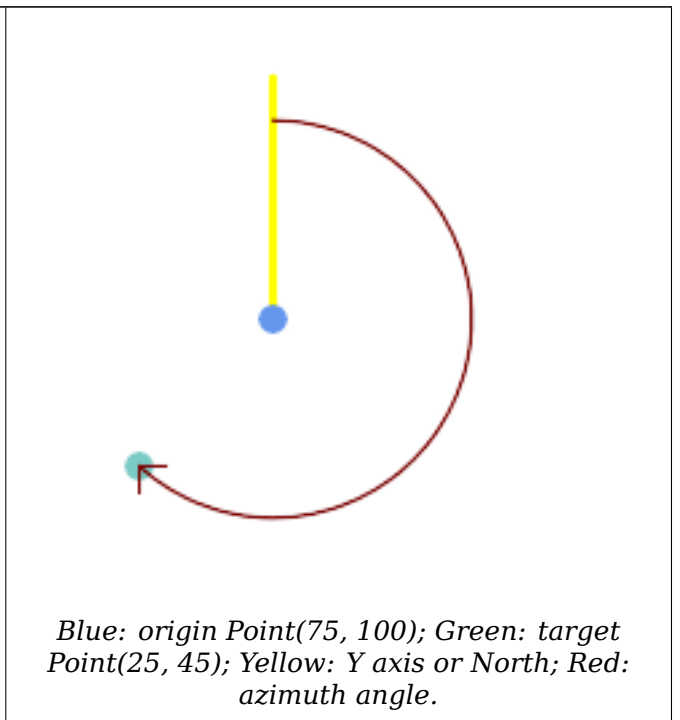
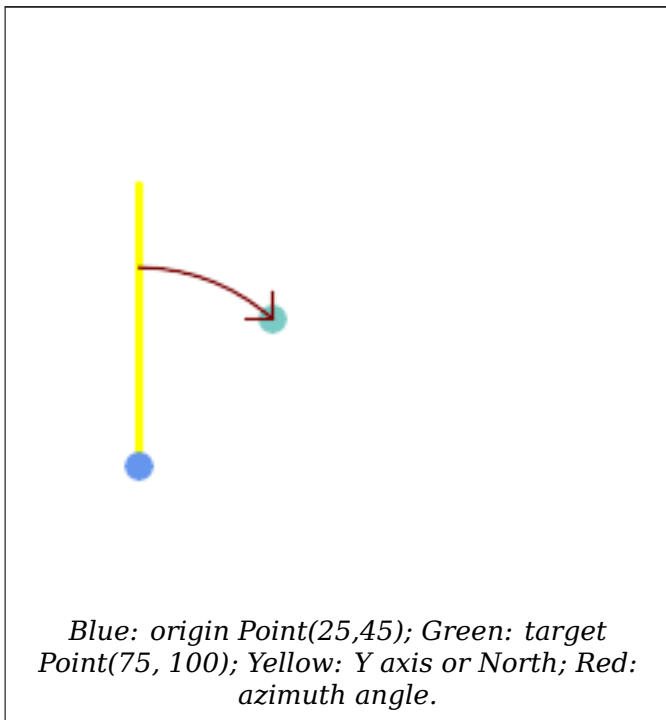
1.1.0

2.0.0

2.2.0 GeographicLib Proj 4.9.0

```
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;
```

dega_b	degb_a
42.2736890060937	222.273689006094



ST\_Angle, ST\_Translate, ST\_Project, PostgreSQL Math Functions

### 7.12.3 ST\_Angle

ST\_Angle — 3 (longest)

## Synopsis

float **ST\_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);  
 float **ST\_Angle**(geometry line1, geometry line2);

⊠

⊠ 3 ⊠ (longest) ⊠.

**Variation 1:** computes the angle enclosed by the points P1-P2-P3. If a 4th point provided computes the angle points P1-P2 and P3-P4

**Variation 2:** computes the angle between two vectors S1-E1 and S2-E2, defined by the start and end points of the input lines

⊠. PostgreSQL `degrees()` ⊠.

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

⊠

⊠

```
SELECT degrees(ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)'));
```

```
degrees

 270
```

Angle between vectors defined by four points

```
SELECT degrees(ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)'));
```

```
degrees

269.999999999999
```

Angle between vectors defined by the start and end points of lines

```
SELECT degrees(ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)'));
```

```
degrees

 45
```

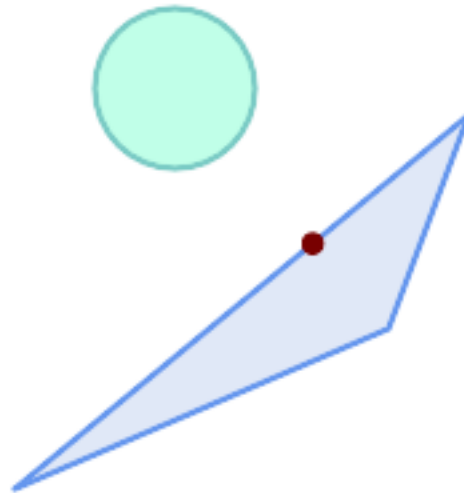
⊠

## ST\_Azimuth

### 7.12.4 ST\_ClosestPoint

`ST_ClosestPoint` — Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.





The closest point on polygon A to polygon B

```
SELECT ST_AsText(ST_ClosestPoint(
 'POLYGON ((190 150, 20 10, 160 70, 190 150))',
 ST_Buffer('POINT(80 160)', 30)
)) As ptwkt;

POINT(131.59149149528952 101.89887534906197)
```

☒☒

[ST\\_3DClosestPoint](#), [ST\\_Distance](#), [ST\\_LongestLine](#), [ST\\_ShortestLine](#), [ST\\_MaxDistance](#)

### 7.12.5 ST\_3DClosestPoint

`ST_3DClosestPoint` — g2 ☒☒☒☒☒☒ g1 ☒☒☒☒ 3 ☒☒☒☒☒☒☒☒☒☒☒. ☒☒☒☒☒☒ 3D ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_3DClosestPoint**(geometry g1, geometry g2);

☒☒

g2 ☒☒☒☒☒☒ g1 ☒☒☒☒ 3 ☒☒☒☒☒☒☒☒☒☒☒. ☒☒☒☒☒☒ 3D ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒. 3D ☒☒☒☒☒☒ 3D ☒☒☒☒ 3D ☒☒☒☒☒☒.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

☒☒☒☒: 2.2.0 ☒☒☒☒ 2D ☒☒☒☒☒☒☒☒☒☒☒, (☒☒☒☒☒☒☒ Z ☒ 0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒) 2D ☒☒☒☒☒☒☒☒☒. 2D ☒ 3D ☒☒☒, ☒☒☒ Z ☒☒☒☒☒ Z ☒ 0 ☒☒☒☒☒☒☒☒☒☒☒.

☒☒



<pre> -- 3D, 2D ST_ClosestPoint (closest point) SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,        ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_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; </pre>	<pre> cp3d_line_pt   cp2d_line_pt -----+----- POINT(54.6993798867619 128.935022917228 11.5475869506606)   POINT(73.0769230769231 115.384615384615) </pre>
<pre> -- 3D, 2D ST_ClosestPoint (closest point) SELECT ST_AsEWKT(ST_3DClosestPoint(line,pt)) AS cp3d_line_pt,        ST_AsEWKT(ST_ClosestPoint(line,pt)) As cp2d_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; </pre>	<pre> cp3d_line_pt   cp2d_line_pt -----+----- POINT(54.6993798867619 128.935022917228 11.5475869506606)   POINT(50 75) </pre>
<pre> -- 3D, 2D ST_ClosestPoint (closest point) SELECT ST_AsEWKT(ST_3DClosestPoint(poly, mline)) As cp3d,        ST_AsEWKT(ST_ClosestPoint(poly, mline)) As cp2d 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; </pre>	<pre> cp3d   cp2d -----+----- POINT(39.993580415989 54.1889925532825 5)   POINT(20 40) </pre>

☒☒

[ST\\_AsEWKT](#), [ST\\_ClosestPoint](#), [ST\\_3DDistance](#), [ST\\_3DShortestLine](#)

### 7.12.6 ST\_Distance

ST\_Distance — ☒☒☒☒☒☒ 3 ☒☒☒☒ (longest) ☒☒☒☒☒☒☒☒.

## Synopsis

```
float ST_Distance(geometry g1, geometry g2);
float ST_Distance(geography geog1, geography geog2, boolean use_spheroid = true);
```

⊠

⊠, ⊠ 3 ⊠ (SRS ⊠) ⊠.

For **geography** types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If `use_spheroid` is false, a faster spherical calculation is used.

- ✔ This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).
- ✔ This method implements the SQL/MM specification. SQL-MM 3: 5.1.23
- ✔ This method supports Circular Strings and Curves.

1.5.0 ⊠. ⊠. ⊠.

⊠: 2.1.0 ⊠. ⊠ [Making Geography faster](#) ⊠.

⊠: 2.1.0 ⊠.

⊠: 2.2.0 ⊠ GeographicLib ⊠. ⊠ Proj 4.9.0 ⊠.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

⊠

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance(
 'SRID=4326;POINT(-72.1235 42.3521)::geometry,
 'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)::geometry);

0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
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)) ←
 ;

167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by  $\cos(\text{lat})$  to account for distortion)

```
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);

123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
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) ←
);

123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
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)) ←
;

126.664256056812
```

Geography

Same as geometry example but note units in meters - use sphere for slightly faster and less accurate computation.

```
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 ;

spheroid_dist | sphere_dist
-----+-----
123.802076746848 | 123.475736916397
```

Geography

[ST\\_3DDistance](#), [ST\\_DWithin](#), [ST\\_DistanceSphere](#), [ST\\_DistanceSpheroid](#), [ST\\_MaxDistance](#), [ST\\_HausdorffDistance](#), [ST\\_FrechetDistance](#), [ST\\_Transform](#)

### 7.12.7 ST\_3DDistance


**ST\_3DDistance** — [Geography](#), [Geometry](#) (SRS [Geography](#)) 3 [Geography](#) [Geometry](#).

#### Synopsis

float **ST\_3DDistance**(geometry g1, geometry g2);

Geography

[Geography](#), [Geometry](#) 3 [Geography](#) [Geometry](#) (SRS [Geography](#)) [Geometry](#).

 This function supports 3d and will not drop the z-index.

- ✔ This function supports Polyhedral surfaces.
  - ✔ This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3 2.0.0.
- 2.2.0: 2.2.0, 2D & 3D Z & 0.
- Changed: 3.0.0 - SFCGAL version removed

☒

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same units as final.
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;

dist_3d | dist_2d
-----+-----
127.295059324629 | 126.66425605671
```

```
-- Multilinestring and polygon both 3d and 2d distance
-- Same example as 3D closest point example
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;

dist3d | dist2d
-----+-----
0.716635696066337 | 0
```

☒

[ST\\_Distance](#), [ST\\_3DClosestPoint](#), [ST\\_3DDWithin](#), [ST\\_3DMaxDistance](#), [ST\\_3DShortestLine](#), [ST\\_Transform](#)

### 7.12.8 ST\_DistanceSphere

ST\_DistanceSphere — PostGIS 1.5.

#### Synopsis

float **ST\_DistanceSphere**(geometry geom1lonlatA, geometry geom1lonlatB, float8 radius=6371008);

¶¶

¶¶¶¶¶ 2 ¶¶¶¶¶. SRID ¶¶¶¶¶. **ST\_DistanceSpheroid** ¶¶¶¶¶, ¶¶¶¶¶. PostGIS 1.5 ¶¶¶¶¶.

1.5 ¶¶¶¶¶. 1.5 ¶¶¶¶¶.

¶¶¶¶: 2.2.0 ¶¶¶¶¶ ST\_Distance\_Sphere ¶¶¶¶¶.

¶¶

```

SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
 ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
 As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As ←
 numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
 ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
 As min_dist_line_point_meters
FROM
 (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
 as foo;
 dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
-----+-----+-----+-----
 70424.47 | 70438.00 | 0.72900 | 65871.18

```

¶¶

**ST\_Distance, ST\_DistanceSpheroid**

**7.12.9 ST\_DistanceSpheroid**

ST\_DistanceSpheroid — ¶¶¶¶¶. PostGIS 1.5 ¶¶¶¶¶.

**Synopsis**

float **ST\_DistanceSpheroid**(geometry geom1lonlatA, geometry geom1lonlatB, spheroid measurement\_spheroid)

¶¶

¶¶¶¶¶. ¶¶¶¶¶ **ST\_LengthSpheroid** ¶¶¶¶¶. PostGIS 1.5 ¶¶¶¶¶.



**Note**

¶¶¶¶¶ SRID ¶¶¶¶¶. ¶¶¶¶¶.

1.5. ST\_Distance\_Spheroid. 1.5. ST\_DistanceSphere.

2.2.0 ST\_Distance\_Spheroid.

SQL

```

SELECT round(CAST(
 ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ←
 ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
 As numeric),2) As dist_meters_spheroid,
 round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 ←
 38)',4326)) As numeric),2) As dist_meters_sphere,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
 ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ←
 As dist_utm11_meters
FROM
 (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As geom) ←
 as foo;
dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
-----+-----+-----
70454.92 | 70424.47 | 70438.00

```

SQL

ST\_Distance, ST\_DistanceSphere

7.12.10 ST\_FrechetDistance

ST\_FrechetDistance — 3 (shortest).

Synopsis

```
float ST_FrechetDistance(geometry g1, geometry g2, float densifyFrac = -1);
```

SQL

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on Computing Discrete Fréchet Distance. The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional densifyFrac is specified, this function performs a segment densification before computing the discrete Fréchet distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.



Note

ST\_FrechetDistance. ST\_FrechetDistance.

**Note**

The smaller `densifyFrac` we specify, the more accurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

GEOS

Availability: 2.4.0 - requires GEOS  $\geq$  3.7.0

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, ↵
 50 50, 100 0)::geometry');
 st_frechetdistance

 70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)::geometry, 'LINESTRING (0 0, 50 50, 100 ↵
 0)::geometry, 0.5);
 st_frechetdistance

 50
(1 row)
```

**ST\_HausdorffDistance**

### 7.12.11 ST\_HausdorffDistance

`ST_HausdorffDistance` — (shortest)

#### Synopsis

```
float ST_HausdorffDistance(geometry g1, geometry g2);
float ST_HausdorffDistance(geometry g1, geometry g2, float densifyFrac);
```

Returns the **Hausdorff distance** between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

The function actually computes the “Discrete Hausdorff Distance”. This is the Hausdorff distance computed at discrete points on the geometries. The *densifyFrac* parameter can be specified, to provide a more accurate answer by densifying segments before computing the discrete Hausdorff distance. Each segment is split into a number of equal-length subsegments whose fraction of the segment length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.



**Note**

This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important case is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

1.5.0



*Hausdorff distance (red) and distance (yellow) between two lines*

```
SELECT ST_HausdorffDistance(geomA, geomB),
 ST_Distance(geomA, geomB)
FROM (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) AS t;
st_hausdorffdistance | st_distance
-----+-----
37.26206567625497 | 20
```

**Example:** Hausdorff distance with densification.

```
SELECT ST_HausdorffDistance(
 'LINESTRING (130 0, 0 0, 0 150)>:::geometry,
 'LINESTRING (10 10, 10 150, 130 10)>:::geometry,
 0.5);

70
```

**Example:** For each building, find the parcel that best represents it. First we require that the parcel intersect with the building geometry. DISTINCT ON guarantees we get each building listed only once. ORDER BY .. ST\_HausdorffDistance selects the parcel that is most similar to the building.

```
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);
```











### 7.12.16 ST\_LongestLine

ST\_LongestLine — Returns the 2-dimensional longest line between the points of two geometries.

#### Synopsis

geometry **ST\_LongestLine**(geometry g1, geometry g2);

Returns

Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on g1 and ends on g2.

The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by **ST\_MaxDistance**.

If 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**.

1.5.0

Example

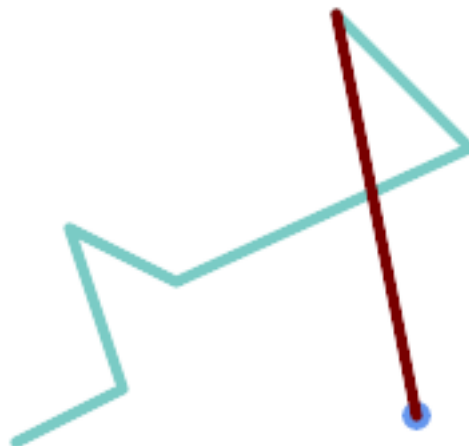


Diagram illustrating the longest line between a polygon and a point.

```
SELECT ST_AsText(ST_LongestLine(
 'POINT (160 40)',
 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) AS lline;

LINESTRING(160 40,130 190)
```





```

-- 3D, 2D
SELECT ST_AsEWKT(ST_3DLongestLine(line,pt)) AS lol3d_line_pt,
 ST_AsEWKT(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;

lol3d_line_pt | lol2d_line_pt
-----+-----
LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)

-- 3D, 2D
SELECT ST_AsEWKT(ST_3DLongestLine(poly, mline)) As lol3d,
 ST_AsEWKT(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;

lol3d | lol2d
-----+-----
LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)

```

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_3DShortestLine](#), [ST\\_3DMaxDistance](#)

### 7.12.18 ST\_MaxDistance

ST\_MaxDistance — 2

#### Synopsis

float **ST\_MaxDistance**(geometry g1, geometry g2);

2 . g1 g2

2 . g1 g2

1.5.0









specified by the SRID, otherwise it is exclusive to WGS84. If use\_spheroid = false, then calculations will approximate a sphere instead of a spheroid.

`ST_Perimeter2D`, `ST_Perimeter`.

 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1 s2.1.5.1](#)

 This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4

Version: 2.0.0

Usage:

`ST_Perimeter` (geometry) returns double precision. SRID: EPSG:2249

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,743238 2967416))', 2249));
st_perimeter

122.630744000095
(1 row)
```

```
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));
st_perimeter

845.227713366825
(1 row)
```

Usage:

`ST_Perimeter` (geometry) returns double precision. SRID: WGS84

```
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog)/0.3048 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;

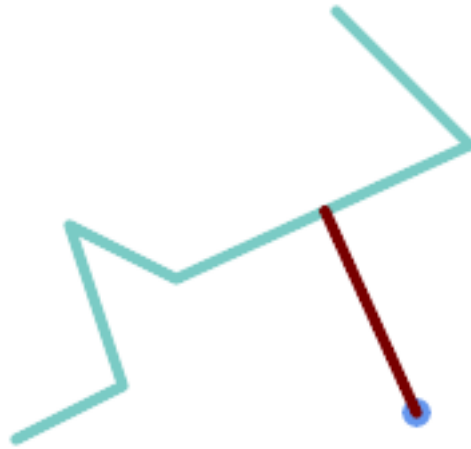
per_meters | per_ft
-----+-----
37.3790462565251 | 122.634666195949

-- MultiPolygon example --
SELECT ST_Perimeter(geog) As per_meters, ST_Perimeter(geog,false) As per_sphere_meters, ↔
ST_Perimeter(geog)/0.3048 As per_ft
```





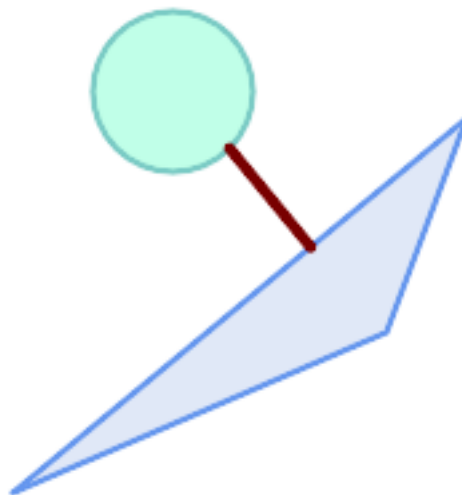
☒☒



*Shortest line between Point and LineString*

```
SELECT ST_AsText(ST_ShortestLine(
 'POINT (160 40)',
 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
) As sline;

LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



*Shortest line between Polygons*

```
SELECT ST_AsText(ST_ShortestLine(
 'POLYGON ((190 150, 20 10, 160 70, 190 150))',
 ST_Buffer('POINT(80 160)', 30)
)) AS llinewkt;

LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```





```

-- 3D, 2D
SELECT ST_AsEWKT(ST_3DShortestLine(line,pt)) AS shl3d_line_pt,
 ST_AsEWKT(ST_ShortestLine(line,pt)) As shl2d_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;

shl2d_line_pt shl3d_line_pt | ←
-----+-----
LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ←
(50 75,50 74)

-- 3D, 2D
SELECT ST_AsEWKT(ST_3DShortestLine(poly, mline)) As shl3d,
 ST_AsEWKT(ST_ShortestLine(poly, mline)) As shl2d
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;
shl3d ←
| shl2d
-----+-----
LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529 ←
5.03423778139177) | LINESTRING(20 40,20 40)

```

☒☒

[ST\\_3DClosestPoint](#), [ST\\_3DDistance](#), [ST\\_LongestLine](#), [ST\\_ShortestLine](#), [ST\\_3DMaxDistance](#)

## 7.13 Overlay Functions

### 7.13.1 ST\_ClipByBox2D

ST\_ClipByBox2D — Computes the portion of a geometry falling within a rectangle.

#### Synopsis

geometry **ST\_ClipByBox2D**(geometry geom, box2d box);

☒☒

Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way. Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).

GEOS ☒☒☒☒☒

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

☒☒

[ST\\_Intersection](#), [ST\\_MakeBox2D](#), [ST\\_MakeEnvelope](#)

## 7.13.2 ST\_Difference

**ST\_Difference** — Computes a geometry representing the part of geometry A that does not intersect geometry B.

### Synopsis

geometry **ST\_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

☒☒

Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to  $A - ST\_Intersection(A,B)$ . If A is completely contained in B then an empty atomic geometry of appropriate type is returned.

 **Note!**

### Note

This is the only overlay function where input order matters. `ST_Difference(A, B)` always returns a portion of A.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

GEOS ☒☒☒☒☒

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)

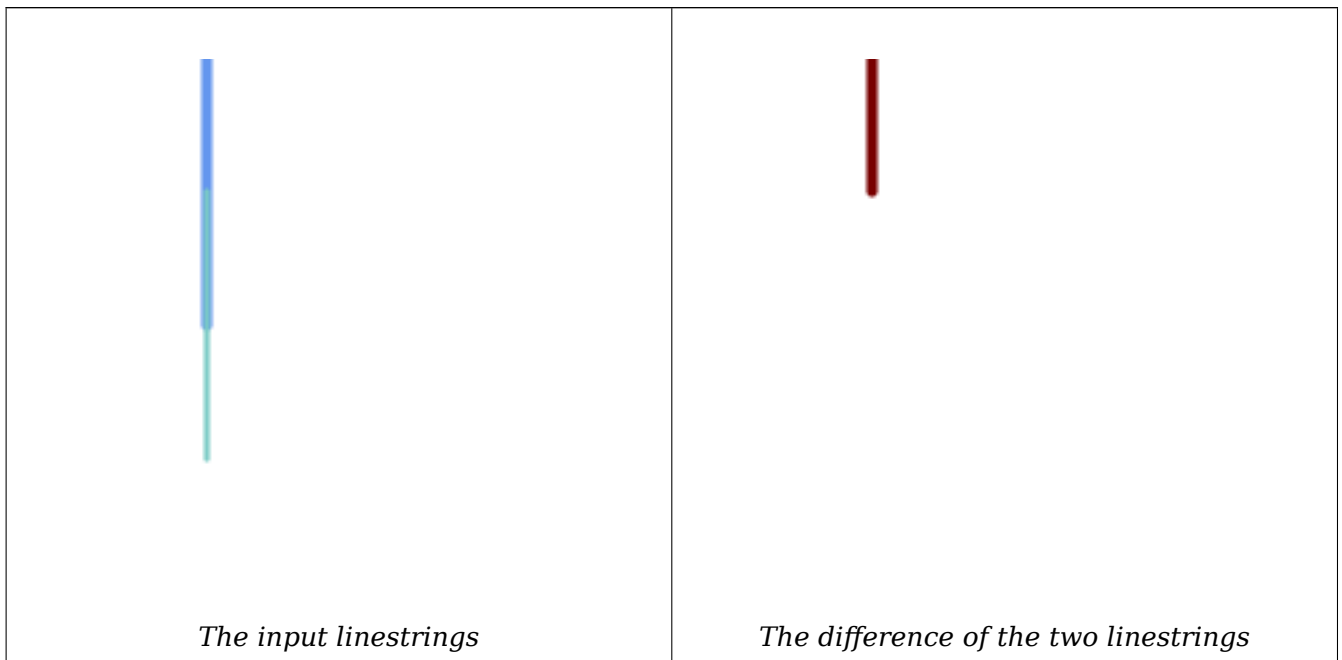


This method implements the SQL/MM specification. SQL-MM 3: 5.1.20



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

☒☒



The difference of 2D linestrings.

```
SELECT ST_AsText(
 ST_Difference(
 'LINESTRING(50 100, 50 200)::geometry',
 'LINESTRING(50 50, 50 150)::geometry'
)
);
```

```
st_astext

LINESTRING(50 150,50 200)
```

The difference of 3D points.

```
SELECT ST_AsEWKT(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
));
```

```
st_asewkt

MULTIPOINT(-118.6 38.329 6, -118.58 38.38 5)
```

☒☒

[ST\\_SymDifference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 7.13.3 ST\_Intersection

`ST_Intersection` — Computes a geometry representing the shared portion of geometries A and B.

## Synopsis

```
geometry ST_Intersection(geometry geomA , geometry geomB , float8 gridSize = -1);
geography ST_Intersection(geography geogA , geography geogB);
```



Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

`ST_Intersection` in conjunction with `ST_Intersects` is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

---

### Note



For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.



### Warning

This function will drop the M coordinate values if present.



### Warning

If working with 3D geometries, you may want to use SFGCAL based `ST_3DIntersection` which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

---

GEOS 

Enhanced: 3.1.0 accept a `gridSize` parameter

Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.5 support for geography data type was introduced.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)



This method implements the SQL/MM specification. SQL-MM 3: 5.1.18



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

---

☒☒

```

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2)':: ←
 geometry));
 st_astext

GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2)':: ←
 geometry));
 st_astext

POINT(0 0)

```

Clip all lines (trails) by country. Here we assume country geom are POLYGON or MULTIPOLYGONS. NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING because we don't care about trails that just share a point. The dump is needed to expand a geometry collection into individual single MULT\* parts. The below is fairly generic and will work for polys, etc. by just changing the where clause.

```

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;

```

For polys e.g. polygon landmarks, you can also use the sometimes faster hack that buffering anything by 0.0 except a polygon results in an empty geometry collection. (So a geometry collection containing polys, lines and points buffered by 0.0 would only leave the polygons and dissolve the collection shell.)

```

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));

```

## Examples: 2.5Dish

Note this is not a true intersection, compare to the same example using [ST\\_3DIntersection](#).

```

select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS ←
 linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

 st_astext

LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)

```

☒☒

[ST\\_3DIntersection](#), [ST\\_Difference](#), [ST\\_Union](#), [ST\\_Dimension](#), [ST\\_Dump](#), [ST\\_Force2D](#), [ST\\_SymDifference](#), [ST\\_Intersects](#), [ST\\_Multi](#)

### 7.13.4 ST\_MemUnion

`ST_MemUnion` — Aggregate function which unions geometries in a memory-efficient but slower way

#### Synopsis

geometry **ST\_MemUnion**(geometry set geomfield);

☒☒

An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.



#### Note

Produces the same result as [ST\\_Union](#), but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the `ST_Union` aggregate which first accumulates an array and then unions the contents using a fast algorithm.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

☒☒

```
SELECT id,
 ST_MemUnion(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

☒☒

[ST\\_Union](#)

### 7.13.5 ST\_Node

`ST_Node` — Nodes a collection of lines.

#### Synopsis

geometry **ST\_Node**(geometry geom);

☒☒

Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to [ST\\_Polygonize](#).

[ST\\_UnaryUnion](#) can also be used to node and dissolve linework. It provides an option to specify a `gridSize`, which can provide simpler and more robust output. See also [ST\\_Union](#) for an aggregate variant.



This function supports 3d and will not drop the z-index.

GEOS ☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Changed: 2.4.0 this function uses `GEOSNode` internally instead of `GEOSUnaryUnion`. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4.

☒☒

Noding a 3D LineString which self-intersects

```
SELECT ST_AsText(
 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))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText(
 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))
```

☒☒

[ST\\_UnaryUnion](#), [ST\\_AsBinary](#)

### 7.13.6 ST\_Split

`ST_Split` — Returns a collection of geometries created by splitting a geometry by another geometry.

#### Synopsis

geometry **ST\_Split**(geometry input, geometry blade);

☒☒

The function supports splitting a `LineString` by a `(Multi)Point`, `(Multi)LineString` or `(Multi)Polygon` boundary, or a `(Multi)Polygon` by a `LineString`. When a `(Multi)Polygon` is used as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of `ST_Union`. Applying `ST_Union` to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).



#### Note

If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using `ST_Snap` with a small tolerance.

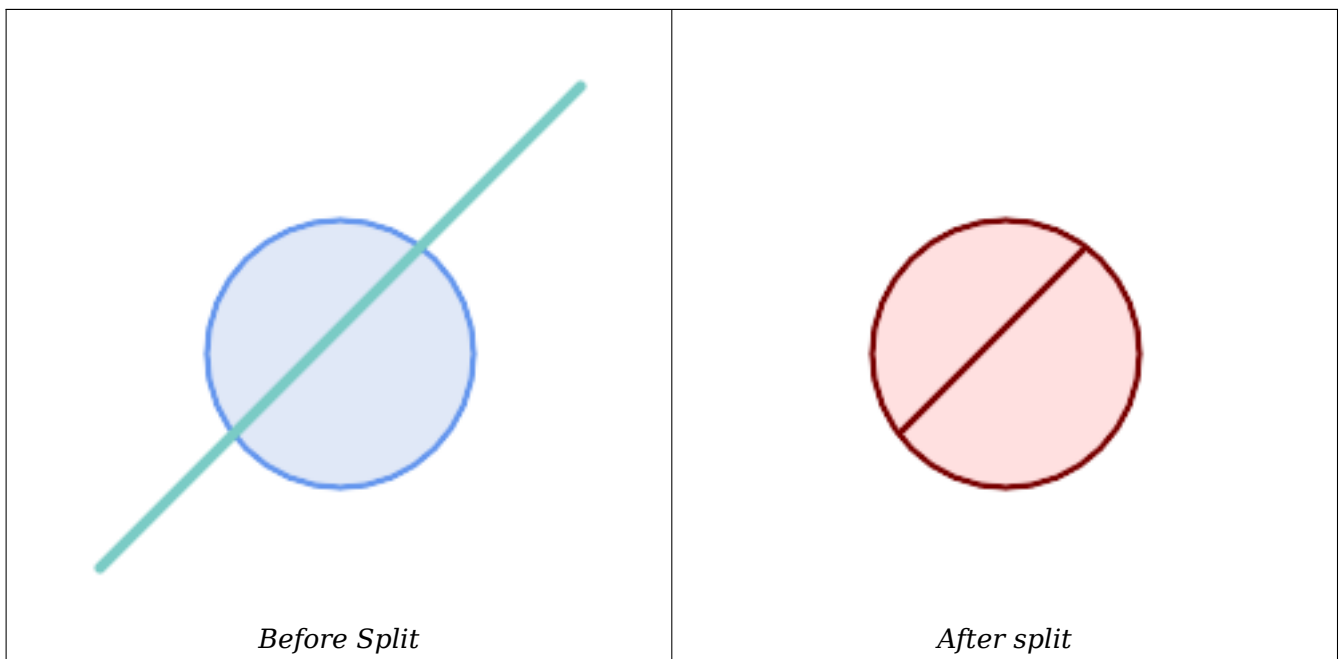
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.

☒☒

Split a Polygon by a Line.



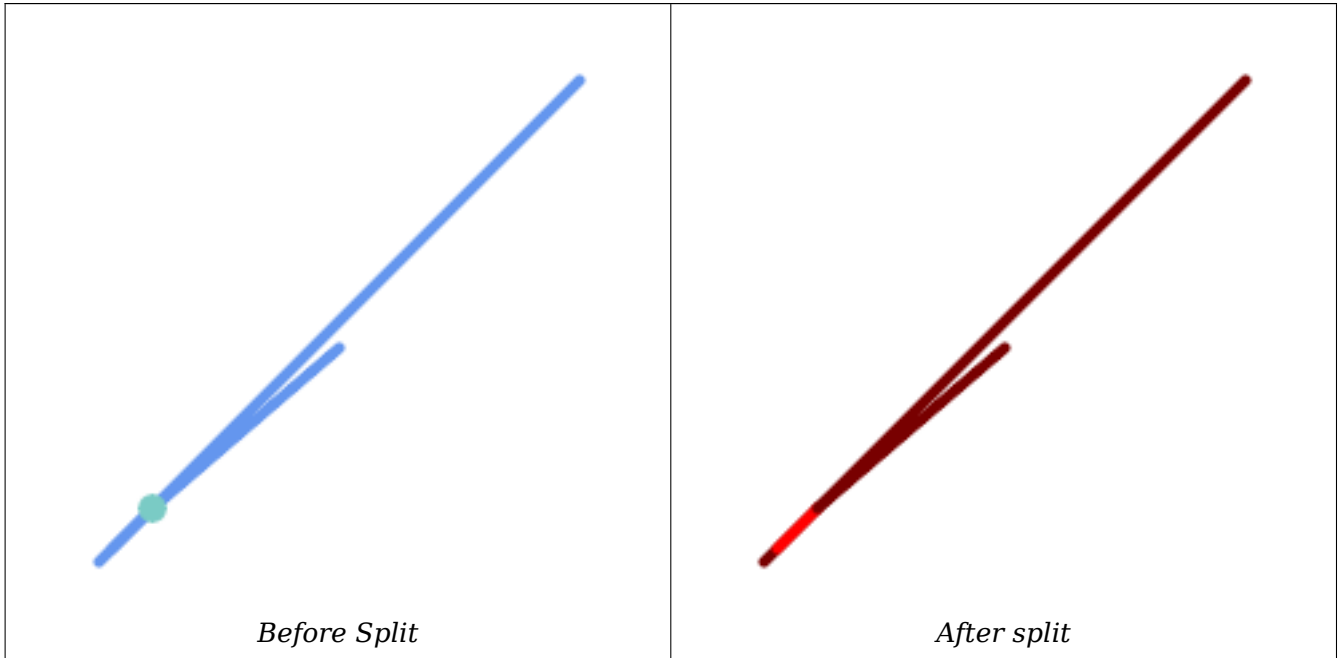
```
SELECT ST_AsText(ST_Split(
 ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
 ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
));

-- result --
GEOMETRYCOLLECTION(
```



```
POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564 ←
 70.8658283817455,..),
POLYGON(..)
)
```

Split a MultiLineString by a Point, where the point lies exactly on both LineStrings elements.



```
SELECT ST_AsText(ST_Split(
 'MULTILINESTRING((10 10, 190 190)', (15 15, 30 30, 100 90))',
 ST_Point(30,30)) As split;

split

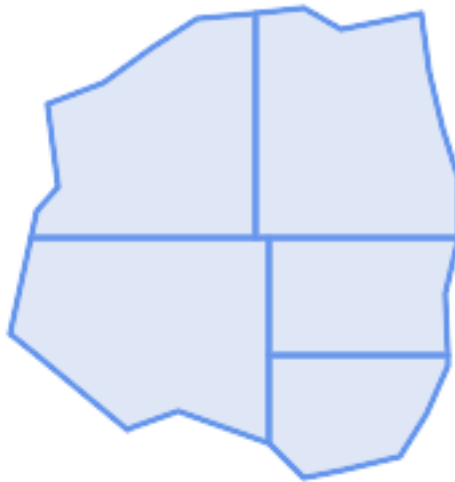
GEOMETRYCOLLECTION(
 LINESTRING(10 10,30 30),
 LINESTRING(30 30,190 190),
 LINESTRING(15 15,30 30),
 LINESTRING(30 30,100 90)
)
```

Split a LineString by a Point, where the point does not lie exactly on the line. Shows using **ST\_Snap** to snap the line to the point to allow it to be split.

```
WITH data AS (SELECT
 'LINESTRING(0 0, 100 100)::geometry AS line,
 'POINT(51 50):: geometry AS point
)
SELECT ST_AsText(ST_Split(ST_Snap(line, point, 1), point)) AS snapped_split,
 ST_AsText(ST_Split(line, point)) AS not_snapped_not_split
FROM data;

 snapped_split | not_snapped_not_split |
-----+-----
GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION(←
 LINESTRING(0 0,100 100))
```



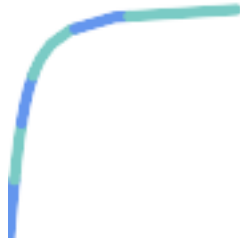


Subdivided to maximum 10 vertices

```
SELECT row_number() OVER() As rn, ST_AsText(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);
```

rn	wkt
1	POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 23))
2	POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))
3	POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))
4	POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114 184,114 100,29.8260869565217 100))
5	POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114 100,114 184))

**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.



*The densified and split lines.*

```
SELECT ST_AsText(ST_Subdivide(
 ST_Segmentize('LINESTRING(0 0, 85 85)::geography,
 1200000)::geometry, 8));
```

```
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)
```

**Example:** Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
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;
```

**Example:** Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since `ST_Subdivide` is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
```

```
SELECT pkey, ST_Subdivide(geom) AS geom
FROM original_geoms;
```



[ST\\_ClipByBox2D](#), [ST\\_Segmentize](#), [ST\\_Split](#), [ST\\_NPoints](#)

### 7.13.8 ST\_SymDifference

`ST_SymDifference` — Computes a geometry representing the portions of geometries A and B that do not intersect.

#### Synopsis

geometry **ST\_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);



Returns a geometry representing the portions of geometries A and B that do not intersect. This is equivalent to `ST_Union(A,B) - ST_Intersection(A,B)`. It is called a symmetric difference because `ST_SymDifference(A,B) = ST_SymDifference(B,A)`.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)


GEOS 

Enhanced: 3.1.0 accept a `gridSize` parameter.

Requires GEOS  $\geq$  3.9.0 to use the `gridSize` parameter

 This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)

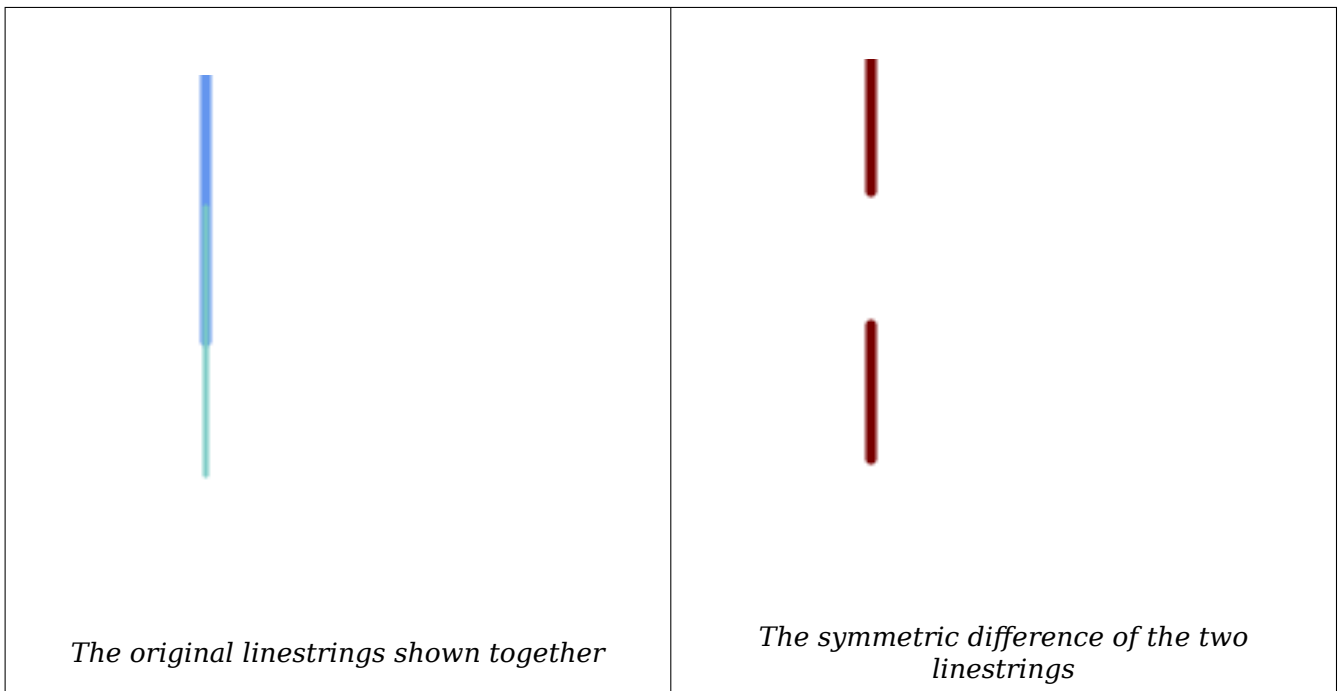
 This method implements the SQL/MM specification. SQL-MM 3: 5.1.21

 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.



---

---



```
--Safe for 2d - symmetric difference of 2 linestrings
SELECT ST_AsText(
 ST_SymDifference(
 ST_GeomFromText('LINESTRING(50 100, 50 200)'),
 ST_GeomFromText('LINESTRING(50 50, 50 150)')
)
);
```

```
st_astext

MULTILINESTRING((50 150,50 200),(50 50,50 100))
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_SymDifference(ST_GeomFromEWKT('LINESTRING(1 2 1, 1 4 2)'),
 ST_GeomFromEWKT('LINESTRING(1 1 3, 1 3 4)')))
```

```
st_astext

MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))
```

☒☒

[ST\\_Difference](#), [ST\\_Intersection](#), [ST\\_Union](#)

### 7.13.9 ST\_UnaryUnion

`ST_UnaryUnion` — Computes the union of the components of a single geometry.

#### Synopsis

```
geometry ST_UnaryUnion(geometry geom, float8 gridSize = -1);
```

☒☒

A single-input variant of **ST\_Union**. The input may be a single geometry, a MultiGeometry, or a GeometryCollection. The union is applied to the individual elements of the input.

This function can be used to fix MultiPolygons which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use **ST\_MakeValid**.

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them **simple**. (**ST\_Node** also does this, but it does not provide the gridSize option.)

It is possible to combine ST\_UnaryUnion with **ST\_Collect** to fine-tune how many geometries are be unioned at once. This allows trading off between memory usage and compute time, striking a balance between ST\_Union and **ST\_MemUnion**.

If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

**ST\_Union**, **ST\_MemUnion**, **ST\_MakeValid**, **ST\_Collect**, **ST\_Node**

### 7.13.10 ST\_Union

**ST\_Union** — Computes a geometry representing the point-set union of the input geometries.

#### 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);
```

☒☒

Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants:

**Two-input variant:** returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned.

**Array variant:** returns a geometry that is the union of an array of geometries.

**Aggregate variant:** returns a geometry that is the union of a rowset of geometries. The ST\_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on

rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

See [ST\\_UnaryUnion](#) for a non-aggregate, single-input variant.

The ST\_Union array and set variants use the fast Cascaded Union algorithm described in <http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html>

A gridSize can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)



#### Note

[ST\\_Collect](#) may sometimes be used in place of ST\_Union, if the result is not required to be non-overlapping. ST\_Collect is usually faster than ST\_Union because it performs no processing on the collected geometries.

GEOS ☒☒☒☒☒

ST\_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use [ST\\_LineMerge](#) to sew LineStrings.

NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.4.0 - ST\_Union was enhanced. ST\_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)



#### Note

Aggregate version is not explicitly defined in OGC SPEC.



This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.



This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

☒☒

Aggregate example

```
SELECT id,
 ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Non-Aggregate example



```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

st_astext

MULTIPOINT(-2 3,1 2)

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext

POINT(1 2)
```

### 3D example - sort of supports 3D (and with mixed dimensions!)

```
select ST_AsEWKT(ST_Union(geom))
from (
 select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::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;

st_asewkt

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)));
```

### 3d example not mixing dimensions

```
select ST_AsEWKT(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;

st_asewkt

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)));

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
 ST_GeomFromText('LINESTRING(3 4, 4 5)']))) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))
```

☒☒

[ST\\_Collect](#), [ST\\_UnaryUnion](#), [ST\\_MemUnion](#), [ST\\_Intersection](#), [ST\\_Difference](#), [ST\\_SymDifference](#)

## 7.14 ☒☒☒☒☒☒

### 7.14.1 ST\_Buffer

`ST_Buffer` — Computes a geometry covering all points within a given distance from a geometry.

#### 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);
```

☒☒

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specified by providing a list of blank-separated key=value pairs as follows:

- `'quad_segs=#'` : number of line segments used to approximate a quarter circle (default is 8).
- `'endcap=round|flat|square'` : endcap style (defaults to "round"). `'butt'` is accepted as a synonym for `'flat'`.
- `'join=round|mitre|bevel'` : join style (defaults to "round"). `'miter'` is accepted as a synonym for `'mitre'`.
- `'mitre_limit=#.#'` : mitre ratio limit (only affects mitered join style). `'miter_limit'` is accepted as a synonym for `'mitre_limit'`.
- `'side=both|left|right'` : `'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.

---

#### Note



For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline

---

**Note**

Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. [ST\\_MakeValid](#) can also be used for this purpose.

**Note**

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use [ST\\_DWithin](#).

**Note**

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Enhanced: 2.5.0 - [ST\\_Buffer](#) geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

Availability: 1.5 - [ST\\_Buffer](#) was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

GEOS    

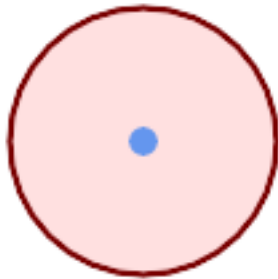


This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3](#)



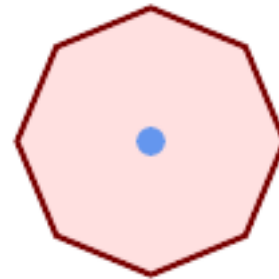
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30



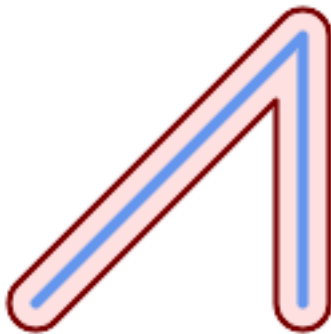
*quad\_segs=8* (☒☒☒)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=8');
```



*quad\_segs=2* (☒☒)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2');
```



*endcap=round join=round* (☒☒☒)

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'endcap=round join=round');
```



*endcap=square*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'endcap=square join=round');
```



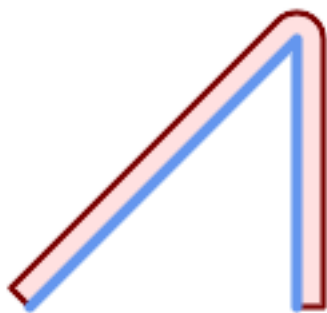
*join=bevel*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'join=bevel');
```



*join=mitre mitre\_limit=5.0* (☒☒☒☒☒☒☒☒)

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'join=mitre mitre_limit=5.0');
```



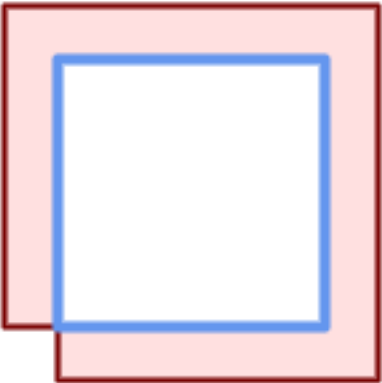
*side=left*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'side=left');
```



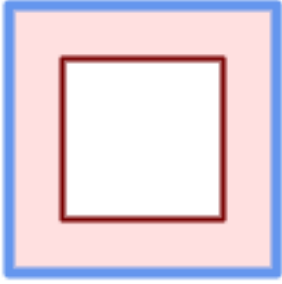
*side=right*

```
SELECT ST_Buffer(
 ST_GeomFromText(
 'LINESTRING(50 50,150 150,150 50)'
), 10, 'side=right');
```



*right-hand-winding, polygon boundary  
side=left*

```
SELECT ST_Buffer(
ST_ForceRHR(
ST_Boundary(
 ST_GeomFromText(
'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'),
), 20, 'side=left');
```



*right-hand-winding, polygon boundary  
side=right*

```
SELECT ST_Buffer(
ST_ForceRHR(
ST_Boundary(
 ST_GeomFromText(
'POLYGON ((50 50, 50 150, 150 150, 150 50, 50 50))'),
), 20, 'side=right');
```

```
--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)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
-----+-----
 33 | 9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- Note coordinates are in NAD 83 long lat which we transform
to Mass state plane meter and then buffer to get measurements in meters;
SELECT ST_AsText(ST_Buffer(
ST_Transform(
ST_SetSRID(ST_Point(-71.063526, 42.35785),4269), 26986)
,100,2)) As octagon;

POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```







☒☒

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTS, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGS, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONS, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use [ST\\_PointOnSurface](#).

New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

Availability: 2.4.0 support for geography was introduced.



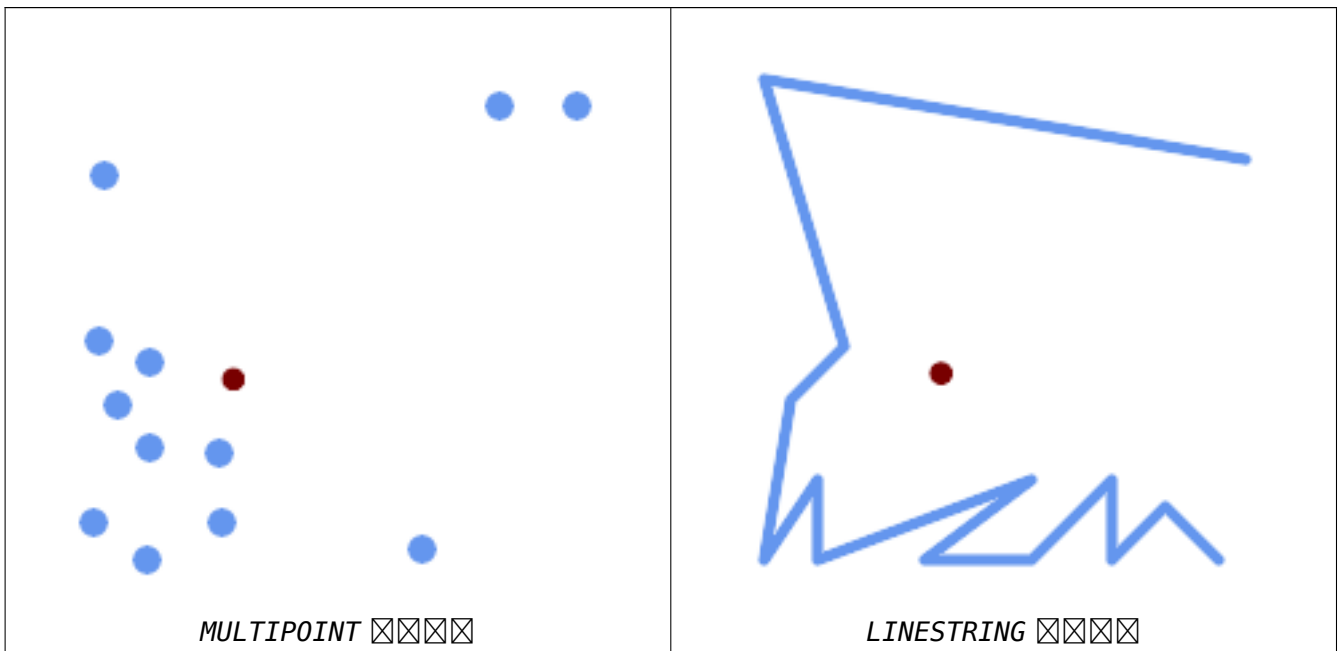
This method implements the [OGC Simple Features Implementation Specification for SQL 1.1](#).

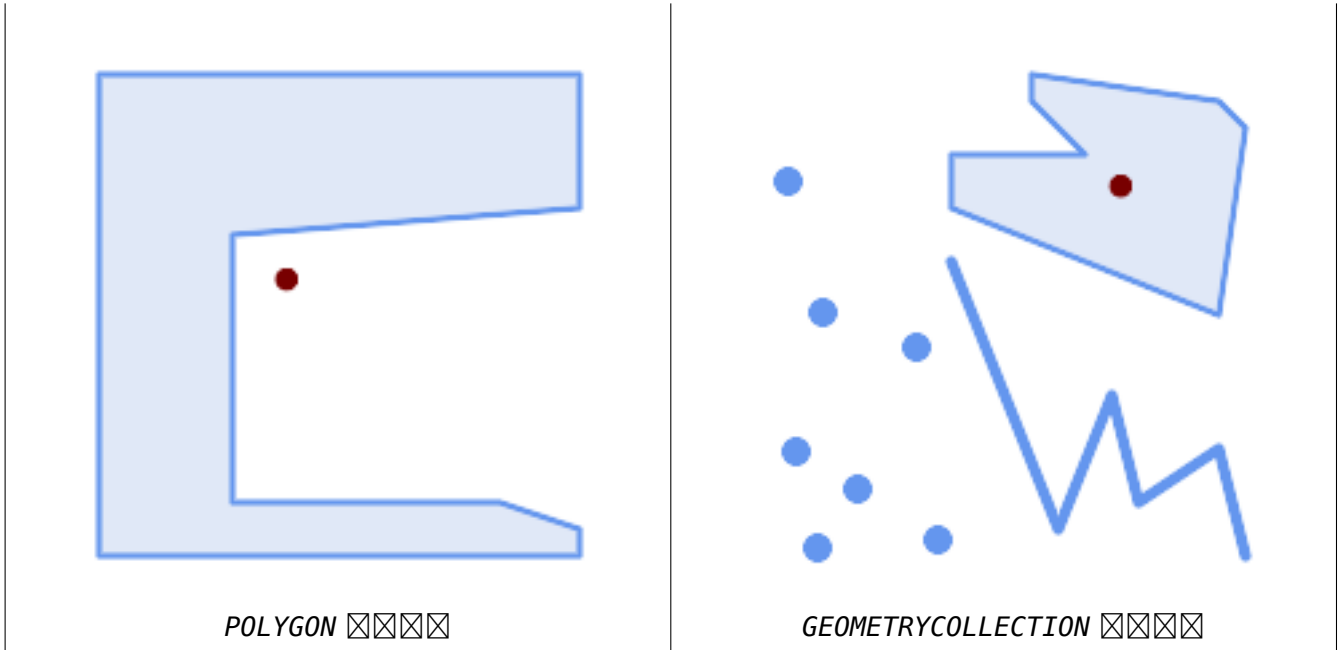


This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

☒☒

In the following illustrations the red dot is the centroid of the source geometry.





```

SELECT ST_AsText(ST_Centroid('MULTIPOINT (-1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2 0, 6 0, 7 8, 9 8, 10 6)'));
 st_astext

POINT(2.30769230769231 3.30769230769231)
(1 row)

SELECT ST_AsText(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 ;

POINT(0.5 1)

SELECT ST_AsText(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;

POINT(0.5 1)

```

☒☒

[ST\\_PointOnSurface](#), [ST\\_GeometricMedian](#)

### 7.14.4 ST\_ChaikinSmoothing

`ST_ChaikinSmoothing` — Returns a smoothed version of a geometry, using the Chaikin algorithm

#### Synopsis

`geometry` **ST\_ChaikinSmoothing**(`geometry geom`, `integer nIterations = 1`, `boolean preserveEndpoints = false`);

☒☒

Smooths a linear or polygonal geometry using **Chaikin's algorithm**. The degree of smoothing is controlled by the `nIterations` parameter. On each iteration, each interior vertex is replaced by two vertices located at 1/4 of the length of the line segments before and after the vertex. A reasonable degree of smoothing is provided by 3 iterations; the maximum is limited to 5.

If `preserveEndpoints` is true, the endpoints of Polygon rings are not smoothed. The endpoints of LineStrings are always preserved.



**Note**

The number of vertices doubles with each iteration, so the result geometry may have many more points than the input. To reduce the number of points use a simplification function on the result (see [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#) and [ST\\_SimplifyVW](#)).

The result has interpolated values for the Z and M dimensions when present.



This function supports 3d and will not drop the z-index.

Availability: 2.5.0

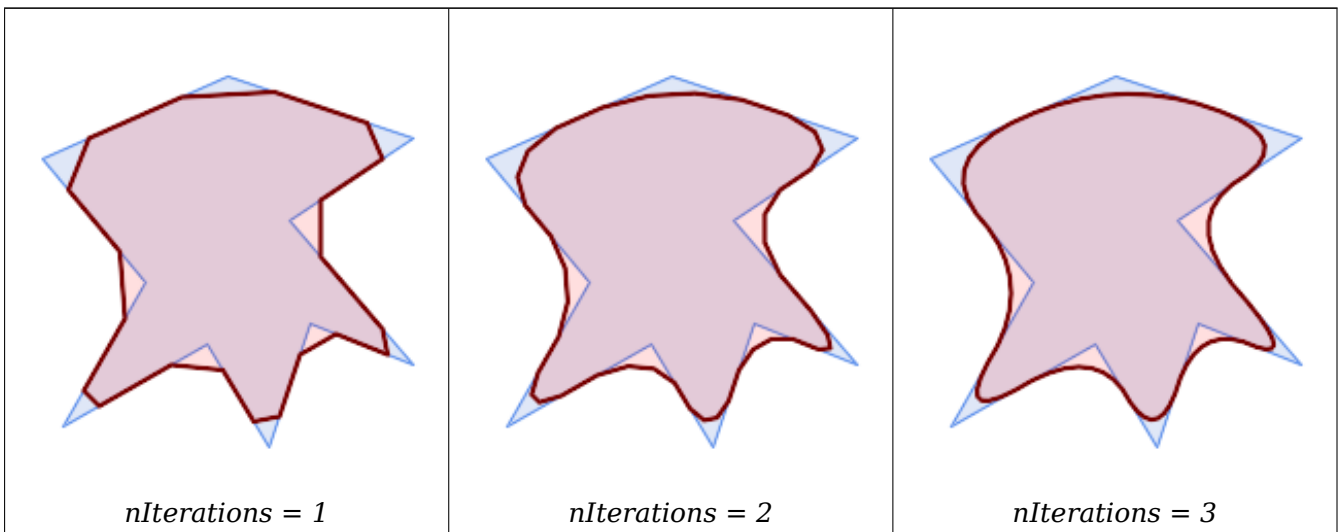
☒☒

Smoothing a triangle:

```
SELECT ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT 'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) AS foo;

 smoothed
b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''- ←
 b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
 '-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b'' ←
 '-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-b''b''-
POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))
```

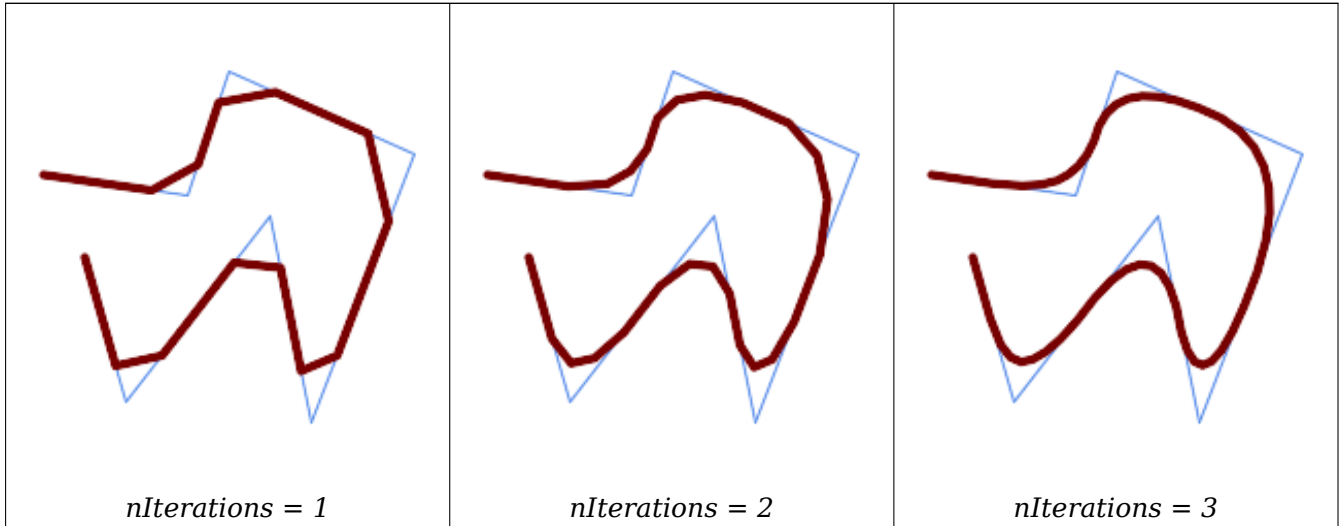
Smoothing a Polygon using 1, 2 and 3 iterations:



```
SELECT 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))',
generate_series(1, 3));
```

Smoothing a LineString using 1, 2 and 3 iterations:



```
SELECT ST_ChaikinSmoothing(
 'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100) ←
 ',
 generate_series(1, 3));
```

☒☒

[ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#)

### 7.14.5 ST\_ConcaveHull

`ST_ConcaveHull` — Computes a possibly concave geometry that contains all input geometry vertices

#### Synopsis

```
geometry ST_ConcaveHull(geometry param_geom, float param_pctconvex, boolean param_allow_holes
= false);
```

☒☒

A concave hull is a (usually) concave geometry which contains the input, and whose vertices are a subset of the input vertices. In the general case the concave hull is a Polygon. The concave hull of two or more collinear points is a two-point LineString. The concave hull of one or more identical points is a Point. The polygon will not contain holes unless the optional `param_allow_holes` argument is specified as true.

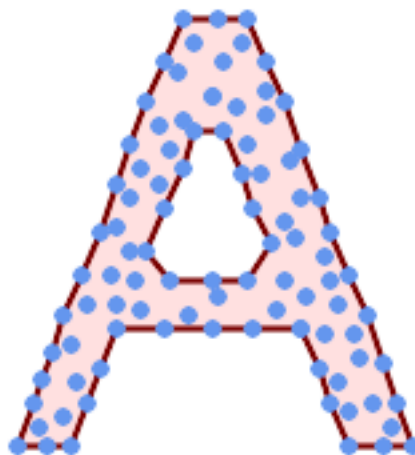
One can think of a concave hull as “shrink-wrapping” a set of points. This is different to the [convex hull](#), which is more like wrapping a rubber band around the points. A concave hull generally has a smaller area and represents a more natural boundary for the input points.



```

, (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));
---st_astext--
POLYGON ((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, 173 134, 161 134, 150 133, 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, 10 72, 21 73, 31 ←
74, 42 74, 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))

```



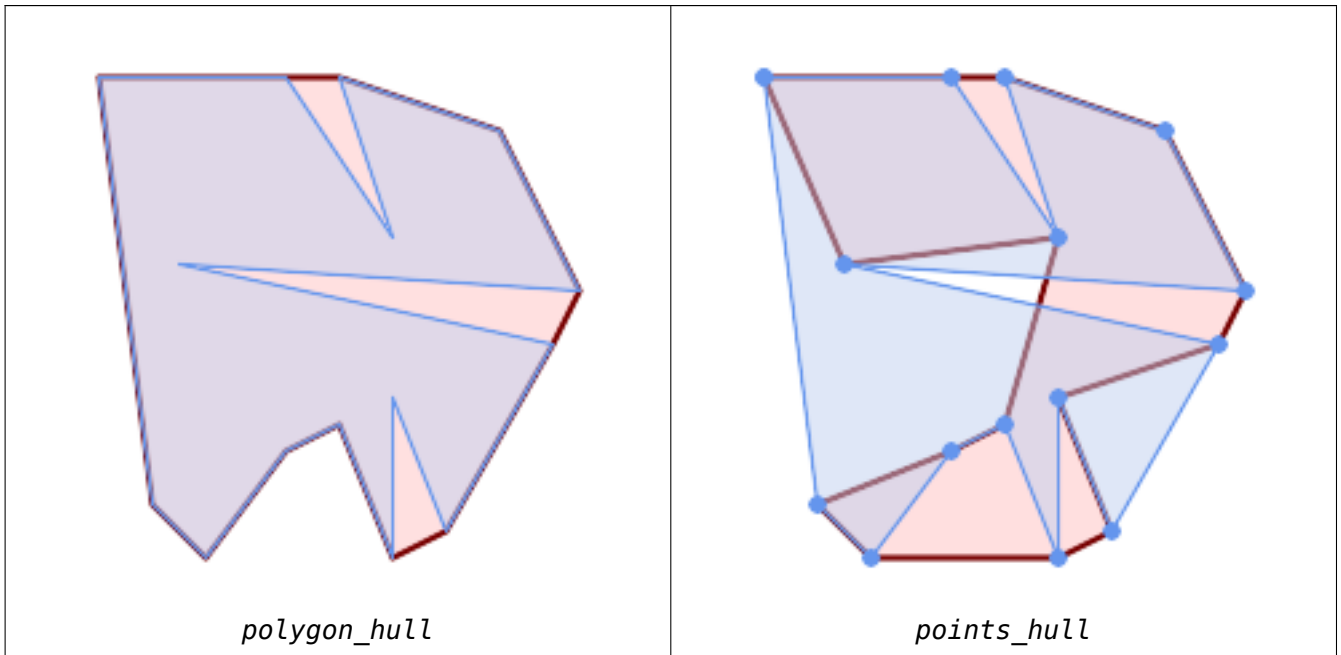
*Concave Hull of a MultiPoint, allowing holes*

```

SELECT ST_AsText(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));
---st_astext--
POLYGON ((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, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20, ←
37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ←
121 96, 114 109, 110 122, 103 138, 92 138, 88 124))
```



Comparing a concave hull of a Polygon to the concave hull of the constituent points. The hull respects the boundary of the polygon, whereas the points-based hull does not.

```
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;
```

Using with ST\_Collect to compute the concave hull of a geometry set.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
 ST_ConcaveHull(ST_Collect(obs_pnt), 0.3) AS geom
FROM disease_obs
GROUP BY disease_type;
```

☒☒

[ST\\_ConvexHull](#), [ST\\_Collect](#), [ST\\_AlphaShape](#), [ST\\_OptimalAlphaShape](#)

### 7.14.6 ST\_ConvexHull

ST\_ConvexHull — Computes the convex hull of a geometry.

#### Synopsis

geometry **ST\_ConvexHull**(geometry geomA);

☒☒

Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a **concave hull** which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations.

In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use **ST\_Collect** to aggregate them into a geometry collection (e.g. `ST_ConvexHull(ST_Collect(geom))`).

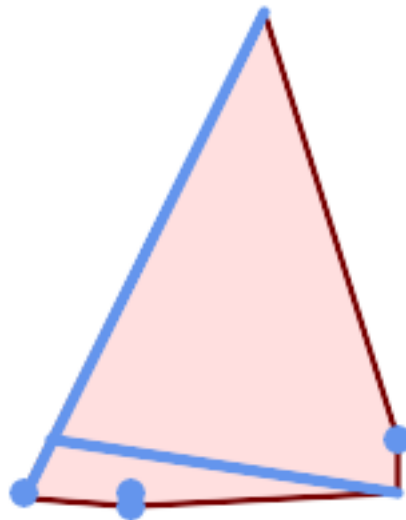
GEOS ☒☒☒☒☒

✔ This method implements the **OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3**

✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16

✔ This function supports 3d and will not drop the z-index.

☒☒



*Convex Hull of a MultiLineString and a MultiPoint*

```
SELECT ST_AsText(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)')
)));
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

Using with `ST_Collect` to compute the convex hulls of geometry sets.



```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
 ST_ConvexHull(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```

☒☒

[ST\\_Collect](#), [ST\\_ConcaveHull](#), [ST\\_MinimumBoundingCircle](#)

### 7.14.7 ST\_DelaunayTriangles

ST\_DelaunayTriangles — Returns the Delaunay triangulation of the vertices of a geometry.

#### Synopsis

geometry **ST\_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

☒☒

Computes the [Delaunay triangulation](#) of the vertices of the input geometry. The optional tolerance can be used to snap nearby input vertices together, which improves robustness in some situations. The result geometry is bounded by the convex hull of the input vertices. The result geometry representation is determined by the flags code:

- 0 - a GEOMETRYCOLLECTION of triangular POLYGONS (default)
- 1 - a MULTILINESTRING of the edges of the triangulation
- 2 - A TIN of the triangulation

GEOS ☒☒☒☒☒

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

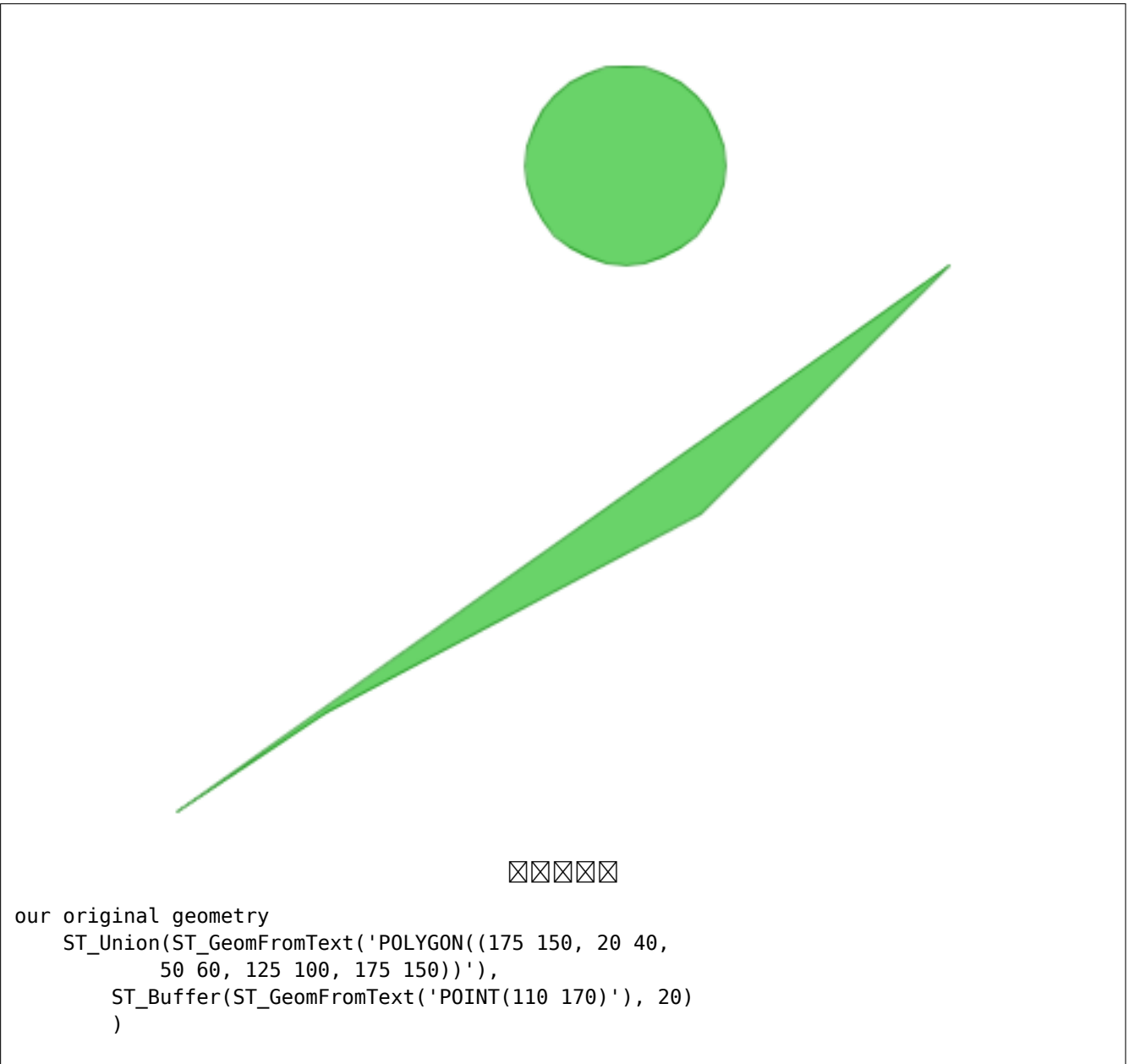
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

---

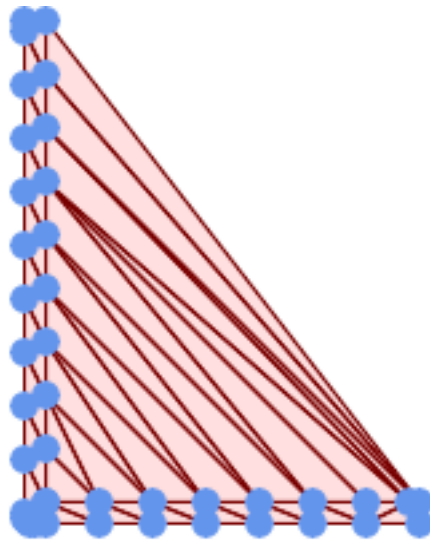


---









```
-- ☒☒☒☒☒☒☒ 55 ☒☒☒☒☒☒ 45 ☒☒☒☒☒☒☒☒☒☒
```

this produces a table of 42 points that form an L shape

```
SELECT (ST_DumpPoints(ST_GeomFromText(
'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)'))).geom
 INTO TABLE l_shape;
```

output as individual polygon triangles

```
SELECT ST_AsText((ST_Dump(geom)).geom) As wkt
FROM (SELECT ST_DelaunayTriangles(ST_Collect(geom)) As geom
FROM l_shape) As foo;
```

wkt

```
POLYGON((6 194,6 190,14 194,6 194))
POLYGON((14 194,6 190,14 174,14 194))
POLYGON((14 194,14 174,154 14,14 194))
POLYGON((154 14,14 174,14 154,154 14))
POLYGON((154 14,14 154,150 14,154 14))
POLYGON((154 14,150 14,154 6,154 14))
```

Example using vertices with Z values.

3D multipoint

```
SELECT ST_AsText(ST_DelaunayTriangles(ST_GeomFromText(
'MULTIPOINT Z(14 14 10, 150 14 100,34 6 25, 20 10 150)')))) As wkt;
```

wkt

```
GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))
,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))
```

☒☒

[ST\\_VoronoiPolygons](#), [ST\\_TriangulatePolygon](#), [ST\\_ConstrainedDelaunayTriangles](#), [ST\\_VoronoiLines](#), [ST\\_Con](#)

### 7.14.8 ST\_FilterByM

ST\_FilterByM — Removes vertices based on their M value

#### Synopsis

geometry **ST\_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

☒☒

Filters out vertex points based on their M-value. Returns a geometry with only vertex points that have a M-value larger or equal to the min value and smaller or equal to the max value. If max-value argument is left out only min value is considered. If fourth argument is left out the m-value will not be in the resulting geometry. If resulting geometry have too few vertex points left for its geometry type an empty geometry will be returned. In a geometry collection geometries without enough points will just be left out silently.

This function is mainly intended to be used in conjunction with ST\_SetEffectiveArea. ST\_EffectiveArea sets the effective area of a vertex in its m-value. With ST\_FilterByM it then is possible to get a simplified version of the geometry without any calculations, just by filtering



#### Note

There is a difference in what ST\_SimplifyVW returns when not enough points meet the criteria compared to ST\_FilterByM. ST\_SimplifyVW returns the geometry with enough points while ST\_FilterByM returns an empty geometry



#### Note

Note that the returned geometry might be invalid



#### Note

This function returns all dimensions, including the Z and M values

Availability: 2.5.0

☒☒

A linestring is filtered

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry) geom ←
) As foo;
```

result

```
 simplified

LINESTRING(5 2,7 25,10 10)
```

☒☒

[ST\\_SetEffectiveArea](#), [ST\\_SimplifyVW](#)

### 7.14.9 ST\_GeneratePoints

`ST_GeneratePoints` — Generates a multipoint of random points contained in a Polygon or MultiPolygon.

#### Synopsis

geometry **ST\_GeneratePoints**(geometry g, integer npoints, integer seed = 0);

☒☒

`ST_GeneratePoints` generates a multipoint consisting of a given number of pseudo-random points which lie within the input area. The optional seed is used to regenerate a deterministic sequence of points, and must be greater than zero.

2.3.0 ☒☒☒☒☒☒☒☒☒☒.

Enhanced: 3.0.0, added seed parameter







---

☒☒

**ST\_Centroid**

### 7.14.11 ST\_LineMerge

ST\_LineMerge — Return the lines formed by sewing together a MultiLineString.

#### Synopsis

```
geometry ST_LineMerge(geometry amultilinestring);
geometry ST_LineMerge(geometry amultilinestring, boolean directed);
```

☒☒

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST\_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged



#### Note

Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

---

GEOS ☒☒☒☒☒

Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

1.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

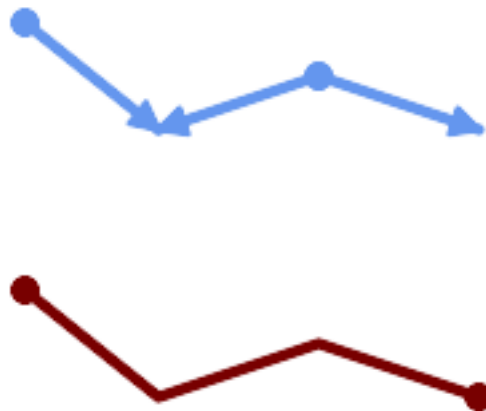


#### Warning

This function strips the M dimension.

---

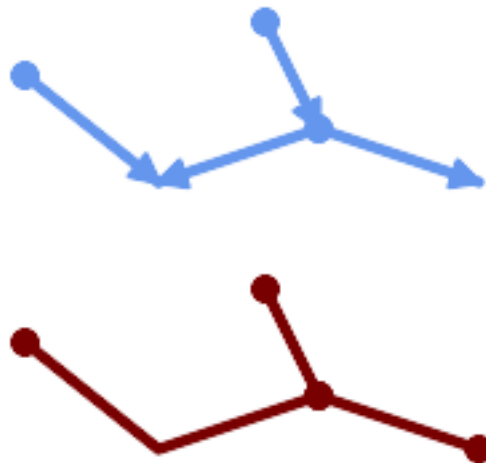
☒☒



*Merging lines with different orientation.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'
));

LINESTRING(10 160,60 120,120 140,180 120)
```



*Lines are not merged across intersections with degree > 2.*

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120 140))'
));

MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

If merging is not possible due to non-touching lines, the original MultiLineString is returned.



☒☒

Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

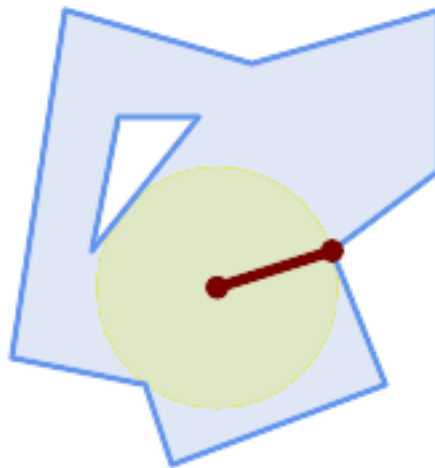
- center - center point of the circle
- nearest - a point on the geometry nearest to the center
- radius - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

Availability: 3.1.0.

Requires GEOS >= 3.9.0.

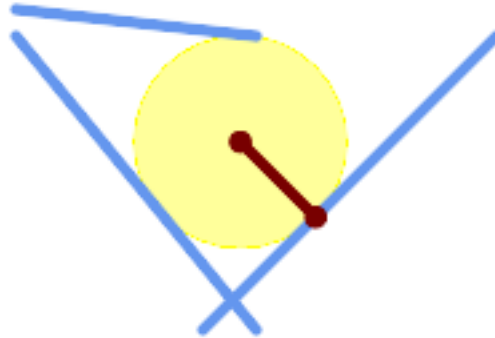
☒☒



*Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.*

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
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))');
```

radius	center	nearest
45.165845650018	POINT(96.953125 76.328125)	POINT(140 90)



*Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.*

☒☒

[ST\\_MinimumBoundingRadius](#), [ST\\_LargestEmptyCircle](#)

### 7.14.13 ST\_LargestEmptyCircle

`ST_LargestEmptyCircle` — Computes the largest circle not overlapping a geometry.

#### Synopsis

(geometry, geometry, double precision) **ST\_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

☒☒

Finds the largest circle which does not overlap a set of point and line obstacles. (Polygonal geometries may be included as obstacles, but only their boundary lines are used.) The center of the circle is constrained to lie inside a polygonal boundary, which by default is the convex hull of the input geometry. The circle center is the point in the interior of the boundary which has the farthest distance from the obstacles. The circle itself is provided by the center point and a nearest point lying on an obstacle determining the circle radius.

The circle center is determined to a given accuracy specified by a distance tolerance, using an iterative algorithm. If the accuracy distance is not specified a reasonable default is used.

Returns a record with fields:

- center - center point of the circle
- nearest - a point on the geometry nearest to the center
- radius - radius of the circle

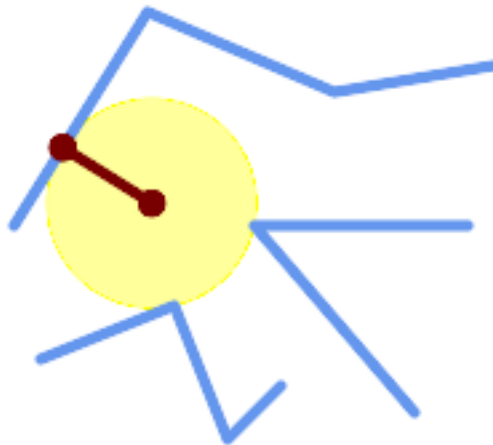
To find the largest empty circle in the interior of a polygon, see [ST\\_MaximumInscribedCircle](#).

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

☒☒

```
SELECT radius,
 center,
 nearest
FROM ST_LargestEmptyCircle(
 'MULTILINESTRING (
 (10 100, 60 180, 130 150, 190 160),
 (20 50, 70 70, 90 20, 110 40),
 (160 30, 100 100, 180 100))');
```



*Largest Empty Circle within a set of lines.*

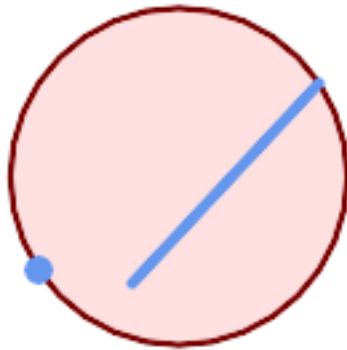
```
SELECT radius,
 center,
 nearest
FROM ST_LargestEmptyCircle(
 ST_Collect(
 'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))'::geometry,
 'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry) ←
 ,
 'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'::geometry
);
```





☒☒

```
SELECT d.disease_type,
 ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
FROM disease_obs As d
GROUP BY d.disease_type;
```



☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ 8 ☒☒☒☒☒☒☒☒.

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
 ST_Collect(
 ST_GeomFromText('LINESTRING(55 75,125 150)'),
 ST_Point(20, 80)), 8
)) As wktmbc;
wktmbc

POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ←
90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ←
70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ←
56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ←
51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ←
56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ←
70.383579256063,20.0366393799278 79.9451031602109,14.20583703063 ←
90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019 ←
127.309642789079,14.2058370306299 139.1462329091,20.0366393799275 ←
150.054896839789,27.883579256063 159.616420743937,
37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921 ←
176.884753327498,
72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003 ←
173.29416296937,107.554896839789 167.463360620072,
117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937 ←
139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))
```

☒☒

**ST\_Collect, ST\_MinimumBoundingRadius, ST\_LargestEmptyCircle, ST\_LongestLine**

### 7.14.15 ST\_MinimumBoundingRadius

`ST_MinimumBoundingRadius` — Returns the center point and radius of the smallest circle that contains a geometry.

#### Synopsis

(geometry, double precision) **ST\_MinimumBoundingRadius**(geometry geom);

☒☒

Computes the center point and radius of the smallest circle that contains a geometry. Returns a record with fields:

- center - center point of the circle
- radius - radius of the circle

Use with [ST\\_Collect](#) to get the minimum bounding circle of a set of geometries.

To compute two points lying on the minimum circle (the "maximum diameter") use [ST\\_LongestLine](#).

2.3.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 65242,26075 65136,26096 65427,26426 65078))');
```

st_astext	radius
POINT(26284.8418027133 65267.1145090825)	247.436045591407

☒☒

[ST\\_Collect](#), [ST\\_MinimumBoundingCircle](#), [ST\\_LongestLine](#)

### 7.14.16 ST\_OrientedEnvelope

`ST_OrientedEnvelope` — Returns a minimum-area rectangle containing a geometry.

#### Synopsis

geometry **ST\_OrientedEnvelope**( geometry geom );

☒☒

Returns the minimum-area rotated rectangle enclosing a geometry. Note that more than one such rectangle may exist. May return a Point or LineString in the case of degenerate inputs.

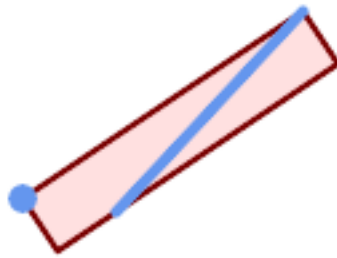
Availability: 2.5.0.

Requires GEOS >= 3.6.0.

☒☒

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));
 st_astext

POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))
```



*Oriented envelope of a point and linestring.*

```
SELECT ST_AsText(ST_OrientedEnvelope(
 ST_Collect(
 ST_GeomFromText('LINESTRING(55 75,125 150)'),
 ST_Point(20, 80))
) As wktenv;
wktenv

POLYGON((19.9999999999997 79.9999999999999,33.0769230769229 ↔
 60.3846153846152,138.076923076924 130.384615384616,125.000000000001 ↔
 150.000000000001,19.9999999999997 79.9999999999999))
```

☒☒

**ST\_Envelope** **ST\_MinimumBoundingCircle**

### 7.14.17 ST\_OffsetCurve

**ST\_OffsetCurve** — Returns an offset line at a given distance and side from an input line.

#### Synopsis

geometry **ST\_OffsetCurve**(geometry line, float signed\_distance, text style\_parameters=“);

ST\_OffsetCurve

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

ST\_OffsetCurve(geometry, float, text)

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

ST\_OffsetCurve(geometry, float, text) = ST\_OffsetCurve(geometry, float, text, text):

- 'quad\_segs=#' : # (quarter circle) (default 8)
- 'join=round|mitre|bevel' : 'round' (round), 'mitre' (miter), 'bevel' (bevel)
- 'mitre limit=#.#' : #.# (mitre limit) (default 2.0). 'mitre\_limit' 'miter\_limit'

GEOS 3.11

Behavior changed in GEOS 3.11 so offset curves now have the same direction as the input line, for both positive and negative offsets.

2.0

Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING




**Note**

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

ST\_Union

ST\_Union(geometry, geometry)

```
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;
```




15, 'quad\_segs=4 join=round'  
15

```
SELECT ST_AsText(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 1,12.2597485145237
 2.1418070123307,
 7.39339828220179 5.39339828220179,
 5.39339828220179 7.39339828220179,
 2.14180701233067 12.2597485145237,1
 18,1 195)
```

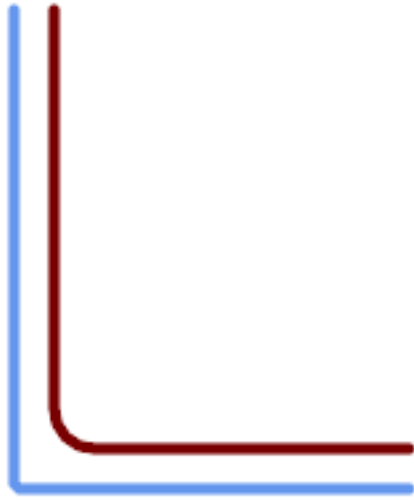


-15, 'quad\_segs=4  
join=round' -15

```
SELECT ST_AsText(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;
```

notsocurvy

```
LINESTRING(31 195,31 31,164 31)
```

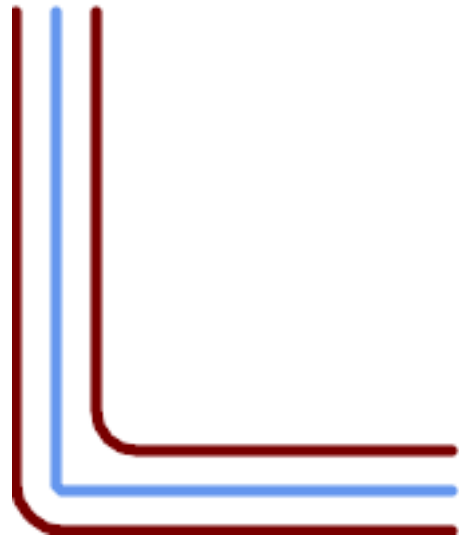


ST\_OffsetCurve(ST\_OffsetCurve(ST\_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16,44 16,24 16,20 16,18 16,17 17,16 18,16 20,16 40,16 60,16 80,16 100,16 120,16 140,16 160,16 180,16 195)') As geom, -30, 'quad\_segs=4 join=round'), -15, 'quad\_segs=4 join=round')) As morecurvy

```
SELECT ST_AsText(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;
```

morecurvy

```
LINESTRING(164 31,46 31,40.2597485145236
32.1418070123307,
35.3933982822018 35.3933982822018,
32.1418070123307 40.2597485145237,31
46,31 195)
```



ST\_OffsetCurve(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)') As geom, -30, 'quad\_segs=4 join=round'), -15, 'quad\_segs=4 join=round')) As parallel\_curves

```
SELECT ST_AsText(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')
)
```

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;
```

parallel curves

```
MULTILINESTRING((164 1,18
1,12.2597485145237 2.1418070123307,
7.39339828220179
5.39339828220179,5.39339828220179 7.39339828220179,
2.14180701233067 12.2597485145237,1 18,1
195),
(164 31,46 31,40.2597485145236
32.1418070123307,35.3933982822018 35.3933982822018,
32.1418070123307 40.2597485145237,31
46,31 195))
```



☒☒

Returns a POINT which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVEPOLYGON). In PostGIS this function also works on line and point geometries.



This method implements the [OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2](#)

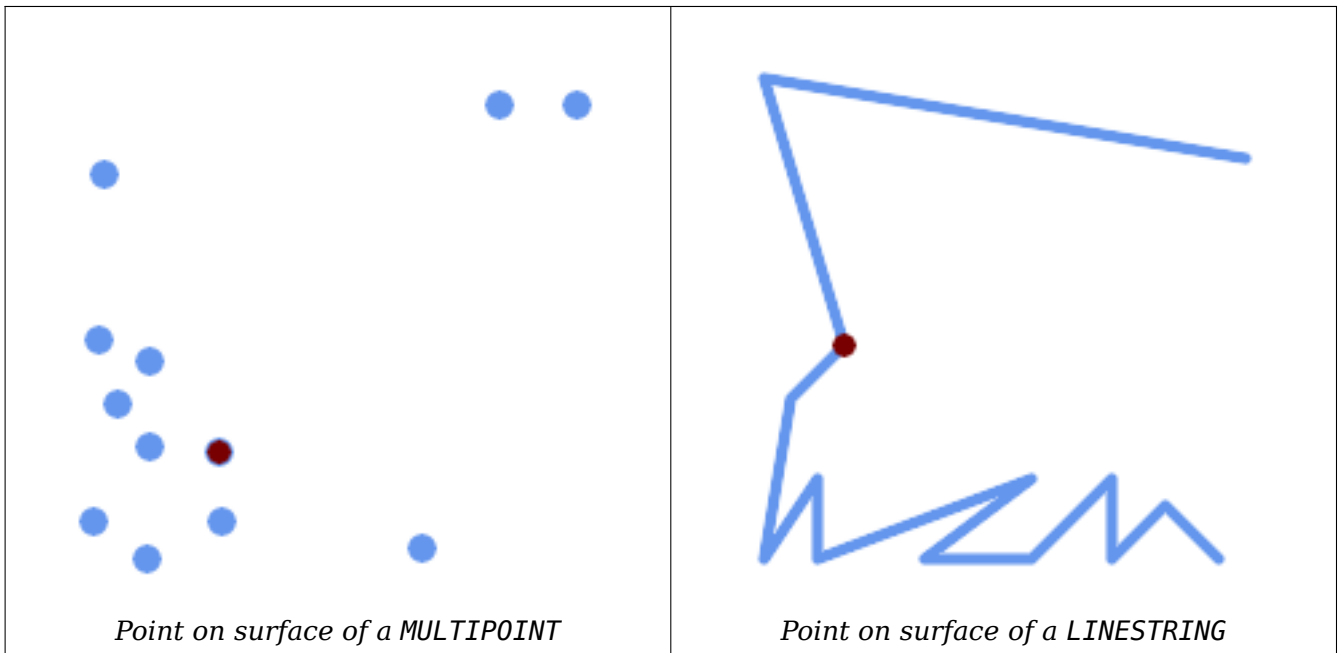


This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST\_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.

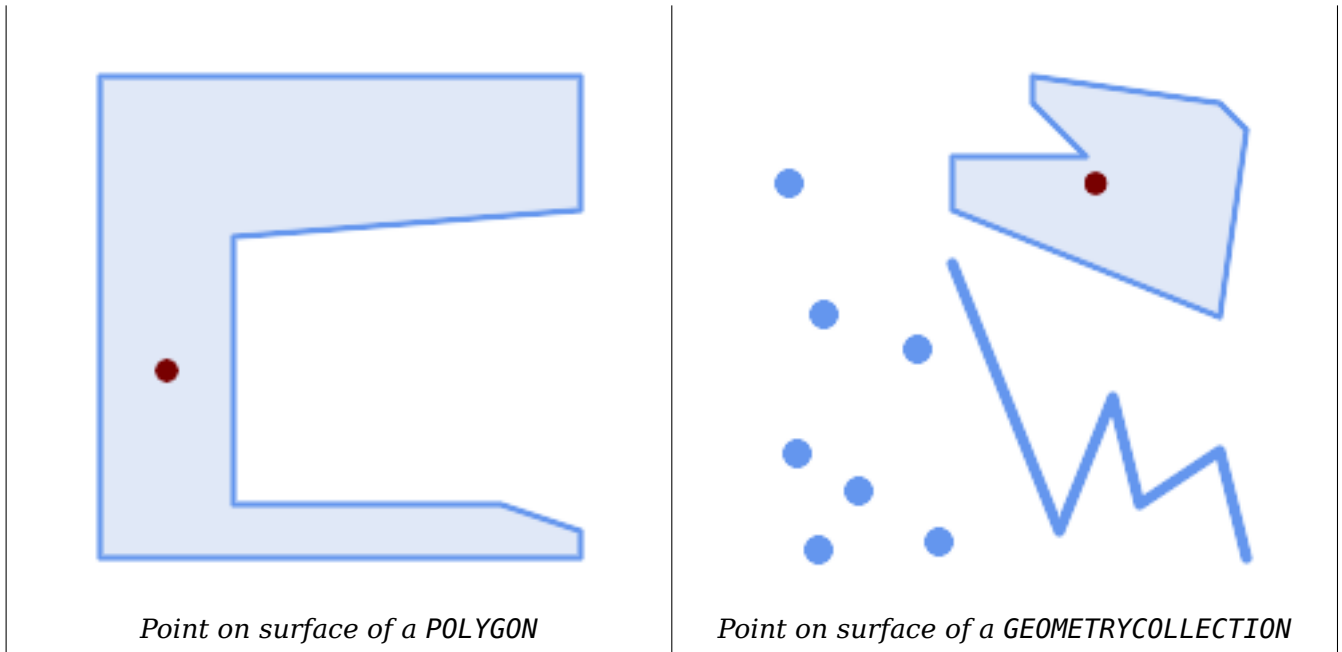


This function supports 3d and will not drop the z-index.

☒☒







```

SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)')::geometry);

POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)')::geometry);

POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))')::geometry);

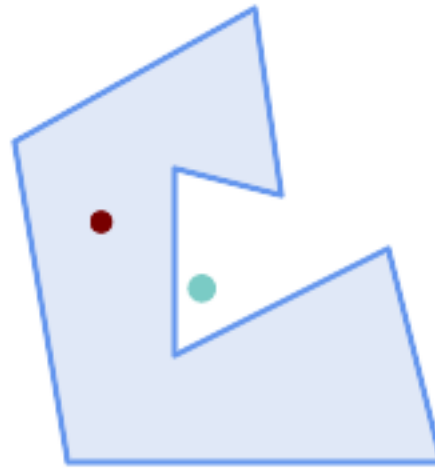
POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));

POINT(0 0 1)

```

**Example:** The result of `ST_PointOnSurface` is guaranteed to lie within polygons, whereas the point computed by `ST_Centroid` may be outside.



*Red: point on surface; Green: centroid*

```
SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
 ST_AsText(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;
```

pt_on_surf	centroid
POINT(62.5 110)	POINT(100.18264840182648 85.11415525114155)

☒☒

[ST\\_Centroid](#), [ST\\_MaximumInscribedCircle](#)

### 7.14.19 ST\_Polygonize

**ST\_Polygonize** — Computes a collection of polygons formed from the linework of a set of geometries.

#### Synopsis

```
geometry ST_Polygonize(geometry set geomfield);
geometry ST_Polygonize(geometry[] geom_array);
```

☒☒

Creates a GeometryCollection containing the polygons formed by the linework of a set of geometries. If the input linework does not form any polygons, an empty GeometryCollection is returned.

This function creates polygons covering all delimited areas. If the result is intended to form a valid polygonal geometry, use [ST\\_BuildArea](#) to prevent holes being filled.



**Note**

The input linework must be correctly noded for this function to work properly. To ensure input is noded use [ST\\_Node](#) on the input geometry before polygonizing.

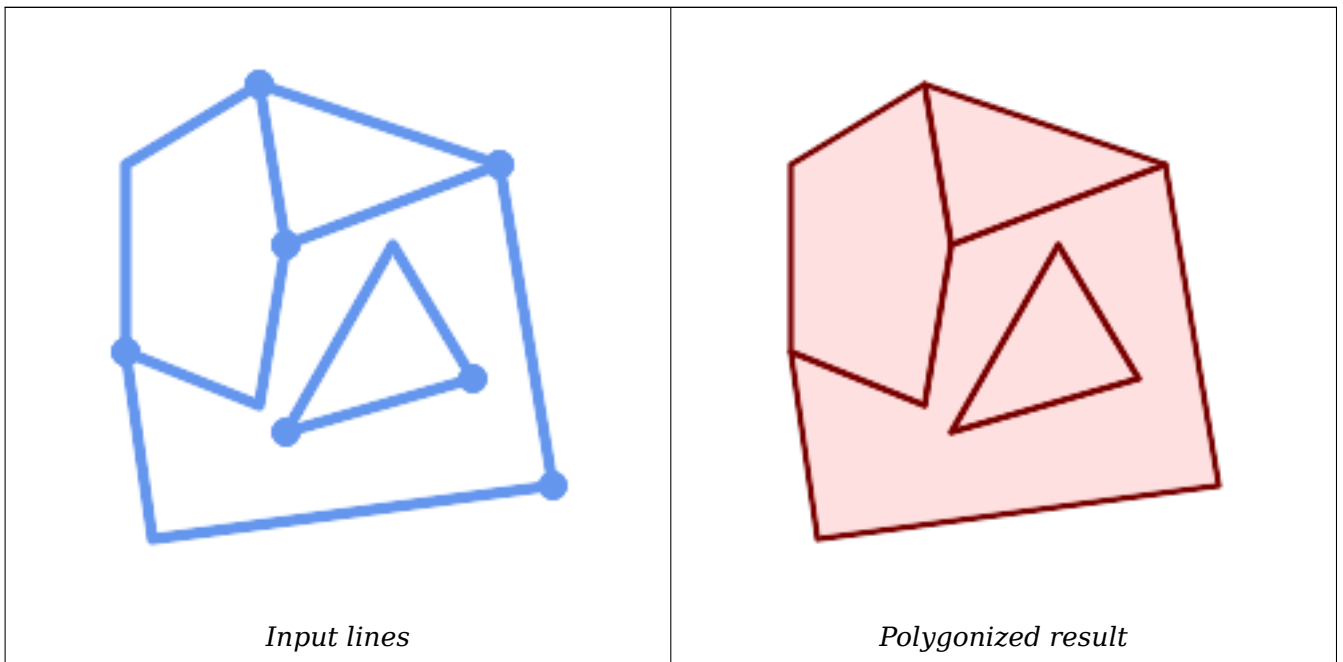


**Note**

GeometryCollections can be difficult to handle with external tools. Use [ST\\_Dump](#) to convert the polygonized result into separate polygons.

GEOS

1.0.0RC1



```

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_AsText(ST_Polygonize(geom))
FROM data;

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 ((20 90, 20 160, 70 190, 80 130, 70 70, 20 90)),
 POLYGON ((160 160, 80 130, 70 190, 160 160)),
 POLYGON ((80 60, 120 130, 150 80, 80 60)))

```

Polygonizing a table of linestrings:

```

SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT geom_4269 FROM ma.suffolk_edges) As foo;

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)))

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((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;

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))

```

☒☒

[ST\\_BuildArea](#), [ST\\_Dump](#), [ST\\_Node](#)

### 7.14.20 ST\_ReducePrecision

`ST_ReducePrecision` — Returns a valid geometry with points rounded to a grid tolerance.

#### Synopsis

geometry **ST\_ReducePrecision**(geometry g, float8 gridsize);

☒☒

Returns a valid geometry with all points rounded to the provided grid tolerance, and features below the tolerance removed.

Unlike [ST\\_SnapToGrid](#) the returned geometry will be valid, with no ring self-intersections or collapsed components.

Precision reduction can be used to:

- match coordinate precision to the data accuracy
- reduce the number of coordinates needed to represent a geometry
- ensure valid geometry output to formats which use lower precision (e.g. text formats such as WKT, GeoJSON or KML when the number of output decimal places is limited).
- export valid geometry to systems which use lower or limited precision (e.g. SDE, Oracle tolerance value)

Availability: 3.1.0.

Requires GEOS >= 3.9.0.





same example but linestring orientation flipped

```
SELECT ST_AsText(
 ST_SharedPaths(
 ST_GeomFromText('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'),
 ST_GeomFromText('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),
 (51 150,101 150,76 175,51 150))')
)
) As wkt
```

wkt

```

GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,
MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))
```

☒☒

[ST\\_Dump](#), [ST\\_GeometryN](#), [ST\\_NumGeometries](#)

## 7.14.22 ST\_Simplify

`ST_Simplify` — Returns a simplified representation of a geometry, using the Douglas-Peucker algorithm.

### Synopsis

```
geometry ST_Simplify(geometry geom, float tolerance);
geometry ST_Simplify(geometry geom, float tolerance, boolean preserveCollapsed);
```

☒☒

Computes a simplified representation of a geometry using the [Douglas-Peucker algorithm](#). The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework. The result may not be valid even if the input is.

The function can be called with any kind of geometry (including GeometryCollections), but only line and polygon elements are simplified. Endpoints of linear geometry are preserved.

The `preserveCollapsed` flag retains small geometries that would otherwise be removed at the given tolerance. For example, if a 1m long line is simplified with a 10m tolerance, when `preserveCollapsed` is true the line will not disappear. This flag is useful for rendering purposes, to prevent very small features disappearing from a map.



### Note

The returned geometry may lose its simplicity (see [ST\\_IsSimple](#)), topology may not be preserved, and polygonal results may be invalid (see [ST\\_IsValid](#)). Use [ST\\_SimplifyPreserveTopology](#) to preserve topology and ensure validity.



**Note**

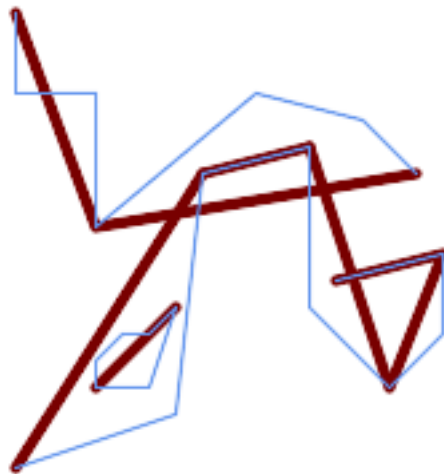
This function does not preserve boundaries shared between polygons. Use `ST_CoverageSimplify` if this is required.

1.2.2

```
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_triangle,
 (ST_Simplify(geom, 100) is null) AS np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS t;
```

np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_triangle	np100_geometrygoesaway
49	33	17	9	4	t

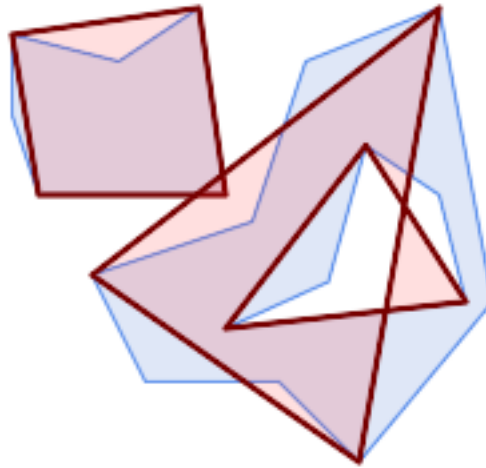
Simplifying a set of lines. Lines may intersect after simplification.



```
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))',
40);
```

Simplifying a MultiPolygon. Polygonal results may be invalid.





```
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)))',
 40);
```

☒☒

[ST\\_IsSimple](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#), [Topology ST\\_Simplify](#)

### 7.14.23 ST\_SimplifyPreserveTopology

`ST_SimplifyPreserveTopology` — Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

#### Synopsis

geometry **ST\_SimplifyPreserveTopology**(geometry geom, float tolerance);

☒☒

Computes a simplified representation of a geometry using a variant of the [Douglas-Peucker algorithm](#) which limits simplification to ensure the result has the same topology as the input. The simplification tolerance is a distance value, in the units of the input SRS. Simplification removes vertices which are within the tolerance distance of the simplified linework, as long as topology is preserved. The result will be valid and simple if the input is.

The function can be called with any kind of geometry (including `GeometryCollections`), but only line and polygon elements are simplified. For polygonal inputs, the result will have the same number of rings (shells and holes), and the rings will not cross. Ring endpoints may be simplified. For linear inputs, the result will have the same number of lines, and lines will not intersect if they did not do so in the original geometry. Endpoints of linear geometry are preserved.



**Note**

This function does not preserve boundaries shared between polygons. Use `ST_CoverageSimplify` if this is required.

GEOS

1.3.3

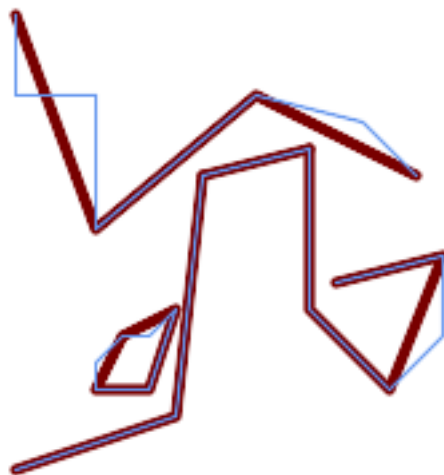


For the same example as `ST_Simplify`, `ST_SimplifyPreserveTopology` prevents oversimplification. The circle can at most become a square.

```
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_square,
 ST_NPoints(ST_SimplifyPreserveTopology(geom, 100)) AS np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) AS geom) AS t;
```

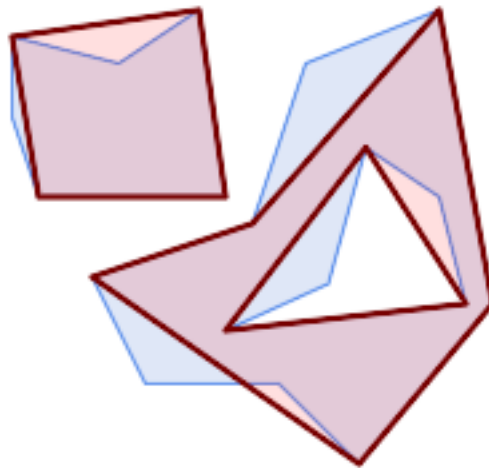
np_before	np01_notbadcircle	np05_notquitecircle	np1_octagon	np10_square	
49	33	17	9	5	↔
	5				↔

Simplifying a set of lines, preserving topology of non-intersecting lines.



```
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))',
 40);
```

Simplifying a MultiPolygon, preserving topology of shells and holes.



```
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)))',
 40);
```

☒☒

[ST\\_Simplify](#), [ST\\_SimplifyVW](#), [ST\\_CoverageSimplify](#)

### 7.14.24 ST\_SimplifyPolygonHull

`ST_SimplifyPolygonHull` — Computes a simplified topology-preserving outer or inner hull of a polygonal geometry.

#### Synopsis

```
geometry ST_SimplifyPolygonHull(geometry param_geom, float vertex_fraction, boolean is_outer = true);
```

☒☒

Computes a simplified topology-preserving outer or inner hull of a polygonal geometry. An outer hull completely covers the input geometry. An inner hull is completely covered by the input geometry. The result is a polygonal geometry formed by a subset of the input vertices. MultiPolygons and holes are handled and produce a result with the same structure as the input.

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the original geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

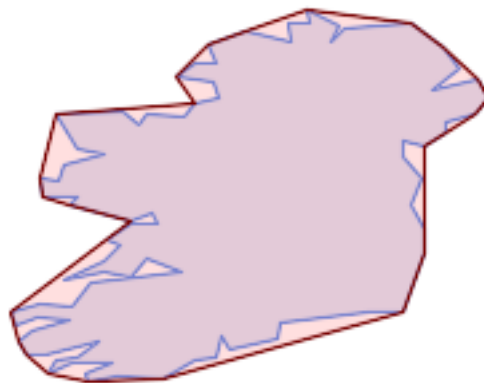
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

GEOS ☒☒☒☒☒

Availability: 3.3.0.

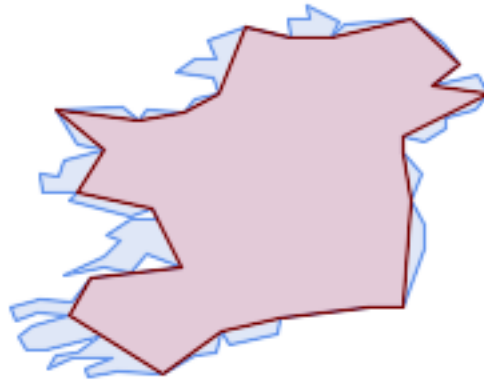
Requires GEOS >= 3.11.0.

☒☒



*Outer hull of a Polygon*

```
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))',
 0.3);
```



*Inner hull of a Polygon*

```
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))',
 0.3, false);
```



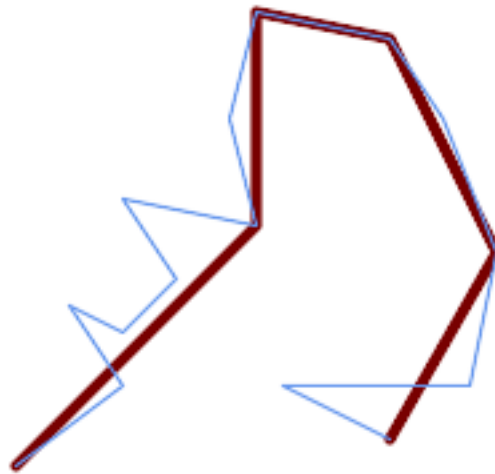
*Outer hull simplification of a MultiPolygon, with segmentization*

```
SELECT ST_SimplifyPolygonHull(
 ST_Segmentize(ST_Letters('xt'), 2.0),
 0.1);
```

☒☒

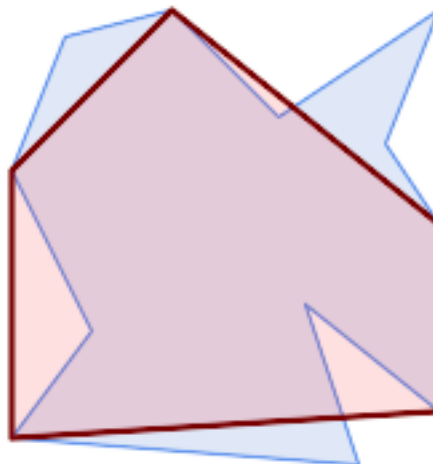
[ST\\_ConvexHull](#), [ST\\_SimplifyVW](#), [ST\\_ConcaveHull](#), [ST\\_Segmentize](#)





```
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)',
 1600);
```

Simplifying a polygon.



```
SELECT ST_SimplifyVW(
 '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))))',
 40);
```

☒☒

[ST\\_SetEffectiveArea](#), [ST\\_Simplify](#), [ST\\_SimplifyPreserveTopology](#), [ST\\_CoverageSimplify](#), [Topology ST\\_Simpl](#)

### 7.14.26 ST\_SetEffectiveArea

`ST_SetEffectiveArea` — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

### Synopsis

geometry **ST\_SetEffectiveArea**(geometry geom, float threshold = 0, integer set\_area = 1);

Geometry

Computes the effective area of a geometry. The effective area is the area of the geometry after removing any holes that are smaller than the specified threshold. The threshold is specified in the same units as the geometry. The set\_area parameter controls whether the area is computed for the entire geometry (1) or for each individual part (2).

When set\_area is 1, the effective area is computed for the entire geometry. When set\_area is 2, the effective area is computed for each individual part of the geometry. The threshold is specified in the same units as the geometry.

For example, a geometry consisting of a large polygon and a small hole will have an effective area that is the area of the large polygon minus the area of the small hole. The effective area will be the same as the area of the large polygon if the small hole is smaller than the specified threshold.



#### Note

The effective area is computed using the `ST_IsSimple` function.



#### Note

The `(topology)` parameter is supported. Use `ST_SimplifyPreserveTopology` to simplify the geometry.



#### Note

The `M` parameter is supported.



#### Note

The `3` parameter is supported.

### 2.2.0 Examples

Geometry

Computes the effective area of a geometry. The effective area is the area of the geometry after removing any holes that are smaller than the specified threshold. The threshold is specified in the same units as the geometry.

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ←
) thrshld_30
FROM (SELECT 'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)::geometry geom) As foo;
-result
all_pts | thrshld_30
-----+-----+
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ←
+38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
```



☒☒

[ST\\_SimplifyVW](#)

### 7.14.27 ST\_TriangulatePolygon

ST\_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

#### Synopsis

geometry **ST\_TriangulatePolygon**(geometry geom);

☒☒

Computes the constrained Delaunay triangulation of polygons. Holes and Multipolygons are supported.

The “constrained Delaunay triangulation” of a polygon is a set of triangles formed from the vertices of the polygon, and covering it exactly, with the maximum total interior angle over all possible triangulations. It provides the “best quality” triangulation of the polygon.

Availability: 3.3.0.

Requires GEOS >= 3.11.0.

☒☒

Triangulation of a square.

```
SELECT ST_AsText(
 ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))');

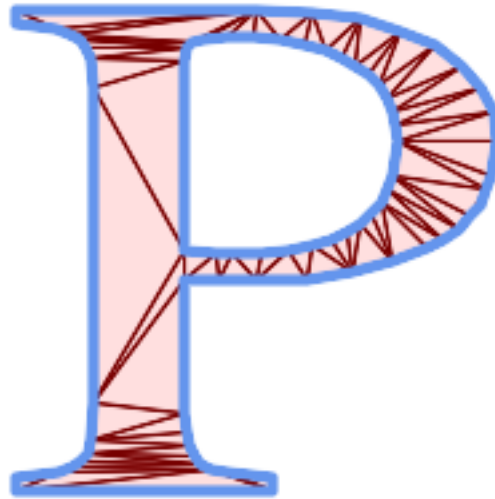
```

st_astext
-----
GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))

☒☒

Triangulation of the letter P.

```
SELECT ST_AsText(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))'
));
```



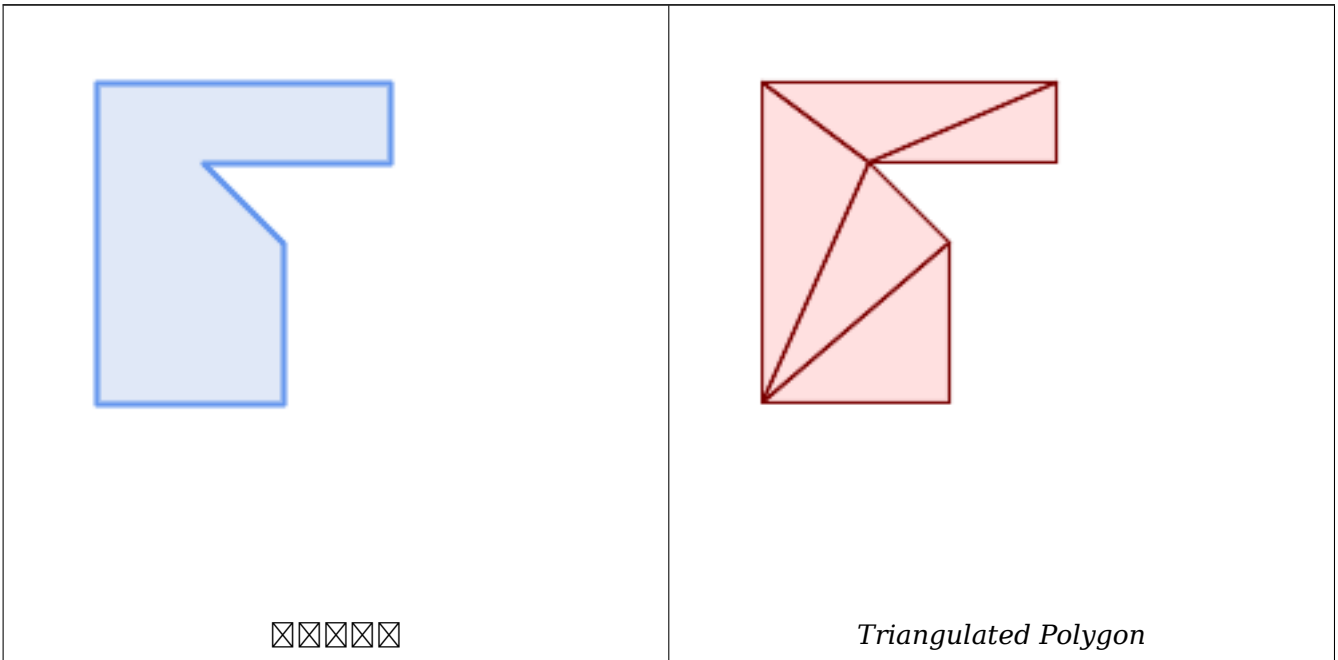
*Polygon Triangulation*

**Same example as ST\_Tessellate**

```
SELECT ST_TriangulatePolygon(
 'POLYGON ((10 190, 10 70, 80 70, 80 130, 50 160, 120 160, 120 190, 10 190 ←
))'::geometry
);
```

ST\_AsText ☒☒☒:

```
GEOMETRYCOLLECTION(POLYGON((50 160,120 190,120 160,50 160))
, POLYGON((10 70,80 130,80 70,10 70))
, POLYGON((50 160,10 70,10 190,50 160))
, POLYGON((120 190,50 160,10 190,120 190))
, POLYGON((80 130,10 70,50 160,80 130)))
```



☒☒

[ST\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_Tessellate](#)

### 7.14.28 ST\_VoronoiLines

ST\_VoronoiLines — Returns the boundaries of the Voronoi diagram of the vertices of a geometry.

#### Synopsis

geometry **ST\_VoronoiLines**( geometry geom , float8 tolerance = 0.0 , geometry extend\_to = NULL );

☒☒

Computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry and returns the boundaries between cells in the diagram as a MultiLineString. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the extend\_to envelope has zero area.

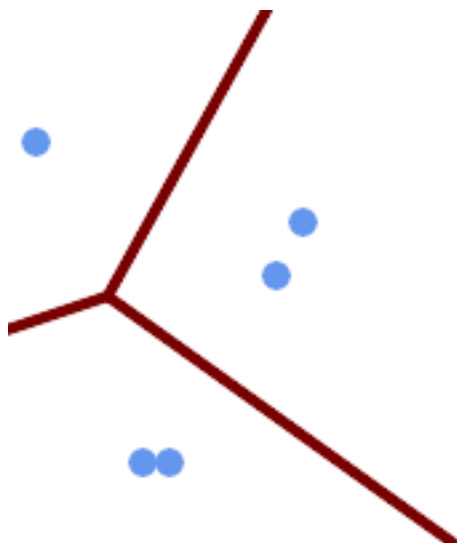
☒☒☒☒☒☒☒☒☒☒:

- **tolerance**: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)
- **extend\_to**: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

GEOS ☒☒☒☒☒

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒



*Voronoi diagram lines, with tolerance of 30 units*

```
SELECT ST_VoronoiLines(
 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,
 30) AS geom;
```

```
ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 92.2727272727273))
```

☒☒

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiPolygons](#)

### 7.14.29 ST\_VoronoiPolygons

`ST_VoronoiPolygons` — Returns the cells of the Voronoi diagram of the vertices of a geometry.

#### Synopsis

```
geometry ST_VoronoiPolygons(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);
```

☒☒

Computes a two-dimensional **Voronoi diagram** from the vertices of the supplied geometry. The result is a `GEOMETRYCOLLECTION` of `POLYGON`s that covers an envelope larger than the extent of the input vertices. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

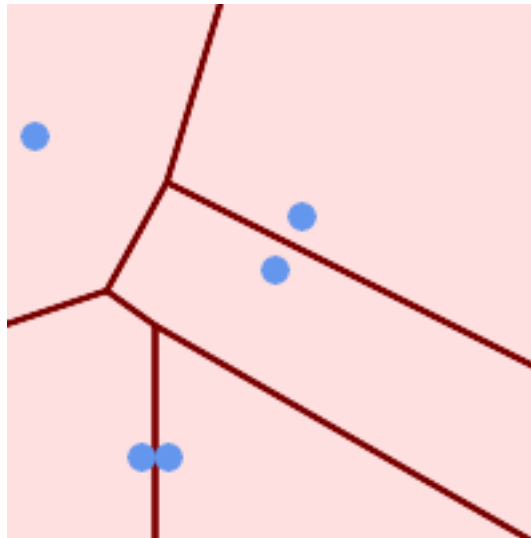
☒☒☒☒☒☒☒☒☒☒:

- **tolerance**: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)
- **extend\_to**: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

GEOS ☒☒☒☒☒

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

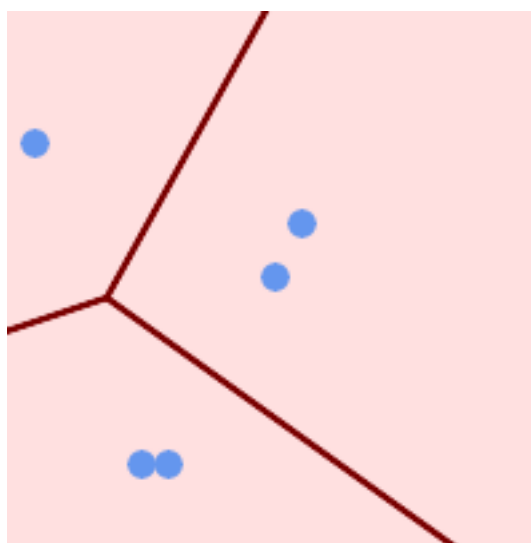


*Points overlaid on top of Voronoi diagram*

```
SELECT ST_VoronoiPolygons(
 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)>::geometry
) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.33333333333333, -110 270,100.5 270,59.3478260869565 ↔
 132.826086956522,36.8181818181818 92.2727272727273, -110 43.3333333333333)),
POLYGON((55 -90, -110 -90, -110 43.3333333333333,36.8181818181818 92.2727272727273,55 ↔
 79.2857142857143,55 -90)),
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818 ↔
 92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230 ↔
 -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



*Voronoi diagram, with tolerance of 30 units*

```
SELECT ST_VoronoiPolygons(
 'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)::geometry,
 30) AS geom;
```

ST\_AsText output

```
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333, -110 270,100.5 270,59.3478260869565 ↔
 132.826086956522,36.8181818181818 92.2727272727273, -110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565 ↔
 132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110 ↔
 43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

☒☒

[ST\\_DelaunayTriangles](#), [ST\\_VoronoiLines](#)

## 7.15 Coverages

### 7.15.1 ST\_CoverageInvalidEdges

`ST_CoverageInvalidEdges` — Window function that finds locations where polygons fail to form a valid coverage.

#### Synopsis

geometry **ST\_CoverageInvalidEdges**(geometry winset geom, float8 tolerance = 0);

☒☒

A window function which checks if the polygons in the window partition form a valid polygonal coverage. It returns linear indicators showing the location of invalid edges (if any) in each polygon.

A set of valid polygons is a valid coverage if the following conditions hold:

- **Non-overlapping** - polygons do not overlap (their interiors do not intersect)
- **Edge-Matched** - vertices along shared edges are identical

As a window function a value is returned for every input polygon. For polygons which violate one or more of the validity conditions the return value is a MULTILINESTRING containing the problematic edges. Coverage-valid polygons return the value NULL. Non-polygonal or empty geometries also produce NULL values.

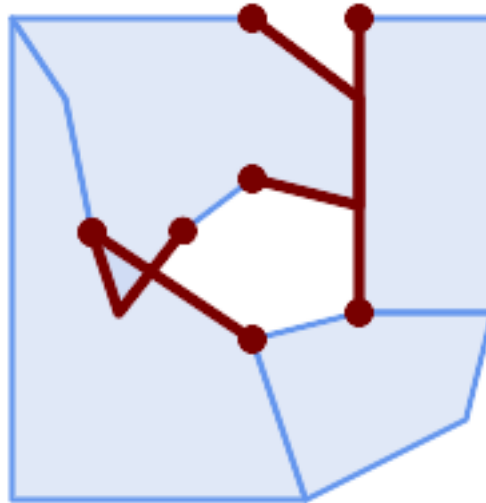
The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned as invalid.

The polygons being checked for coverage validity must also be valid geometries. This can be checked with [ST\\_IsValid](#).

Availability: 3.4.0

Requires GEOS >= 3.12.0

☒☒



*Invalid edges caused by overlap and non-matching vertices*

```
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_AsText(ST_CoverageInvalidEdges(geom) OVER ())
FROM coverage;
```

id	st_astext
1	LINestring (40 110, 100 70)
2	MULTILINestring ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5))
3	LINestring (140 80, 140 190)
4	null

```
-- Test entire table for coverage validity
SELECT true = ALL (
 SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
 FROM coverage
);
```

☒☒

[ST\\_IsValid](#), [ST\\_CoverageUnion](#), [ST\\_CoverageSimplify](#)

### 7.15.2 ST\_CoverageSimplify

ST\_CoverageSimplify — Window function that simplifies the edges of a polygonal coverage.

**Synopsis**

geometry **ST\_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

☒☒

A window function which simplifies the edges of polygons in a polygonal coverage. The simplification preserves the coverage topology. This means the simplified output polygons are consistent along shared edges, and still form a valid coverage.

The simplification uses a variant of the **Visvalingam-Whyatt algorithm**. The *tolerance* parameter has units of distance, and is roughly equal to the square root of triangular areas to be simplified.

To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the *simplifyBoundary* parameter to false.



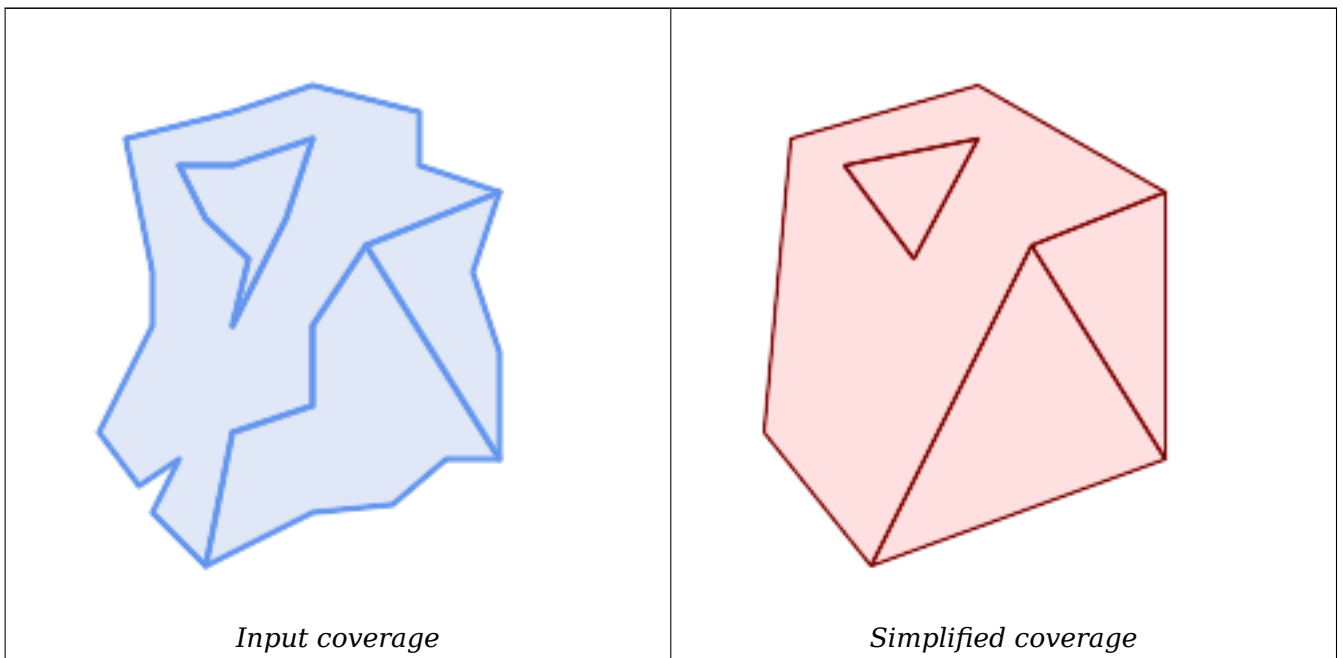
**Note**

If the input is not a valid coverage there may be unexpected artifacts in the output (such as boundary intersections, or separated boundaries which appeared to be shared). Use **ST\_CoverageInvalidEdges** to determine if a coverage is valid.

Availability: 3.4.0

Requires GEOS >= 3.12.0

☒☒



```
WITH coverage(id, geom) AS (VALUES
(1, 'POLYGON ((160 150, 110 130, 90 100, 60 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_AsText(ST_CoverageSimplify(geom, 30) OVER ())
FROM coverage;

id | st_astext
---+-----
 1 | POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125, 90 170, 40 160))
 2 | POLYGON ((40 160, 66 125, 90 170, 40 160))
 3 | POLYGON ((110 130, 160 50, 50 10, 110 130))
 4 | POLYGON ((160 150, 160 50, 110 130, 160 150))

```

☒☒

### ST\_CoverageInvalidEdges

## 7.15.3 ST\_CoverageUnion

**ST\_CoverageUnion** — Computes the union of a set of polygons forming a coverage by removing shared edges.

### Synopsis

geometry **ST\_CoverageUnion**(geometry set geom);

☒☒

An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as **ST\_Union**, but uses the coverage structure to compute the union much faster.

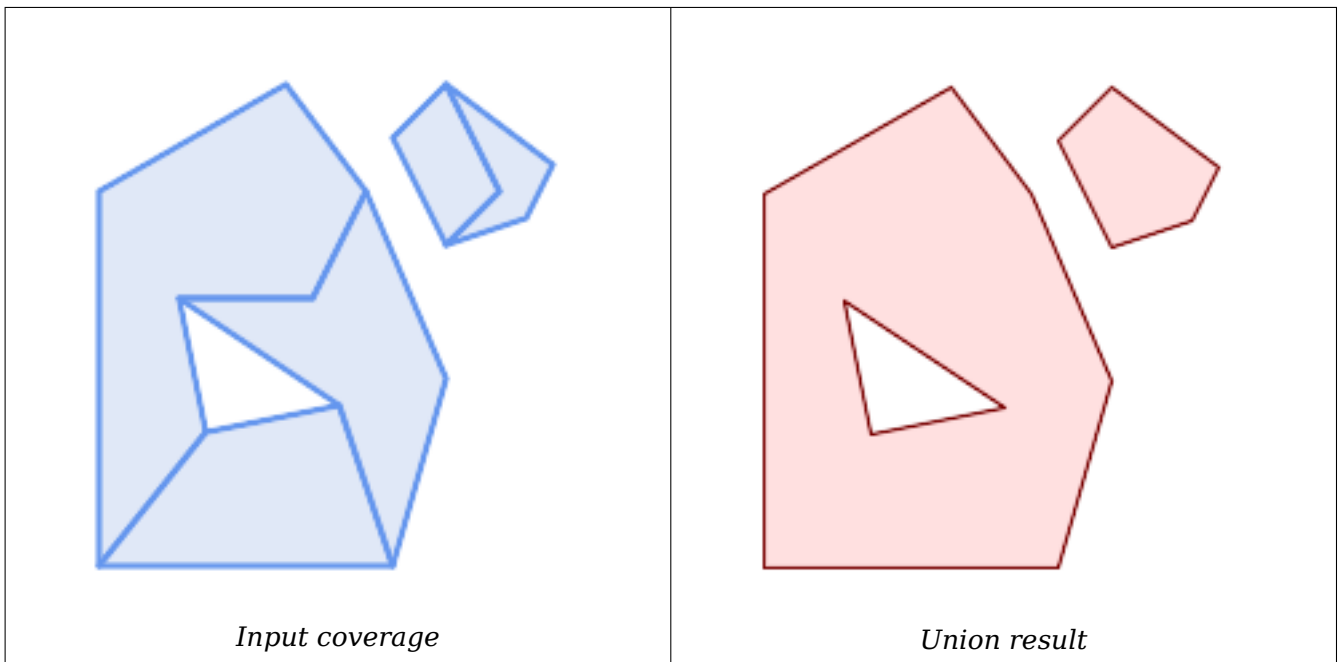


#### Note

If the input is not a valid coverage there may be unexpected artifacts in the output (such as unmerged or overlapping polygons). Use **ST\_CoverageInvalidEdges** to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.8.0

☒☒



```
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_AsText(ST_CoverageUnion(geom))
FROM coverage;

MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ←
 110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))
```

☒☒

[ST\\_CoverageInvalidEdges](#), [ST\\_AsBinary](#)

## 7.16 Affine Transformations

### 7.16.1 ST\_Affine

ST\_Affine — Apply a 3D affine transformation to a geometry.

#### 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);

**ST\_Affine**

Applies a 3D affine transformation to the geometry to do things like translate, rotate, scale in one step.

Version 1: The call

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

represents the transformation matrix

```
/ a b c xoff \
| d e f yoff |
| g h i zoff |
\ 0 0 0 1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

All of the translate / scale functions below are expressed via such an affine transformation.

Version 2: Applies a 2d affine transformation to the geometry. The call

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

represents the transformation matrix

```
/ 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 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

This method is a subcase of the 3D method above.

**ST\_Affine**: 2.0.0 **ST\_Affine**, **ST\_Affine** TIN **ST\_Affine**.

Availability: 1.1.2. Name changed from **Affine** to **ST\_Affine** in 1.2.2



**Note**

1.3.4 **ST\_Affine** (curve) **ST\_Affine**. 1.3.4 **ST\_Affine**.

- This function supports Polyhedral surfaces.
- This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- This function supports 3d and will not drop the z-index.
- This method supports Circular Strings and Curves.

☒☒

```
--Rotate a 3d line 180 degrees about the z axis. Note this is long-hand for doing ↵
ST_Rotate();
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, ↵
 0, 1, 0, 0, 0)) As using_affine,
 ST_AsEWKT(ST_Rotate(geom, pi())) As using_rotate
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
-----+-----
LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AsEWKT(ST_Affine(geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()) ↵
 , 0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As geom) As foo;
 st_asewkt

LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

☒☒

[ST\\_Rotate](#), [ST\\_Scale](#), [ST\\_Translate](#), [ST\\_TransScale](#)

## 7.16.2 ST\_Rotate

ST\_Rotate — Rotates a geometry about an origin point.

### 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);
```

☒☒

Rotates geometry rotRadians counter-clockwise about the origin point. The rotation origin can be specified either as a POINT geometry, or as x and y coordinates. If the origin is not specified, the geometry is rotated about POINT(0 0).

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST\_Rotate in 1.2.2



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
--Rotate 180 degrees
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()));
 st_asewkt

LINESTRING(-50 -160,-50 -50,-100 -50)
(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160
SELECT ST_AsEWKT(ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160));
 st_asewkt

LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)
(1 row)

--Rotate 60 degrees clockwise from centroid
SELECT ST_AsEWKT(ST_Rotate(geom, -pi()/3, ST_Centroid(geom)))
FROM (SELECT 'LINESTRING (50 160, 50 50, 100 50)::geometry AS geom) AS foo;
 st_asewkt

LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)
(1 row)
```

☒☒

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 7.16.3 ST\_RotateX

ST\_RotateX — Rotates a geometry about the X axis.

#### Synopsis

geometry **ST\_RotateX**(geometry geomA, float rotRadians);

☒☒

Rotates a geometry geomA - rotRadians about the X axis.



#### Note

ST\_RotateX(geomA, rotRadians) is short-hand for ST\_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

Availability: 1.1.2. Name changed from RotateX to ST\_RotateX in 1.2.2



This function supports Polyhedral surfaces.



This function supports 3d and will not drop the z-index.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
--Rotate a line 90 degrees along x-axis
SELECT ST_AsEWKT(ST_RotateX(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
 st_asewkt

LINESTRING(1 -3 2,1 -1 1)
```

☒☒

[ST\\_Affine](#), [ST\\_RotateY](#), [ST\\_RotateZ](#)

### 7.16.4 ST\_RotateY

ST\_RotateY — Rotates a geometry about the Y axis.

#### Synopsis

geometry **ST\_RotateY**(geometry geomA, float rotRadians);

☒☒

Rotates a geometry geomA - rotRadians about the y axis.



#### Note

ST\_RotateY(geomA, rotRadians) is short-hand for ST\_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0).

Availability: 1.1.2. Name changed from RotateY to ST\_RotateY in 1.2.2

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.



This function supports Polyhedral surfaces.



This function supports 3d and will not drop the z-index.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
--Rotate a line 90 degrees along y-axis
SELECT ST_AsEWKT(ST_RotateY(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
 st_asewkt

LINESTRING(3 2 -1,1 1 -1)
```

☒☒

[ST\\_Affine](#), [ST\\_RotateX](#), [ST\\_RotateZ](#)

## 7.16.5 ST\_RotateZ

ST\_RotateZ — Rotates a geometry about the Z axis.

### Synopsis

geometry **ST\_RotateZ**(geometry geomA, float rotRadians);

⊠

Rotates a geometry geomA - rotRadians about the Z axis.



#### Note

This is a synonym for ST\_Rotate



#### Note

ST\_RotateZ(geomA, rotRadians) is short-hand for SELECT ST\_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).

⊠: 2.0.0 ⊠, ⊠ TIN ⊠.

Availability: 1.1.2. Name changed from RotateZ to ST\_RotateZ in 1.2.2



#### Note

1.3.4 ⊠ (curve) ⊠. 1.3.4 ⊠.

- ✔ This function supports 3d and will not drop the z-index.
- ✔ This method supports Circular Strings and Curves.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

⊠

```
--Rotate a line 90 degrees along z-axis
SELECT ST_AsEWKT(ST_RotateZ(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), pi()/2));
 st_asewkt

LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis
SELECT ST_AsEWKT(ST_RotateZ(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As foo;
```





- ☑ This method supports Circular Strings and Curves.
- ☑ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ☑ This function supports M coordinates.

☒☒

```

--Version 1: scale X, Y, Z
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75, 0.8));
 st_asewkt

LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 0.75));
 st_asewkt

LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
 ST_MakePoint(0.5, 0.75, 2, -1)));
 st_asewkt

LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)::geometry'));
 st_astext

LINESTRING(1 1,3 3)

```

☒☒

**ST\_Affine, ST\_TransScale**

### 7.16.7 ST\_Translate

ST\_Translate — Translates a geometry by given offsets.

#### Synopsis

```

geometry ST_Translate(geometry g1, float deltax, float deltax);
geometry ST_Translate(geometry g1, float deltax, float deltax, float deltax);

```

☒☒

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.

**Note**

1.3.4 `ST_Translate` (curve) `ST_Translate`. 1.3.4 `ST_Translate`.

1.2.2 `ST_Translate`.

- This function supports 3d and will not drop the z-index.
- This method supports Circular Strings and Curves.

☒☒

Move a point 1 degree longitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('POINT(-71.01 42.37)',4326),1,0)) As ↵
 wgs_transgeomtxt;

 wgs_transgeomtxt

 POINT(-70.01 42.37)
```

Move a linestring 1 degree longitude and 1/2 degree latitude

```
SELECT ST_AsText(ST_Translate(ST_GeomFromText('LINESTRING(-71.01 42.37,-71.11 42.38)',4326) ↵
 ,1,0.5)) As wgs_transgeomtxt;
 wgs_transgeomtxt

 LINESTRING(-70.01 42.87,-70.11 42.88)
```

Move a 3d point

```
SELECT ST_AsEWKT(ST_Translate(CAST('POINT(0 0 0)' As geometry), 5, 12,3));
 st_asewkt

 POINT(5 12 3)
```

Move a curve and a point

```
SELECT ST_AsText(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));

GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ↵
 5)),POINT(2 5))
```

☒☒

[ST\\_Affine](#), [ST\\_AsText](#), [ST\\_GeomFromText](#)

## 7.16.8 ST\_TransScale

`ST_TransScale` — Translates and scales a geometry by given offsets and factors.

### Synopsis

geometry **ST\_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.



**Note**

ST\_TransScale(geomA, deltaX, deltaY, XFactor, YFactor) is short-hand for ST\_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX\*XFactor, deltaY\*YFactor, 0).



**Note**

1.3.4 (curve) 1.3.4

Availability: 1.1.0.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

```
SELECT ST_AsEWKT(ST_TransScale(ST_GeomFromEWKT('LINESTRING(1 2 3, 1 1 1)'), 0.5, 1, 1, 2));
 st_asewkt
```

```

LINESTRING(1.5 6 3,1.5 4 1)
```

```
--Buffer a point to get an approximation of a circle, convert to curve and then translate ↔
1,2 and scale it 3,4
```

```
SELECT ST_AsText(ST_Transscale(ST_LineToCurve(ST_Buffer('POINT(234 567)', 3)),1,2,3,4));
```

```

CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 ↔
2264,698.636038969321 2284.48528137424,714 2276))
```

[ST\\_Affine](#), [ST\\_Translate](#)

## 7.17 Clustering Functions

### 7.17.1 ST\_ClusterDBSCAN

`ST_ClusterDBSCAN` — Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

#### Synopsis

integer `ST_ClusterDBSCAN`(geometry winset geom, float8 eps, integer minpoints);

☒☒

A window function that returns a cluster number for each input geometry, using the 2D [Density-based spatial clustering of applications with noise \(DBSCAN\)](#) algorithm. Unlike `ST_ClusterKMeans`, it does not require the number of clusters to be specified, but instead uses the desired **distance** (eps) and density (minpoints) parameters to determine each cluster.

An input geometry is added to a cluster if it is either:

- A "core" geometry, that is within eps **distance** of at least minpoints input geometries (including itself); or
- A "border" geometry, that is within eps **distance** of a core geometry.

Note that border geometries may be within eps distance of core geometries in more than one cluster. Either assignment would be correct, so the border geometry will be arbitrarily assigned to one of the available clusters. In this situation it is possible for a correct cluster to be generated with fewer than minpoints geometries. To ensure deterministic assignment of border geometries (so that repeated calls to `ST_ClusterDBSCAN` will produce identical results) use an `ORDER BY` clause in the window definition. Ambiguous cluster assignments may differ from other DBSCAN implementations.



#### Note

Geometries that do not meet the criteria to join any cluster are assigned a cluster number of NULL.

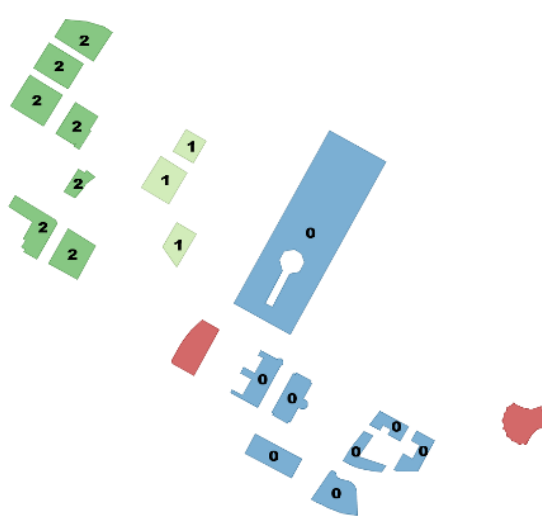
2.3.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.

☒☒

Clustering polygon within 50 meters of each other, and requiring at least 2 polygons per cluster.



*Clusters within 50 meters with at least 2 items per cluster. Singletons have NULL for cid*

```

SELECT name, ST_ClusterDBSCAN(geom, eps = > 50, minpoints = > 2) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
 AND ST_DWithin(geom,
 ST_Transform(
 ST_GeomFromText('POINT ↵
 (-71.04054 42.35141)', 4326), 26986),
 500);

```

bucket	name	
0	Manulife Tower	↵
0	Park Lane Seaport I	↵
0	Park Lane Seaport II	↵
0	Renaissance Boston Waterfront Hotel	↵
0	Seaport Boston Hotel	↵
0	Seaport Hotel & World Trade Center	↵
0	Waterside Place	↵
0	World Trade Center East	↵
1	100 Northern Avenue	↵
1	100 Pier 4	↵
1	The Institute of Contemporary Art	↵
2	101 Seaport	↵
2	District Hall	↵
2	One Marina Park Drive	↵
2	Twenty Two Liberty	↵
2	Vertex	↵
2	Vertex	↵
2	Watermark Seaport	↵
2	Blue Hills Bank Pavilion	↵
NULL	World Trade Center West	↵
NULL		

(20 rows)

A example showing combining parcels with the same cluster number into geometry collections.

```

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;

```



[ST\\_DWithin](#), [ST\\_ClusterKMeans](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.2 ST\_ClusterIntersecting

`ST_ClusterIntersecting` — Aggregate function that clusters input geometries into connected sets.

### Synopsis

```
geometry[] ST_ClusterIntersecting(geometry set g);
```

☒☒

An aggregate function that returns an array of `GeometryCollections` partitioning the input geometries into connected clusters that are disjoint. Each geometry in a cluster intersects at least one other geometry in the cluster, and does not intersect any geometry in other clusters.

2.2.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

```
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 ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

-- result

st_astext

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))
```

☒☒

[ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

## 7.17.3 ST\_ClusterIntersectingWin

`ST_ClusterIntersectingWin` — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.

### Synopsis

```
integer ST_ClusterIntersectingWin(geometry winset geom);
```

☒☒

A window function that builds connected clusters of geometries that intersect. It is possible to traverse all geometries in a cluster without leaving the cluster. The return value is the cluster number that the geometry argument participates in, or null for null inputs.

Availability: 3.4.0

☒☒

```
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,
 ST_AsText(geom),
 ST_ClusterIntersectingWin(geom) OVER () AS cluster
FROM testdata;
```

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

☒☒

[ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_ClusterWithinWin](#)

### 7.17.4 ST\_ClusterKMeans

`ST_ClusterKMeans` — Window function that returns a cluster id for each input geometry using the K-means algorithm.

#### Synopsis

integer **ST\_ClusterKMeans**(geometry winset geom, integer number\_of\_clusters, float max\_radius);

☒☒

Returns **K-means** cluster number for each input geometry. The distance used for clustering is the distance between the centroids for 2D geometries, and distance between bounding box centers for 3D geometries. For POINT inputs, M coordinate will be treated as weight of input and has to be larger than 0.

`max_radius`, if set, will cause `ST_ClusterKMeans` to generate more clusters than `k` ensuring that no cluster in output has radius larger than `max_radius`. This is useful in reachability analysis.

Enhanced: 3.2.0 Support for max\_radius

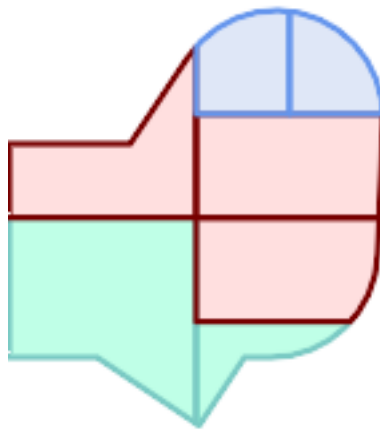
Enhanced: 3.1.0 Support for 3D geometries and weights

2.3.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
 ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ←
 40, 'endcap=square'),12) As geom;
```



*Parcels color-coded by cluster number (cid)*

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
FROM parcels;
```

cid	parcel_id	geom
0	001	0103000000...
0	002	0103000000...
1	003	0103000000...
0	004	0103000000...
1	005	0103000000...
2	006	0103000000...
2	007	0103000000...

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
FROM parcels;
```



cid	parcel_id	type
1	005	commercial
1	003	commercial
2	007	commercial
0	001	commercial
1	004	residential
0	002	residential
2	006	residential

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:

```
create table kontur_population_3000km_clusters as
select
 geom,
 ST_ClusterKMeans(
 ST_Force4D(
 ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
 mvalue => population -- set clustering to be weighed by population
),
 20, -- aim to generate at least 20 clusters
 max_radius => 3000000 -- but generate more to make each under 3000 km radius
) over () as cid
from
 kontur_population;
```



*World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth’s curvature.*

☒☒

[ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersectingWin](#), [ST\\_ClusterWithinWin](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterWithin](#), [ST\\_Subdivide](#), [ST\\_Force3D](#), [ST\\_Force4D](#),

### 7.17.5 ST\_ClusterWithin

`ST_ClusterWithin` — Aggregate function that clusters geometries by separation distance.

#### Synopsis

```
geometry[] ST_ClusterWithin(geometry set g, float8 distance);
```



An aggregate function that returns an array of GeometryCollections, where each collection is a cluster containing some input geometries. Clustering partitions the input geometries into sets in which each geometry is within the specified *distance* of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST\_ClusterWithin is equivalent to running [ST\\_ClusterDBSCAN](#) with `minpoints => 0`.

2.2.0



This method supports Circular Strings and Curves.



```
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 ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext

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))
```



[ST\\_ClusterWithinWin](#), [ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#)

### 7.17.6 ST\_ClusterWithinWin

ST\_ClusterWithinWin — Window function that returns a cluster id for each input geometry, clustering using separation distance.

#### Synopsis

integer **ST\_ClusterWithinWin**(geometry winset geom, float8 distance);



A window function that returns a cluster number for each input geometry. Clustering partitions the geometries into sets in which each geometry is within the specified distance of at least one other geometry in the same cluster. Distances are Cartesian distances in the units of the SRID.

ST\_ClusterWithinWin is equivalent to running [ST\\_ClusterDBSCAN](#) with `minpoints => 0`.

Availability: 3.4.0



This method supports Circular Strings and Curves.

☒☒

```
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,
 ST_AsText(geom),
 ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
FROM testdata;
```

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

☒☒

[ST\\_ClusterWithin](#), [ST\\_ClusterDBSCAN](#), [ST\\_ClusterIntersecting](#), [ST\\_ClusterIntersectingWin](#),

## 7.18 Bounding Box Functions

### 7.18.1 Box2D

Box2D — Returns a BOX2D representing the 2D extent of a geometry.

#### Synopsis

box2d **Box2D**(geometry geom);

☒☒

Returns a **box2d** representing the 2D extent of the geometry.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT Box2D(ST_GeomFromText('LINESTRING(1 2, 3 4, 5 6)'));
```

```
box2d
```

```

```

```
BOX(1 2,5 6)
```

```
SELECT Box2D(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227 150406)'));
```

```
box2d
```

```

```

```
BOX(220186.984375 150406,220288.25 150506.140625)
```

☒☒

**Box3D, ST\_GeomFromText**

## 7.18.2 Box3D

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

### Synopsis

```
box3d Box3D(geometry geom);
```

☒☒

Returns a **box3d** representing the 3D extent of the geometry.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒☒.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT Box3D(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 5, 5 6 5)'));
```

```
Box3d
```

```

```

```
BOX3D(1 2 3,5 6 5)
```

```
SELECT Box3D(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 1,220227 150406 1)'));
```

```
Box3d
```

```

```

```
BOX3D(220227 150406 1,220268 150415 1)
```



## 7.18.4 ST\_Expand

ST\_Expand — Returns a bounding box expanded from another bounding box or a geometry.

### 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);
```

☒☒

Returns a bounding box expanded from the bounding box of the input, either by specifying a single distance with which the box should be expanded on both axes, or by specifying an expansion distance for each axis. Uses double-precision. Can be used for distance queries, or to add a bounding box filter to a query to take advantage of a spatial index.

In addition to the version of ST\_Expand accepting and returning a geometry, variants are provided that accept and return **box2d** and **box3d** data types.

Distances are in the units of the spatial reference system of the input.

ST\_Expand is similar to **ST\_Buffer**, except while buffering expands a geometry in all directions, ST\_Expand expands the bounding box along each axis.



#### Note

Pre version 1.3, ST\_Expand was used in conjunction with **ST\_Distance** to do indexable distance queries. For example, `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. This has been replaced by the simpler and more efficient **ST\_DWithin** function.

Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒



#### Note

Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```

--10 meter expanded box around bbox of a linestring
SELECT CAST(ST_Expand(ST_GeomFromText('LINESTRING(2312980 110676,2312923 110701,2312892 ←
 110714)', 2163),10) As box2d);
 st_expand

BOX(2312882 110666,2312990 110724)

--10 meter expanded 3D box of a 3D box
SELECT ST_Expand(CAST('BOX3D(778783 2951741 1,794875 2970042.61545891 10)' As box3d),10)
 st_expand

BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

--10 meter geometry astext rep of a expand box around a point geometry
SELECT ST_AsEWKT(ST_Expand(ST_GeomFromEWKT('SRID=2163;POINT(2312980 110676)'),10));
 st_asewkt ←

SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970 ←
 110666))

```

☒☒

[ST\\_Buffer](#), [ST\\_DWithin](#), [ST\\_SRID](#)

### 7.18.5 ST\_Extent

`ST_Extent` — Aggregate function that returns the bounding box of geometries.

#### Synopsis

`box2d ST_Extent(geometry set geomfield);`

☒☒

An aggregate function that returns a `box2d` bounding box that bounds a set of geometries. The bounding box coordinates are in the spatial reference system of the input geometries. `ST_Extent` is similar in concept to Oracle Spatial/Locator's `SDO_AGGR_MBR`.

**Note**

`ST_Extent` returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use [ST\\_3DExtent](#).

**Note**

The returned `box3d` value does not include a SRID. Use [ST\\_SetSRID](#) to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

2.0.0 supports Polyhedral surfaces, TIN and Triangulated Irregular Network Surfaces (TIN).

- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒



#### Note

Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
 st_bextent

BOX(739651.875 2908247.25,794875.8125 2970042.75)

--Return extent of each category of geometries
SELECT ST_Extent(geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

 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

--Force back into a geometry
-- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(geom),2249) as bextent FROM sometable;

 bextent

SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
794875.8125 2908247.25,739651.875 2908247.25))
```

☒

[ST\\_EstimatedExtent](#), [ST\\_3DExtent](#), [ST\\_SetSRID](#)

## 7.18.6 ST\_3DExtent

`ST_3DExtent` — Aggregate function that returns the 3D bounding box of geometries.

### Synopsis

```
box3d ST_3DExtent(geometry set geomfield);
```



☒☒

An aggregate function that returns a **box3d** (includes Z ordinate) bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.



**Note**

The returned box3d value does not include a SRID. Use **ST\_SetSRID** to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

☒☒☒☒: 2.0.0 ☒☒☒☒☒☒☒☒, ☒☒☒☒ TIN ☒☒☒☒☒☒☒☒☒☒.

Changed: 2.0.0 In prior versions this used to be called ST\_Extent3D



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
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;
 b3extent

BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(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;
 b3extent

BOX3D(1 0 0,4 2 2)
```

☒☒

**ST\_Extent**, **ST\_Force3DZ**, **ST\_SetSRID**

### 7.18.7 ST\_MakeBox2D

ST\_MakeBox2D — Creates a BOX2D defined by two 2D point geometries.

**Synopsis**

box2d **ST\_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

☒☒

Creates a **box2d** defined by two Point geometries. This is useful for doing range queries.

☒☒

```
--Return all features that fall reside 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)
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)
```

☒☒

**ST\_Point**, **ST\_SetSRID**, **ST\_SRID**

**7.18.8 ST\_3DMakeBox**

ST\_3DMakeBox — Creates a BOX3D defined by two 3D point geometries.

**Synopsis**

box3d **ST\_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

☒☒

Creates a **box3d** defined by two 3D Point geometries.



This function supports 3D and will not drop the z-index.

Changed: 2.0.0 In prior versions this used to be called ST\_MakeBox3D

☒☒

```
SELECT ST_3DMakeBox(ST_MakePoint(-989502.1875, 528439.5625, 10),
 ST_MakePoint(-987121.375 ,529933.1875, 10)) As abb3d

--bb3d--

BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)
```

☒☒

[ST\\_MakePoint](#), [ST\\_SetSRID](#), [ST\\_SRID](#)

### 7.18.9 ST\_XMax

ST\_XMax — Returns the X maxima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_XMax(box3d aGeomorBox2DorBox3D);
```

☒☒

Returns the X maxima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However, it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_XMax('BOX3D(1 2 3, 4 5 6)');
st_xmax

4

SELECT ST_XMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmax

5

SELECT ST_XMax(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmax

3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
st_xmax

220288.248780547
```

☒☒

`ST_XMin, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin`

### 7.18.10 ST\_XMin

`ST_XMin` — Returns the X minima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_XMin(box3d aGeomorBox2DorBox3D);
```

☒☒

Returns the X minima of a 2D or 3D bounding box or a geometry.

**Note**

Although this function is only defined for `box3d`, it also works for `box2d` and geometry values due to automatic casting. However it will not accept a geometry or `box2d` text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```

SELECT ST_XMin('BOX3D(1 2 3, 4 5 6)');
st_xmin

1

SELECT ST_XMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
st_xmin

1

SELECT ST_XMin(CAST('BOX(-3 2, 3 4)' As box2d));
st_xmin

-3
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_xmin

220186.995121892

```

☒☒

**ST\_XMax, ST\_YMax, ST\_YMin, ST\_ZMax, ST\_ZMin**

### 7.18.11 ST\_YMax

ST\_YMax — Returns the Y maxima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_YMax(box3d aGeomorBox2DorBox3D);
```

☒☒

Returns the Y maxima of a 2D or 3D bounding box or a geometry.

**Note**

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_YMax('BOX3D(1 2 3, 4 5 6)');
```

```
st_ymax
```

```

```

```
5
```

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
st_ymax
```

```

```

```
6
```

```
SELECT ST_YMax(CAST('BOX(-3 2, 3 4)' As box2d));
```

```
st_ymax
```

```

```

```
4
```

--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D ↔

```
SELECT ST_YMax('LINESTRING(1 3, 5 6)');
```

```
--ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

```
st_ymax
```

```

```

```
150506.126829327
```

☒☒

`ST_XMin, ST_XMax, ST_YMin, ST_ZMax, ST_ZMin`

### 7.18.12 ST\_YMin

ST\_YMin — Returns the Y minima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_YMin(box3d aGeomorBox2DorBox3D);
```

☒☒

Returns the Y minima of a 2D or 3D bounding box or a geometry.

**Note**

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_YMin('BOX3D(1 2 3, 4 5 6)');
```

```
st_ymin
```

```

```

```
2
```

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(1 3 4, 5 6 7)'));
```

```
st_ymin
```

```

```

```
3
```

```
SELECT ST_YMin(CAST('BOX(-3 2, 3 4)' As box2d));
```

```
st_ymin
```

```

```

```
2
```

--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D ↔

```
SELECT ST_YMin('LINESTRING(1 3, 5 6)');
```

```
--ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

```
st_ymin
```

```

```

```
150406
```

☒☒

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_ZMax](#), [ST\\_ZMin](#)

### 7.18.13 ST\_ZMax

`ST_ZMax` — Returns the Z maxima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_ZMax(box3d aGeomorBox2DorBox3D);
```

☒☒

Returns the Z maxima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for `box3d`, it also works for `box2d` and geometry values due to automatic casting. However it will not accept a geometry or `box2d` text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_ZMax('BOX3D(1 2 3, 4 5 6)');
```

```
st_zmax
```

```

```

```
6
```

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
```

```
st_zmax
```

```

```

```
7
```

```
SELECT ST_ZMax('BOX3D(-3 2 1, 3 4 1)');
```

```
st_zmax
```

```

```

```
1
```

--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to a BOX3D ↔

```
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');
```

```
--ERROR: BOX3D parser - doesn't start with BOX3D(
```

```
SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)'));
```

```
st_zmax
```

```

```

```
3
```

☒☒

[ST\\_GeomFromEWKT](#), [ST\\_XMin](#), [ST\\_XMax](#), [ST\\_YMax](#), [ST\\_YMin](#), [ST\\_ZMax](#)

### 7.18.14 ST\_ZMin

ST\_ZMin — Returns the Z minima of a 2D or 3D bounding box or a geometry.

#### Synopsis

```
float ST_ZMin(box3d aGeomorBox2DorBox3D);
```

☒☒

Returns the Z minima of a 2D or 3D bounding box or a geometry.



#### Note

Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.



This function supports 3d and will not drop the z-index.



This method supports Circular Strings and Curves.

☒☒

```
SELECT ST_ZMin('BOX3D(1 2 3, 4 5 6)');
st_zmin

3

SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING(1 3 4, 5 6 7)'));
st_zmin

4

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1) ');
st_zmin

1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to
a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');
--ERROR: BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227
150406 3)'));
st_zmin

1
```



XX

ST\_GeomFromEWKT, ST\_GeomFromText, ST\_XMin, ST\_XMax, ST\_YMax, ST\_YMin, ST\_ZMax

## 7.19 XXXXXXXXXX (Linear Referencing)

### 7.19.1 ST\_LineInterpolatePoint

ST\_LineInterpolatePoint — Returns a point interpolated along a line at a fractional location.

#### Synopsis

geometry **ST\_LineInterpolatePoint**(geometry a\_linestring, float8 a\_fraction);  
 geography **ST\_LineInterpolatePoint**(geography a\_linestring, float8 a\_fraction, boolean use\_spheroid = true);

XX

XXXXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXXXXXX. XXXXXXX 0 X 1 XXXX Float8 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ST\_LineLocatePoint XXXXXXXXX.



**Note**  
 This function computes points in 2D and then interpolates values for Z and M, while ST\_3DLineInterpolatePoint computes points in 3D and only interpolates the M value.

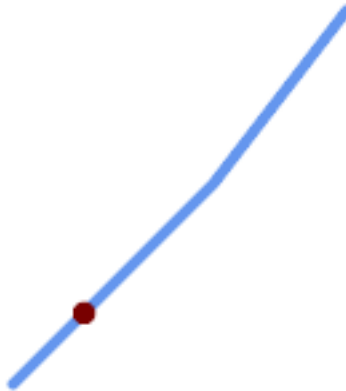


**Note**  
 1.1.1 XXXXXXXXXXXXXXXX M X Z X (XXXXXXXX) XXXXXXXX. XXXXXXXXXXXXXXXXXXX 0.0 XXXXX XXXXXXX.

0.8.2 XXXXXXXXXXXXXXXX. 1.1.1 XXXXX Z X M XXXXXXXXXXXXX.  
XXXXX: 2.1.0 XXXXX, X 2.0.x XXXXXXXXXXXXXXXX ST\_Line\_Interpolate\_Point XXXXX.



This function supports 3d and will not drop the z-index.



20% (0.20)

-- The point 20% along a line

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(
 'LINESTRING(25 50, 100 125, 150 190)',
 0.2));

POINT(51.5974135047432 76.5974135047432)
```

The mid-point of a 3D line:

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint('
 LINESTRING(1 2 3, 4 5 6, 6 7 8)',
 0.5));

POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText(ST_LineInterpolatePoint(line.geom,
 ST_LineLocatePoint(line.geom, 'POINT(4 3)'))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;

POINT(3 4)
```



[ST\\_LineInterpolatePoints](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

## 7.19.2 ST\_3DLineInterpolatePoint

`ST_3DLineInterpolatePoint` — Returns a point interpolated along a 3D line at a fractional location.

### Synopsis

geometry **ST\_3DLineInterpolatePoint**(geometry a\_linestring, float8 a\_fraction);

Float8



**Note**

`ST_LineInterpolatePoint` computes points in 2D and then interpolates the values for Z and M, while this function computes points in 3D and only interpolates the M value.

2.1.0



This function supports 3d and will not drop the z-index.

Return point 20% along 3D line

```
SELECT ST_AsText(
 ST_3DLineInterpolatePoint('LINESTRING(25 50 70, 100 125 90, 150 190 200)',
 0.20));
st_asetext

POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

`ST_LineInterpolatePoint`, `ST_LineInterpolatePoint`, `ST_LineMerge`

### 7.19.3 ST\_LineInterpolatePoints

`ST_LineInterpolatePoints` — Returns points interpolated along a line at a fractional interval.

**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);

Returns one or more points interpolated along a line at a fractional interval. The first argument must be a LINestring. The second argument is a float8 between 0 and 1 representing the spacing between the points as a fraction of line length. If the third argument is false, at most one point will be constructed (which is equivalent to `ST_LineInterpolatePoint`.)

If the result has zero or one points, it is returned as a POINT. If it has two or more points, it is returned as a MULTIPOINT.

Availability: 2.5.0



This function supports 3d and will not drop the z-index.



This function supports M coordinates.







**Note**

This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with **ST\_LineMerge**.



**Note**

1.1.1 **M** **Z** ( **M** ) **Z**. **M**. **Z**.

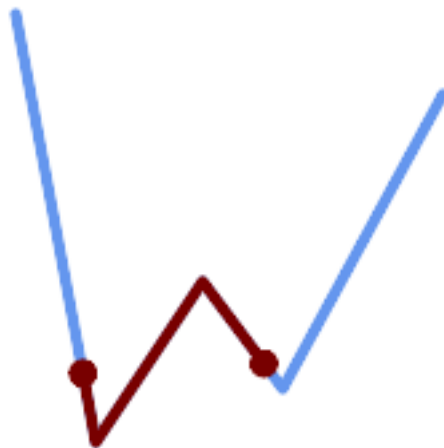
Enhanced: 3.4.0 - Support for geography was introduced.

2.1.0, 2.0.x **ST\_Line\_Substring**.

1.1.0. 1.1.1 **Z** **M**.



This function supports 3d and will not drop the z-index.



1/3 (0.333, 0.666)

```
SELECT ST_AsText(ST_LineSubstring('LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)',
 0.333, 0.666));
```

```
LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862
 49.36542599789519)
```

If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring('LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
```

```
POINT(69.2846934853974 94.2846934853974)
```

A query to cut a LineString into sections of length 100 or shorter. It uses generate\_series() with a CROSS JOIN LATERAL to produce the equivalent of a FOR loop.

```

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_AsText(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)
 -- skip last i if line length is exact multiple of sublen
 WHERE (sublen * i) / len <
> 1.0
) AS d2;

```

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)

Geography implementation measures along a spheroid, geometry along a line

```

SELECT ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611):: ↵
 geography, 0.333, 0.666),6) AS geog_sub
, ST_AsText(ST_LineSubstring('LINESTRING(-118.2436 34.0522, -71.0570 42.3611)::geometry, ↵
 0.333, 0.666),6) AS geom_sub;

geog_sub | LINESTRING(-104.167064 38.854691,-87.674646 41.849854)
geom_sub | LINESTRING(-102.530462 36.819064,-86.817324 39.585927)

```

☒☒

[ST\\_Length](#), [ST\\_LineExtend](#), [ST\\_LineInterpolatePoint](#), [ST\\_LineMerge](#)

### 7.19.6 ST\_LocateAlong

`ST_LocateAlong` — Returns the point(s) on a geometry that match a measure value.

#### Synopsis

geometry **ST\_LocateAlong**(geometry geom\_with\_measure, float8 measure, float8 offset = 0);

☒☒

Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported.

If `offset` is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right.

 Note

**Note**

Use this function only for linear geometries with an M component

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

1.1.0 `ST_Locate_Along_Measure(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0)`.

`2.0.0` `ST_Locate_Along_Measure(geometry geom, float8 measure_start, float8 measure_end, float8 offset = 0, float8 offset_side = 'left')`.



This function supports M coordinates.



This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

☒☒

```
SELECT ST_AsText(
 ST_LocateAlong(
 'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
 3));

MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

☒☒

[ST\\_LocateBetween](#), [ST\\_LocateBetweenElevations](#), [ST\\_InterpolatePoint](#)

## 7.19.7 ST\_LocateBetween

`ST_LocateBetween` — Returns the portions of a geometry that match a measure range.

### Synopsis

geometry **ST\_LocateBetween**(geometry geom, float8 measure\_start, float8 measure\_end, float8 offset = 0);



### ST\_Locate\_Between\_Measures

Clipping a non-convex POLYGON may produce invalid geometry. The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard.

1.1.0 ST\_Locate\_Between\_Measures  
2.0.0 ST\_Locate\_Along\_Measure

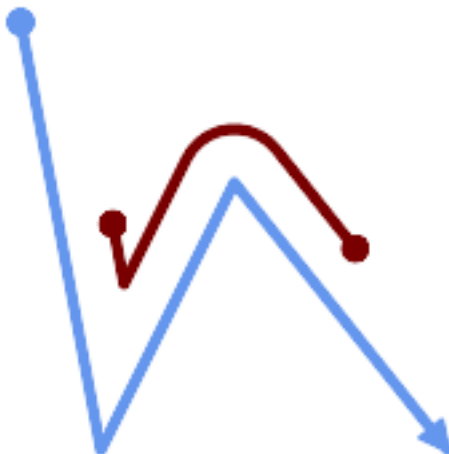
Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

- ✓ This function supports M coordinates.
- ✓ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

### Example

```
SELECT ST_AsText(
 ST_LocateBetween(
 'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
 1.5, 3));

GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



A LineString with the section between measures 2 and 8, offset to the left

```
SELECT ST_AsText(ST_LocateBetween(
 ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
 2, 8,
 20
));

```

```
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303 ←
82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168 ←
128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555 ←
135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769 ←
139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893 ←
139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828 ←
137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606 ←
132.49390095108848,145.31017306064817 95.37790486135405))
```

☒☒

[ST\\_LocateAlong](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

### 7.19.8 ST\_LocateBetweenElevations

`ST_LocateBetweenElevations` — Returns the portions of a geometry that lie in an elevation (Z) range.

#### Synopsis

geometry **ST\_LocateBetweenElevations**(geometry geom, float8 elevation\_start, float8 elevation\_end);

☒☒

Returns a geometry (collection) with the portions of a geometry that lie in an elevation (Z) range.

Clipping a non-convex POLYGON may produce invalid geometry.

1.4.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.



This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_AsText(
 ST_LocateBetweenElevations(
 'LINESTRING(1 2 3, 4 5 6)::geometry,
 2, 4));
```

st\_astext

-----  
MULTILINESTRING Z ((1 2 3,2 3 4))

```
SELECT ST_AsText(
 ST_LocateBetweenElevations(
 'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
 6, 9)) As ewelev;
```

ewelev

-----  
GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))

☒☒

[ST\\_Dump](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

### 7.19.9 ST\_InterpolatePoint

`ST_InterpolatePoint` — Returns an interpolated measure value of a linear measured geometry at the location closest to the given point (M ☒☒) ☒☒☒☒☒☒☒☒.

**Synopsis**

float8 **ST\_InterpolatePoint**(geometry linear\_geom\_with\_measure, geometry point);

☒☒

Returns an interpolated measure value of a linear measured geometry at the location closest to the given point.



**Note**  
Use this function only for linear geometries with an M component

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒.

✔ This function supports 3d and will not drop the z-index.

☒☒

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');

 10
```

☒☒

[ST\\_AddMeasure](#), [ST\\_LocateAlong](#), [ST\\_LocateBetween](#)

### 7.19.10 ST\_AddMeasure

`ST_AddMeasure` — Interpolates measures along a linear geometry.

**Synopsis**

geometry **ST\_AddMeasure**(geometry geom\_mline, float8 measure\_start, float8 measure\_end);

. . . . .

1.5.0 .



This function supports 3d and will not drop the z-index.

```
SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0, 2 0, 4 0)'),1,4)) As ewelev;
 ewelev

LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRING(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
 ewelev

LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('LINESTRINGM(1 0 4, 2 0 4, 4 0 4)'),10,40)) As ewelev;
 ewelev

LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST_AsText(ST_AddMeasure(
ST_GeomFromEWKT('MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4)'),10,70)) As ←
 ewelev;
 ewelev

MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))
```

## 7.20 Trajectory Functions

### 7.20.1 ST\_IsValidTrajectory

`ST_IsValidTrajectory` — Tests if the geometry is a valid trajectory.


#### Synopsis

boolean `ST_IsValidTrajectory`(geometry line);

Tests if a geometry encodes a valid trajectory. A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next.

Valid trajectories are expected as input to spatio-temporal functions like [ST\\_ClosestPointOfApproach](#)

2.2.0

 This function supports 3d and will not drop the z-index.



```
-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
 ST_MakePointM(0,0,1),
 ST_MakePointM(0,1,2)
));
t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE: Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
st_isvalidtrajectory

f
```



**ST\_ClosestPointOfApproach**

## 7.20.2 ST\_ClosestPointOfApproach

**ST\_ClosestPointOfApproach** — Returns a measure at the closest point of approach of two trajectories.

### Synopsis

float8 **ST\_ClosestPointOfApproach**(geometry track1, geometry track2);




Returns the smallest measure at which points interpolated along the given trajectories are the least distance apart.

Inputs must be valid trajectories as checked by **ST\_IsValidTrajectory**. Null is returned if the trajectories do not overlap in their M ranges.

To obtain the actual points at the computed measure use **ST\_LocateAlong** .

2.2.0

 This function supports 3d and will not drop the z-index.



```

-- 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
WITH inp AS (SELECT
 ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) a,
 ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) 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) t,
 ST_Distance(pa,pb) distance,
 ST_AsText(pa, 2) AS pa, ST_AsText(pb, 2) AS pb
FROM points, cpa;

```

t	distance	pa	
	pb		
2015-05-26 10:45:31.034483-07	1.9603683315139542	POINT ZM (7.59 0 3.79 1432662331.03)	←
POINT ZM (9.1 1.24 3.93 1432662331.03)			

☒☒

[ST\\_IsValidTrajectory](#), [ST\\_DistanceCPA](#), [ST\\_LocateAlong](#), [ST\\_AddMeasure](#)

### 7.20.3 ST\_DistanceCPA

`ST_DistanceCPA` — Returns the distance between the closest point of approach of two trajectories.

#### Synopsis

float8 `ST_DistanceCPA`(geometry track1, geometry track2);

☒☒

Returns the distance (in 2D) between two trajectories at their closest point of approach.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). Null is returned if the trajectories do not overlap in their M ranges.

2.2.0 ☒☒☒☒☒☒☒☒☒☒.



This function supports 3d and will not drop the z-index.



```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS (SELECT
 ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) a,
 ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

 distance

1.96036833151395
```



[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_AddMeasure](#), [|=|](#)

## 7.20.4 ST\_CPAWithin

`ST_CPAWithin` — Tests if the closest point of approach of two trajectories is within the specified distance.

### Synopsis

boolean `ST_CPAWithin`(geometry track1, geometry track2, float8 dist);



Tests whether two moving objects have ever been closer than the specified distance.

Inputs must be valid trajectories as checked by [ST\\_IsValidTrajectory](#). False is returned if the trajectories do not overlap in their M ranges.

2.2.0 .



This function supports 3d and will not drop the z-index.



```
WITH inp AS (SELECT
 ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) a,
 ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)::geometry,
 extract(epoch from '2015-05-26 10:00'::timestampz),
 extract(epoch from '2015-05-26 11:00'::timestampz)
) b
```

```
)
SELECT ST_CPWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;

 st_cpawithin | distance
-----+-----
 t | 1.96521473776207
```

☒☒

[ST\\_IsValidTrajectory](#), [ST\\_ClosestPointOfApproach](#), [ST\\_DistanceCPA](#), [|](#) [=](#)

## 7.21 Version Functions

### 7.21.1 PostGIS\_Extensions\_Upgrade

PostGIS\_Extensions\_Upgrade — Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.

#### Synopsis

```
text PostGIS_Extensions_Upgrade(text target_version=null);
```

☒☒

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 un-packaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis\_raster, postgis\_sfcgal, postgis\_topology, and postgis\_tiger\_geocoder. Availability: 2.5.0



#### Note

Changed: 3.4.0 to add target\_version argument.

Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems.

Changed: 3.0.0 to repackage loose extensions and support postgis\_raster.

☒☒

```
SELECT PostGIS_Extensions_Upgrade();
```

```
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
NOTICE: Extension postgis_tiger_geocoder is not available or not packagable for some reason
 reason

 postgis_extensions_upgrade
-----+-----
Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```



☒☒

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` — Reports full PostGIS version and build configuration infos.

#### Synopsis

text `PostGIS_Full_Version()`;

☒☒

Reports full PostGIS version and build configuration infos. Also informs about synchronization between libraries and scripts suggesting upgrades as needed.

Enhanced: 3.4.0 now includes extra PROJ configurations `NETWORK_ENABLED`, `URL_ENDPOINT` and `DATABASE_PATH` of `proj.db` location

☒☒

```
SELECT PostGIS_Full_Version();
 postgis_full_version

POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↔
-1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↔
org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL ↔
3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↔
="0.5.0 (Internal)" TOPOLOGY RASTER
(1 row)
```

☒☒

Section [3.4](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Wagyu\\_Version](#), [PostGIS\\_Version](#)

### 7.21.3 PostGIS\_GEOS\_Version

`PostGIS_GEOS_Version` — Returns the version number of the GEOS library.

#### Synopsis

text `PostGIS_GEOS_Version()`;

☒☒

Returns the version number of the GEOS library, or NULL if GEOS support is not enabled.

☒☒

```
SELECT PostGIS_GEOS_Version();
 postgis_geos_version

3.12.0dev-CAPI-1.18.0
(1 row)
```

☒☒

[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` — Returns the version number of the GEOS library against which PostGIS was built.

#### Synopsis

text `PostGIS_GEOS_Compiled_Version()`;

☒☒

Returns the version number of the GEOS library, or against which PostGIS was built.

Availability: 3.4.0

☒☒

```
SELECT PostGIS_GEOS_Compiled_Version();
 postgis_geos_compiled_version

3.12.0
(1 row)
```

☒☒

[PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Full\\_Version](#)

### 7.21.5 PostGIS\_Liblwgeom\_Version

`PostGIS_Liblwgeom_Version` — Returns the version number of the liblwgeom library. This should match the version of PostGIS.

#### Synopsis

text `PostGIS_Liblwgeom_Version()`;

☒☒

Returns the version number of the liblwgeom library/

☒☒

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version

3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

☒☒

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

### 7.21.6 PostGIS\_LibXML\_Version

`PostGIS_LibXML_Version` — Returns the version number of the libxml2 library.

#### Synopsis

text `PostGIS_LibXML_Version()`;

☒☒

Returns the version number of the LibXML2 library.

1.5 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
SELECT PostGIS_LibXML_Version();
postgis_libxml_version

2.9.10
(1 row)
```

☒☒

[PostGIS\\_Full\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Version](#)

### 7.21.7 PostGIS\_Lib\_Build\_Date

`PostGIS_Lib_Build_Date` — Returns build date of the PostGIS library.

## Synopsis

text **PostGIS\_Lib\_Build\_Date()**;

☒☒

Returns build date of the PostGIS library.

☒☒

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date

 2023-06-22 03:56:11
(1 row)
```

## 7.21.8 PostGIS\_Lib\_Version

PostGIS\_Lib\_Version — Returns the version number of the PostGIS library.

### Synopsis

text **PostGIS\_Lib\_Version()**;

☒☒

Returns the version number of the PostGIS library.

☒☒

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version

 3.4.0dev
(1 row)
```

☒☒

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Version](#)

## 7.21.9 PostGIS\_PROJ\_Version

PostGIS\_PROJ\_Version — Returns the version number of the PROJ4 library.

### Synopsis

text **PostGIS\_PROJ\_Version()**;

---

☒☒

Returns the version number of the PROJ library and some configuration options of proj.

Enhanced: 3.4.0 now includes NETWORK\_ENABLED, URL\_ENDPOINT and DATABASE\_PATH of proj.db location

☒☒

```
SELECT PostGIS_PROJ_Version();
 postgis_proj_version

7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↔
 proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

☒☒

[PostGIS\\_PROJ\\_Compiled\\_Version](#), [PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.21.10 PostGIS\_PROJ\_Compiled\_Version

`PostGIS_PROJ_Compiled_Version` — Returns the version number of the PROJ library against which PostGIS was built.

#### Synopsis

text **`PostGIS_PROJ_Compiled_Version()`**;

☒☒

Returns the version number of the PROJ library, or against which PostGIS was built.

Availability: 3.5.0

☒☒

```
SELECT PostGIS_PROJ_Compiled_Version();
 postgis_proj_compiled_version

9.1.1
(1 row)
```

☒☒

[PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Full\\_Version](#)

### 7.21.11 PostGIS\_Wagyu\_Version

PostGIS\_Wagyu\_Version — Returns the version number of the internal Wagyu library.

#### Synopsis

text **PostGIS\_Wagyu\_Version()**;

☒☒

Returns the version number of the internal Wagyu library, or NULL if Wagyu support is not enabled.

☒☒

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version

0.5.0 (Internal)
(1 row)
```

☒☒

[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_PROJ\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML2\\_Version](#), [PostGIS\\_Version](#)

### 7.21.12 PostGIS\_Scripts\_Build\_Date

PostGIS\_Scripts\_Build\_Date — Returns build date of the PostGIS scripts.

#### Synopsis

text **PostGIS\_Scripts\_Build\_Date()**;

☒☒

Returns build date of the PostGIS scripts.

1.0.0RC1 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
SELECT PostGIS_Scripts_Build_Date();
 postgis_scripts_build_date

2023-06-22 03:56:11
(1 row)
```



[PostGIS\\_Full\\_Version](#), [PostGIS\\_GEOS\\_Version](#), [PostGIS\\_Lib\\_Version](#), [PostGIS\\_LibXML\\_Version](#), [PostGIS\\_Version](#)

### 7.21.13 PostGIS\_Scripts\_Installed

PostGIS\_Scripts\_Installed — Returns version of the PostGIS scripts installed in this database.

#### Synopsis

text **PostGIS\_Scripts\_Installed**();



Returns version of the PostGIS scripts installed in this database.



#### Note

If the output of this function doesn't match the output of [PostGIS\\_Scripts\\_Released](#) you probably missed to properly upgrade an existing database. See the [Upgrading](#) section for more info.

Availability: 0.9.0



```
SELECT PostGIS_Scripts_Installed();
 postgis_scripts_installed

 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```



[PostGIS\\_Full\\_Version](#), [PostGIS\\_Scripts\\_Released](#), [PostGIS\\_Version](#)

### 7.21.14 PostGIS\_Scripts\_Released

PostGIS\_Scripts\_Released — Returns the version number of the postgis.sql script released with the installed PostGIS lib.

#### Synopsis

text **PostGIS\_Scripts\_Released**();

---



Returns the version number of the postgis.sql script released with the installed PostGIS lib.

**Note**

Starting with version 1.1.0 this function returns the same value of `PostGIS_Lib_Version`. Kept for backward compatibility.

Availability: 0.9.0



```
SELECT PostGIS_Scripts_Released();
 postgis_scripts_released

3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```



`PostGIS_Full_Version`, `PostGIS_Scripts_Installed`, `PostGIS_Lib_Version`

## 7.21.15 PostGIS\_Version

`PostGIS_Version` — Returns PostGIS version number and compile-time options.

### Synopsis

text `PostGIS_Version()`;



Returns PostGIS version number and compile-time options.



```
SELECT PostGIS_Version();
 postgis_version

3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```



`PostGIS_Full_Version`, `PostGIS_GEOS_Version`, `PostGIS_Lib_Version`, `PostGIS_LibXML_Version`, `PostGIS_PROJ_Version`



## 7.22 PostGIS GUC(Grand Unified Custom Variable)

### 7.22.1 postgis.backend

`postgis.backend` — GEOS or SFCGAL. `geos` or `sfcgal`.

PostGIS `sfcgal` GUC. GEOS or SFCGAL. `geos` or `sfcgal`.

2.1.0

```
set postgis.backend = sfcgal;
```

```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

Chapter 8

### 7.22.2 postgis.gdal\_datapath

`postgis.gdal datapath` — GDAL `GDAL_DATA`. `GDAL_DATA`.

GDAL `GDAL_DATA` PostgreSQL GUC. `postgis.gdal_datapath` `GDAL`.

GDAL (hard-coded) `GDAL`.



**Note**

PostgreSQL `postgresql.conf`.

2.2.0



**Note**

GDAL `GDAL_DATA`.

[REDACTED]

```
postgis.gdal_datapath [REDACTED].
```

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

[REDACTED]

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

[REDACTED]

### PostGIS\_GDAL\_Version, ST\_Transform

## 7.22.3 postgis.gdal\_enabled\_drivers

`postgis.gdal_enabled_drivers` — PostGIS [REDACTED] GDAL [REDACTED]. GDAL [REDACTED] `GDAL_SKIP` [REDACTED].

[REDACTED]

PostGIS [REDACTED] GDAL [REDACTED]. GDAL [REDACTED] `GDAL_SKIP` [REDACTED]. PostgreSQL [REDACTED] `postgresql.conf` [REDACTED]. [REDACTED]

PostgreSQL [REDACTED] `POSTGIS_GDAL_ENABLED_DRIVERS` [REDACTED] (pass) [REDACTED] `postgis.gdal_enabled_drivers` [REDACTED].

[REDACTED] GDAL [REDACTED]. [REDACTED] `GDAL` [REDACTED] [REDACTED]. [REDACTED].

### Note

`postgis.gdal_enabled_drivers` [REDACTED]. [REDACTED]



- `DISABLE_ALL` [REDACTED] GDAL [REDACTED]. `DISABLE_ALL` [REDACTED], `postgis.gdal_enabled_drivers` [REDACTED].
- `ENABLE_ALL` [REDACTED] GDAL [REDACTED].
- `VSI CURL` [REDACTED] GDAL [REDACTED] `/vsicurl/` [REDACTED].

`postgis.gdal_enabled_drivers` [REDACTED] `DISABLE_ALL` [REDACTED], `DB` [REDACTED], `ST_FromGDALRaster()`, `ST_AsGDALRaster()`, `ST_AsTIFF()`, `ST_AsJPEG()` [REDACTED] `ST_AsPNG()` [REDACTED].



### Note

[REDACTED] PostGIS [REDACTED], `postgis.gdal_enabled_drivers` [REDACTED] `DISABLE_ALL` [REDACTED].



**Note**

GDAL\_SKIP GDAL [Configuration Options](#).

2.2.0

postgis.gdal\_enabled\_drivers

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

PostgreSQL 9.4

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SELECT pg_reload_conf();
```

```
SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SET postgis.gdal_enabled_drivers = default;
```

GDAL

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
```

GDAL

```
SET postgis.gdal_enabled_drivers = 'DISABLE_ALL';
```

[ST\\_FromGDALRaster](#), [ST\\_AsGDALRaster](#), [ST\\_AsTIFF](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [postgis.enable\\_outdb\\_raster](#)

### 7.22.4 postgis.enable\_outdb\_rasters

postgis.enable\_outdb\_rasters — DB

DB PostgreSQL postgresql.conf

PostgreSQL 0 POSTGIS\_ENABLE\_OUTDB\_RASTERS (pass) postgis.enable\_outdb\_rasters



**Note**

postgis.enable\_outdb\_rasters, GUC postgis.enable\_outdb\_rasters



**Note**

PostGIS, `postgis.enable_outdb_rasters`.

2.2.0.

`postgis.enable_outdb_rasters`.

```
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 for specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. You need to reconnect to the database for changes to take effect.

```
--writes to postgres.auto.conf
ALTER SYSTEM postgis.enable_outdb_rasters = true;
--Reloads postgres conf
SELECT pg_reload_conf();
```

[postgis.gdal\\_enabled\\_drivers](#) [postgis.gdal\\_vsi\\_options](#)

### 7.22.5 postgis.gdal\_vsi\_options

`postgis.gdal_vsi_options` — DB

A string configuration to set options used when working with an out-db raster. **Configuration options** control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Availability: 3.2.0

`postgis.enable_outdb_rasters`.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY=
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxx ↔
AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
```

[postgis.enable\\_outdb\\_rasters](#) [postgis.gdal\\_enabled\\_drivers](#)

## 7.23 Troubleshooting Functions

### 7.23.1 PostGIS\_AddBBox

PostGIS\_AddBBox —

#### Synopsis

geometry **PostGIS\_AddBBox**(geometry geomA);



#### Note

This method supports Circular Strings and Curves.

```
UPDATE sometable
SET geom = PostGIS_AddBBox(geom)
WHERE PostGIS_HasBBox(geom) = false;
```

[PostGIS\\_DropBBox](#), [PostGIS\\_HasBBox](#)

### 7.23.2 PostGIS\_DropBBox

PostGIS\_DropBBox —

#### Synopsis

geometry **PostGIS\_DropBBox**(geometry geomA);



☒☒

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

☒☒

[PostGIS\\_AddBBox](#), [PostGIS\\_DropBBox](#)

## Chapter 8

# SFCGAL Functions Reference

SFCGAL is a 2D & 3D geometry engine based on CGAL & C++ (wrapper) libraries. It is a library, not a function, and it is used to create and manipulate geometry objects.

SFCGAL is available at <http://www.sfcgal.org>. It is used in PostGIS via the `postgis_sfcgal` extension.

## 8.1 SFCGAL Management Functions

### 8.1.1 `postgis_sfcgal_version`

`postgis_sfcgal_version` — Returns SFCGAL version.

#### Synopsis

```
text postgis_sfcgal_version(void);
```

⚠

⚠ SFCGAL is not installed.

2.1.0 ⚠

✔ This method needs SFCGAL backend.

⚠

`postgis_sfcgal_full_version`

### 8.1.2 `postgis_sfcgal_full_version`

`postgis_sfcgal_full_version` — Returns the full version of SFCGAL in use including CGAL and Boost versions



## Synopsis

text **postgis\_sfcgal\_version**(void);



Returns the full version of SFCGAL in use including CGAL and Boost versions

Availability: 3.3.0



This method needs SFCGAL backend.



[postgis\\_sfcgal\\_version](#)

## 8.2 SFCGAL Accessors and Setters

### 8.2.1 CG\_ForceLHR

CG\_ForceLHR — LHR(Left Hand Reverse;     .

## Synopsis

geometry **CG\_ForceLHR**(geometry geom);



Availability: 3.5.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.2.2 CG\_IsPlanar

CG\_IsPlanar —           .

## Synopsis

boolean **CG\_IsPlanar**(geometry geom);

---

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.2.3 CG\_IsSolid

CG\_IsSolid — .

#### Synopsis

boolean CG\_IsSolid(geometry geom1);

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.2.4 CG\_MakeSolid

CG\_MakeSolid — . ,

#### Synopsis

geometry CG\_MakeSolid(geometry geom1);

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).



## 8.2.5 CG\_Orientation

CG\_Orientation — (orientation)

### Synopsis

```
integer CG_Orientation(geometry geom);
```

Availability: 3.5.0

-  This method needs SFCGAL backend.
-  This function supports 3d and will not drop the z-index.

## 8.2.6 CG\_Area

CG\_Area — Calculates the area of a geometry

### Synopsis

```
double precision CG_Area(geometry geom);
```

Calculates the area of a geometry.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the area.

Availability: 3.5.0

-  This method needs SFCGAL backend.

```
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))');
 cg_area

 25
(1 row)
```

¶¶

[ST\\_3DArea](#), [ST\\_Area](#)

### 8.2.7 CG\_3DArea






CG\_3DArea — 3 ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶ 0 ¶¶¶¶¶¶¶¶¶¶.

#### Synopsis

```
floatCG_3DArea(geometry geom1);
```

¶¶

Availability: 3.5.0

-  This method needs SFCGAL backend.
-  This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

¶¶

¶¶: ¶¶¶¶¶ KWT ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶  
¶. ¶¶¶¶¶¶¶¶¶¶, ¶¶¶¶¶¶¶¶¶¶¶¶.

```
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);
```

cube_surface_area	solid_surface_area
6	0

¶¶

[CG\\_Area](#), [CG\\_MakeSolid](#), [CG\\_IsSolid](#), [CG\\_Area](#)

### 8.2.8 CG\_Volume

CG\_Volume — 3 ¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶¶. ¶¶¶¶¶¶¶¶¶¶¶ (¶¶¶¶¶¶¶¶¶¶) 0 ¶¶¶¶¶¶¶¶¶¶.

## Synopsis

```
float CG_Volume(geometry geom1);
```

☒☒

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as CG\_3DVolume)

☒☒

When closed surfaces are created with WKT, they are treated as areal rather than solid. To make them solid, you need to use [CG\\_MakeSolid](#). Areal geometries have no volume. Here is an example to demonstrate.

```
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);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1
```

☒☒

[CG\\_3DArea](#), [CG\\_MakeSolid](#), [CG\\_IsSolid](#)

## 8.2.9 ST\_ForceLHR

ST\_ForceLHR — LHR(Left Hand Reverse; ☒☒☒☒) ☒☒☒☒☒☒☒☒.

### Synopsis

```
geometry ST_ForceLHR(geometry geom);
```

☒☒



**Warning**

**ST\_ForceLHR** is deprecated as of 3.5.0. Use **CG\_ForceLHR** instead.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**8.2.10 ST\_IsPlanar**

ST\_IsPlanar — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

**Synopsis**

boolean **ST\_IsPlanar**(geometry geom);

☒☒



**Warning**

**ST\_IsPlanar** is deprecated as of 3.5.0. Use **CG\_IsPlanar** instead.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒. ☒☒ 2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ 2.1 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**8.2.11 ST\_IsSolid**

ST\_IsSolid — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

**Synopsis**

boolean **ST\_IsSolid**(geometry geom1);

☒☒



**Warning**

**ST\_IsSolid** is deprecated as of 3.5.0. Use **CG\_IsSolid** instead.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**8.2.12 ST\_MakeSolid**

ST\_MakeSolid — ☒☒☒☒☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒☒☒☒☒☒. ☒☒☒☒☒☒☒☒☒, ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒ TIN ☒☒☒☒☒☒☒.

**Synopsis**

geometry **ST\_MakeSolid**(geometry geom1);

☒☒



**Warning**

**ST\_MakeSolid** is deprecated as of 3.5.0. Use **CG\_MakeSolid** instead.

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**8.2.13 ST\_Orientation**

ST\_Orientation — ☒☒☒☒☒ (orientation) ☒☒☒☒☒☒☒.

**Synopsis**

integer **ST\_Orientation**(geometry geom);





```
SELECT ST_3DArea(geom) As cube_surface_area,
 ST_3DArea(ST_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);

cube_surface_area | solid_surface_area
-----+-----
6 | 0
```

☒☒

[ST\\_Area](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#), [ST\\_Area](#)

### 8.2.15 ST\_Volume

ST\_Volume — 3D volume of a geometry. Returns 0 for non-3D geometries.

#### Synopsis

float **ST\_Volume**(geometry geom1);

☒☒



#### Warning

**ST\_Volume** is deprecated as of 3.5.0. Use **CG\_Volume** instead.

2.2.0 **ST\_Volume**.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST\_3DVolume)

☒☒

WKT `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)) )`, `ST_MakeSolid`

```

SELECT ST_Volume(geom) As cube_surface_vol,
 ST_Volume(ST_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);

cube_surface_vol | solid_surface_vol
-----+-----
0 | 1

```

☒☒

[ST\\_3DArea](#), [ST\\_MakeSolid](#), [ST\\_IsSolid](#)

## 8.3 SFCGAL Processing and Relationship Functions

### 8.3.1 CG\_Intersection

**CG\_Intersection** — Computes the intersection of two geometries

#### Synopsis

geometry **CG\_Intersection**( geometry geomA , geometry geomB );

☒☒

Computes the intersection of two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a geometry representing the intersection.

Availability: 3.5.0



This method needs SFCGAL backend.

☒☒☒☒

```

SELECT ST_AsText(CG_Intersection('LINESTRING(0 0, 5 5)', 'LINESTRING(5 0, 0 5)'));
 cg_intersection

POINT(2.5 2.5)
(1 row)

```

☒☒

[ST\\_3DIntersection](#), [ST\\_Intersection](#)

### 8.3.2 CG\_Intersects

CG\_Intersects — Tests if two geometries intersect (they have at least one point in common)

#### Synopsis

boolean **CG\_Intersects**( geometry geomA , geometry geomB );

☒☒

Returns true if two geometries intersect. Geometries intersect if they have any point in common.

Performed by the SFCGAL module



#### Note

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING (2 0, 0 2) '::geometry);
cg_intersects

f
(1 row)
SELECT CG_Intersects('POINT(0 0)::geometry, 'LINESTRING (0 0, 0 2) '::geometry);
cg_intersects

t
(1 row)
```

☒☒

[CG\\_3DIntersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

### 8.3.3 CG\_3DIntersects

CG\_3DIntersects — Tests if two 3D geometries intersect

**Synopsis**

boolean **CG\_3DIntersects**( geometry geomA , geometry geomB );

☒☒

Tests if two 3D geometries intersect. 3D geometries intersect if they have any point in common in the three-dimensional space.

Performed by the SFCGAL module

**Note**

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```
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)))');
 cg_3dintersects

 t
 (1 row)
```

☒☒

[CG\\_Intersects](#), [ST\\_3DIntersects](#), [ST\\_Intersects](#), [ST\\_Disjoint](#)

**8.3.4 CG\_Difference**

**CG\_Difference** — Computes the geometric difference between two geometries

**Synopsis**

geometry **CG\_Difference**( geometry geomA , geometry geomB );



Computes the geometric difference between two geometries. The resulting geometry is a set of points that are present in geomA but not in geomB.

Performed by the SFCGAL module

**Note**

NOTE: this function returns a geometry.

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).






















```
SELECT ST_AsText(CG_Difference('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::geometry, 'LINESTRING ↵
(0 0, 2 2)'::geometry));
cg_difference

POLYGON((0 0,1 0,1 1,0 1,0 0))
(1 row)
```



[ST\\_3DDifference](#), [ST\\_Difference](#)

### 8.3.5 ST\_3DDifference

ST\_3DDifference — 3                   .

#### Synopsis

geometry **ST\_3DDifference**(geometry geom1, geometry geom2);

**Warning**

[ST\\_3DDifference](#) is deprecated as of 3.5.0. Use [CG\\_3DDifference](#) instead.

geom2 ST\_3DDifference(geom1).

2.2.0

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.3.6 CG\_3DDifference

CG\_3DDifference — 3

#### Synopsis

geometry CG\_3DDifference(geometry geom1, geometry geom2);



#### Warning

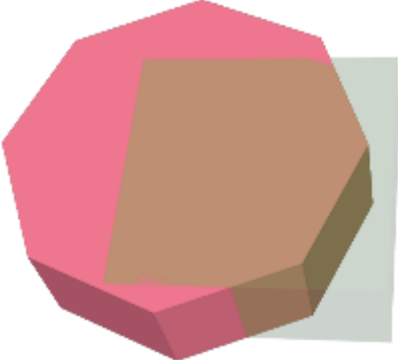
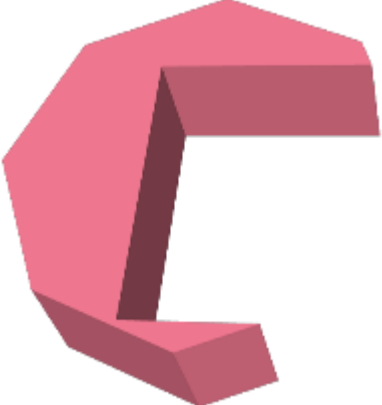
CG\_3DDifference is deprecated as of 3.5.0. Use ST\_3DDifference instead.

geom2 ST\_3DDifference(geom1).

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

PostGIS ST\_AsX3D 3 X3Dom HTML HTML

<pre>SELECT CG_Extrude(ST_Buffer(   ST_GeomFromText('POINT(100 90)'),   50, 'quad_segs=1'),0,0,30) AS geom2;</pre>  <p>3 <i>geom2</i></p>	<pre>SELECT CG_3DDifference(geom1,geom2)   AS geom1,   CG_Extrude(   ST_Buffer(ST_GeomFromText('POINT(80 80)'),   50, 'quad_segs=1'),0,0,30) AS geom2 ) As t;</pre>  <p><i>geom2</i></p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DUnion](#)

### 8.3.7 CG\_Distance

**CG\_Distance** — Computes the minimum distance between two geometries

#### Synopsis

double precision **CG\_Distance**( geometry geomA , geometry geomB );

Computes the minimum distance between two geometries.

Performed by the SFCGAL module



**Note**

NOTE: this function returns a double precision value representing the distance.

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```
SELECT CG_Distance('LINESTRING(0.0 0.0,-1.0 -1.0)', 'LINESTRING(3.0 4.0,4.0 5.0)');
 cg_distance

 2.0
(1 row)
```

☒☒

[CG\\_3DDistance](#), [CG\\_Distance](#)

### 8.3.8 CG\_3DDistance

CG\_3DDistance — Computes the minimum 3D distance between two geometries

#### Synopsis

double precision **CG\_3DDistance**( geometry geomA , geometry geomB );

☒☒

Computes the minimum 3D distance between two geometries.

Performed by the SFCGAL module



#### Note

NOTE: this function returns a double precision value representing the 3D distance.

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒☒☒

```
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))');
 cg_3ddistance

 1
(1 row)
```



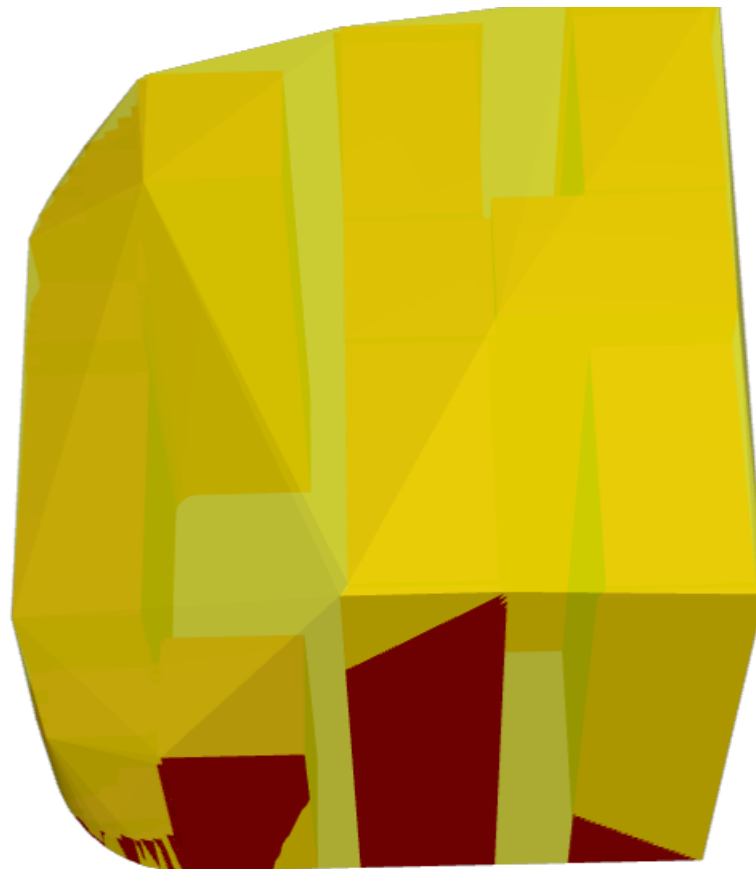


☒☒

```
SELECT ST_AsText(CG_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3 , 5 7 5, 6 3 5) ↔
'::geometry));
```

```
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)))
```

```
WITH f AS (SELECT i, CG_Extrude(geom, 0,0, i) AS geom
FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
)
SELECT CG_3DConvexHull(ST_Collect(f.geom))
FROM f;
```



*Original geometry overlaid with 3D convex hull*

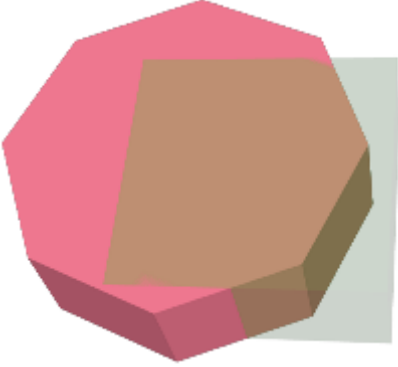
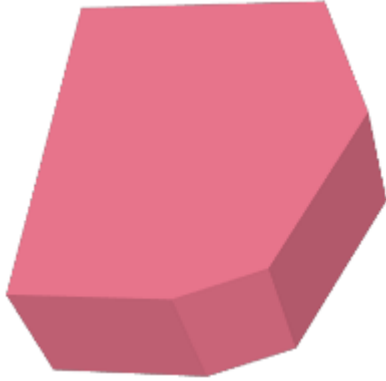
☒☒

[ST\\_Letters](#), [ST\\_AsX3D](#)

### 8.3.11 ST\_3DIntersection

ST\_3DIntersection — 3 ☒☒☒☒☒☒☒☒☒☒.



<pre>SELECT CG_Extrude(ST_Buffer(   ST_GeomFromText('POINT(100 90)'),   CG_Extrude(ST_Buffer(ST_GeomFromText('POINT(80 80)'),     50, '     quad_segs=1'),0,0,30) AS geom2;</pre>  <p>3 . geom2</p>	<pre>SELECT CG_3DIntersection(geom1,geom2)   50, '   quad_segs=2'),0,0,30) AS geom1,   quad_segs=2'),0,0,30) AS geom1,   CG_Extrude(ST_Buffer(ST_GeomFromText('PO   50, '   quad_segs=1'),0,0,30) AS geom2 ) As t;</pre>  <p>geom1 geom2</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3

```
SELECT ST_AsText(CG_3DIntersection(linestring, polygon)) As wkt
FROM ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS
linestring
CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

wkt

LINESTRING Z (1 1 8,0.5 0.5 8)
```

() Z

```
SELECT ST_AsText(CG_3DIntersection(
ST_GeomFromText('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))'::geometry))
```

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)))

(ST\_Dimension 3)

```
SELECT ST_AsText(CG_3DIntersection(CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1)
,0,0,30),
CG_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1),2,0,10)));
```

POLYHEDRALSURFACE Z (((13.33333333333333 13.33333333333333 10,20 20 0,20 20
10,13.33333333333333 13.33333333333333 10)),
((20 20 10,16.66666666666667 23.33333333333333 10,13.33333333333333 13.33333333333333
10,20 20 10)),

```
((20 20 0,16.6666666666667 23.3333333333333 10,20 20 10,20 20 0)),
((13.3333333333333 13.3333333333333 10,10 10 0,20 20 0,13.3333333333333 ←
 13.3333333333333 10)),
((16.6666666666667 23.3333333333333 10,12 28 10,13.3333333333333 13.3333333333333 ←
 10,16.6666666666667 23.3333333333333 10)),
((20 20 0,9.99999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
((10 10 0,9.99999999999995 30 0,20 20 0,10 10 0)),((13.3333333333333 ←
 13.3333333333333 10,12 12 10,10 10 0,13.3333333333333 13.3333333333333 10)),
((12 28 10,12 12 10,13.3333333333333 13.3333333333333 10,12 28 10)),
((16.6666666666667 23.3333333333333 10,9.99999999999995 30 0,12 28 ←
 10,16.6666666666667 23.3333333333333 10)),
((10 10 0,0 20 0,9.99999999999995 30 0,10 10 0)),
((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
((9.99999999999995 30 0,11 29 10,12 28 10,9.99999999999995 30 0)),((0 20 0,2 20 ←
 10,9.99999999999995 30 0,0 20 0)),
((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28 ←
 10,11 29 10,11 11 10,12 28 10)),
((9.99999999999995 30 0,2 20 10,11 29 10,9.99999999999995 30 0)),((11 11 10,11 29 ←
 10,2 20 10,11 11 10)))
```

### 8.3.13 CG\_Union

CG\_Union — Computes the union of two geometries

#### Synopsis

geometry **CG\_Union**( geometry geomA , geometry geomB );

☒☒

Computes the union of two geometries.

Performed by the SFCGAL module



**Note**

NOTE: this function returns a geometry representing the union.

Availability: 3.5.0



This method needs SFCGAL backend.

☒☒☒☒

```
SELECT CG_Union('POINT(.5 0)', 'LINESTRING(-1 0,1 0)');
 cg_union

LINESTRING(-1 0,0.5 0,1 0)
(1 row)
```

---

 ☒☒

[ST\\_3DUnion](#), [ST\\_AsBinary](#)

### 8.3.14 ST\_3DUnion

ST\_3DUnion — Perform 3D union.

#### Synopsis

geometry **ST\_3DUnion**(geometry geom1, geometry geom2);  
 geometry **ST\_3DUnion**(geometry set g1field);

 ☒☒
 

---



#### Warning

[ST\\_3DUnion](#) is deprecated as of 3.5.0. Use [CG\\_3DUnion](#) instead.

---

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

Availability: 3.3.0 aggregate variant was added

- ✔ This method needs SFCGAL backend.
- ✔ This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

**Aggregate variant:** returns a geometry that is the 3D union of a rowset of geometries. The ST\_3DUnion() function is an “aggregate” function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

### 8.3.15 CG\_3DUnion

CG\_3DUnion — Perform 3D union.

#### Synopsis

geometry **CG\_3DUnion**(geometry geom1, geometry geom2);  
 geometry **CG\_3DUnion**(geometry set g1field);

---



☒☒

[CG\\_Extrude](#), [ST\\_AsX3D](#), [CG\\_3DIntersection](#) [CG\\_3DDifference](#)

### 8.3.16 ST\_AlphaShape

ST\_AlphaShape — Computes an Alpha-shape enclosing a geometry

#### Synopsis

geometry **ST\_AlphaShape**(geometry geom, float alpha, boolean allow\_holes = false);

☒☒

**Warning**

[ST\\_AlphaShape](#) is deprecated as of 3.5.0. Use [CG\\_AlphaShape](#) instead.

---

Computes the [Alpha-Shape](#) of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the [convex hull](#).

### 8.3.17 CG\_AlphaShape

CG\_AlphaShape — Computes an Alpha-shape enclosing a geometry

#### Synopsis

geometry **CG\_AlphaShape**(geometry geom, float alpha, boolean allow\_holes = false);

☒☒

Computes the [Alpha-Shape](#) of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the [convex hull](#).

The “closeness of fit” is controlled by the alpha parameter, which can have values from 0 to infinity. Smaller alpha values produce more concave results. Alpha values greater than some data-dependent value produce the convex hull of the input.

---

**Note**

Following the CGAL implementation, the alpha value is the *square* of the radius of the disc used in the Alpha-Shape algorithm to “erode” the Delaunay Triangulation of the input points. See [CGAL Alpha-Shapes](#) for more information. This is different from the original definition of alpha-shapes, which defines alpha as the radius of the eroding disc.

---



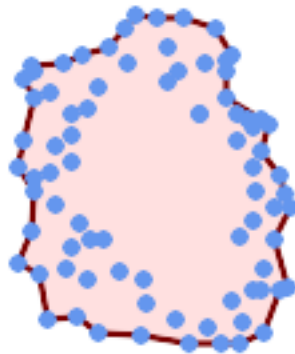
The computed shape does not contain holes unless the optional `allow_holes` argument is specified as `true`.

This function effectively computes a concave hull of a geometry in a similar way to `ST_ConcaveHull`, but uses CGAL and a different algorithm.

Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1.



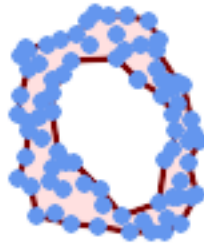
This method needs SFCGAL backend.



*Alpha-shape of a MultiPoint (same example As `CG_OptimalAlphaShape`)*

```
SELECT ST_AsText(CG_AlphaShape('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,80.2));
```

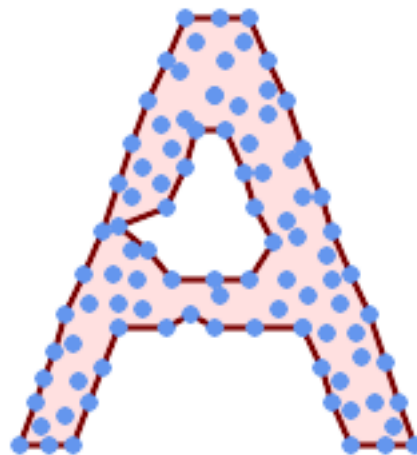
```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,
37 23,30 22,28 33,23 36,26 44,27 54,23 60,24 67,27 77,
24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,
64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53))
```



Alpha-shape of a MultiPoint, allowing holes (same example as [CG\\_OptimalAlphaShape](#))

```
SELECT ST_AsText(CG_AlphaShape('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, 100.1,true))
```

```
POLYGON((89 53,91 50,87 42,90 30,84 19,78 16,73 16,65 16,53 18,43 19,30 22,28 33,23 36,
26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,
76 88,75 84,83 72,85 71,88 58,89 53),(36 61,36 68,40 75,43 80,60 81,68 73,77 67,
81 60,82 54,81 47,78 43,76 27,62 22,54 32,44 42,38 46,36 61))
```



Alpha-shape of a MultiPoint, allowing holes (same example as [ST\\_ConcaveHull](#))

```
SELECT ST_AsText(CG_AlphaShape(
 '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,102.2, true));
```

```
POLYGON((26 20,32 36,35 45,39 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 64,114 64,99 64,90 69,81 64,63 64,57 ←
 49,52 36,46 20,37 20,26 20),
 (74 93,81 109,88 124,92 138,103 138,110 122,114 109,121 96,112 82,99 82,83 ←
 82,74 93))
```

☒☒

[ST\\_ConcaveHull, CG\\_OptimalAlphaShape](#)

### 8.3.18 CG\_ApproxConvexPartition

CG\_ApproxConvexPartition — Computes approximal convex partition of the polygon geometry

#### Synopsis

```
geometry CG_ApproxConvexPartition(geometry geom);
```

☒☒

Computes approximal convex partition of the polygon geometry (using a triangulation).

---

**Note**

A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. CG\_ApproxConvexPartition and CG\_GreeneApproxConvexPartition functions produce approximately optimal convex partitions. Both these functions produce convex decompositions by first decomposing the polygon into simpler polygons; CG\_ApproxConvexPartition uses a triangulation and CG\_GreeneApproxConvexPartition a monotone partition. These two functions both guarantee that they will produce no more than four times the optimal number of convex pieces but they differ in their runtime complexities. Though the triangulation-based approximation algorithm often results in fewer convex pieces, this is not always the case.

---

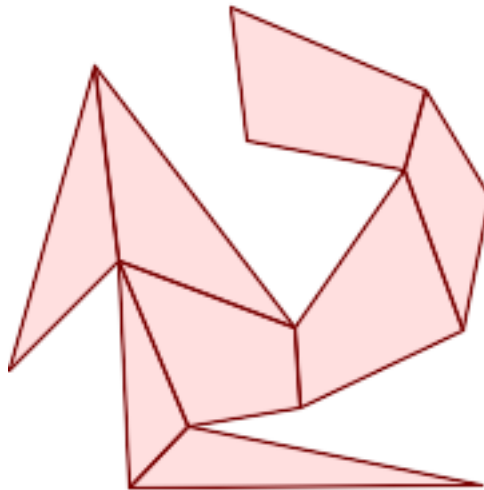
 Note!

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

✔ This method needs SFCGAL backend.

☒☒



*Approximal Convex Partition (same example As [CG\\_YMonotonePartition](#), [CG\\_GreeneApproxConvexPartition](#) and [CG\\_OptimalConvexPartition](#))*

```
SELECT ST_AsText(CG_ApproxConvexPartition('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));
```

```
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)))
```

☒☒

[CG\\_YMonotonePartition](#), [CG\\_GreeneApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

### 8.3.19 ST\_ApproximateMedialAxis

ST\_ApproximateMedialAxis — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

```
geometry ST_ApproximateMedialAxis(geometry geom);
```

---

 ☒☒
**Warning**

`ST_ApproximateMedialAxis` is deprecated as of 3.5.0. Use `CG_ApproximateMedialAxis` instead.

---

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

2.2.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.



This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.3.20 `CG_ApproximateMedialAxis`

`CG_ApproximateMedialAxis` — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **`CG_ApproximateMedialAxis`**(geometry geom);

☒☒

Return an approximate medial axis for the areal input based on its straight skeleton. Uses an SFCGAL specific API when built against a capable version (1.2.0+). Otherwise the function is just a wrapper around `CG_StraightSkeleton` (slower case).

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.

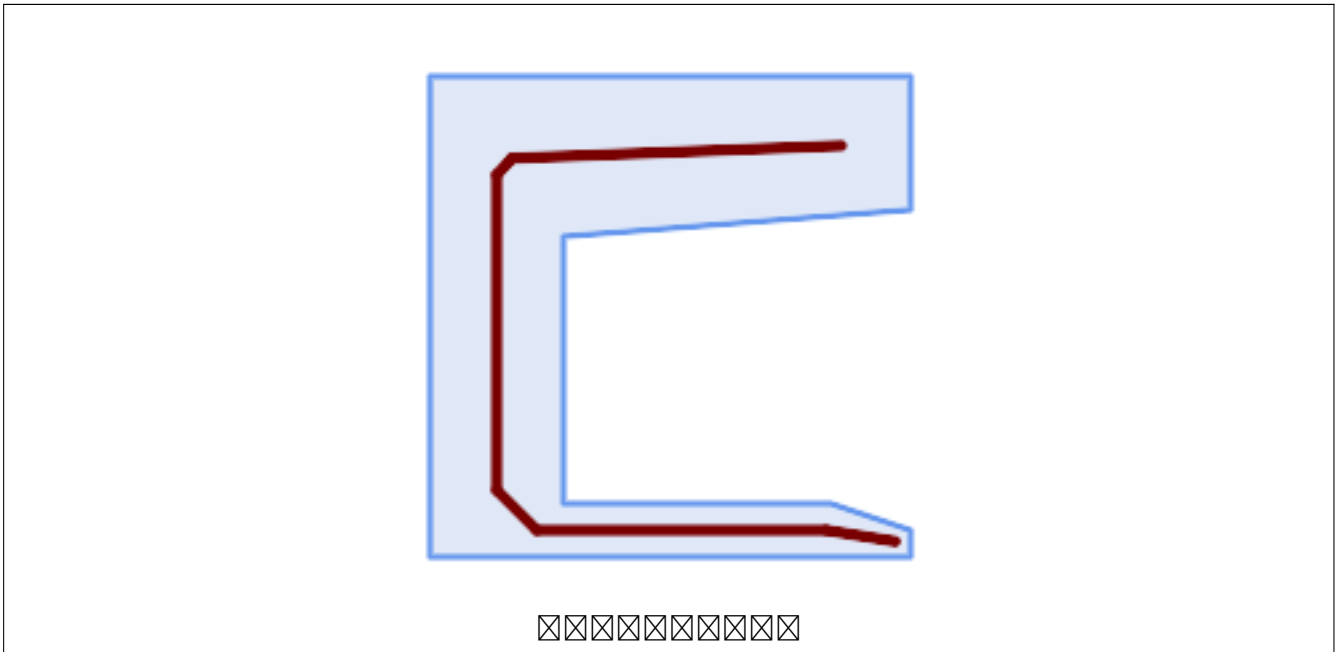


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT CG_ApproximateMedialAxis(ST_GeomFromText('POLYGON ((190 190, 10 190, 10 10, 190 10, ↵
 190 20, 160 30, 60 30, 60 130, 190 140, 190 190))'));
```

---



☒☒

[CG\\_StraightSkeleton](#)

### 8.3.21 ST\_ConstrainedDelaunayTriangles

ST\_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

#### Synopsis

```
geometry ST_ConstrainedDelaunayTriangles(geometry g1);
```

☒☒



**Warning**

[ST\\_ConstrainedDelaunayTriangles](#) is deprecated as of 3.5.0. Use [CG\\_ConstrainedDelaunayTriangles](#) instead.

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.

✔ This method needs SFCGAL backend.

2.1.0 ☒☒☒☒☒☒☒☒☒☒.

✔ This function supports 3d and will not drop the z-index.

### 8.3.22 CG\_ConstrainedDelaunayTriangles

CG\_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

#### Synopsis

```
geometry CG_ConstrainedDelaunayTriangles(geometry g1);
```



#### Warning

**CG\_ConstrainedDelaunayTriangles** is deprecated as of 3.5.0. Use **CG\_ConstrainedDelaunayTriangles** instead.

---

Return a **Constrained Delaunay triangulation** around the vertices of the input geometry. Output is a TIN.



This method needs SFCGAL backend.

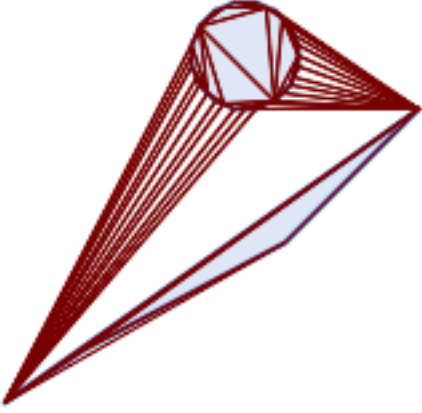
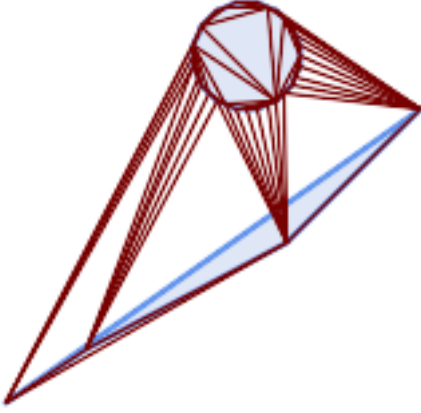
2.1.0          .



This function supports 3d and will not drop the z-index.

---

 <p><i>CG_ConstrainedDelaunayTriangles</i> of 2 polygons</p> <pre>select CG_ConstrainedDelaunayTriangles(   ST_Union(     POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),     ST_Buffer('POINT(110 170)::geometry, 20)   ) );</pre>	 <p><i>ST_DelaunayTriangles</i> of 2 polygons. Triangle edges cross polygon boundaries.</p> <pre>select ST_DelaunayTriangles(   ST_Union(     POLYGON((175 150, 20 40, 50 60, 125 100, 175 150)),     ST_Buffer('POINT(110 170)::geometry, 20)   ) );</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

☒☒

*ST\_DelaunayTriangles, ST\_TriangulatePolygon, CG\_Tessellate, ST\_ConcaveHull, ST\_Dump*

### 8.3.23 ST\_Extrude

ST\_Extrude — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_Extrude**(geometry geom, float x, float y, float z);

☒☒



**Warning**

*ST\_Extrude* is deprecated as of 3.5.0. Use *CG\_Extrude* instead.



2.1.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.3.24 CG\_Extrude

CG\_Extrude —

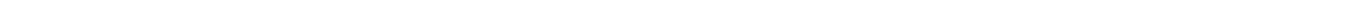
#### Synopsis




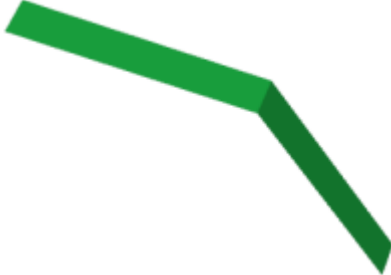
geometry **CG\_Extrude**(geometry geom, float x, float y, float z);

Availability: 3.5.0

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

PostGIS [ST\\_AsX3D](#) 3 [X3Dom HTML](#) [HTML](#)



<pre>SELECT ST_Buffer(ST_GeomFromText('POINT (100 90)'),   50, 'quad_segs=2'),0,0,30);</pre>  <p>3D visualization of the buffer.</p>	<pre>CG_Extrude(ST_Buffer(ST_GeomFromText('POINT (100 90)'),   50, 'quad_segs=2'),0,0,30);</pre>  <p>Z 30 PolyhedralSurfaceZ.</p>
<pre>SELECT ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  <p>3D visualization of the line string.</p>	<pre>SELECT CG_Extrude(   ST_GeomFromText('LINESTRING(50 50, 100 90, 95 150)')</pre>  <p>Z PolyhedralSurfaceZ.</p>

☒☒

ST\_AsX3D, CG\_ExtrudeStraightSkeleton

### 8.3.25 CG\_ExtrudeStraightSkeleton

CG\_ExtrudeStraightSkeleton — Straight Skeleton Extrusion

#### Synopsis

geometry **CG\_ExtrudeStraightSkeleton**(geometry geom, float roof\_height, float body\_height = 0);

☒☒

Computes an extrusion with a maximal height of the polygon geometry.

#### Note



Perhaps the first (historically) use-case of straight skeletons: given a polygonal roof, the straight skeleton directly gives the layout of each tent. If each skeleton edge is lifted from the plane a height equal to its offset distance, the resulting roof is "correct" in that water will always fall down to the contour edges (the roof's border), regardless of where it falls on the roof. The function computes this extrusion aka "roof" on a polygon. If the argument body\_height > 0, so the polygon is extruded like with CG\_Extrude(polygon, 0, 0, body\_height). The result is an union of these polyhedralsurfaces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

☒☒

```
SELECT ST_AsText(CG_ExtrudeStraightSkeleton('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))', 3.0, 2.0));
```

```
POLYHEDRALSURFACE Z (((0 0 0,0 4 0,4 4 0,4 5 0,5 5 0,5 0 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 ←
0,1 1 0)),((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 5 2,4 5 0,4 4 0)),((4 5 0,4 5 2,5 5 2,5 5 0,4 5 0)),((5 5 0,5 5 2,5 0 2,5 0 0,5 5 ←
0)),((5 0 0,5 0 2,0 0 2,0 0 0,5 0 0)),((1 1 0,1 1 2,2 1 2,2 1 0,1 1 0)),((2 1 0,2 1 2,2 ←
2 2,2 2 0,2 1 0)),((2 2 0,2 2 2,1 2 2,1 2 0,2 2 0)),((1 2 0,1 2 2,1 1 2,1 1 0,1 2 0) ←
,((4 5 2,5 5 2,4 4 2,4 5 2)),((2 1 2,5 0 2,0 0 2,2 1 2)),((5 5 2,5 0 2,4 4 2,5 5 2)),((2 ←
1 2,0 0 2,1 1 2,2 1 2)),((1 2 2,1 1 2,0 0 2,1 2 2)),((0 4 2,2 2 2,1 2 2,0 4 2)),((0 4 ←
2,1 2 2,0 0 2,0 4 2)),((4 4 2,5 0 2,2 2 2,4 4 2)),((4 4 2,2 2 2,0 4 2,4 4 2)),((2 2 2,5 ←
0 2,2 1 2,2 2 2)),((0.5 2.5 2.5,0 0 2,0.5 0.5 2.5,0.5 2.5 2.5)),((1 3 3,0 4 2,0.5 2.5 ←
2.5,1 3 3)),((0.5 2.5 2.5,0 4 2,0 0 2,0.5 2.5 2.5)),((2.5 0.5 2.5,5 0 2,3.5 1.5 3.5,2.5 ←
0.5 2.5)),((0 0 2,5 0 2,2.5 0.5 2.5,0 0 2)),((0.5 0.5 2.5,0 0 2,2.5 0.5 2.5,0.5 0.5 2.5) ←
),((4.5 3.5 2.5,5 2,4.5 4.5 2.5,4.5 3.5 2.5)),((3.5 2.5 3.5,3.5 1.5 3.5,4.5 3.5 ←
2.5,3.5 2.5 3.5)),((4.5 3.5 2.5,5 0 2,5 2,4.5 3.5 2.5)),((3.5 1.5 3.5,5 0 2,4.5 3.5 ←
2.5,3.5 1.5 3.5)),((5 5 2,4 5 2,4.5 4.5 2.5,5 5 2)),((4.5 4.5 2.5,4 4 2,4.5 3.5 2.5,4.5 ←
4.5 2.5)),((4.5 4.5 2.5,4 5 2,4 4 2,4.5 4.5 2.5)),((3 3 3,0 4 2,1 3 3,3 3 3)),((3.5 2.5 ←
3.5,4.5 3.5 2.5,3 3 3,3.5 2.5 3.5)),((3 3 3,4 4 2,0 4 2,3 3 3)),((4.5 3.5 2.5,4 4 2,3 3 ←
3,4.5 3.5 2.5)),((2 1 2,1 1 2,0.5 0.5 2.5,2 1 2)),((2.5 0.5 2.5,2 1 2,0.5 0.5 2.5,2.5 ←
0.5 2.5)),((1 1 2,1 2 2,0.5 2.5 2.5,1 1 2)),((0.5 0.5 2.5,1 1 2,0.5 2.5 2.5,0.5 0.5 2.5) ←
),((1 3 3,2 2 2,3 3 3,1 3 3)),((0.5 2.5 2.5,1 2 2,1 3 3,0.5 2.5 2.5)),((1 3 3,1 2 2,2 2 ←
2,1 3 3)),((2 2 2,2 1 2,2.5 0.5 2.5,2 2 2)),((3.5 2.5 3.5,3 3 3,3.5 1.5 3.5,3.5 2.5 3.5) ←
),((3.5 1.5 3.5,2 2 2,2.5 0.5 2.5,3.5 1.5 3.5)),((3 3 3,2 2 2,3.5 1.5 3.5,3 3 3)))
```

☒☒

[ST\\_Extrude](#), [CG\\_StraightSkeleton](#)

### 8.3.26 CG\_GreeneApproxConvexPartition

CG\_GreeneApproxConvexPartition — Computes approximal convex partition of the polygon geometry

#### Synopsis

geometry **CG\_GreeneApproxConvexPartition**(geometry geom);

☒☒

Computes approximal monotone convex partition of the polygon geometry.

---

#### Note

A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. CG\_ApproxConvexPartition and CG\_GreeneApproxConvexPartition functions produce approximately optimal convex partitions. Both these functions produce convex decompositions by first decomposing the polygon into simpler polygons; CG\_ApproxConvexPartition uses a triangulation and CG\_GreeneApproxConvexPartition a monotone partition. These two functions both guarantee that they will produce no more than four times the optimal number of convex pieces but they differ in their runtime complexities. Though the triangulation-based approximation algorithm often results in fewer convex pieces, this is not always the case.

---

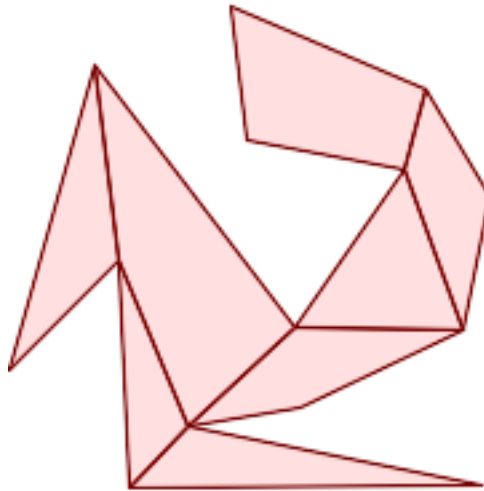
Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.

☒☒



*Greene Approximal Convex Partition (same example As [CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#) and [CG\\_OptimalConvexPartition](#))*

```
SELECT ST_AsText(CG_GreeneApproxConvexPartition('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));

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)))
```

☒☒

[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_OptimalConvexPartition](#)

### 8.3.27 ST\_MinkowskiSum

ST\_MinkowskiSum — ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

#### Synopsis

geometry **ST\_MinkowskiSum**(geometry geom1, geometry geom2);

☒☒



#### Warning

**ST\_MinkowskiSum** is deprecated as of 3.5.0. Use [CG\\_MinkowskiSum](#) instead.

CGAL 2D MinkowskiSum

CGAL 2D MinkowskiSum (motion planning) CAD(computer-aided design) Wikipedia Minkowski addition

CGAL 2D MinkowskiSum (CGAL, CGAL, CGAL) CGAL 3D MinkowskiSum, Z CGAL 0 CGAL 2D MinkowskiSum

CGAL 2D MinkowskiSum

2.1.0

This method needs SFCGAL backend.

### 8.3.28 CG\_MinkowskiSum

CG\_MinkowskiSum — CGAL 2D MinkowskiSum

#### Synopsis

geometry CG\_MinkowskiSum(geometry geom1, geometry geom2);

CG

CGAL 2D MinkowskiSum

CGAL 2D MinkowskiSum (motion planning) CAD(computer-aided design) Wikipedia Minkowski addition

CGAL 2D MinkowskiSum (CGAL, CGAL, CGAL) CGAL 3D MinkowskiSum, Z CGAL 0 CGAL 2D MinkowskiSum

CGAL 2D MinkowskiSum

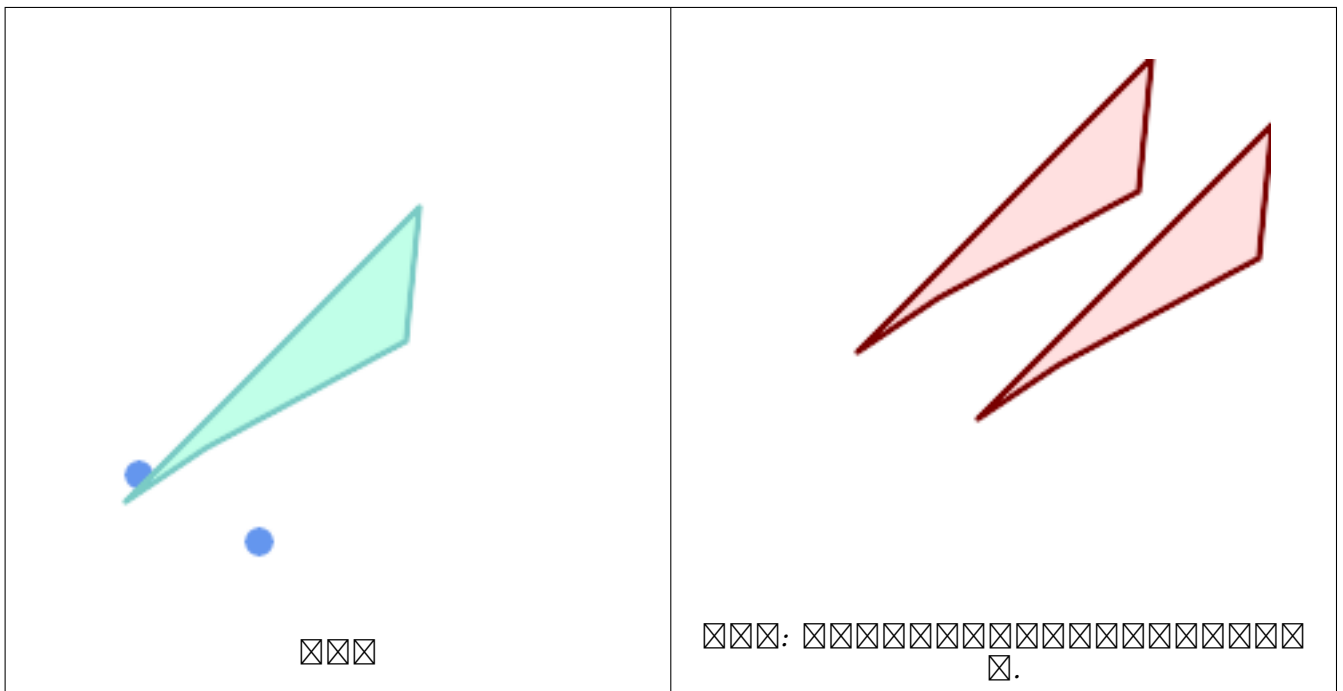
Availability: 3.5.0

This method needs SFCGAL backend.

CG

CGAL 2D MinkowskiSum





```
SELECT CG_MinkowskiSum(mp, poly)
 FROM (SELECT 'MULTIPOINT(25 50,70 25)::geometry As mp,
 'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))::geometry As poly
) As foo

-- wkt --
MULTIPOLYGON(
 ((70 115,100 135,175 175,225 225,70 115)),
 ((120 65,150 85,225 125,275 175,120 65))
)
```

### 8.3.29 ST\_OptimalAlphaShape

ST\_OptimalAlphaShape — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

#### Synopsis

geometry **ST\_OptimalAlphaShape**(geometry geom, boolean allow\_holes = false, integer nb\_components = 1);

☒☒



**Warning**

ST\_OptimalAlphaShape is deprecated as of 3.5.0. Use CG\_OptimalAlphaShape instead.



Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of  $\alpha$  chosen so that:

1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.3.0 - requires SFCGAL  $\geq$  1.4.1.



This method needs SFCGAL backend.

### 8.3.30 CG\_OptimalAlphaShape

`CG_OptimalAlphaShape` — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

#### Synopsis

```
geometry CG_OptimalAlphaShape(geometry geom, boolean allow_holes = false, integer nb_components = 1);
```



Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of  $\alpha$  chosen so that:

1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

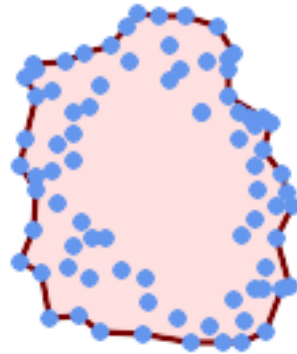
The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.5.0 - requires SFCGAL  $\geq$  1.4.1.



This method needs SFCGAL backend.

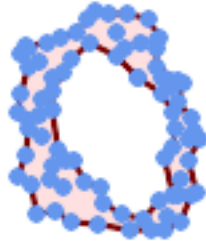
☒☒



*Optimal alpha-shape of a MultiPoint (same example as [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_OptimalAlphaShape('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));

POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
 33,23 36,
 26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
 97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



*Optimal alpha-shape of a MultiPoint, allowing holes (same example as [CG\\_AlphaShape](#))*

```
SELECT ST_AsText(CG_OptimalAlphaShape('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, allow_holes => true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28 ←
33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 ←
97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43 ←
80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54 ←
32,48 34,44 42,38 46,36 61))
```



[ST\\_ConcaveHull](#), [CG\\_AlphaShape](#)

### 8.3.31 CG\_OptimalConvexPartition

`CG_OptimalConvexPartition` — Computes an optimal convex partition of the polygon geometry

#### Synopsis

geometry **CG\_OptimalConvexPartition**(geometry geom);



Computes an optimal convex partition of the polygon geometry.



**Note**

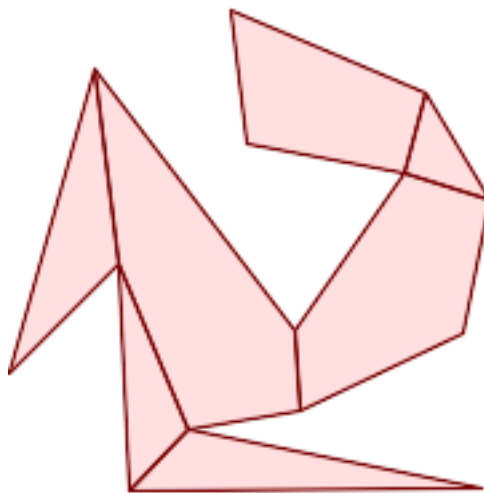
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. `CG_OptimalConvexPartition` produces a partition that is optimal in the number of pieces.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



*Optimal Convex Partition (same example As [CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#) and [CG\\_GreeneApproxConvexPartition](#))*

```
SELECT ST_AsText(CG_OptimalConvexPartition('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));
```

```
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)))
```



[CG\\_YMonotonePartition](#), [CG\\_ApproxConvexPartition](#), [CG\\_GreeneApproxConvexPartition](#)

### 8.3.32 CG\_StraightSkeleton

`CG_StraightSkeleton` — ☒☒☒☒☒☒☒☒☒ (straight skeleton) ☒☒☒☒☒☒.

**Synopsis**

geometry **CG\_StraightSkeleton**(geometry geom, boolean use\_distance\_as\_m = false);

☒☒

Availability: 3.5.0

Requires SFCGAL >= 1.3.8 for option use\_distance\_as\_m

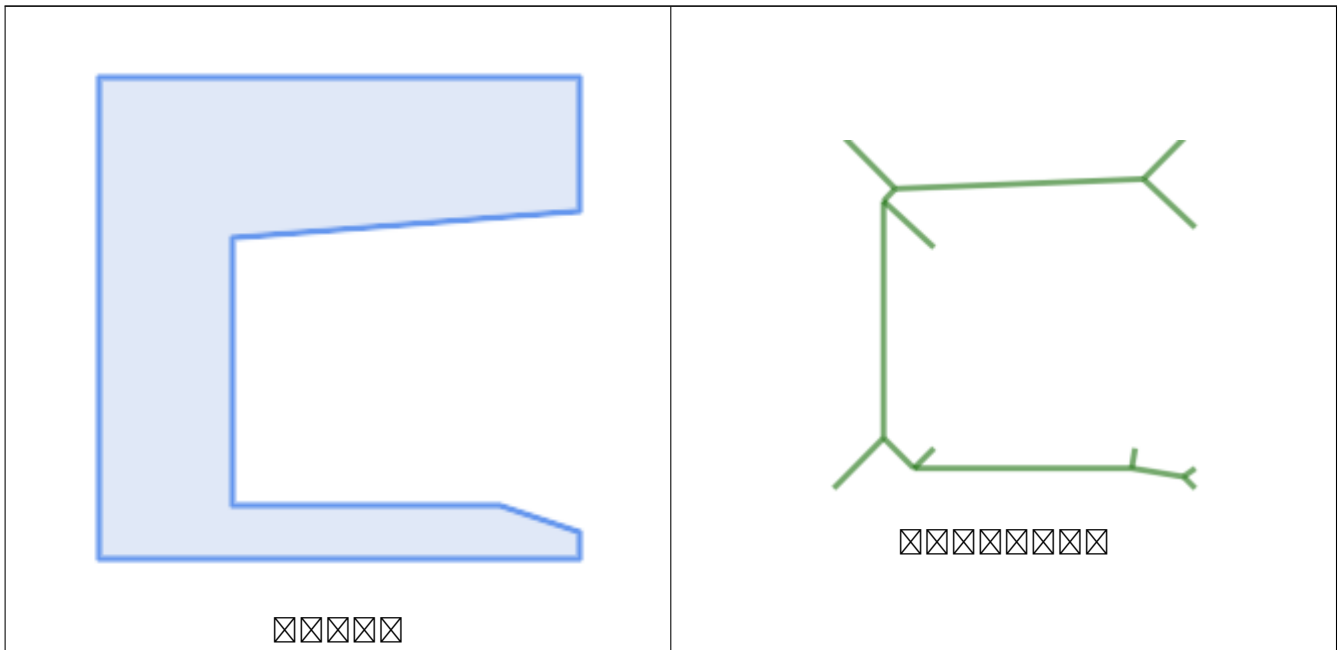
- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT CG_StraightSkeleton(ST_GeomFromText('POLYGON ((190 190, 10 190, 10 10, 190 10, 190 ←
20, 160 30, 60 30, 60 130, 190 140, 190 190))'));
```

```
ST_AsText(CG_StraightSkeleton('POLYGON((0 0,1 0,1 1,0 1,0 0))', true);
```

```
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))
```



☒☒

**CG\_ExtrudeStraightSkeleton**

### 8.3.33 ST\_StraightSkeleton

ST\_StraightSkeleton — ☒☒☒☒☒☒☒☒☒ (straight skeleton) ☒☒☒☒☒☒☒.

### Synopsis

geometry **ST\_StraightSkeleton**(geometry geom);

☒☒



#### Warning

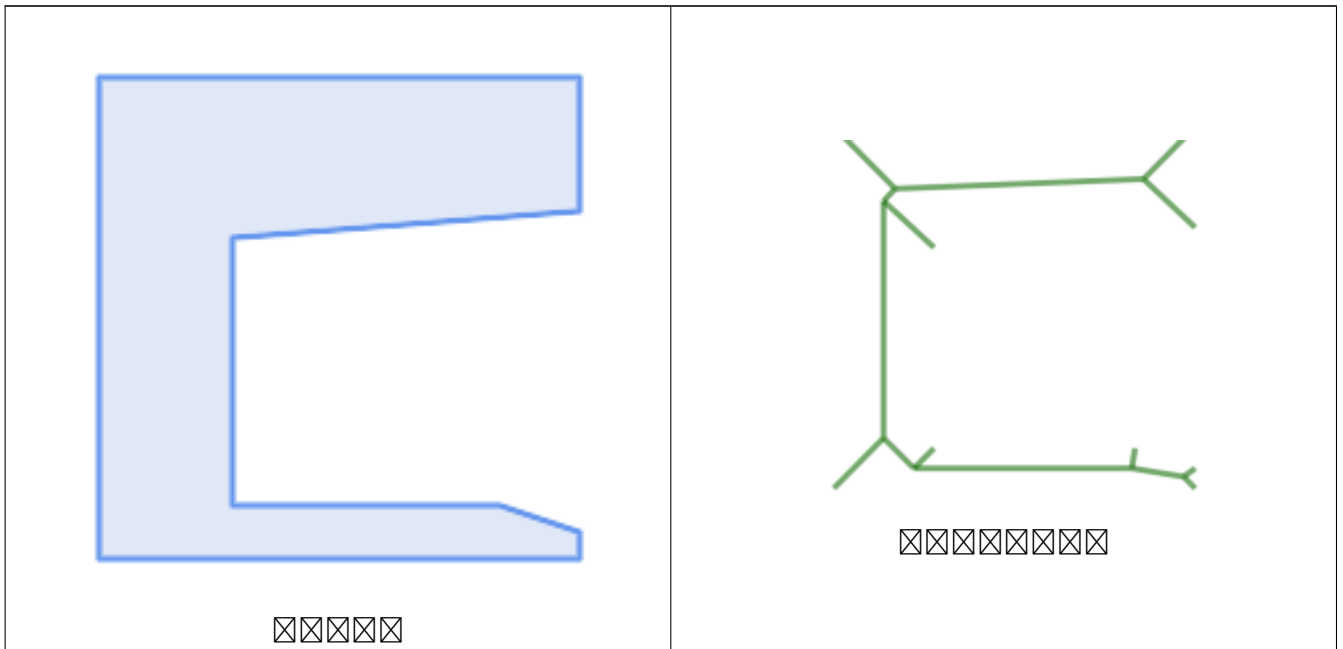
**ST\_StraightSkeleton** is deprecated as of 3.5.0. Use **CG\_StraightSkeleton** instead.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

- ✔ This method needs SFCGAL backend.
- ✔ This function supports 3d and will not drop the z-index.
- ✔ This function supports Polyhedral surfaces.
- ✔ This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON ((190 190, 10 190, 10 10, 190 10, 190 20, 160 30, 60 30, 60 130, 190 140, 190 190))'));
```



☒☒

**CG\_ExtrudeStraightSkeleton**

### 8.3.34 ST\_Tesselate

ST\_Tesselate — 將多邊形 (tessellation) 轉換為 TIN 格式。返回 TIN 格式的多邊形。

#### Synopsis

geometry ST\_Tesselate(geometry geom);

返回



#### Warning

ST\_Tesselate 已於 3.5.0 版本中棄用。請使用 CG\_Tesselate 代替。





[格式] 將多邊形 (tessellation) 轉換為 TIN 格式。返回 TIN 格式的多邊形。



#### Note

ST\_TriangulatePolygon 與此功能類似，但它返回一個多邊形集合，而不是 TIN，並且僅適用於 2D 幾何。

2.1.0 版本中新增。

-  This method needs SFCGAL backend.
-  This function supports 3d and will not drop the z-index.
-  This function supports Polyhedral surfaces.
-  This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

### 8.3.35 CG\_Tesselate

CG\_Tesselate — 將多邊形 (tessellation) 轉換為 TIN 格式。返回 TIN 格式的多邊形。

#### Synopsis

geometry CG\_Tesselate(geometry geom);

返回

[格式] 將多邊形 (tessellation) 轉換為 TIN 格式。返回 TIN 格式的多邊形。

**Note**

**ST\_TriangulatePolygon** does similar to this function except that it returns a geometry collection of polygons instead of a TIN and also only works with 2D geometries.

---

Availability: 3.5.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.


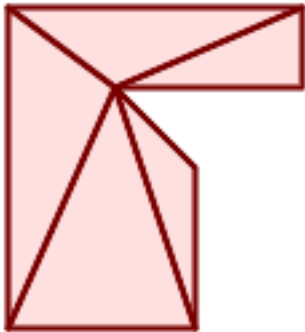


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).







<pre>SELECT 'POLYGON (( 10 190, 10 70, 80 70, ← 80 130, 50 160, 120 160, 120 190, 10 190 )</pre>  <p style="text-align: center;">☒☒☒☒☒</p>	<pre>SELECT     CG_Tesselate(' ← POLYGON (( 10 190, 10 70, 80 70, 80 130, 50 160, ; ST_AsText ☒☒☒: ::geometry; TIN(( (80 130, 50 160, 80 70, 80 130)), ((50 ← 160, 10 190, 10 70, 50 160)), ((80 70, 50 160, 10 70, 80 ← 70)), ((120 160, 120 190, 50 160, 120 160)), ((120 190, 10 190, 50 ← 160, 120 190)))</pre>  <p style="text-align: center;">☒☒☒☒☒☒☒☒</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

☒☒

[CG\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#)

### 8.3.36 CG\_Triangulate

CG\_Triangulate — Triangulates a polygonal geometry

#### Synopsis

```
geometry CG_Triangulate(geometry geom);
```

☒☒

Triangulates a polygonal geometry.

Performed by the SFCGAL module

**Note**

NOTE: this function returns a geometry representing the triangulated result.

Availability: 3.5.0



This method needs SFCGAL backend.



```
SELECT CG_Triangulate('POLYGON((0.0 0.0,1.0 0.0,1.0 1.0,0.0 1.0,0.0 0.0),(0.2 0.2,0.2 0.8,0.8 0.8,0.8 0.2,0.2 0.2))');
 cg_triangulate

 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)))
(1 row)
```



[CG\\_ConstrainedDelaunayTriangles](#), [ST\\_DelaunayTriangles](#), [ST\\_TriangulatePolygon](#)

### 8.3.37 CG\_Visibility

**CG\_Visibility** — Compute a visibility polygon from a point or a segment in a polygon geometry

#### Synopsis

```
geometry CG_Visibility(geometry polygon, geometry point);
geometry CG_Visibility(geometry polygon, geometry pointA, geometry pointB);
```



Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0



This method needs SFCGAL backend.



This function supports 3d and will not drop the z-index.



This function supports Polyhedral surfaces.

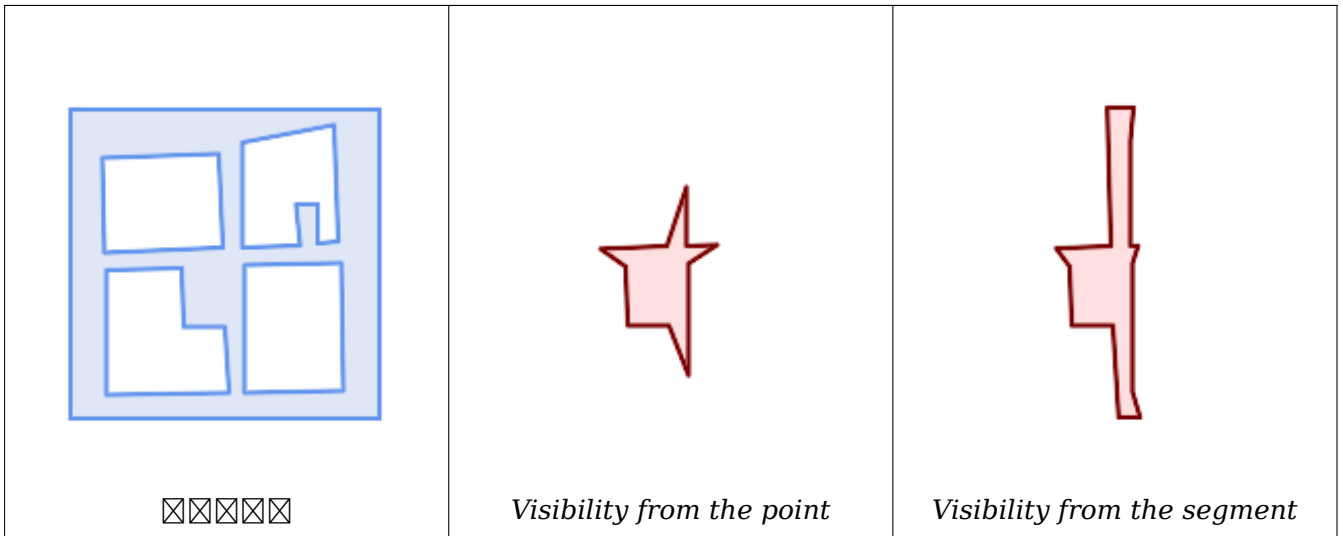


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

☒☒

```
SELECT CG_Visibility('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, 'POINT(91 87)'::geometry);
```

```
SELECT CG_Visibility('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, 'POINT(78.5 68)'::geometry, 'POINT(98.5 68)'::geometry);
```



### 8.3.38 CG\_YMonotonePartition

CG\_YMonotonePartition — Computes y-monotone partition of the polygon geometry

#### Synopsis

```
geometry CG_YMonotonePartition(geometry geom);
```

☒☒

Computes y-monotone partition of the polygon geometry.

#### Note



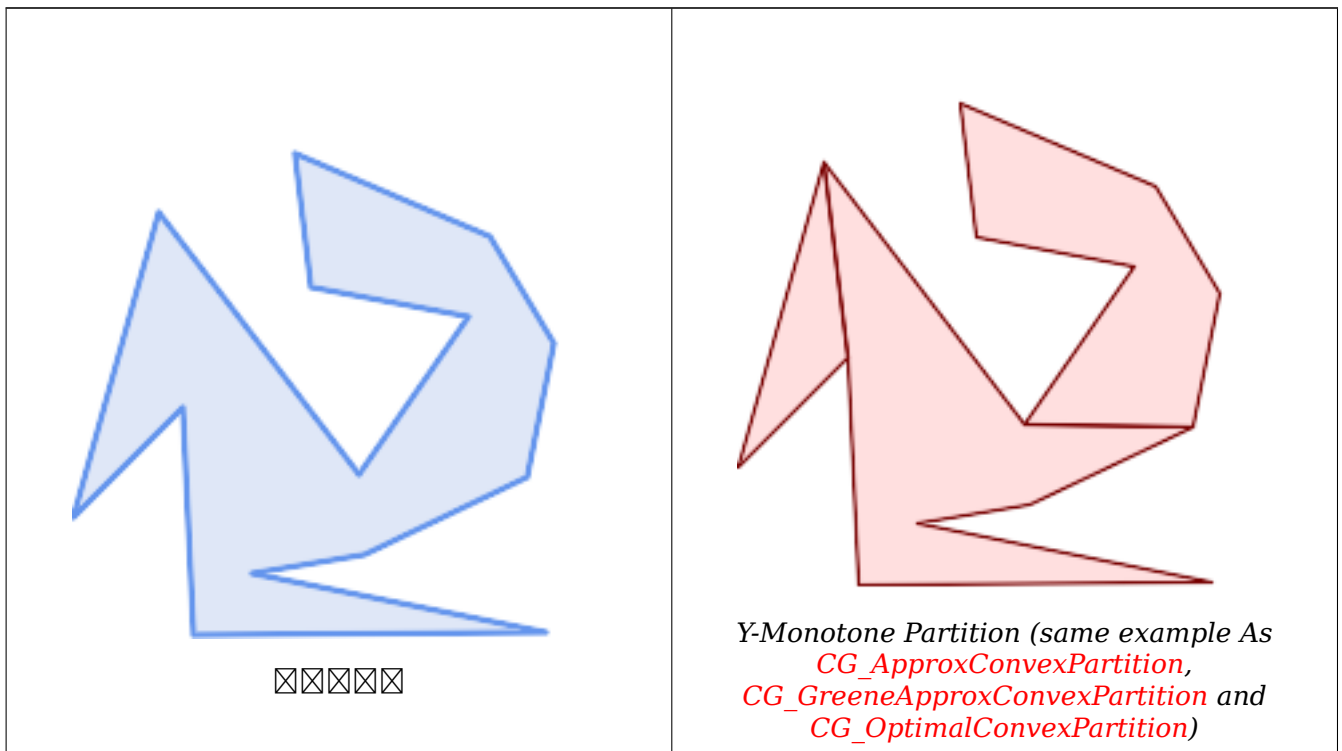
A partition of a polygon P is a set of polygons such that the interiors of the polygons do not intersect and the union of the polygons is equal to the interior of the original polygon P. A y-monotone polygon is a polygon whose vertices  $v_1, \dots, v_n$  can be divided into two chains  $v_1, \dots, v_k$  and  $v_k, \dots, v_n, v_1$ , such that any horizontal line intersects either chain at most once. This algorithm does not guarantee a bound on the number of polygons produced with respect to the optimal number.

Availability: 3.5.0 - requires SFCGAL >= 1.5.0.

Requires SFCGAL >= 1.5.0

✔ This method needs SFCGAL backend.

☒☒



```
SELECT ST_AsText(CG_YMonotonePartition('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));
```

```
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)))
```

☒☒

**CG\_ApproxConvexPartition, CG\_GreeneApproxConvexPartition, CG\_OptimalConvexPartition**

# Chapter 9

## Topology

PostGIS (face), (edge), (node) .

Sandro Santilli’s presentation at PostGIS Day Paris 2011 conference gives a good synopsis of PostGIS Topology and where it is headed [Topology with PostGIS 2.0 slide deck](#).

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in [PostGIS Topology PGConf EU 2012](#).

GIS [US Census Topologically Integrated Geographic Encoding and Referencing System \(TIGER\)](#) . PostGIS [Topology\\_Load\\_Tiger](#) .

PostGIS PostGIS , PostGIS . PostGIS 2.0.0 , , , SQL-MM .

[PostGIS Topology Wiki](#) .

topology .

SQL/MM ST\_ , PostGIS .

Topology support is build by default starting with PostGIS 2.0, and can be disabled specifying --without-topology configure option at build time as described in [Chapter 2](#)

### 9.1

#### 9.1.1 getfaceedges\_returntype

getfaceedges\_returntype — A composite type that consists of a sequence number and an edge number.

A composite type that consists of a sequence number and an edge number. This is the return type for ST\_GetFaceEdges and GetNodeEdges functions.

1. sequence : SRID topology.topology .
2. edge : .

### 9.1.2 TopoGeometry

TopoGeometry — A composite type representing a topologically defined geometry.

TopoGeometry, ID TopoGeometry topology\_id, layer\_id, id, type

1. topology\_id: SRID topology.topology
2. layer\_id: TopoGeometry layer\_id topology\_id layer\_id topology.layers (unique reference)
3. id: ,
4. 1 4 type. 1: [ ] , 2: [ ] , 3: [ ] , 4: [ ] .


#### CreateTopoGeom

### 9.1.3 validatetopology\_returntype

validatetopology\_returntype — A composite type that consists of an error message and id1 and id2 to denote location of error. This is the return type for ValidateTopology.

2 ValidateTopology ID id1 id2

1. error (varchar): coincident nodes(), edge crosses node(), edge not simple(), edge end node geometry mismatch(), edge start node geometry mismatch(), face overlaps face(), face within face()
2. id1: (edge)/ (face)/ (node)
3. id2: 2 ,

### ValidateTopology

## 9.2

### 9.2.1 TopoElement

TopoElement — TopoGeometry 2

TopoGeometry 1 2

TopoGeometry, (topological primitive) (1: node, 2: edge, 3: face) TopoGeometry TopoGeometry

**Note**  
 TopoGeometry, TopoGeometry, TopoGeometry topology.layer

```
SELECT te[1] AS id, te[2] AS type FROM
(SELECT ARRAY[1,2]::topology.topoelement AS te) f;
id | type
---+-----
1 | 2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
te

{1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoelement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR: value for domain topology.topoelement violates check constraint "dimensions"
```

GetTopoGeomElements, TopoElementArray, TopoGeometry, TopoGeom\_addElement, TopoGeom\_remElement

### 9.2.2 TopoElementArray

TopoElementArray — An array of TopoElement objects.





`topology.layer`

[child\_layer] (NULL) , ( ) TopoGeometry . (child\_layer TopoGeometry ) TopoGeometry .

(AddTopoGeometryColumn ID ) TopoGeometry .

Valid feature\_types are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ←
-- schema_name would be different
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');
```

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

[DropTopoGeometryColumn](#), [toTopoGeom](#), [CreateTopology](#), [CreateTopoGeom](#)

### 9.3.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Renames a topogeometry column

#### Synopsis

`topology.layer` **RenameTopoGeometryColumn**(regclass layer\_table, name feature\_column, name new\_name)

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

☒☒

[AddTopoGeometryColumn, RenameTopology](#)

### 9.3.3 DropTopology

**DropTopology** — `DropTopology`: Removes a topology from the database, dropping the `topology.topology` table, the `geometry_columns` table, and the `geometry_columns` table.

#### Synopsis

```
integer DropTopology(varchar topology_schema_name);
```

☒☒

`DropTopology` removes a topology from the database, dropping the `topology.topology` table, the `geometry_columns` table, and the `geometry_columns` table. It also drops the `geometry_columns` table. It also drops the `geometry_columns` table.

Availability: 1.1

☒☒

```
ma_topo DropTopology topology.topology geometry_columns
```

```
SELECT topology.DropTopology('ma_topo');
```

☒☒

[DropTopoGeometryColumn](#)

### 9.3.4 RenameTopology

**RenameTopology** — Renames a topology

#### Synopsis

```
varchar RenameTopology(varchar old_name, varchar new_name);
```

☒☒

Renames a topology schema, updating its metadata record in the `topology.topology` table.

Availability: 3.4.0



☒☒

Adds missing entries to the `topology.layer` table by inspecting topology constraints on tables. This function is useful for fixing up entries in topology catalog after restores of schemas with topo data.

It returns the list of entries created. Returned columns are `schema_name`, `table_name`, `feature_column`.

2.3.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

```
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');
-- this will return no records because this feature is already registered
SELECT *
 FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
 FROM topology.Populate_Topology_Layer();

SELECT topology_id,layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-----+-----+-----
strk | parcels | topo
(1 row)

topology_id | layer_id | sn | tn | fc
-----+-----+-----+-----+-----
 2 | 2 | strk | parcels | topo
(1 row)
```

☒☒

[AddTopoGeometryColumn](#)

### 9.3.7 TopologySummary

`TopologySummary` — Takes a topology name and provides summary totals of types of objects in topology.

#### Synopsis

text **TopologySummary**(varchar topology\_schema\_name);

☒☒

Takes a topology name and provides summary totals of types of objects in topology.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
SELECT topology.topologysummary('city_data');
 topologysummary

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
```

☒☒

**Topology\_Load\_Tiger**

**9.3.8 ValidateTopology**

ValidateTopology — Returns a set of validate\_topology\_returntype objects detailing issues with topology.

**Synopsis**

setof validate\_topology\_returntype **ValidateTopology**(varchar toponame, geometry bbox);

☒☒

Returns a set of **validate\_topology\_returntype** objects detailing issues with topology, optionally limiting the check to the area specified by the **bbox** parameter.

List of possible errors, what they mean and what the returned ids represent are displayed below:

☒☒	id1	id2	Meaning
coincident nodes	Identifier of first node.	Identifier of second node.	Two nodes have the same geometry.
edge crosses node(☒☒☒☒☒☒☒☒☒☒)	Identifier of the edge.	Identifier of the node.	An edge has a node in its interior. See <b>ST_Relate</b> .

	<b>id1</b>	<b>id2</b>	<b>Meaning</b>
invalid edge( <code>edge_id</code> )	Identifier of the edge.		An edge geometry is invalid. See <a href="#">ST_IsValid</a> .
edge not simple( <code>edge_id</code> )	Identifier of the edge.		An edge geometry has self-intersections. See <a href="#">ST_IsSimple</a> .
edge crosses edge( <code>edge_id1</code> , <code>edge_id2</code> )	Identifier of first edge.	Identifier of second edge.	Two edges have an interior intersection. See <a href="#">ST_Relate</a> .
edge start node geometry mismatch( <code>edge_id</code> , <code>start_node_id</code> )	Identifier of the edge.	Identifier of the indicated start node.	The geometry of the node indicated as the starting node for an edge does not match the first point of the edge geometry. See <a href="#">ST_StartPoint</a> .
edge end node geometry mis-match( <code>edge_id</code> , <code>end_node_id</code> )	Identifier of the edge.	Identifier of the indicated end node.	The geometry of the node indicated as the ending node for an edge does not match the last point of the edge geometry. See <a href="#">ST_EndPoint</a> .
face without edges( <code>face_id</code> )	Identifier of the orphaned face.		No edge reports an existing face on either of its sides (left_face, right_face).
face has no rings( <code>face_id</code> )	Identifier of the partially-defined face.		Edges reporting a face on their sides do not form a ring.
face has wrong mbr	Identifier of the face with wrong mbr cache.		Minimum bounding rectangle of a face does not match minimum bounding box of the collection of edges reporting the face on their sides.
hole not in advertised face	Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> .		A ring of edges reporting a face on its exterior is contained in different face.
not-isolated node has not- containing_face	Identifier of the ill-defined node.		A node which is reported as being on the boundary of one or more edges is indicating a containing face.
isolated node has containing_face	Identifier of the ill-defined node.		A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.

	<b>id1</b>	<b>id2</b>	<b>Meaning</b>
isolated node has wrong containing_face	Identifier of the misrepresented node.		A node which is not reported as being on the boundary of any edges indicates a containing face which is not the actual face containing it. See <a href="#">GetFaceContainingPoint</a> .
invalid next_right_edge	Identifier of the misrepresented edge.	Signed id of the edge which should be indicated as the next right edge.	The edge indicated as the next edge encountered walking on the right side of an edge is wrong.
invalid next_left_edge	Identifier of the misrepresented edge.	Signed id of the edge which should be indicated as the next left edge.	The edge indicated as the next edge encountered walking on the left side of an edge is wrong.
mixed face labeling in ring	Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> .		Edges in a ring indicate conflicting faces on the walking side. This is also known as a "Side Location Conflict".
non-closed ring	Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> .		A ring of edges formed by following next_left_edge/next_right_edge attributes starts and ends on different nodes.
face has multiple shells	Identifier of the contended face.	Signed identifier of an edge, identifying the ring. See <a href="#">GetRingEdges</a> .	More than a one ring of edges indicate the same face on its interior.

1.0.0

2.0.0, (false positive)

2.2.0 'edge crosses node' id1 id2

Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks.

```
SELECT * FROM topology.ValidateTopology('ma_topo');
 error | id1 | id2
-----+-----+-----
face without edges | 1 |
```

[validatetopology\\_returntype](#), [Topology\\_Load\\_Tiger](#)



### 9.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

#### Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

☒☒

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

☒☒

[ValidateTopology](#)

### 9.3.10 FindTopology

FindTopology — Returns a topology record by different means.

#### 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);

☒☒

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

☒☒

```
SELECT name(findTopology('features.land_parcel', 'feature'));
 name

city_data
(1 row)
```

☒☒

[FindLayer](#)

### 9.3.11 FindLayer

FindLayer — Returns a topology.layer record by different means.

#### 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);
```

☒☒

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

☒☒

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id

 1
(1 row)
```

☒☒

**FindTopology**

## 9.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.



#### Note

That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

## 9.5 ☒☒☒☒☒

### 9.5.1 CreateTopology

CreateTopology — Creates a new topology schema and registers it in the topology.topology table.

## Synopsis

```
integer CreateTopology(varchar topology_schema_name);
integer CreateTopology(varchar topology_schema_name, integer srid);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec);
integer CreateTopology(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);
```

☒☒

Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`, and `relation`) are created in the schema. It returns the id of the topology.

The `srid` is the [spatial reference system](#) SRID for the topology.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

This is similar to the SQL/MM [ST\\_InitTopoGeo](#) but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting `hasZ`

☒☒

Create a topology schema called `ma_topo` that stores edges and nodes in Massachusetts State Plane-meters (SRID = 26986). The tolerance represents 0.5 meters since the spatial reference system is meter-based.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Create a topology for Rhode Island called `ri_topo` in spatial reference system State Plane-feet (SRID = 3438)

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid

2
```

☒☒

Section [4.5](#), [ST\\_InitTopoGeo](#), [Topology\\_Load\\_Tiger](#)

## 9.5.2 CopyTopology

`CopyTopology` — Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema

### Synopsis

```
integer CopyTopology(varchar existing_topology_name, varchar new_name);
```

☒☒

Creates a new topology with name `new_name`, with SRID and precision copied from `existing_topology_name`. The nodes, edges and faces in `existing_topology_name` are copied into the new topology, as well as Layers and their associated TopoGeometries.



**Note**

The new rows in the `topology.layer` table contain synthetic values for `schema_name`, `table_name` and `feature_column`. This is because the TopoGeometry objects exist only as a definition and are not yet available in a user-defined table.

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

Make a backup of a topology called `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

☒☒

Section [4.5, CreateTopology, RenameTopology](#)

### 9.5.3 ST\_InitTopoGeo

`ST_InitTopoGeo` — Creates a new topology schema and registers it in the `topology.topology` table.

#### Synopsis

```
text ST_InitTopoGeo(varchar topology_schema_name);
```

☒☒

This is the SQL-MM equivalent of [CreateTopology](#). It lacks options for spatial reference system and tolerance. It returns a text description of the topology creation, instead of the topology id.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

☒☒

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
 astopocreation
```

```

Topology-Geometry 'topo_schema_to_create' (id:7) created.
```



### 9.5.5 TopoGeo\_AddPoint

TopoGeo\_AddPoint —  (split)  .

#### Synopsis

integer **TopoGeo\_AddPoint**(varchar atopolgy, geometry apoint, float8 tolerance);



Adds a point to an existing topology and returns its identifier. The given point will snap to existing nodes or edges within given tolerance. An existing edge may be split by the snapped point.

2.0.0 .



[TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddNode](#), [CreateTopology](#)

### 9.5.6 TopoGeo\_AddLineString

TopoGeo\_AddLineString — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns edge identifiers.

#### Synopsis

SETOF integer **TopoGeo\_AddLineString**(varchar atopolgy, geometry aline, float8 tolerance);



Adds a linestring to an existing topology and returns a set of edge identifiers forming it up. 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.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0 .



[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddPolygon](#), [TopoGeo\\_LoadGeometry](#), [AddEdge](#), [CreateTopology](#)

## 9.5.7 TopoGeo\_AddPolygon

`TopoGeo_AddPolygon` — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

### Synopsis

SETOF integer **TopoGeo\_AddPolygon**(varchar atopolology, geometry apoly, float8 tolerance);



Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

2.0.0     .



[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [TopoGeo\\_LoadGeometry](#), [AddFace](#), [CreateTopology](#)

## 9.5.8 TopoGeo\_LoadGeometry

`TopoGeo_LoadGeometry` — Load a geometry into an existing topology, snapping and splitting as needed.

### Synopsis

void **TopoGeo\_LoadGeometry**(varchar atopolology, geometry ageom, float8 tolerance);



Loads a geometry into an existing topology. The given geometry will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split as a consequence of the load.



#### Note

Updating statistics about topologies being loaded via this function is up to caller, see [maintaining statistics during topology editing and population](#).

Availability: 3.5.0

[TopoGeo\\_AddPoint](#), [TopoGeo\\_AddLineString](#), [TopoGeo\\_AddPolygon](#), [CreateTopology](#)

## 9.6

### 9.6.1 ST\_AddIsoNode

`ST_AddIsoNode` — Add an isolated node to a topology. `faceid` is the ID of the face. `nodeid` is the ID of the node. `apoint` is the geometry of the node. `SRID` is the SRID of the node. `apoint` is NULL if the node is a point.

#### Synopsis

integer `ST_AddIsoNode`(varchar atopology, integer aface, geometry apoint);

`atopology` is the topology name. `aface` is the ID of the face. `apoint` is the geometry of the node. `SRID` is the SRID of the node. `apoint` is NULL if the node is a point.

`SRID` is the SRID of the node. `apoint` is the geometry of the node. `apoint` is NULL if the node is a point. `apoint` is the geometry of the node. `apoint` is NULL if the node is a point.

`aface` is NULL if the node is a point. `apoint` is the geometry of the node. `apoint` is NULL if the node is a point.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

[AddNode](#), [CreateTopology](#), [DropTopology](#), [ST\\_Intersects](#)

### 9.6.2 ST\_AddIsoEdge

`ST_AddIsoEdge` — Add an isolated edge to a topology. `anode` and `anothernode` are the IDs of the nodes. `alinesring` is the geometry of the edge. `SRID` is the SRID of the edge. `alinesring` is NULL if the edge is a line string.

#### Synopsis

integer `ST_AddIsoEdge`(varchar atopology, integer anode, integer anothernode, geometry alinesring);





### 9.6.4 ST\_AddEdgeModFace

ST\_AddEdgeModFace — `integer, geometry, integer, integer, geometry`

#### Synopsis

integer **ST\_AddEdgeModFace**(varchar atopolgy, integer anode, integer anothernode, geometry acurve);



#### Note

..., (universe face) ...

ID ...

...

NULL ..., node ..., acurve LINGSTRING ..., anode anothernode acurve ...

acurve (SRID) ...

2.0 ...



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

[ST\\_RemEdgeModFace](#)

[ST\\_AddEdgeNewFaces](#)

### 9.6.5 ST\_RemEdgeNewFace

ST\_RemEdgeNewFace — `integer, integer`

#### Synopsis

integer **ST\_RemEdgeNewFace**(varchar atopolgy, integer anedge);

ST

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

ST

ST

ST\_RemEdgeModFace

ST\_AddEdgeNewFaces

### 9.6.6 ST\_RemEdgeModFace

ST\_RemEdgeModFace — Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.

#### Synopsis

integer ST\_RemEdgeModFace(varchar atopology, integer anedge);

ST

Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with ST\_AddEdgeModFace. Returns the id of the face which is preserved.

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15





☒☒

Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly.

2.0 ☒☒☒☒☒☒☒☒☒☒☒☒.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

☒☒

[ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

### 9.6.10 ST\_NewEdgeHeal

`ST_NewEdgeHeal` — Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.

#### Synopsis

```
int ST_NewEdgeHeal(varchar atopology, integer anedge, integer anotheredge);
```

☒☒

Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Returns the id of the new edge replacing the healed ones. Updates all existing joined edges and relationships accordingly.

2.0 ☒☒☒☒☒☒☒☒☒☒☒☒.



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

☒☒

[ST\\_ModEdgeHeal](#) [ST\\_ModEdgeSplit](#) [ST\\_NewEdgesSplit](#)

### 9.6.11 ST\_MoveIsoNode

`ST_MoveIsoNode` — Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move.

#### Synopsis

```
text ST_MoveIsoNode(varchar atopology, integer anode, geometry apoint);
```



☒☒

```
-- Add an edge --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) ←
', 26986)) As edgeid;
-- result-
edgeid

 2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2, ST_GeomFromText('POINT(227578.5 893913.5)', ←
26986)) As newnodeid;
newnodeid

 6
```

☒☒

[ST\\_ModEdgeSplit](#) [ST\\_ModEdgeHeal](#) [ST\\_NewEdgeHeal](#) [AddEdge](#)

### 9.6.13 ST\_RemoveIsoNode

**ST\_RemoveIsoNode** — Removes an isolated node from a topology. `ST_RemoveIsoNode` (`text topology`, `integer anode`) returns `text`, the name of the topology.

#### Synopsis

text **ST\_RemoveIsoNode**(varchar atopology, integer anode);

☒☒

`ST_RemoveIsoNode` (`text topology`, `integer anode`) returns `text`, the name of the topology.

Availability: 1.1



This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

☒☒

```
-- Remove an isolated node with no face --
SELECT topology.ST_RemoveIsoNode('ma_topo', 7) As result;
result

Isolated node 7 removed
```

☒☒

[ST\\_AddIsoNode](#)





**Note**

tolerance = 0 ST\_Intersects, ST\_DWithin.

GEOS

2.0.0

**AddEdge**

```
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;
with1mtol | withnotol
-----+-----
 2 | 0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
-- get error --
ERROR: Two or more edges found
```

**AddEdge, GetNodeByPoint, GetFaceByPoint**

## 9.7.2 GetFaceByPoint

GetFaceByPoint — Finds face intersecting a given point.

### Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

Finds a face referenced by a Point, with given tolerance.

The function will effectively look for a face intersecting a circle having the point as center and the tolerance as radius.

If no face intersects the given query location, 0 is returned (universal face).

If more than one face intersect the query location an exception is thrown.

2.0.0

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

☒☒

```
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;

with1mtol | withnotol
-----+-----
1 | 0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR: Two or more faces found
```

☒☒

[GetFaceContainingPoint](#), [AddFace](#), [GetNodeByPoint](#), [GetEdgeByPoint](#)

### 9.7.3 GetFaceContainingPoint

`GetFaceContainingPoint` — Finds the face containing a point.

#### Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

☒☒

Returns the id of the face containing a point.

An exception is thrown if the point falls on a face boundary.



#### Note

The function relies on a valid topology, using edge linking and face labeling.

Availability: 3.2.0

☒☒

[ST\\_GetFaceGeometry](#)

### 9.7.4 GetNodeByPoint

`GetNodeByPoint` — Finds the node-id of a node at a point location.

## Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 toll1);

☒☒

Retrieves the id of a node at a point location.

The function returns an integer (id-node) given a topology, a POINT and a tolerance. If tolerance = 0 means exact intersection, otherwise retrieves the node from an interval.

If apoint doesn't intersect a node, returns 0 (zero).

If use tolerance > 0 and there is more than one node near the point then an exception is thrown.



### Note

☒☒☒☒ tolerance = 0 ☒☒☒ ST\_Intersects ☒, ☒☒☒☒☒☒☒ ST\_DWithin ☒☒☒☒☒☒.

GEOS ☒☒☒☒☒

2.0.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

☒☒☒☒☒☒ **AddEdge** ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
nearnode

 2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

----get error--
ERROR: Two or more nodes found
```

☒☒

**AddEdge, GetEdgeByPoint, GetFaceByPoint**

## 9.7.5 GetTopologyID

GetTopologyID — ☒☒☒☒☒☒☒☒☒☒ topology.topology ☒☒☒☒☒☒☒☒☒☒ ID ☒☒☒☒☒☒☒.

### Synopsis

integer **GetTopologyID**(varchar toponame);







### 9.7.10 GetRingEdges

GetRingEdges —

#### Synopsis

getfaceedges\_returntype **GetRingEdges**(varchar atopolgy, integer aring, integer max\_edges=null);

GetRingEdges returns a set of edges. The sequence (sequence) is the ID(edgeid) of the edge. The first edge is the outer boundary.

max\_edges is NULL, all edges are returned. If max\_edges is not NULL, only the first max\_edges edges are returned.

max\_edges is NULL, all edges are returned. If max\_edges is not NULL, only the first max\_edges edges are returned.



#### Note

GetRingEdges returns a set of edges.

2.0.0

[ST\\_GetFaceEdges](#), [GetNodeEdges](#)

### 9.7.11 GetNodeEdges

GetNodeEdges —

#### Synopsis

getfaceedges\_returntype **GetNodeEdges**(varchar atopolgy, integer anode);

GetNodeEdges returns a set of edges. The sequence (sequence) is the ID of the node. The first edge is the outer boundary.



#### Note

GetNodeEdges returns a set of edges.

2.0







**Note**

`aline` `srid` `srid`. `LINESTRING` `LINESTRING`.

GEOS

**Warning**

`AddEdge` is deprecated as of 3.5.0. Use `TopoGeo_AddLineString` instead.

2.0.0

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9
 893900.4)', 26986)) As edgeid;
-- result-
edgeid

1

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;
-- result --
edgeid

2

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9
 893900.4,
 227704.5 893778.5)', 26986)) As edgeid;
-- gives error --
ERROR: Edge intersects (not on endpoints) with existing edge 1
```

[TopoGeo\\_AddLineString](#), [CreateTopology](#), Section 4.5

### 9.8.4 AddFace

`AddFace` — (face primitive)

#### Synopsis

integer **AddFace**(varchar toponame, geometry apolygon, boolean force\_new=false);













```
--use to verify what has happened --
SELECT * FROM
 topology.TopologySummary('topo_boston_test');

-- summary--
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
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
 FROM topo_boston_test.face f
 WHERE f.face_id
> 0 -- don't consider the universe face
 AND NOT EXISTS (-- check that no TopoGeometry references the face
 SELECT * FROM topo_boston_test.relation
 WHERE layer_id = 1 AND element_id = f.face_id
);
```

☒☒

[CreateTopology](#), [AddTopoGeometryColumn](#), [CreateTopoGeom](#), [TopologySummary](#), [clearTopoGeom](#)

### 9.9.3 TopoElementArray\_Agg

**TopoElementArray\_Agg** — Returns a `topoelementarray` for a set of `element_id`, type arrays (`topoelements`).

#### Synopsis

`topoelementarray` **TopoElementArray\_Agg**(`topoelement set tefield`);

☒☒

**TopoElement** ☒☒☒☒☒☒ **TopoElementArray** ☒☒☒☒☒☒☒☒☒☒.

2.0.0 ☒☒☒☒☒☒☒☒☒☒.

☒☒

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
 FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
 tea

{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```





☒☒

TopoGeometry ☒☒☒☒☒☒☒ **TopoElement** ☒☒☒☒☒☒☒.

2.3 ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

☒☒

**TopoGeom\_addElement, CreateTopoGeom**

### 9.10.4 TopoGeom\_addTopoGeom

TopoGeom\_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

#### Synopsis

topogeometry **TopoGeom\_addTopoGeom**(topogeometry tgt, topogeometry src);

☒☒

Adds the elements of a **TopoGeometry** to the definition of another TopoGeometry, possibly changing its cached type (type attribute) to a collection, if needed to hold all elements in the source object.

The two TopoGeometry objects need be defined against the *same* topology and, if hierarchically defined, need be composed by elements of the same child layer.

Availability: 3.2

☒☒

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopogeom(
 TopoGeom_addTopoGeom(
 clearTopoGeom(tg_overall),
 tg_specific1
),
 tg_specific2
);
```

☒☒

**TopoGeom\_addElement, clearTopoGeom, CreateTopoGeom**

### 9.10.5 toTopoGeom

toTopoGeom — Adds a geometry shape to an existing topo geometry.

Refer to [toTopoGeom](#).

## 9.11 TopoGeometry

### 9.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Returns a topoelementarray (an array of topoelements) containing the topological elements and type of the given TopoGeometry (primitive elements).

#### Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer\_id, integer tg\_id);  
 topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

`GetTopoGeomElementArray` (TopoGeometry) returns TopoElementArray. `GetTopoGeomElements` (TopoGeometry) returns TopoElementArray.

tg\_id topology.layer layer\_id TopoGeometry TopoGeometry ID.

Availability: 1.1

[GetTopoGeomElements](#), [TopoElementArray](#)

### 9.11.2 GetTopoGeomElements

GetTopoGeomElements — Returns a set of topoelement objects containing the topological element\_id,element\_id of the given TopoGeometry (primitive elements).

#### Synopsis

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer\_id, integer tg\_id);  
 setof topoelement **GetTopoGeomElements**(topogeometry tg);



## 9.12 TopoGeometry

### 9.12.1 AsGML

AsGML — TopoGeometry GML.

#### 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 gmlversion);
```

TopoGeometry GML GML3. nsprefix\_in gml nsprefix (non-qualified) (15) (1) ST\_AsGML.

visitedTable (xlink:xref) 'element\_type' 'element\_id' (2) element\_type element\_id, orientation:

```
CREATE TABLE visited (
 element_type integer, element_id integer,
 unique(element_type, element_id)
);
```

idprefix, gmlver ST\_AsGML 3 2.0.0.

**CreateTopoGeom**

```
SELECT topology.AsGML(topo) As rdgml
FROM ri.roads
WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
 <gml:directedEdge>
 <gml:Edge gml:id="E1">
 <gml:directedNode orientation="-">
```









### Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

⊠

⊠ TopoGeometry (⊠, ⊠, ⊠) ⊠



#### Note

⊠ TopoGeometry ⊠. ⊠ TopoGeometry ⊠.

1.1.0 ⊠



This function supports 3d and will not drop the z-index.

⊠

⊠

[GetTopoGeomElements](#), [ST\\_Equals](#)

### 9.13.2 Intersects

Intersects — ⊠ TopoGeometry ⊠

### Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

⊠

⊠ TopoGeometry ⊠



#### Note

This function not supported for topogeometries that are geometry collections. It also can not compare topogeometries from different topologies. Also not currently supported for hierarchical topogeometries (topogeometries composed of other topogeometries).

1.1.0 ⊠



This function supports 3d and will not drop the z-index.

☒☒

☒☒

[ST\\_Intersects](#)

## 9.14 Importing and exporting Topologies

Once you have created topologies, and maybe associated topological layers, you might want to export them into a file-based format for backup or transfer into another database.

Using the standard dump/restore tools of PostgreSQL is problematic because topologies are composed by a set of tables (4 for primitives, an arbitrary number for layers) and records in metadata tables (topology.topology and topology.layer). Additionally, topology identifiers are not univoque across databases so that parameter of your topology will need to be changes upon restoring it.

In order to simplify export/restore of topologies a pair of executables are provided: `pgtopo_export` and `pgtopo_import`. Example usage:

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

### 9.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` commandline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version 1 the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge\_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

### 9.14.2 Using the Topology importer

The `pgtopo_import` script takes a `pgtopo_export` format topology dump and a name to give to the topology to be created and outputs an SQL script reconstructing the topology and associated layers.

The generated SQL file will contain statements that create a topology with the given name, load primitive data in it, restores and registers all topology layers by properly linking all TopoGeometry values to their correct topology.

By default `pgtopo_import` reads the dump from the standard input so that it can be used in conjunction with `pgtopo_export` in a pipeline. You can optionally specify an input filename with the `-f` commandline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers *AFTER* resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

# Chapter 10

## 10. Raster Data

### 10.1 Raster Data

The `raster2pgsql` utility is a raster loader executable that loads GDAL supported raster formats into SQL suitable for loading into a PostGIS raster table. It is capable of loading folders of raster files as well as creating overviews of rasters.

#### 10.1.1 raster2pgsql

The `raster2pgsql` is a raster loader executable that loads GDAL supported raster formats into SQL suitable for loading into a PostGIS raster table. It is capable of loading folders of raster files as well as creating overviews of rasters.

Since the `raster2pgsql` is compiled as part of PostGIS most often (unless you compile your own GDAL library), the raster types supported by the executable will be the same as those compiled in the GDAL dependency library. To get a list of raster types your particular `raster2pgsql` supports use the `-G` switch.



#### Note

The `-f` switch (factor) is used to specify the resolution of the raster. For more information, see <http://trac.osgeo.org/postgis/ticket/1764>.

##### 10.1.1.1 Example Usage

Load a 100x100 raster into a table:

```
-s use srid 4326
-I create spatial index
-C use standard raster constraints
-M vacuum analyze after load
*.tif load all these files
-F include a filename column in the raster table
-t tile the output 100x100
public.demelevation load into this table
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql
```

```
-d connect to this database
-f read this file after connecting
psql -d gisdb -f elev.sql
```



-r **regular blocking** (regular blocking) 选项 (选项) 选项。 -C 选项 选项。

选项: 选项

-s <SRID> 选项 SRID 选项。 选项 0 选项, 选项 SRID 选项 选项。

-b **BAND** 选项 (1-选项) 选项。 选项, 选项 (,) 选项 选项。

-t **TILE\_SIZE** 选项。 TILE\_SIZE 选项 x 选项 选项, 选项"auto" 选项 选项。

-P 选项 (padding) 选项。

-R, --register 选项 (DB 选项) 选项。 选项 (选项) 选项。

-l **OVERVIEW\_FACTOR** 选项。 选项 (,) 选项。 选项 o overview\_factor\_table 选项, 选项 overview\_factor 选项 选项 (placeholder) 选项 table 选项。 选项 选项, -R 选项。 选项 SQL 选项 选项。

-N **NODATA** "NODATA" 选项 NODATA 选项。

选项

-f **COLUMN** 选项。 选项'rast' 选项。

-F 选项。

-n **COLUMN** 选项。 -F 选项。

-q PostgreSQL 选项。

-I 选项 GiST 选项。

-M 选项 (vacuum analyze) 选项。

-k Keeps empty tiles and skips NODATA value checks for each raster band. Note you save time in checking, but could end up with far more junk rows in your database and those junk rows are not marked as empty tiles.

-T **tablespace** 选项。 -X 选项 (选项 选项) 选项。

-X **tablespace** 选项。 -I 选项 选项。

-Y **max\_rows\_per\_copy=50** Use copy statements instead of insert statements. Optionally specify max\_rows\_per\_copy; default 50 when not specified.

-e 选项, 选项 (transaction) 选项。

-E **ENDIAN** 选项 (endianness) 选项。 XDR 选项 0, 选项 选项 NDR 选项 1 选项。 选项, NDR 选项。

-V **version** 选项。 选项 0 选项。 选项, 0 选项。

### 10.1.2 PostGIS 选项

选项。 选项, 选项 选项。 选项。

1. 选项:



```
CREATE TABLE myrasters(rid serial primary key, rast raster);
```

- 2. `ST_MakeEmptyRaster`, `ST_AddBand`, `ST_AsRaster`, `ST_Union`, `ST_MapAlgebra` (algebra), `ST_Transform`

```
CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist(ST_ConvexHull(rast));
```

(convex hull) `ST_ConvexHull`



**Note** PostGIS 2.0 (envelop) ...

- 4. `AddRasterConstraints`

### 10.1.3 Using "out db" cloud rasters

The raster2pgsql tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are stored remotely in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the "cloud optimized GeoTIFF" format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is reading from the cloud raster and writing to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xx \
raster2pgsql \
-s 990000 \
-t 256x256 \
-I \
-R \
/vsis3/your.bucket.com/your_file.tif \
your_table \
| psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, `postgis.enable_outdb_rasters` and `postgis.gdal_enabled_drivers`.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```





- `o_table_catalog` ...
- `o_table_schema` ...
- `o_table_name` ...
- `o_raster_column` ...
- `r_table_catalog` ...
- `r_table_schema` ...
- `r_table_name` ...
- `r_raster_column` ...
- `overview_factor` ... raster2pgsql ... 1 ... 2 ... 4 ... 125x125 ... 5000x5000 ... (5000\*5000)/(125\*125) = 1600 ... o\_2 (l=2) ... (1600/2^2) = 400 ... o\_3 (l=3) ... (1600/2^3) = 200 ... raster2pgsql ... 2^overview\_factor ...

### 10.3 PostGIS

The fact that PostGIS raster provides you with SQL functions to render rasters in known image formats gives you a lot of options for rendering them. For example you can use OpenOffice / LibreOffice for rendering as demonstrated in [Rendering PostGIS Raster graphics with LibreOffice Base Reports](#). In addition you can use a wide variety of languages as demonstrated in this section.

#### 10.3.1 ST\_ASPNG PHP

..., PHP PostgreSQL `ST_AsGDALRaster` 1, 2, 3 PHP (request stream) ... PHP "img src" HTML ...

(combine) WGS84 (union) `ST_Transform` `ST_ASPNG` PNG ...

[http://mywebserver/test\\_raster.php?srid=2249](http://mywebserver/test_raster.php?srid=2249)

...

```
<?php
/** contents of test_raster.php */
$conn_str = 'dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters ←
**/
if (!empty($_REQUEST['srid']) && is_numeric($_REQUEST['srid'])){
```

```

 $input_srid = intval($_REQUEST['srid']);
 }
 else { $input_srid = 26986; }
 /** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 */
 $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]);
 ?>

```

### 10.3.2 ST\_AsPNG ASP.NET C#

npqsql PostgreSQL .NET [ST AsGDALRaster](#) 1, 2, 3 PHP (request stream) [HTML](#) `img src`

npqsql PostgreSQL .NET <http://npqsql.projects.postgresql.org/> ASP.NET bin

(combine) WGS84 [ST Union](#) (union) [ST Transform](#) [ST AsPNG](#) PNG

C# Section 10.3.1

[http://mywebserver/test\\_raster.php?srid=2249](http://mywebserver/test_raster.php?srid=2249)

```

-- web.config connection string section --
<connectionStrings>
 <add name="DSN"
 connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ←
 mypwd"/>
</connectionStrings>

```

```

// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npqsql;

public class TestRaster : IHttpHandler
{
 public void ProcessRequest(HttpContext context)
 {
 context.Response.ContentType = "image/png";
 }
}

```





```

 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);
 }
}
}

```

### 10.3.4 PLPython SQL

PLPython is a procedural language extension for PostgreSQL. PLPythonu is a PLPythonu3u extension.

```

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
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
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;

```

```

write_file

C:/temp/slices1.png
C:/temp/slices2.png
C:/temp/slices3.png
C:/temp/slices4.png
C:/temp/slices5.png

```

### 10.3.5 PSQL

PostgreSQL has a PSQL extension. PSQL is a PSQL extension. PSQL is a PSQL extension.

PostgreSQL has a PSQL extension. PSQL is a PSQL extension.

```

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);
-- you'll get an output something like --
oid | num_bytes
-----+-----
2630819 | 74860

```



```
-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
\lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```







4.  $(a-b) = a < x < b$

'( (a-b) a-b

**pixeltype text ST\_BandPixelType**

**nodataval double precision** NODATA. ,

: 2 255 NODATA 8BUI

```
SELECT ROW(2, '0-100:1-10, 101-500:11-150,501 - 10000: 151-254', '8BUI', 255)::reclassarg;
```

: 1 NODATA 1BB

```
SELECT ROW(1, '0-100]:0, (100-255:1', '1BB', NULL)::reclassarg;
```

**ST\_Reclass**

### 11.1.6 summarystats

summarystats — ST\_SummaryStats ST\_SummaryStatsAgg

**ST\_SummaryStats** ST\_SummaryStatsAgg

**count integer**

**sum double precision**

**mean double precision**

**stddev double precision**

**min double precision**

**max double precision**

**ST\_SummaryStats, ST\_SummaryStatsAgg**

### 11.1.7 unionarg

unionarg — UNION ST\_Union

¶

UNION ST\_Union

nband integer 1-

uniontype text UNION ST\_Union

¶

ST\_Union

11.2

11.2.1 AddRasterConstraints

AddRasterConstraints — Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.

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 no\_data\_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=false, boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true, boolean extent=true );

¶

raster\_columns, rastschema, srid SPATIAL\_REF\_SYS

raster2pgsql

Section 10.2.1

- blocksize X Y
• blocksize\_x X ( )
• blocksize\_y Y ( )
• extent
• num\_bands

















WHERE short\_name = 'JPEG') As g;

oname	otype	descrip
PROGRESSIVE	boolean	whether to generate a progressive JPEG
QUALITY	int	good=100, bad=0, default=75
WORLDFILE	boolean	whether to geneate 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)

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';
```

```
<CreationOptionList>
 <Option name="COMPRESS" type="string-select">
 <Value
>NONE</Value>
 <Value
>LZW</Value>
 <Value
>PACKBITS</Value>
 <Value
>JPEG</Value>
 <Value
>CCITTRLE</Value>
 <Value
>CCITTFAX3</Value>
 <Value
>CCITTFAX4</Value>
 <Value
>DEFLATE</Value>
 </Option>
 <Option name="PREDICTOR" type="int" description="Predictor Type"/>
 <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
 <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←
 ="6"/>
 <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16 ←
 (9-15), sub-uint32 (17-31)"/>
 <Option name="INTERLEAVE" type="string-select" default="PIXEL">
 <Value
>BAND</Value>
 <Value
>PIXEL</Value>
 </Option>
 <Option name="TILED" type="boolean" description="Switch to tiled format"/>
 <Option name="TFW" type="boolean" description="Write out world file"/>
 <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
 <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
 <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
 <Option name="PHOTOMETRIC" type="string-select">
 <Value
>MINISBLACK</Value>
 <Value
```

```

>MINISWHITE</Value>
 <Value
>PALETTE</Value>
 <Value
>RGB</Value>
 <Value
>CMYK</Value>
 <Value
>YCBCR</Value>
 <Value
>CIELAB</Value>
 <Value
>ICCLAB</Value>
 <Value
>ITULAB</Value>
 </Option>
 <Option name="SPARSE_OK" type="boolean" description="Can newly created files have ↵
 missing blocks?" default="FALSE"/>
 <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↵
 "/>
 <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
 <Value
>GDALGeoTIFF</Value>
 <Value
>GeoTIFF</Value>
 <Value
>BASELINE</Value>
 </Option>
 <Option name="PIXELTYPE" type="string-select">
 <Value
>DEFAULT</Value>
 <Value
>SIGNEDBYTE</Value>
 </Option>
 <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↵
 ">
 <Value
>YES</Value>
 <Value
>NO</Value>
 <Value
>IF_NEEDED</Value>
 <Value
>IF_SAFER</Value>
 </Option>
 <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force ↵
 endianness of created file. For DEBUG purpose mostly">
 <Value
>NATIVE</Value>
 <Value
>INVERTED</Value>
 <Value
>LITTLE</Value>
 <Value
>BIG</Value>
 </Option>
 <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy ↵
 of overviews of source dataset (CreateCopy())"/>
</CreationOptionList>

-- Output the create options XML column for GTiff as a table --
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,
array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM st_gdaldrivers()
WHERE short_name = 'GTiff') As g;
```

oname	otype	descrip	vals
COMPRESS	string-select		NONE, LZW, ↩
PREDICTOR	int	Predictor Type ↩	
JPEG_QUALITY	int	JPEG quality 1-100 ↩	
ZLEVEL	int	DEFLATE compression level 1-9 ↩	
NBITS	int	BITS for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31) ↩	
INTERLEAVE	string-select		BAND, PIXEL ↩
TILED	boolean	Switch to tiled format ↩	
TFW	boolean	Write out world file ↩	
RPB	boolean	Write out .RPB (RPC) file ↩	
BLOCKXSIZE	int	Tile Width ↩	
BLOCKYSIZE	int	Tile/Strip Height ↩	
PHOTOMETRIC	string-select		MINISBLACK, ↩
SPARSE_OK	boolean	Can newly created files have missing blocks? ↩	
ALPHA	boolean	Mark first extrasample as being alpha ↩	
PROFILE	string-select		GDALGeoTIFF, ↩
PIXELTYPE	string-select		DEFAULT, ↩
BIGTIFF	string-select	Force creation of BigTIFF file ↩	
ENDIANNESS	string-select	Force endianness of created file. For DEBUG purpose ↩	
COPY_SRC_OVERVIEWS	boolean	Force copy of overviews of source dataset (CreateCopy ↩)	

(19 rows)



[ST\\_AsGDALRaster](#), [ST\\_SRID](#), [postgis.gdal\\_enabled\\_drivers](#)



## 11.2.9 ST\_Contour

ST\_Contour — Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).

### 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);
```

☒☒

Generates a set of vector contours from the provided raster band, using the [GDAL contouring algorithm](#).

When the `fixed_levels` parameter is a non-empty array, the `level_interval` and `level_base` parameters are ignored.

Input parameters are:

**rast** The raster to generate the contour of

**bandnumber** The band to generate the contour of

**level\_interval** The elevation interval between contours generated

**level\_base** The "base" relative to which contour intervals are applied, this is normally zero, but could be different. To generate 10m contours at 5, 15, 25, ... the `LEVEL_BASE` would be 5.

**fixed\_levels** The elevation interval between contours generated

**polygonize** If true, contour polygons will be created, rather than polygon lines.

Return values are a set of records with the following attributes:

**geom** The geometry of the contour line.

**id** A unique identifier given to the contour line by GDAL.

**value** The raster value the line represents. For an elevation DEM input, this would be the elevation of the output contour.

Availability: 3.2.0

☒☒

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels => ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT st_astext(geom), id, value
FROM c;
```

☒☒

[ST\\_InterpolateRaster](#)



### 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);

SRID. SRID (, SRID) .



**Note**

() .

2.1.0

**UpdateGeometrySRID**

### 11.2.12 ST\_CreateOverview

ST\_CreateOverview —

### Synopsis

regclass **ST\_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

. (1/factor).

raster\_overviews, .

'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', 'Lanczos'. GDAL Warp resampling methods.

2.2.0

Output to generally better quality but slower to product format

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Output to faster to process default nearest neighbor

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```



```

-- 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
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[]
)
);

-- output meta data of raster bands to verify all is right --
SELECT (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
FROM dummy_rast WHERE rid = 10) AS foo;
--result --
pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----
1BB | | f |
4BUI | | f |

-- output meta data of raster -
SELECT (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
FROM dummy_rast WHERE rid = 10) AS foo;
-- result --
upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |
numbands
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | 0 | 100 | 100 | 1 | -1 | 0 | 0 | 0 |
2

```

Example: ST\_AddBand

```

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[]
);

bandnum | pixeltype | nodatavalue | isoutdb | path
-----+-----+-----+-----+-----
1 | 8BUI | 0 | f |
2 | 32BF | -1 | f |
3 | 32BUI | 12 | f |
4 | 16BUI | 2 | f |

```

```
-- 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: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ←
)
-- 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
SELECT
 mouse,
 ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

☒☒: ☒☒☒ **DB** ☒☒☒☒

```
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[]
);
```

bandnum	pixeltype	nodataval	isoutdb	path
1	8BUI		t	/home/raster/mytestraster.tif
2	8BUI		t	/home/raster/mytestraster.tif
3	8BUI		t	/home/raster/mytestraster.tif

☒☒

[ST\\_BandMetaData](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [ST\\_MetaData](#), [ST\\_NumBands](#), [ST\\_Reclass](#)

### 11.3.2 ST\_AsRaster

ST\_AsRaster — PostGIS ☒☒☒ 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);





**Note**

PostGIS, TIN, GDAL

**PostGIS: PNG**



PostGIS

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```



PostGIS

```
-- the bands map to RGB bands - the value (118,154,118) - teal --
SELECT ST_AsPNG(
 ST_AsRaster(
 ST_Buffer(
 ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
 200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

PostGIS

[ST\\_BandPixelType](#), [ST\\_Buffer](#), [ST\\_GDALDrivers](#), [ST\\_AsGDALRaster](#), [ST\\_AsPNG](#), [ST\\_AsJPEG](#), [ST\\_SRID](#)

### 11.3.3 ST\_Band

**ST\_Band** — Returns the band number of a pixel in a raster. Returns the band number of a pixel in a raster.



### 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=,);
```

⊠

Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters or export of only selected bands of a raster or rearranging the order of bands in a raster. If no band is specified or any of specified bands does not exist in the raster, then all bands are returned. Used as a helper function in various functions such as for deleting a band.



#### Warning

⊠ nbands ⊠, ⊠, ⊠ '1,2,3' ⊠. ⊠ ST\_Band(rast, '1@2@3', '@') ⊠. ⊠ ST\_Band(rast, '{1,2,3}'::int[]); ⊠. PostGIS ⊠ text ⊠.

2.0.0 ⊠.

⊠

```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and then reclassified as a 2BUI ←
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;
```

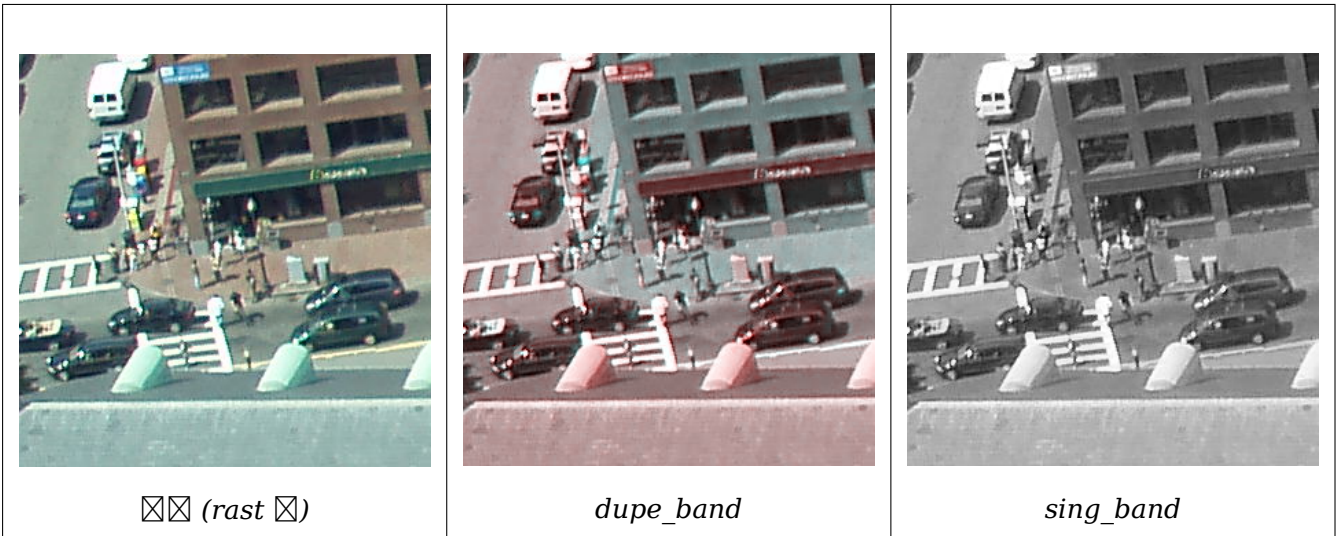
numb1	pix1	numb2	pix2
1	8BUI	1	2BUI

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
 FROM dummy_rast WHERE rid=2;
```

```
num_bands

2
```

```
-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
 FROM dummy_rast
WHERE rid=2;
```



```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
 ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

☒☒

[ST\\_AddBand](#), [ST\\_NumBands](#), [ST\\_Reclass](#), Chapter 11

### 11.3.4 ST\_MakeEmptyCoverage

ST\_MakeEmptyCoverage — Cover georeferenced area with a grid of empty raster tiles.

#### 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);

☒☒

Create a set of raster tiles with [ST\\_MakeEmptyRaster](#). Grid dimension is width & height. Tile dimension is tilewidth & tileheight. The covered georeferenced area is from upper left corner (upperleftx, upperlefty) to lower right corner (upperleftx + width \* scalex, upperlefty + height \* scaley).



**Note**

Note that scaley is generally negative for rasters and scalex is generally positive. So lower right corner will have a lower y value and higher x value than the upper left corner.

2.2.0 ☒☒☒☒☒☒☒☒☒☒.

SQL

Create 16 tiles in a 4x4 grid to cover the WGS84 area from upper left corner (22, 77) to lower right corner (55, 33).

```
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;
```

upperleftx numbands	upperlefty	width	height	scalex	scaley	skewx	skewy	srid	
22	33	1	1	8.25	-11	0	0	4326	↔
30.25	0	33	1	8.25	-11	0	0	4326	↔
38.5	0	33	1	8.25	-11	0	0	4326	↔
46.75	0	33	1	8.25	-11	0	0	4326	↔
22	0	22	1	8.25	-11	0	0	4326	↔
30.25	0	22	1	8.25	-11	0	0	4326	↔
38.5	0	22	1	8.25	-11	0	0	4326	↔
46.75	0	22	1	8.25	-11	0	0	4326	↔
22	0	11	1	8.25	-11	0	0	4326	↔
30.25	0	11	1	8.25	-11	0	0	4326	↔
38.5	0	11	1	8.25	-11	0	0	4326	↔
46.75	0	11	1	8.25	-11	0	0	4326	↔
22	0	0	1	8.25	-11	0	0	4326	↔
30.25	0	0	1	8.25	-11	0	0	4326	↔
38.5	0	0	1	8.25	-11	0	0	4326	↔
46.75	0	0	1	8.25	-11	0	0	4326	↔

SQL

### ST\_MakeEmptyRaster

#### 11.3.5 ST\_MakeEmptyRaster

ST\_MakeEmptyRaster — (width & height), xmin X ymin, scalex, scaley (scalex, scaley, skewx & skewy) SRID (SRID) (skewx & skewy). Returns an empty raster with the specified width, height, xmin, ymin, scalex, scaley, skewx, skewy, SRID and SRID. SRID is optional, defaults to 0(unknown) if not specified.



### 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);

padwithnodata = FALSE, padwithnodata = TRUE, NODATA (padding) (NODATA) nodataval NODATA.



#### Note

DB, DB.

### 2.1.0

```
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 ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
 ST_DumpValues(rast)
FROM baz;
```

## st\_dumpvalues

```

(1,"{{1,1,1},{1,1,1},{1,1,1}}")
(2,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{2,2,2},{2,2,2},{2,2,2}}")
(2,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{3,3,3},{3,3,3},{3,3,3}}")
(2,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{4,4,4},{4,4,4},{4,4,4}}")
(2,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{5,5,5},{5,5,5},{5,5,5}}")
(2,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{6,6,6},{6,6,6},{6,6,6}}")
(2,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{7,7,7},{7,7,7},{7,7,7}}")
(2,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{8,8,8},{8,8,8},{8,8,8}}")
(2,"{{80,80,80},{80,80,80},{80,80,80}}")
(1,"{{9,9,9},{9,9,9},{9,9,9}}")
(2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)
```

```
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 ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
 ST_DumpValues(rast)
FROM baz;
```

## st\_dumpvalues

```

(1,"{{10,10,10},{10,10,10},{10,10,10}}")
(1,"{{20,20,20},{20,20,20},{20,20,20}}")
(1,"{{30,30,30},{30,30,30},{30,30,30}}")
(1,"{{40,40,40},{40,40,40},{40,40,40}}")
(1,"{{50,50,50},{50,50,50},{50,50,50}}")
(1,"{{60,60,60},{60,60,60},{60,60,60}}")
(1,"{{70,70,70},{70,70,70},{70,70,70}}")
(1,"{{80,80,80},{80,80,80},{80,80,80}}")
```







upperleftx  
upperlefty

ESRI:

scalex  
skewy  
skewx  
scaley  
upperleftx + scalex\*0.5  
upperlefty + scaley\*0.5

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
FROM dummy_rast WHERE rid=1;
```

esri_ref	gdal_ref
2.0000000000	2.0000000000
0.0000000000	0.0000000000
0.0000000000	0.0000000000
3.0000000000	3.0000000000
1.5000000000	0.5000000000
2.0000000000	0.5000000000

[ST\\_SetGeoReference](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#)

### 11.4.2 ST\_Height

ST\_Height —

#### Synopsis

integer **ST\_Height**(raster rast);

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;
```

rid	rastheight
1	20
2	5

☒☒

**ST\_Width**

**11.4.3 ST\_IsEmpty**

ST\_IsEmpty — Returns true if the raster is empty (width = 0, height = 0). Returns false otherwise.

**Synopsis**

boolean **ST\_IsEmpty**(raster rast);

☒☒

Returns true if the raster is empty (width = 0, height = 0). Returns false otherwise.  
 2.0.0 Returns true if the raster is empty.

☒☒

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
f |

SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----+
t |
```

☒☒

**ST\_HasNoBand**

**11.4.4 ST\_MemSize**

ST\_MemSize — Returns the memory size of the raster in bytes (KB) if the raster is empty.

**Synopsis**

integer **ST\_MemSize**(raster rast);





**Example:**

```
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;
```

rastheight	pixheight	scalex	scaley	skewx	skewy
20	3	2	3	0	0
5	0.05	0.05	-0.05	0	0

**Example: 0**

```
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;
```

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

**Example:**

[ST\\_PixelWidth](#), [ST\\_ScaleX](#), [ST\\_ScaleY](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 11.4.8 ST\_PixelWidth

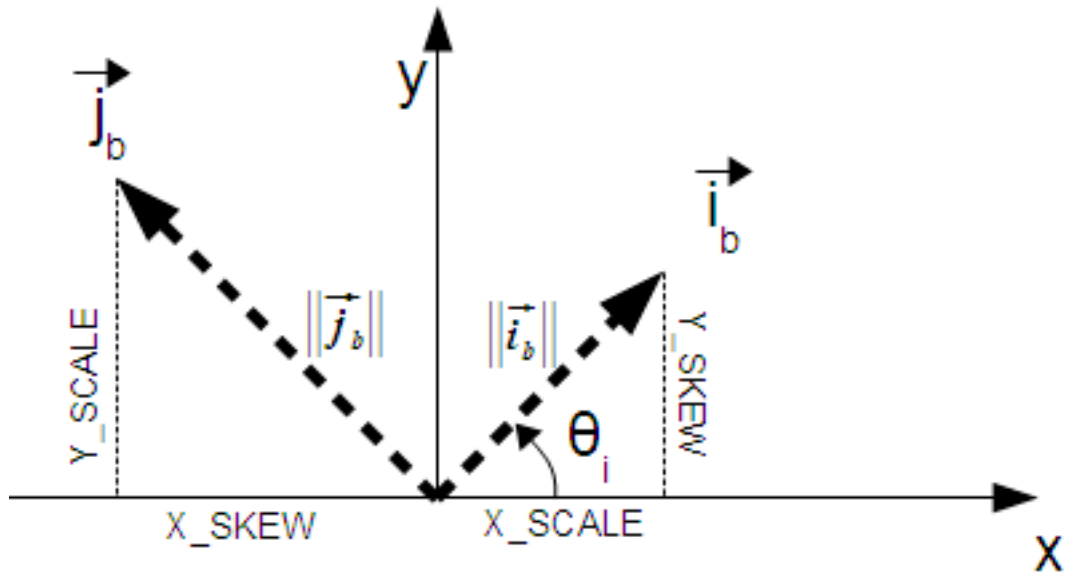
`ST_PixelWidth` — Returns the pixel width of a raster.

#### Synopsis

double precision **ST\_PixelWidth**(raster rast);

**Example:**

`ST_PixelWidth(rast)` returns the pixel width of the raster. `ST_PixelWidth(rast, 1)` returns the pixel width of the raster in the x-direction. `ST_PixelWidth(rast, 2)` returns the pixel width of the raster in the y-direction.



$i$   
 $j$

**Example 1:**

```
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;
```

rastwidth	pixwidth	scalex	scaley	skewx	skewy
10	2	2	3	0	0
5	0.05	0.05	-0.05	0	0

**Example 2:** 0

```
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;
```

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

**Example 3:**

**ST\_PixelHeight, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY**

### 11.4.9 ST\_ScaleX

ST\_ScaleX — X

#### Synopsis

float8 ST\_ScaleX(raster rast);

X.   
 : 2.0.0 WKTRaster ST\_PixelSizeX

```
SELECT rid, ST_ScaleX(rast) As rastpixmapwidth
FROM dummy_rast;
```

rid	rastpixmapwidth
1	2
2	0.05

#### ST\_Width

### 11.4.10 ST\_ScaleY

ST\_ScaleY — Y

#### Synopsis

float8 ST\_ScaleY(raster rast);

Y.   
 : 2.0.0 WKTRaster ST\_PixelSizeY

```
SELECT rid, ST_ScaleY(rast) As rastpixmapheight
FROM dummy_rast;
```

rid	rastpixmapheight
1	3
2	-0.05





☒☒

[ST\\_RasterToWorldCoordX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#)

### 11.4.12 ST\_RasterToWorldCoordX

ST\_RasterToWorldCoordX — Returns the world coordinate X of the pixel at row 1 column 1.

#### Synopsis

float8 **ST\_RasterToWorldCoordX**(raster rast, integer xcolumn);  
 float8 **ST\_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

☒☒

Returns the world coordinate X of the pixel at row 1 column 1 of the raster. If the raster is skewed, the world coordinate X is returned for the pixel at row 1 column 1 of the unskewed raster.



#### Note

ST\_RasterToWorldCoordX, x column. If the raster is skewed, ST\_ScaleX, ST\_SkewX, ST\_SetSkew. Returns the world coordinate X of the pixel at row 1 column 1.

☒☒☒: 2.1.0 replaced ST\_Raster2WorldCoordX.

☒☒

```
-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As xlcoord,
 ST_RasterToWorldCoordX(rast,2) As x2coord,
 ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

rid	xlcoord	x2coord	pixelx
1	0.5	2.5	2
2	3427927.75	3427927.8	0.05

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As xlcoord,
 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;
```

rid	xlcoord	x2coord	pixelx
1	0.5	203.5	2
2	3427927.75	3428128.8	0.05

☒☒

[ST\\_ScaleX](#), [ST\\_RasterToWorldCoordY](#), [ST\\_SetSkew](#), [ST\\_SkewX](#)

### 11.4.13 ST\_RasterToWorldCoordY

ST\_RasterToWorldCoordY — 返回栅格中指定行 Y 的地理坐标。返回类型为 1 的浮点数组。

#### Synopsis

```
float8 ST_RasterToWorldCoordY(raster rast, integer yrow);
float8 ST_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);
```

☒☒

返回栅格中指定行 Y 的地理坐标。返回类型为 1 的浮点数组。如果指定了 X 列，则返回 X 和 Y 的地理坐标。返回类型为 2 的浮点数组。



#### Note

返回栅格中指定行 Y 的地理坐标。返回类型为 1 的浮点数组。如果指定了 X 列，则返回 X 和 Y 的地理坐标。返回类型为 2 的浮点数组。

☒☒☒: 2.1.0 版本中 ST\_Raster2WorldCoordY 函数已弃用。

☒☒

```
-- non-skewed raster providing row is sufficient
SELECT rid, ST_RasterToWorldCoordY(rast,1) As ylcoord,
 ST_RasterToWorldCoordY(rast,3) As y2coord,
 ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

rid	ylcoord	y2coord	pixely
1	0.5	6.5	3
2	5793244	5793243.9	-0.05

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As ylcoord,
 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;
```

rid	ylcoord	y2coord	pixely
1	0.5	107	3
2	5793244	5793344.4	-0.05



☒☒

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000 :
2	0	0	0.0500000000 : 0.0000000000 : 0.0000000000 : -0.0500000000 : 3427927.7500000000 : 5793244.0000000000

☒☒

[ST\\_GeoReference](#), [ST\\_SkewY](#), [ST\\_SetSkew](#)

### 11.4.16 ST\_SkewY

ST\_SkewY — ☒☒☒☒ Y ☒☒☒ (☒☒☒☒☒☒☒☒) ☒☒☒☒☒☒.

#### Synopsis

float8 **ST\_SkewY**(raster rast);

☒☒

☒☒☒☒ Y ☒☒☒ (☒☒☒☒☒☒☒☒) ☒☒☒☒☒☒. ☒☒☒☒☒☒ ☒☒☒☒☒☒ ☒☒☒☒☒☒☒.

☒☒

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000 :



2.1.0

```

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
)
);

 st_summary

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)

```

[ST\\_MetaData](#), [ST\\_BandMetaData](#), [ST\\_Summary](#) [ST\\_Extent](#)

### 11.4.19 ST\_UpperLeftX

ST\_UpperLeftX —

#### Synopsis

float8 **ST\_UpperLeftX**(raster rast);

```

SELECT rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;

```

rid	ulx
1	0.5
2	3427927.75













**Note**

If `isoutdb` is False, `path`, `outdbbandnum`, `filesize` and `filetimestamp` are NULL. If `outdb` access is disabled, `filesize` and `filetimestamp` will also be NULL.

Enhanced: 2.5.0 to include `outdbbandnum`, `filesize` and `filetimestamp` for `outdb` rasters.

Example 1

```
SELECT
 rid,
 (foo.md).*
FROM (
 SELECT
 rid,
 ST_BandMetaData(rast, 1) AS md
 FROM dummy_rast
 WHERE rid=2
) As foo;
```

rid	pixeltype	nodatavalue	isoutdb	path	outdbbandnum
2	8BUI		f		

Example 2

```
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
 *
FROM ST_BandMetadadata(
 (SELECT rast FROM foo),
 ARRAY[1,3,2]::int[]
);
```

bandnum	pixeltype	nodatavalue	isoutdb	outdbbandnum	filesize	filetimestamp	path
1	8BUI		t	1	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
3	8BUI		t	3	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif
2	8BUI		t	2	12345	1521807257	/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected.tif

Example

[ST\\_MetaData](#), [ST\\_BandPixelType](#)

### 11.5.2 ST\_BandNoDataValue

ST\_BandNoDataValue — Returns the NODATA value for a given band number. If the band number is 1, the NODATA value is the value of the first band.

#### Synopsis

double precision **ST\_BandNoDataValue**(raster rast, integer bandnum=1);

Parameters

**bandnum** NODATA value for the given band number.

Return Value

```
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;
```

bnval1	bnval2	bnval3
0	0	0

Notes

#### ST\_NumBands

### 11.5.3 ST\_BandIsNoData

ST\_BandIsNoData — Returns TRUE if the NODATA value for a given band number is the same as the NODATA value for the first band.

#### Synopsis

boolean **ST\_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);  
 boolean **ST\_BandIsNoData**(raster rast, boolean forceChecking=true);

Parameters

**band** NODATA value for the given band number. If the band number is 1, the NODATA value is the value of the first band. If **forceChecking** is TRUE, the NODATA value for the given band number is compared to the NODATA value for the first band. If **forceChecking** is FALSE, the NODATA value for the given band number is compared to the NODATA value for the first band only if the first band is not NULL.

2.0.0 Added.



#### Note

ST\_BandIsNoData (rast, band, TRUE) is equivalent to ST\_SetBandNoDataValue(rast, band, TRUE) = TRUE. ST\_SetBandIsNoData(rast, band, TRUE) is equivalent to ST\_SetBandIsNoData(rast, band, TRUE) = TRUE. ST\_SetBandIsNoData(rast, TRUE) is equivalent to ST\_SetBandIsNoData(rast, TRUE) = TRUE.

☒☒

```

-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
 = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
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

```

☒☒

**ST\_BandNoDataValue, ST\_NumBands, ST\_SetBandNoDataValue, ST\_SetBandIsNoData**

## 11.5.4 ST\_BandPath

`ST_BandPath` — Returns the path of a band stored in file system. `bandnum` defaults to 1.

### Synopsis

```
text ST_BandPath(raster rast, integer bandnum=1);
```

Examples

```
SELECT ST_BandPath(rast, 1) FROM dummy_rast WHERE rid = 1;
```

Results

Results

## 11.5.5 ST\_BandFileSize

`ST_BandFileSize` — Returns the file size of a band stored in file system. If no `bandnum` specified, 1 is assumed.

### Synopsis

```
bigint ST_BandFileSize(raster rast, integer bandnum=1);
```

Examples

Returns the file size of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with `ST_BandPath()` and `ST_BandFileTimestamp()` so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

Results

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;
```

```
st_bandfilesize

240574
```



- 32BSI - 32 signed integers
- 32BUI - 32 unsigned integers
- 32BF - 32 floating point numbers
- 64BF - 64 floating point numbers

Example

```
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;
```

btype1	btype2	btype3
8BUI	8BUI	8BUI

Example

**ST\_NumBands**

**11.5.8 ST\_MinPossibleValue**

ST\_MinPossibleValue — Returns the minimum possible value for a pixel type.

**Synopsis**

integer **ST\_MinPossibleValue**(text pixeltype);

Example

Example: Returns the minimum possible value for a pixel type.

Example

```
SELECT ST_MinPossibleValue('16BSI');
 st_minpossiblevalue

 -32768

SELECT ST_MinPossibleValue('8BUI');
 st_minpossiblevalue

 0
```





☒☒

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(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;
```

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, ..

☒☒

[ST\\_DumpAsPolygons](#), [ST\\_PixelAsPolygons](#), [ST\\_PixelAsPoint](#), [ST\\_PixelAsPoints](#), [ST\\_PixelAsCentroid](#), [ST\\_PixelAsCentroids](#), [ST\\_Intersection](#), [ST\\_AsText](#)

### 11.6.2 ST\_PixelAsPolygons

**ST\_PixelAsPolygons** — Returns the geometry of the pixel at the given X, Y coordinates.

#### Synopsis

setof record **ST\_PixelAsPolygons**(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

☒☒

Returns a record with the following fields: *geom* (geometry), *val* (double precision), *x* (integer), *y* (integer).

Return record format: *geom geometry*, *val* double precision, *x* integer, *y* integers.

**Note!** When *exclude\_nodata\_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

**Note!** **ST\_PixelAsPolygons** returns a record with the following fields: *geom* (geometry), *val* (double precision), *x* (integer), *y* (integer). **ST\_DumpAsPolygons** returns a record with the following fields: *geom* (geometry), *val* (double precision), *x* (integer), *y* (integer).

2.0.0 Returns a record with the following fields: *geom* (geometry), *val* (double precision), *x* (integer), *y* (integer).

☒☒☒: 2.1.0 Returns a record with the following fields: *geom* (geometry), *val* (double precision), *x* (integer), *y* (integer).

☒☒☒☒: 2.1.1 Returns a record with the following fields: *geom* (geometry), *val* (double precision), *x* (integer), *y* (integer).



### 11.6.4 ST\_PixelAsPoints

ST\_PixelAsPoints — Returns a set of record (x, y, val) for each pixel in the raster. The x and y coordinates are the pixel's center coordinates.

#### Synopsis

setof record **ST\_PixelAsPoints**(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

Parameters

**rast**: raster. **band**: integer. **exclude\_nodata\_value**: boolean. **geom geometry**, **val** double precision, **x** integer, **y** integers.

Return record format: *geom geometry*, *val* double precision, *x* integer, *y* integers.



**Note**

When *exclude\_nodata\_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

2.1.0 Returns a set of record (x, y, val).

Parameters: 2.1.1 **exclude\_nodata\_value** boolean.

Example

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM dummy_rast WHERE rid = 2) foo;
```

x	y	val	st_astext
1	1	253	POINT(3427927.75 5793244)
2	1	254	POINT(3427927.8 5793244)
3	1	253	POINT(3427927.85 5793244)
4	1	254	POINT(3427927.9 5793244)
5	1	254	POINT(3427927.95 5793244)
1	2	253	POINT(3427927.75 5793243.95)
2	2	254	POINT(3427927.8 5793243.95)
3	2	254	POINT(3427927.85 5793243.95)
4	2	253	POINT(3427927.9 5793243.95)
5	2	249	POINT(3427927.95 5793243.95)
1	3	250	POINT(3427927.75 5793243.9)
2	3	254	POINT(3427927.8 5793243.9)
3	3	254	POINT(3427927.85 5793243.9)
4	3	252	POINT(3427927.9 5793243.9)
5	3	249	POINT(3427927.95 5793243.9)
1	4	251	POINT(3427927.75 5793243.85)
2	4	253	POINT(3427927.8 5793243.85)
3	4	254	POINT(3427927.85 5793243.85)
4	4	254	POINT(3427927.9 5793243.85)
5	4	253	POINT(3427927.95 5793243.85)
1	5	252	POINT(3427927.75 5793243.8)
2	5	250	POINT(3427927.8 5793243.8)
3	5	254	POINT(3427927.85 5793243.8)
4	5	254	POINT(3427927.9 5793243.8)
5	5	254	POINT(3427927.95 5793243.8)





### 11.6.7 ST\_Value

ST\_Value — columnx, rowy, exclude\_nodata\_value, nodata, exclude\_nodata\_value.

#### 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);

columnx, rowy, exclude\_nodata\_value, nodata, exclude\_nodata\_value.

The allowed values of the resample parameter are "nearest" which performs the default nearest-neighbor resampling, and "bilinear" which performs a bilinear interpolation to estimate the value between pixel centers.

2.1.0 exclude\_nodata\_value  
 2.0.0 exclude\_nodata\_value

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
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;

rid | b1pval | b2pval
-----+-----+-----
 2 | 252 | 79

-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```

```
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;

rid | b1pval | b2pval | b3pval
-----+-----+-----+-----
 2 | 253 | 78 | 70
```

```

--- Get all values in bands 1,2,3 of each pixel --
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);

```

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 --
SELECT ST_AsText(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);

```

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 particular polygon --
SELECT ST_AsText(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;
```

```
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
```

```
SELECT ST_AsText(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;
```

```
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)))
```



```

 2, 3, 0.
),
 3, 5, 0.
),
 4, 2, 0.
),
 5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
1 | 1

```

```

-- pixel 2x3 is NODATA
SELECT
 ST_Value(rast, 2, 3) AS value,
 ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
 SELECT
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_SetValue(
 ST_AddBand(
 ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
 '8BUI'::text, 1, 0
),
 1, 1, 0.
),
 2, 3, 0.
),
 3, 5, 0.
),
 4, 2, 0.
),
 5, 4, 0.
) AS rast
) AS foo

value | nearestvalue
-----+-----
| 1

```



**ST\_Neighborhood, ST\_Value**

### 11.6.9 ST\_SetZ

**ST\_SetZ** — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.

#### Synopsis

geometry **ST\_SetZ**(raster rast, geometry geom, text resample=nearest, integer band=1);

☒☒

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use **bilinear interpolation** to calculate a value that takes neighboring cells into account also.

Availability: 3.2.0

☒☒

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
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_AsText(
 ST_SetZ(
 rast,
 band => 1,
 geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
 resample => 'bilinear'
))
FROM test_raster

 st_astext

LINESTRING Z (1 1.9 38,1 0.2 27)
```

☒☒

**ST\_Value, ST\_SetSRID**

### 11.6.10 ST\_SetM

`ST_SetM` — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.

**Synopsis**

geometry **ST\_SetM**(raster rast, geometry geom, text resample=nearest, integer band=1);

☒☒

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimensions using the requested resample algorithm.

The resample parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use **bilinear interpolation** to calculate a value that takes neighboring cells into account also.

Availability: 3.2.0

☒☒

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
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_AsText(
 ST_SetM(
 rast,
 band => 1,
 geom => 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)::geometry,
 resample => 'bilinear'
))
FROM test_raster

 st_astext

LINESTRING M (1 1.9 38,1 0.2 27)
```

☒☒

**ST\_Value**, **ST\_SetSRID**



```

 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

 st_neighborhood

{{NULL,1,1},{1,1,1},{1,NULL,1}}

```

```

-- pixel 2x3 is NODATA
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

 st_neighborhood

{{1,1,1},{1,NULL,1},{1,1,1}}

```

```

-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
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

 st_neighborhood

```







☒☒: ☒☒ 1

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
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[][]
)
) AS poly
) foo
ORDER BY 1, 2;

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 ST_SetValues() does the following...

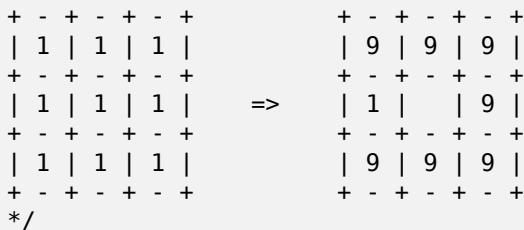
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 9 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | =
> | 9 | | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 9 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
SELECT
 (poly).x,

```

```
(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[][]
)
) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	9
1	2	9
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

/\*  
The ST\_SetValues() does the following...



```
*/
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[[[]]]
)
) AS poly
) foo
ORDER BY 1, 2;
```

x	y	val
1	1	9
1	2	9
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

1	1	9
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| | 1 | 1 | | | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | => | 1 | | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 9 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
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
)
) AS poly
) foo
ORDER BY 1, 2;

```

x	y	val
1	1	
1	2	1
1	3	9
2	1	9
2	2	
2	3	9
3	1	9
3	2	9
3	3	9

☒☒: ☒☒ 2

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | => | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
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
)
) AS poly
) foo
ORDER BY 1, 2;

```

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

```

/*
This example is like the previous one. Instead of nosetvalue = -1, nosetvalue = NULL

The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | => | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
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, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double ←
 precision[[]], NULL::double precision
)
) AS poly
) foo
ORDER BY 1, 2;

```

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

☒☒: ☒☒ 3

```

/*
The ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | => | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
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
)
) AS poly
) foo
ORDER BY 1, 2;

```

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 ST_SetValues() does the following...

+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 1 | 1 |
+ - + - + - + + - + - + - +
| 1 | | 1 | => | 1 | | 9 |
+ - + - + - + + - + - + - +
| 1 | 1 | 1 | | 1 | 9 | 9 |
+ - + - + - + + - + - + - +
*/
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
)
) AS poly
) foo
ORDER BY 1, 2;

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

```

☒☒: ☒☒ 5

```

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
)
SELECT
 rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;

```

rid	gid	st_dumpvalues
1	1	(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,1,NULL, ← NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	2	(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL ← ,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	3	(1,"{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL, ← NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
1	4	(1,"{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ← NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")

(4 rows)

geomvals geomvals geomvals geomvals geomvals geomvals geomvals geomvals geomvals geomvals.

```

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
)
SELECT
 t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ←
 ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
 AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;

```

rid	gid	gid	st_dumpvalues
1	1	2	(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{ ← NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

geomvals geomvals geomvals geomvals geomvals geomvals geomvals geomvals geomvals geomvals.

```

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
)
SELECT
 t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ←
 ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
 AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;

```

rid	gid	gid	st_dumpvalues
1	2	1	(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ← NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}")

(1 row)

☒☒

[ST\\_Value](#), [ST\\_SetValue](#), [ST\\_PixelAsPolygons](#)

### 11.6.14 ST\_DumpValues

ST\_DumpValues — ☒☒☒☒☒☒☒☒☒ 2 ☒☒☒☒☒☒☒☒☒☒☒☒.

#### 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
);

```

☒☒

☒☒☒☒☒☒☒☒☒ 2 ☒☒☒☒☒☒☒☒☒☒☒☒ (☒☒☒☒☒☒☒☒☒, ☒☒☒☒☒☒☒☒☒☒☒☒). nband ☒ NULL ☒  
☒☒☒☒☒☒☒☒☒, ☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒☒.

2.1.0 ☒☒☒☒☒☒☒☒☒☒☒☒.

☒☒

```

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;

```





## 11.7 地理参考

### 11.7.1 ST\_SetGeoReference

ST\_SetGeoReference — 将地理参考信息设置到 6 波段栅格。支持 GDAL 和 ESRI 格式。GDAL 格式。

#### 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);

参数

georefcoords 6 波段栅格。'GDAL' 或 'ESRI' 格式。GDAL 格式。6 波段栅格 NULL 值。

格式:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```



#### Note

DB 格式，支持 6 波段栅格。ESRI 格式。

更新: 2.1.0 支持 ST\_SetGeoReference(raster, double precision, ...) 格式。

示例

```
WITH foo AS (
 SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
 0 AS rid, (ST_Metadata(rast)).*
FROM foo
UNION ALL
SELECT
 1, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
 2, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
```



### 11.7.3 ST\_SetScale

ST\_SetScale — X Y  $\frac{x}{y}$ .  $\frac{x}{y}$ .

#### Synopsis

raster **ST\_SetScale**(raster rast, float8 xy);  
 raster **ST\_SetScale**(raster rast, float8 x, float8 y);

**Notes**

X Y  $\frac{x}{y}$ .  $\frac{x}{y}$ .  $\frac{x}{y}$ , X Y  $\frac{x}{y}$ .



#### Note

ST\_SetScale  $\frac{x}{y}$  **ST\_Rescale**  $\frac{x}{y}$ .  $\frac{x}{y}$  ( $\frac{x}{y}$ )  $\frac{x}{y}$ . ST\_Rescale  $\frac{x}{y}$ . ST\_SetScale  $\frac{x}{y}$ .

**Changes:** 2.0.0 WKTRaster ST\_SetPixelSize 2.0.0

**Examples**

```
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;

 pixx | pixy | newbox
-----+-----+-----
 1.5 | 1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)
```

```
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;

 pixx | pixy | newbox
-----+-----+-----
 1.5 | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0)
```

**See Also**

[ST\\_ScaleX](#), [ST\\_ScaleY](#), [Box3D](#)

### 11.7.4 ST\_SetSkew

ST\_SetSkew — `ST_SetSkew(raster rast, float8 skewx, float8 skewy)` returns a raster with the same geometry as the input raster, but with the specified skew values. The skew values are applied to the raster's georeference information.

#### Synopsis

```
raster ST_SetSkew(raster rast, float8 skewxy);
raster ST_SetSkew(raster rast, float8 skewx, float8 skewy);
```

Examples

Example 1: Set skew values to 1 and 2. The output raster's georeference information is updated to reflect these values.

Example 2

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	1	2	2.0000000000 : 2.0000000000 : 1.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

```
-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
 ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

rid	skewx	skewy	georef
1	0	0	2.0000000000 : 0.0000000000 : 0.0000000000 : 3.0000000000 : 0.5000000000 : 0.5000000000

See also

[ST\\_GeoReference](#), [ST\\_SetGeoReference](#), [ST\\_SkewX](#), [ST\\_SkewY](#)

### 11.7.5 ST\_SetSRID

ST\_SetSRID — Returns SRID of spatial\_ref\_sys SRID.

#### Synopsis

raster **ST\_SetSRID**(raster rast, integer srid);

SRID.



#### Note

SRID. SRID.

Section 4.5, ST\_SRID

### 11.7.6 ST\_SetUpperLeft

ST\_SetUpperLeft — Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

#### Synopsis

raster **ST\_SetUpperLeft**(raster rast, double precision x, double precision y);

Set the value of the upper left corner of raster to the projected X and Y coordinates

```
SELECT ST_SetUpperLeft(rast, -71.01,42.37)
FROM dummy_rast
WHERE rid = 2;
```

ST\_UpperLeftX, ST\_UpperLeftY



### 11.7.7 ST\_Resample

ST\_Resample — Resamples a raster to a new size and/or projection. The new raster is created by reprojecting the original raster to the new size and/or projection.

#### 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);

Parameters

width, height (width & height), gridx & gridy (gridx & gridy), scalex, scaley, skewx & skewy (scalex, scaley, skewx & skewy) SRID (SRID).

New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

maxerr 0.125



#### Note

GDAL Warp resampling methods

2.0.0. GDAL 1.6.1

Enhanced: 3.4.0 max and min resampling options added

☒☒

```

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;

orig_width | new_width
-----+-----
 200 | 100

```

☒☒

[ST\\_Rescale](#), [ST\\_Resize](#), [ST\\_Transform](#)

### 11.7.8 ST\_Rescale

**ST\_Rescale** — Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is 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);

```

☒☒

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max

- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

scalex and scaley define the new pixel size. scaley must often be negative to get well oriented raster.

scalex scaley, ST\_Resize

maxerr is the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no maxerr is specified, which is the same value used in GDAL gdalwarp utility. If set to zero, no approximation takes place.



**Note**

GDAL Warp resampling methods



**Note**

ST\_Rescale ST\_SetScale

2.0.0 GDAL 1.6.1

Enhanced: 3.4.0 max and min resampling options added

2.1.0 SRID

0.001 0.0015

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269), '8BUI'::text, 1, 0)) width

width

0.001

-- the rescaled raster raster pixel size
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

width

0.0015
```

ST\_Resize, ST\_Resample, ST\_SetScale, ST\_ScaleX, ST\_ScaleY, ST\_Transform

### 11.7.9 ST\_Reskew

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);

#### 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);

skewx

skewy (NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos, NearestNeighbor).

maxerr

0.125



**Note**

GDAL Warp resampling methods



**Note**

ST\_Reskew uses ST\_SetSkew. ST\_SetSkew (skewx, skewy) rotates the raster. ST\_Reskew uses ST\_SetSkew (skewx, skewy, maxerr).

GDAL 1.6.1

SRID

skewx

0.0 0.0015

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269)
, '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
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));

-- result
-0.982793723247329
```

















☒☒

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ←
= 3.
-- In the second band, nodatavalue = 13, pixel value = 4
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. We are going to set it to true
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
```

☒☒

[ST\\_BandNoDataValue](#), [ST\\_NumBands](#), [ST\\_SetBandNoDataValue](#), [ST\\_BandIsNoData](#)

### 11.8.3 ST\_SetBandPath

ST\_SetBandPath — Update the external path and band number of an out-db band

#### Synopsis

raster **ST\_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false)

☒☒

Updates an out-db band's external raster file path and external band number.



#### Note

If force is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for file system changes where the external raster resides.

Availability: 2.5.0

☒☒

```
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,
 *
FROM ST_BandMetadata(
 (SELECT rast FROM foo),
 ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
 2,
 *
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
),
),
```



☒☒

```

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,
 *
FROM ST_BandMetadata(
 (SELECT rast FROM foo),
 ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
 2,
 *
FROM ST_BandMetadata(
 (
 SELECT
 ST_SetBandIndex(
 rast,
 2,
 1
) AS rast
 FROM foo
),
 ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

```

query	bandnum	pixeltype	nodatavalue	isoutdb	path
outdbbandnum					
1	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif   1
1	2	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif   2
1	3	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif   3
2	1	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif   1
2	2	8BUI		t	<b>/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif   1</b>
2	3	8BUI		t	/home/pele/devel/geo/postgis-git/ ↵ raster/test/regress/loader/Projected.tif   3

☒☒

**ST\_BandMetaData, ST\_SetBandPath**

## 11.9

### 11.9.1 ST\_Count

ST\_Count — Returns the number of pixels in the raster that are not equal to the specified value. The value is compared to the pixel values in the raster. If the value is not found, the function returns 0. The function is implemented as a SQL aggregate function. The function signature is: ST\_Count(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, text nodata\_value=NODATA). The function returns a bigint.

#### Synopsis

bigint ST\_Count(raster rast, integer nband=1, boolean exclude\_nodata\_value=true);  
bigint ST\_Count(raster rast, boolean exclude\_nodata\_value);

Examples

Count the number of pixels in the raster that are not equal to the specified value. The function is implemented as a SQL aggregate function. The function signature is: ST\_Count(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, text nodata\_value=NODATA). The function returns a bigint.

**Note** exclude\_nodata\_value, nodata\_value. The function signature is: ST\_Count(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, text nodata\_value=NODATA). The function returns a bigint.

2.2.0 ST\_Count(rastertable, rastercolumn, ...) Returns the number of pixels in the raster that are not equal to the specified value. The function is implemented as a SQL aggregate function. The function signature is: ST\_Count(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, text nodata\_value=NODATA). The function returns a bigint.

2.0.0 ST\_Count(raster rast, boolean exclude\_nodata\_value);

Examples

```
--example will count all pixels not 249 and one will count all pixels. --
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;

rid | exclude_nodata | include_nodata
-----+-----+-----
 2 | 23 | 25
```

Examples

ST\_CountAgg, ST\_SummaryStats, ST\_SetBandNoDataValue

### 11.9.2 ST\_CountAgg

ST\_CountAgg — Returns the number of pixels in the raster that are not equal to the specified value. The function is implemented as a SQL aggregate function. The function signature is: ST\_CountAgg(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, text nodata\_value=NODATA). The function returns a bigint.





### 11.9.3 ST\_Histogram

ST\_Histogram — (bin; ...) returns a table with columns min, max, count, percent, nband.

#### 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);

min

min, max, count, percent columns. nband column 1.



#### Note

exclude\_nodata\_value option. X [a, b] (a, b)

width width: ... width [a, b, c] [a, b, c, a, b, c, a, b, c]

bins (breakout) ...

right ... X [a, b] (a, b)

Changed: 3.1.0 Removed ST\_Histogram(table\_name, column\_name) variant.

2.0.0

1, 2, 3

```
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;
```

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

Example: `ST_Histogram(rast, 2,6)`

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
 FROM dummy_rast
 WHERE rid=2) As foo;
```

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.

```
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;
```

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)

Example

[ST\\_Count](#), [ST\\_SummaryStats](#), [ST\\_SummaryStatsAgg](#)

### 11.9.4 ST\_Quantile

`ST_Quantile` — `(population)` `(quantile)`. `25%`, `50%`, `75%` `(percentile)`.

**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);

(population) (quantile) ., 25%, 50%, 75% (percentile).



**Note**

exclude\_nodata\_value, NODATA.

Changed: 3.1.0 Removed ST\_Quantile(table\_name, column\_name) variant.

2.0.0.

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --
```

```
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;
```

```
quantile | value
-----+-----
 0.25 | 253
 0.75 | 254
```

```
SELECT ST_Quantile(rast, 0.75) As value
FROM dummy_rast WHERE rid=2;
```

```
value

 254
```

```
--real live example. Quantile of all pixels in band 2 intersecting a geometry
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) ←
```





**Note**

sample\_percent 1

2.2.0 ST\_SummaryStats(rastertable, rastercolumn, ...) ST\_SummaryStatsAgg

2.0.0

Example 1

```
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;
```

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

Example 2

PostGIS 64 (102,000, 150x150 134,000) 574

```
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(feats.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;
```

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

SQL: `ST_SummaryStats`

```
-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
 FROM generate_series(1,3) As band) As foo;
```

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 -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
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;
```

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

SQL

[summarystats](#), [ST\\_SummaryStatsAgg](#), [ST\\_Count](#), [ST\\_Clip](#)

### 11.9.6 ST\_SummaryStatsAgg

`ST_SummaryStatsAgg` — Aggregates a raster into a table with columns for count, sum, mean, stddev, min, max. `ST_SummaryStatsAgg` is a `SQL-MM` function. `ST_SummaryStatsAgg` is a `SQL-MM` function.

#### 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);
```

SQL

`ST_SummaryStatsAgg` returns a table with columns for count, sum, mean, stddev, min, max. `ST_SummaryStatsAgg` is a `SQL-MM` function. `ST_SummaryStatsAgg` is a `SQL-MM` function.



#### Note

`ST_SummaryStatsAgg` `exclude_nodata_value` is a `SQL-MM` function. `ST_SummaryStatsAgg` is a `SQL-MM` function.



**Note**

sample\_percent 0 1

2.2.0

```

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,
 round((stats).sum::numeric, 3),
 round((stats).mean::numeric, 3),
 round((stats).stddev::numeric, 3),
 round((stats).min::numeric, 3),
 round((stats).max::numeric, 3)
FROM (
 SELECT
 ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
 FROM foo
) bar;

```

count	round	round	round	round	round
20	-68.584	-3.429	6.571	-10.000	3.142

(1 row)

summarystats, ST\_SummaryStats, ST\_Count, ST\_Clip



### 11.9.7 ST\_ValueCount

ST\_ValueCount — Returns the value and count of pixels in a raster that match the searchvalues. If searchvalues is NULL, the function returns the value and count of all pixels in the raster. If searchvalues is an array, the function returns the value and count for each element in the array. If searchvalues is a single value, the function returns the value and count for that value. If searchvalues is a single value and exclude\_nodata\_value is true, the function returns the value and count for that value, excluding NODATA pixels. If searchvalues is an array and exclude\_nodata\_value is true, the function returns the value and count for each element in the array, excluding NODATA pixels.

#### 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, in-
teger 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 ex-
clude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, dou-
ble 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 pre-
cision roundto=0);
bigint ST_ValueCount(text rastertable, text rastercolumn, integer nband, double precision search-
value, double precision roundto=0);

```

Examples

SELECT ST\_ValueCount(rast, 1, NULL) AS value, count FROM rastertable;

SELECT ST\_ValueCount(rast, 1, 1) AS value, count FROM rastertable;
SELECT ST\_ValueCount(rast, 1, 1, 1) AS value, count FROM rastertable;



**Note** exclude\_nodata\_value is true, NODATA pixels are excluded.

2.0.0 Returns the value and count of pixels in a raster that match the searchvalues.

Examples

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will count only pixels of band 1 that are not 249. --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvc).value;
```

value	count
250	2
251	1
252	2
253	6
254	12

```
-- Example will count all pixels of band 1 including 249 --
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvc).value;
```

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
```

```
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
 FROM dummy_rast WHERE rid=2) As foo
 ORDER BY (pvc).value;
```

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
```

```
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
```

























### 11.12.2 ST\_ColorMap

`ST_ColorMap` — Returns a raster of type 8BUI (grayscale, RGB, RGBA) with 4 bands. The first band is the input raster.

#### 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);`

`rast`

`nband` integer colormap text 8BUI 4

`colormap` text 8BUI

`method` text, 1

`colormap` text:

- `grayscale` - grayscale - 8BUI (shades of gray)
- `pseudocolor` - 8BUI(rgba) 4 bands, 0-255
- `fire` - 8BUI(rgba) 4 bands, 0-255
- `bluered` - 8BUI(rgba) 4 bands, 0-255

`colormap` (text) text 5 bands, 0-255 (rgba) 0-255. `nodata` text `nv, null` text `nodata` text.

```
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
```

`colormap` text GDAL text (color-relief) text `gdaldem` text.

`method` text:

- `INTERPOLATE` -
- `EXACT` -
- `NEAREST` -



#### Note

ColorBrewer text.



**Warning**

ST\_SetBandNoDataValue NODATA NODATA

2.1.0

```
-- setup test raster table --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

INSERT INTO funky_shapes(rast)
WITH ref AS (
 SELECT ST_MakeEmptyRaster(200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
 ST_Union(rast)
FROM (
 SELECT
 ST_AsRaster(
 ST_Rotate(
 ST_Buffer(
 ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
 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
 ST_NumBands(rast) As n_orig,
 ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
 ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
 ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
 ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
 ST_NumBands(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As nred
FROM funky_shapes;
```

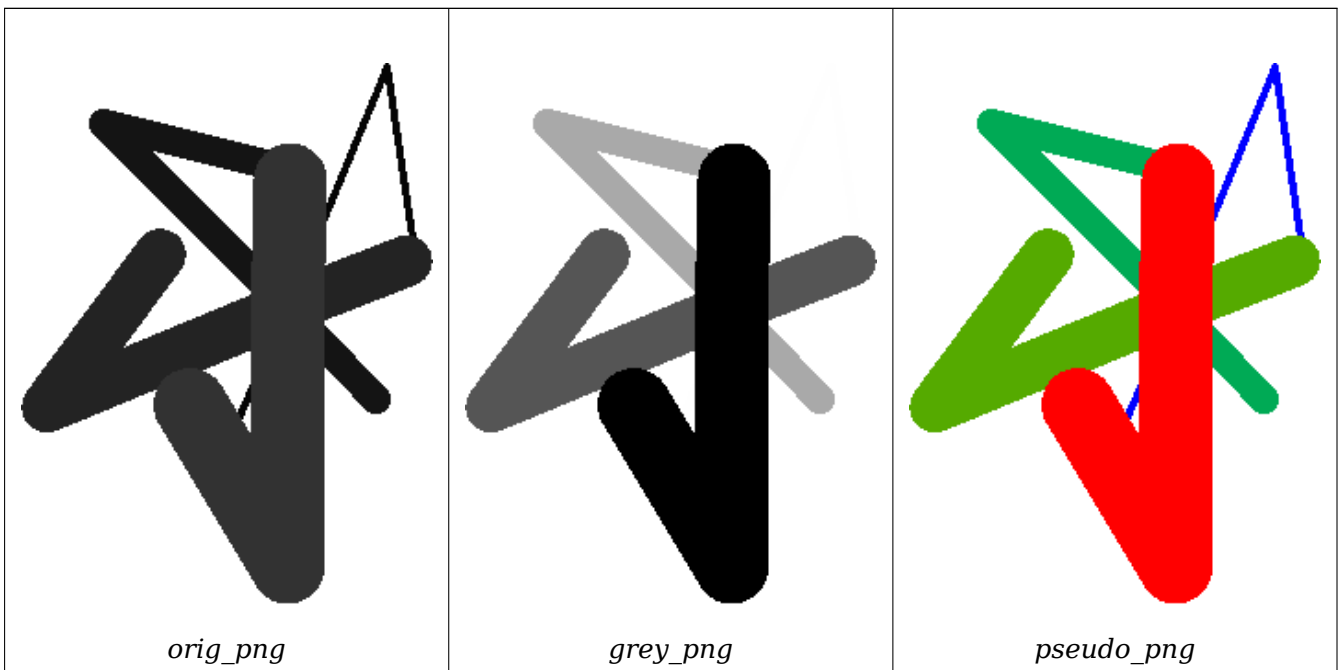
n_orig	ngrey	npseudo	nfire	nbluered	nred
1	1	4	4	4	3

## ST\_AsPNG

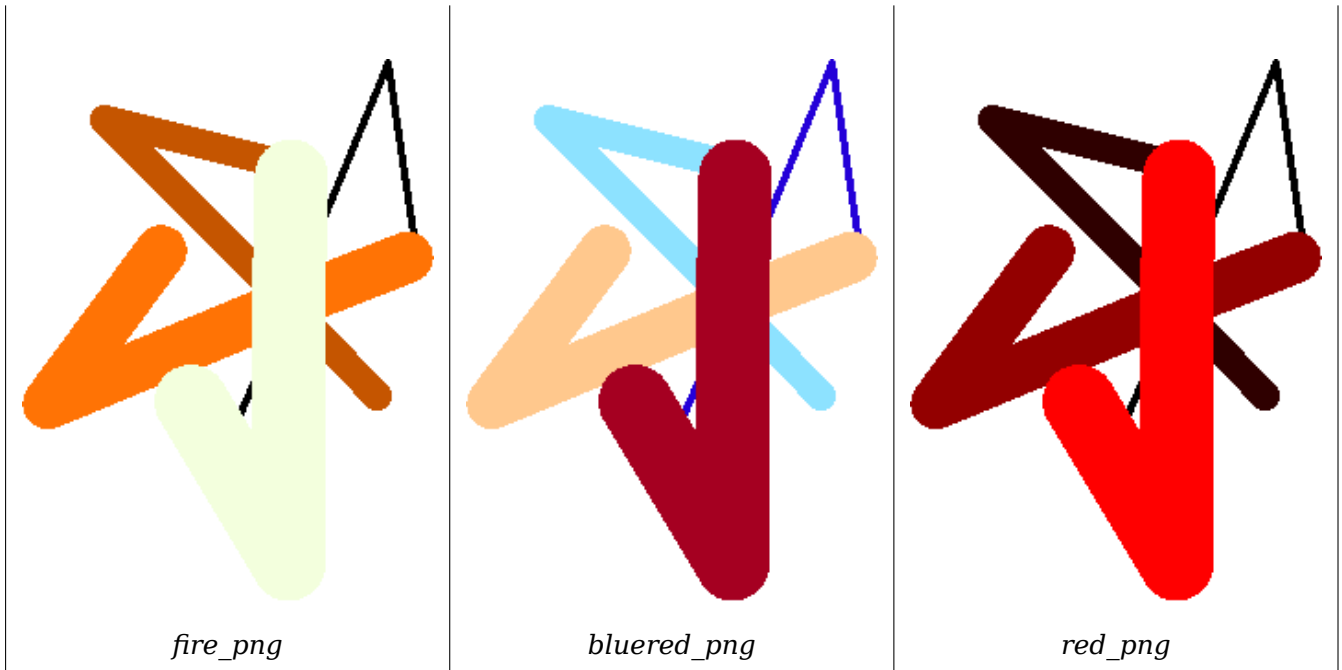
```

SELECT
 ST_AsPNG(rast) As orig_png,
 ST_AsPNG(ST_ColorMap(rast,1,'greyscale')) As grey_png,
 ST_AsPNG(ST_ColorMap(rast,1,'pseudocolor')) As pseudo_png,
 ST_AsPNG(ST_ColorMap(rast,1,'nfire')) As fire_png,
 ST_AsPNG(ST_ColorMap(rast,1,'bluered')) As bluered_png,
 ST_AsPNG(ST_ColorMap(rast,1, '
100% 255 0 0
80% 160 0 0
50% 130 0 0
30% 30 0 0
20% 60 0 0
0% 0 0 0
nv 255 255 255
')) As red_png
FROM funky_shapes;

```







☒☒

[ST\\_AsPNG](#), [ST\\_AsRaster](#) [ST\\_MapAlgebra](#) (callback function version), [ST\\_Grayscale](#) [ST\\_NumBands](#), [ST\\_Reclass](#), [ST\\_SetBandNoDataValue](#), [ST\\_Union](#)

### 11.12.3 ST\_Grayscale

`ST_Grayscale` — Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

#### 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);

☒☒

Create a raster with one 8BUI band given three input bands (from one or more rasters). Any input band whose pixel type is not 8BUI will be reclassified using [ST\\_Reclass](#).



#### Note

This function is not like [ST\\_ColorMap](#) with the `grayscale` keyword as `ST_ColorMap` operates on only one band while this function expects three bands for RGB. This function applies the following equation for converting RGB to Grayscale:  $0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE$

Availability: 2.5.0

## ☒☒: ☒☒ 1

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
 SELECT ST_AddBand(
 ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
 '/tmp/apple.png'::text,
 NULL::int[]
) AS rast
)
SELECT
 ST_AsPNG(rast) AS original_png,
 ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```

*original\_png**grayscale\_png*

## ☒☒: ☒☒ 2

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
 SELECT ST_AddBand(
 ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
 '/tmp/apple.png'::text,
 NULL::int[]
) AS rast
)
SELECT
 ST_AsPNG(rast) AS original_png,
 ST_AsPNG(ST_Grayscale(
 ARRAY[
 ROW(rast, 1)::rastbandarg, -- red
 ROW(rast, 2)::rastbandarg, -- green
 ROW(rast, 3)::rastbandarg, -- blue
]::rastbandarg[]
)) AS grayscale_png
```

FROM apple;

[ST\\_AsPNG](#), [ST\\_Reclass](#), [ST\\_ColorMap](#)

### 11.12.4 ST\_Intersection

ST\_Intersection — Returns the intersection of a geometry and a raster. The geometry is converted to a polygon if it is not already one.

#### 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 geom);
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);
```

ST\_Intersection returns the intersection of a geometry and a raster. The geometry is converted to a polygon if it is not already one.

geomval ST\_Intersection(geometry geom, raster rast, integer band\_num). (ST\_DumpAsPolygon) ST\_Intersection(geometry geom, raster rast, integer band\_num) PostGIS. WHERE ST\_Intersects(geom, rast) AND ST\_Intersection(geom, rast, band\_num) IS NOT NULL.

geomval ST\_Intersection(raster rast, geometry geom). (ST\_DumpAsPolygon) ST\_Intersection(raster rast, geometry geom) PostGIS. WHERE ST\_Intersects(geom, rast) AND ST\_Intersection(geom, rast) IS NOT NULL.

raster ST\_Intersection(raster rast1, raster rast2, double precision[] nodataval). ST\_MapAlgebraExpr(rast1, rast2, nodataval, 2, ST\_Intersection, nodataval).

raster ST\_Intersection(raster rast1, raster rast2, text returnband, double precision[] nodataval). ST\_MapAlgebraExpr(rast1, rast2, nodataval, 2, ST\_Intersection, returnband, nodataval).

raster ST\_Intersection(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval). ST\_MapAlgebraExpr(rast1, rast2, nodataval, 2, ST\_Intersection, band1, band2, nodataval).

raster ST\_Intersection(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval). ST\_MapAlgebraExpr(rast1, rast2, nodataval, 2, ST\_Intersection, returnband, band1, band2, nodataval).



Note

ST\_MapAlgebraExpr NODATA



Note

ST\_Clip



Note

ST\_Intersects ST\_Intersection

2.0.0

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

```

SELECT
 foo.rid,
 foo.gid,
 ST_AsText((foo.geomval).geom) As geomwkt,
 (foo.geomval).val
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, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8 5793243.8)')),
 (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
) As g(gid,geom)
 WHERE A.rid = 2
) As foo;

```

rid	gid	geomwkt	val
2	1	POINT(3427928 5793243.85)	249
2	1	POINT(3427928 5793243.85)	253
2	2	POINT(3427927.85 5793243.75)	254
2	2	POINT(3427927.8 5793243.8)	251
2	2	POINT(3427927.8 5793243.8)	253
2	2	LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)	252
2	2	MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...)	250
2	3	GEOMETRYCOLLECTION EMPTY	

geomval

ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

### 11.12.5 ST\_MapAlgebra (callback function version)

ST\_MapAlgebra (callback function version) — `ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltypename=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);`  
`ST_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltypename=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);`  
`ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixeltypename=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);`  
`ST_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltypename=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);`  
`ST_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixeltypename=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);`

#### Synopsis

raster **ST\_MapAlgebra**(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltypename=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);  
raster **ST\_MapAlgebra**(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltypename=FIRST, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);  
raster **ST\_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, text pixeltypename=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 pixeltypename=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 pixeltypename=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);

`ST_MapAlgebra`

`ST_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltypename=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);`

**rast, rast1, rast2, rastbandargset** `TEXT` (代数) `TEXT`

`rastbandargset` `TEXT` / `TEXT`. `TEXT` 1 `TEXT`.

**nband, nband1, nband2** `TEXT`. `nband` `TEXT`. `nband1` `TEXT` `rast1` `TEXT` `nband2` `TEXT` 2 `TEXT` 2 `TEXT` `rast2` `TEXT`.

**callbackfunc** The callbackfunc parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position integer[][], VARIADIC userargs text[])
RETURNS double precision
AS $$
BEGIN
 RETURN 0;
END;
$$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The callbackfunc must have three arguments: a 3-dimension double precision array, a 2-dimension integer array and 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 third argument userargs is for passing through any user-specified arguments.





```

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

```

PostgreSQL 9.0

```

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
)
WITH 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[],
 'raster_nmapalgebra_test(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_Metadatas(rast)),
 (ST_BandMetadatas(rast, 1)),
 ST_Value(rast, 1, 1, 1)
FROM bar;

```

☒☒: ☒☒ 2 ☒ 3

☒☒☒ 1 ☒, ☒☒☒☒

```

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

```

☒☒☒ 1 ☒, ☒☒ 1 ☒

```

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

```

☒☒: ☒☒ 4

☒☒☒ 2 ☒, ☒☒ 2 ☒

```

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, 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

```

☒☒: **mask** ☒☒☒☒

```

WITH foo AS (SELECT
 ST_SetBandNoDataValue(
 ST_SetValue(ST_SetValue(ST_AsRaster(
 ST_Buffer(
 ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 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)
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], ←
 int[], text[])::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' 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 'mask weighted only consider neighbors, exclude center multi otehr pixel values by ←
 2' 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;

```

---





4. SECOND -

**nodataexpr** rast1 NODATA rast2 rast2, rast2

**nodata2expr** rast2 NODATA rast1 rast1, rast1

**nodatanodataval** rast1 rast2 NODATA

• expression, nodataexpr nodata2expr

1. [rast1] - rast1
2. [rast1.val] - rast1
3. [rast1.x] - rast1 1-
4. [rast1.y] - rast1 1-
5. [rast2] - rast2
6. [rast2.val] - rast2
7. [rast2.x] - rast2 1-
8. [rast2.y] - rast2 1-

: 1 2

```
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;
```

: 3 4

```
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, 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;
```

[rastbandarg](#), [ST\\_Union](#), [ST\\_MapAlgebra \(callback function version\)](#)

### 11.12.7 ST\_MapAlgebraExpr

ST\_MapAlgebraExpr — `expression` 1 `band`: PostgreSQL `expression`, `band` 1 `band`. `band` 1 `band`.

#### 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)

`band`



#### Warning

**ST\_MapAlgebraExpr** 2.1.0 `expression` PostgreSQL `expression`, `band` 1 `band`. `band` 1 `band`.

`expression` (rast) `expression` PostgreSQL `expression`, `band` 1 `band`. `band` 1 `band`. `band` 1 `band`.

`pixeltype` `pixeltype`, `pixeltype` NULL `pixeltype`, `pixeltype` rast `pixeltype`.

`band` [rast], 1-`band` [rast.x], 1-`band` [rast.y] `band`.

2.0.0 `band`.

`band`

`band` 2 `band` (modulo) `band` 1 `band`.

```
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;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0

250		0
254		0
254		0

CREATE TABLE dummy\_rast (rid integer, rast raster, map\_rast2 raster);  
 INSERT INTO dummy\_rast (rid, rast, map\_rast2) VALUES (2, ST\_MakeEmptyRaster(256, 256, 0, 0, 256, 256, 2BUI, 1, 0));

```
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;
```

origval		mapval
249		1
250		1
251		1
252		2
253		3
254		3

```
SELECT
 ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;
```

b1pixtyp
2BUI







```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8 ←
 BUI',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;

--map them -
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
```

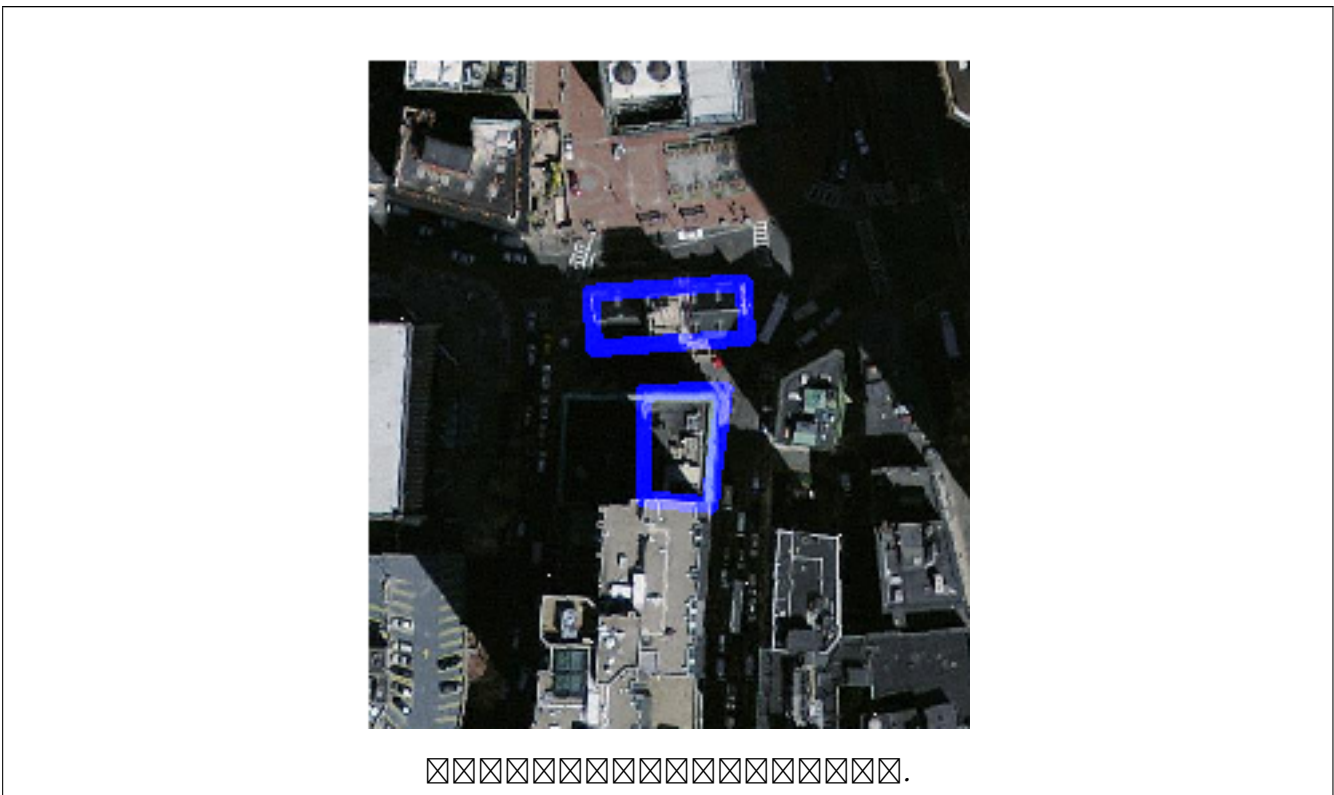




```

-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
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)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
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;

```



XX

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 — XXXX 1 XX: XXXXXXXXXXXXXXX PostgreSQL XXXXXXXXXXXXXXX, XXXXXXXXXXXXXXX, XX 1 XXXXXXXXXXXXXXX. XXXXXXXXXXXXXXX, XX 1 XXXX XX.

**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);
```

⊠



**Warning**

**ST\_MapAlgebraFct** 2.1.0 is deprecated. Use **ST\_MapAlgebra (callback function version)** instead.

**rast** (rast) ⊠ onerasteruserfunc ⊠ PostgreSQL ⊠, ⊠ 1 ⊠. **band** ⊠, ⊠ 1 ⊠. ⊠, ⊠, ⊠ 1 ⊠.

**pixeltype** ⊠, ⊠. **pixeltype** ⊠ NULL ⊠, ⊠ rast ⊠.

The onerasteruserfunc parameter must be the name and signature of a SQL or PL/pgSQL function, cast to a regprocedure. A very simple and quite useless PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT [])
 RETURNS FLOAT
 AS $$ BEGIN
 RETURN 0.0;
 END; $$
LANGUAGE 'plpgsql' IMMUTABLE;
```

The userfunction may accept two or three arguments: a float value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell (regardless of the raster datatype). The second argument is the position of the current processing cell in the form '{x,y}'. The third argument indicates that all remaining parameters to **ST\_MapAlgebraFct** shall be passed through to the userfunction.

Passing a regprodedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(float,integer[],text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

**userfunction** ⊠ variadic text ⊠. **ST\_MapAlgebraFct** ⊠ userfunction ⊠, args ⊠.



**Note**

(`ST_MapAlgebraFct`) VARIADIC `ST_MapAlgebraFct`, PostgreSQL `Query Language (SQL) Functions` "SQL Functions with Variable Numbers of Arguments".



**Note**

`ST_MapAlgebraFct`, `userfunction` `text[]`.

2.0.0

`ST_MapAlgebraFct` 2 `ST_MapAlgebraFct` (modulo) 1 `ST_MapAlgebraFct`.

```
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;
```

origval	mapval
253	1
254	0
253	1
253	1
254	0
254	0
250	0
254	0
254	0

`ST_MapAlgebraFct` NODATA `ST_MapAlgebraFct` (0) `ST_MapAlgebraFct` 2BUI, 1 `ST_MapAlgebraFct`.

```
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
```

```

 no_data := args[1];
 END IF;
 IF pixel < 251 THEN
 RETURN 1;
 ELSIF pixel = 252 THEN
 RETURN 2;
 ELSIF pixel
> 252 THEN
 RETURN 3;
 ELSE
 RETURN no_data;
 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;

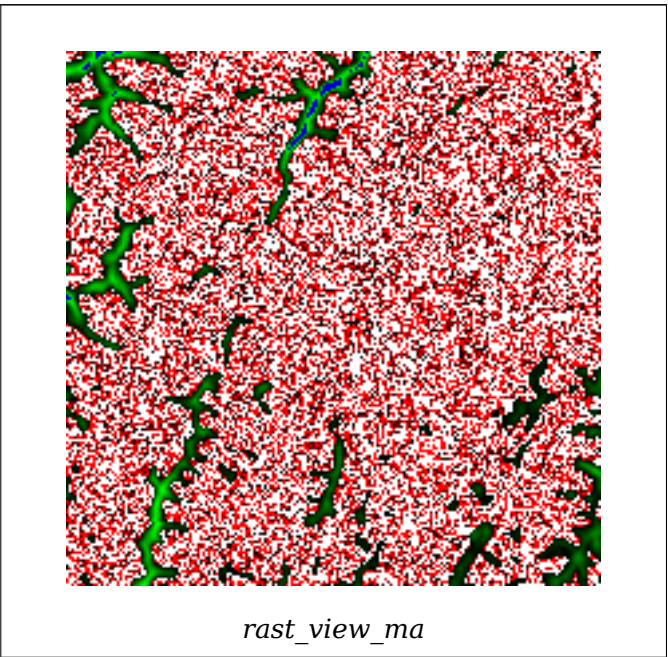
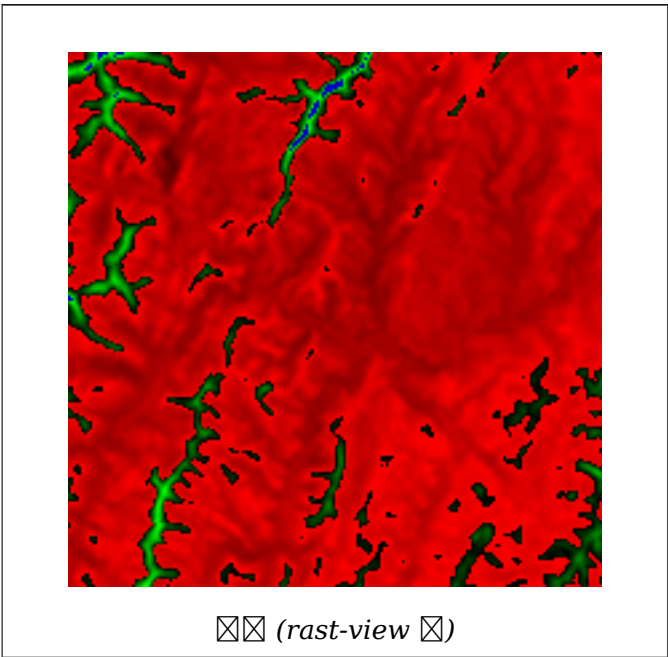
origval | mapval
-----+-----
 249 | 1
 250 | 1
 251 | 1
 252 | 2
 253 | 3
 254 | 3

SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

b1pixtyp

2BUI

```







rast1, rast2 two\_rasterfunc PostgreSQL, 1  
band1 band2, 1. raster  
raster, raster, 1.

pixeltype, pixeltype NULL  
raster1.

The two\_rasterfunc parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
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;
```

The two\_rasterfunc may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in rast1 (regardless of the raster datatype). The second argument is an individual raster cell value in rast2. The third argument is the position of the current processing cell in the form '{x,y}'. The fourth argument indicates that all remaining parameters to ST\_MapAlgebraFct shall be passed through to the two\_rasterfunc.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The fourth argument to the two\_rasterfunc is a variadic text array. All trailing text arguments to any ST\_MapAlgebraFct call are passed through to the specified two\_rasterfunc, and are contained in the userargs argument.

**Note!** (VARIADIC), PostgreSQL Query Language (SQL) Functions "SQL Functions with Variable Numbers of Arguments".

**Note!** two\_rasterfunc text[].

2.0.0.

:

```
-- define our user defined function --
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;

-- prep our test table of rasters
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 with ST_AddBand
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')

```









reclassarg

pixeltype reclassargset, reclassarg

2.0.0

2 8BUI 4BUI 101 254 NODATA

```
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;
```

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

1, 2, 3 1BB, 4BUI, 4BUI reclassarg

```
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;
```

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

**32BF**

32BF ((8BUI,8BUI,8BUI)

```
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)
]
);
```

[ST\\_AddBand](#), [ST\\_Band](#), [ST\\_BandPixelType](#), [ST\\_MakeEmptyRaster](#), [reclassarg](#), [ST\\_Value](#)

### 11.12.13 ST\_Union

ST\_Union — 1

#### 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);
```

1. uniontype LAST(), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE



**Note**

In order for rasters to be unioned, they must all have the same alignment. Use [ST\\_SameAlignment](#) and [ST\\_NotSameAlignmentReason](#) for more details and help. One way to fix alignment issues is to use [ST\\_Resample](#) and use the same reference raster for alignment.

2.0.0

: 2.1.0 (C).

2.1.0 ST\_Union(rast, unionarg)

: 2.1.0 ST\_Union(rast) 1 PostGIS

: 2.1.0 ST\_Union(rast, uniontype) 4

:

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01','dem02') GROUP BY filename;
```

:

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: 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[]
SELECT ST_Union(rast)
FROM aeriAls.boston
WHERE ST_Intersects(rast, ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ←
);
```

:

,

```
-- this creates a multi band raster collecting all the tiles that intersect a line
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) ←
);
```

unionarg, ST\_Envelope, ST\_ConvexHull, ST\_Clip, ST\_Union

## 11.13

### 11.13.1 ST\_Distinct4ma

ST\_Distinct4ma —

### Synopsis

float8 **ST\_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);  
double precision **ST\_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC user-args);



#### Note

1 **ST\_MapAlgebraFctNgb**



#### Note

2 **ST\_MapAlgebra (callback function version)**



#### Warning

2.1.0 **ST\_MapAlgebraFctNgb** 1

2.0.0

: 2.1.0 2

```

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;
 rid | st_value
-----+-----
 2 | 3
(1 row)

```

**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** —

Synopsis

double precision ST\_InvDistWeight4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

(Inverse Distance Weighted method)

userargs 2. (力率) (k). 1. ...

z-hat(x\_o) = (sum\_{j=1}^m z(x\_j) d\_ij^{-k}) / (sum\_{j=1}^m d\_ij^{-k})

k = (power factor), 0 <= 1



Note

ST\_MapAlgebra (callback function version)

2.1.0

-- NEEDS EXAMPLE

ST\_MapAlgebra (callback function version), ST\_MinDist4ma

11.13.3 ST\_Max4ma

ST\_Max4ma

Synopsis

float8 ST\_Max4ma(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision ST\_Max4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

¶

ST\_MapAlgebraFctNgb 函数。

¶ 2 参数, 参数 userargs 支持 NODATA 选项。



**Note**

¶ 1 `ST_MapAlgebraFctNgb` 函数。



**Note**

¶ 2 `ST_MapAlgebra (callback function version)` 函数。



**Warning**

2.1.0 版本 `ST_MapAlgebraFctNgb` 函数 1 参数。

2.0.0 版本。

更新: 2.1.0 版本 2 参数。

¶

```

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;
 rid | st_value
-----+-----
 2 | 254
(1 row)

```

¶

[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` — 函数。

















```

]::double precision[][]
) AS rast
)
SELECT
 ST_DumpValues(ST_Aspect(rast, 1, '32BF'))
FROM foo

```

```

(1,"{{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270
2227,180,161.565048217773,135}}")
(1 row)

```

**11.14.2**

PostgreSQL 9.1

```

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;

```

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_HillShade](#), [ST\\_Slope](#)

## 11.14.2 ST\_HillShade

`ST_HillShade` —



```
(1,"{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008, ←
NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ←
,147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}")
(1 row)
```

Example 2

Example 2. PostgreSQL 9.1

```
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_Hillshade(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;
```

Example

[ST\\_MapAlgebra \(callback function version\)](#), [ST\\_TRI](#), [ST\\_TPI](#), [ST\\_Roughness](#), [ST\\_Aspect](#), [ST\\_Slope](#)

### 11.14.3 ST\_Roughness

`ST_Roughness` — DEM (roughness)

#### Synopsis

raster `ST_Roughness`(raster rast, integer nband, raster customextent, text pixeltype="32BF", boolean interpolate\_nodata=FALSE );





**例 1**

```

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

(1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681285858154,26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438905681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)

```

**例 2**

例 2. PostgreSQL 9.1

```

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

```









ORDER BY val;

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))

[geomval](#), [ST\\_Value](#), [ST\\_Polygon](#), [ST\\_ValueCount](#)

### 11.15.4 ST\_Envelope

ST\_Envelope — [ST\\_Envelope](#).

#### Synopsis

geometry **ST\_Envelope**(raster rast);

[ST\\_Envelope](#) SRID [ST\\_Envelope](#). [ST\\_Envelope](#) float8 [ST\\_Envelope](#).

[ST\\_Envelope](#) ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)).

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;
```

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))

[ST\\_Envelope](#), [ST\\_AsText](#), [ST\\_SRID](#)

### 11.15.5 ST\_MinConvexHull

ST\_MinConvexHull — [ST\\_MinConvexHull](#) NODATA [ST\\_MinConvexHull](#).

**Synopsis**

geometry **ST\_MinConvexHull**(raster rast, integer nband=NULL);

NODATA . nband NULL, .

2.1.0 .

```

WITH foo AS (
 SELECT
 ST_SetValues(
 ST_SetValues(
 ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, '8 ←
 BUI', 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
 ST_AsText(ST_ConvexHull(rast)) AS hull,
 ST_AsText(ST_MinConvexHull(rast)) AS mhull,
 ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
 ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo

 hull | mhull | ←
 mhull_1 | mhull_2 |
-----+-----+-----
POLYGON((0 0,9 0,9 -9,0 -9,0 0)) | POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3)) | POLYGON((3 -3,9 ←
-3,9 -6,3 -6,3 -3)) | POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))

```



















¶

ST\_Covers(rastB, rastA) 返回 rastA 是否被 rastB 完全覆盖。如果 rastA 或 rastB 为 NULL，则返回 NULL。如果 rastA 或 rastB 包含 NODATA 值，则返回 NULL。



**Note**

ST\_Covers 与 ST\_Contains 类似，但 ST\_Covers 要求 rastA 完全被 rastB 覆盖。



**Note**

ST\_Covers(ST\_Polygon(raster), geometry) 与 ST\_Covers(geometry, ST\_Polygon(raster)) 返回相同的布尔值。

### 2.1.0 版本更改

¶

```
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;
```

rid	rid	st_covers
2	1	f
2	2	t

¶

[ST\\_Intersects](#), [ST\\_CoveredBy](#)

## 11.17.4 ST\_CoveredBy

ST\_CoveredBy — 返回 rastA 是否被 rastB 完全覆盖。

### Synopsis

```
boolean ST_CoveredBy(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean ST_CoveredBy(raster rastA , raster rastB);
```

¶

ST\_CoveredBy(rastA, rastB) 返回 rastA 是否被 rastB 完全覆盖。如果 rastA 或 rastB 为 NULL，则返回 NULL。如果 rastA 或 rastB 包含 NODATA 值，则返回 NULL。





**Note**

ST\_CoveredBy(geometry, ST\_Polygon(raster)) is deprecated. Use ST\_CoveredBy(ST\_Polygon(raster), geometry) instead.



**Note**

ST\_CoveredBy(ST\_Polygon(raster), geometry) is deprecated. Use ST\_CoveredBy(geometry, ST\_Polygon(raster)) instead.

2.1.0 ST\_CoveredBy(geometry, ST\_Polygon(raster)).

SQL

```
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;
```

rid	rid	st_coveredby
2	1	f
2	2	t

SQL

**ST\_Intersects, ST\_Covers**

**11.17.5 ST\_Disjoint**

ST\_Disjoint — Returns true if rasterA and rasterB do not intersect.

**Synopsis**

```
boolean ST_Disjoint(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean ST_Disjoint(raster rastA , raster rastB);
```

SQL

ST\_Disjoint(rastA, rastB) returns true if rasterA and rasterB do not intersect. If either raster is NULL, the result is NULL. If either raster has a NODATA value, the result is NULL.



**Note**

ST\_Disjoint(geometry, ST\_Polygon(raster)) is deprecated. Use ST\_Disjoint(ST\_Polygon(raster), geometry) instead.





**Note**

PostGIS 2.0.0 introduced ST\_Intersects(geometry, raster) and ST\_Intersects(raster, geometry).

PostGIS 2.0.0 introduced ST\_Intersects(geometry, raster) and ST\_Intersects(raster, geometry).



**Warning**

PostGIS 2.1.0 introduced ST\_Intersects(geometry, raster) and ST\_Intersects(raster, geometry).

SQL

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

st_intersects

t
```

SQL

**ST\_Intersection, ST\_Disjoint**

**11.17.7 ST\_Overlaps**

ST\_Overlaps — Returns true if rasterA overlaps rasterB.

**Synopsis**

boolean **ST\_Overlaps**( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean **ST\_Overlaps**( raster rastA , raster rastB );

SQL

ST\_Overlaps(rastA, nbandA, rastB, nbandB) returns true if rasterA overlaps rasterB. ST\_Overlaps(rastA, rastB) returns true if rasterA overlaps rasterB. ST\_Overlaps(rastA, rastB) returns NULL if either raster is NULL or either raster has a NODATA value.



**Note**

PostGIS 2.0.0 introduced ST\_Overlaps(rastA, nbandA, rastB, nbandB).



**Note**

ST\_Overlaps(ST\_Polygon(raster), geometry) ST\_Polygon

2.1.0

SQL

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

st_overlaps

f
```

SQL

**ST\_Intersects**

**11.17.8 ST\_Touches**

ST\_Touches — raster rastA raster rastB, TRUE

**Synopsis**

boolean ST\_Touches( raster rastA , integer nbandA , raster rastB , integer nbandB );
boolean ST\_Touches( raster rastA , raster rastB );

SQL

raster rastA raster rastB. raster rastA raster rastB NULL, (NODATA)



**Note**



**Note**

ST\_Touches(ST\_Polygon(raster), geometry) ST\_Polygon

2.1.0











### 11.17.13 ST\_DFullyWithin

ST\_DFullyWithin — raster rastA, raster rastB, integer nbandA, integer nbandB, double precision distance\_of\_srid.

#### 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 );

Notes

ST\_DFullyWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid ) returns true if any pixel in rastA is within distance\_of\_srid of any pixel in rastB. If either raster has NULL or NODATA values, the result is NULL.

ST\_DFullyWithin( raster rastA , raster rastB , double precision distance\_of\_srid ) returns true if any pixel in rastA is within distance\_of\_srid of any pixel in rastB. If either raster has NULL or NODATA values, the result is NULL.



#### Note

distance\_of\_srid (operand) must be non-negative.



#### Note

ST\_DFullyWithin(ST\_Polygon(raster), geometry) is equivalent to ST\_DFullyWithin(ST\_Polygon(raster), ST\_Polygon(geometry)).

#### 2.1.0 Examples

Notes

```
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;
```

rid	rid	st_dfullywithin
2	1	f
2	2	t

Notes

[ST\\_Within](#), [ST\\_DWithin](#)

## 11.18 Raster Tips

### 11.18.1 Out-DB Rasters

#### 11.18.1.1 Directory containing many files

When GDAL opens a file, GDAL eagerly scans the directory of that file to build a catalog of other files. If this directory contains many files (e.g. thousands, millions), opening that file becomes extremely slow (especially if that file happens to be on a network drive such as NFS).

To control this behavior, GDAL provides the following environment variable: `GDAL_DISABLE_READDIR_ON_OPEN`. Set `GDAL_DISABLE_READDIR_ON_OPEN` to `TRUE` to disable directory scanning.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), `GDAL_DISABLE_READDIR_ON_OPEN` can be set in `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment` (where `POSTGRESQL_VERSION` is the version of PostgreSQL, e.g. 9.6 and `CLUSTER_NAME` is the name of the cluster, e.g. maindb). You can also set PostGIS environment variables here as well.

```
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 Maximum Number of Open Files

The maximum number of open files permitted by Linux and PostgreSQL are typically conservative (typically 1024 open files per process) given the assumption that the system is consumed by human users. For Out-DB Rasters, a single valid query can easily exceed this limit (e.g. a dataset of 10 year's worth of rasters with one raster for each day containing minimum and maximum temperatures and we want to know the absolute min and max value for a pixel in that dataset).

The easiest change to make is the following PostgreSQL setting: `max_files_per_process`. The default is set to 1000, which is far too low for Out-DB Rasters. A safe starting value could be 65536 but this really depends on your datasets and the queries run against those datasets. This setting can only be made on server start and probably only in the PostgreSQL configuration file (e.g. `/etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf` in Ubuntu environments).

```
...
- Kernel Resource Usage -

max_files_per_process = 65536 # min 25
 # (change requires restart)
...
```

The major change to make is the Linux kernel's open files limits. There are two parts to this:

- Maximum number of open files for the entire system
- Maximum number of open files per process

### 11.18.1.2.1 Maximum number of open files for the entire system

You can inspect the current maximum number of open files for the entire system with the following example:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

If the value returned is not large enough, add a file to `/etc/sysctl.d/` as per the following example:

```
$ 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 Maximum number of open files per process

We need to increase the maximum number of open files per process for the PostgreSQL server processes.

To see what the current PostgreSQL service processes are using for maximum number of open files, do as per the following example (make sure to have PostgreSQL running):

```
$ ps aux | grep postgres
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
```

```
$ cat /proc/31718/limits
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

In the example above, we inspected the open files limit for Process 31718. It doesn't matter which PostgreSQL process, any of them will do. The response we are interested in is *Max open files*.

We want to increase *Soft Limit* and *Hard Limit* of *Max open files* to be greater than the value we specified for the PostgreSQL setting `max_files_per_process`. In our example, we set `max_files_per_process` to 65536.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), the easiest way to change the *Soft Limit* and *Hard Limit* is to edit `/etc/init.d/postgresql` (SysV) or `/lib/systemd/system/postgresql*.service` (systemd).

Let's first address the SysV Ubuntu case where we add `ulimit -H -n 262144` and `ulimit -n 131072` to `/etc/init.d/postgresql`.

```
...
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)
 ...
```

Now to address the systemd Ubuntu case. We will add `LimitNOFILE=131072` to every `/lib/systemd/system/postgresql*.service` file in the `[Service]` section.

```
...
[Service]

LimitNOFILE=131072

...

[Install]
WantedBy=multi-user.target
...
```

After making the necessary systemd changes, make sure to reload the daemon

```
systemctl daemon-reload
```

# Chapter 12

## PostGIS Extras

This chapter documents features found in the extras folder of the PostGIS source tarballs and source repository. These are not always packaged with PostGIS binary releases, but are usually PL/pgSQL based or standard shell scripts that can be run as is.

### 12.1

**PAGC standardizer** (fork) (PAGC PostgreSQL)

(lexicon; lex) (gazetteer; gaz)

```
CREATE EXTENSION address_standardizer;
address_standardizer PostgreSQL
address_standardizer_data_us
CREATE EXTENSION address_standardizer_data_us;
```

PostGIS extensions/address\_standardizer

Section 2.3

#### 12.1.1

(macro) (micro)

/

(zip code) (Perl) parseaddress-api.c

(Perl) parseaddress-api.c

### 12.1.2

#### 12.1.2.1 stdaddr

stdaddr — standardize\_address

standardize\_address PAGC Postal Attributes

rules table

This method needs address\_standardizer extension.

building (0) : 123 Main St

house\_num (1) : 75 State Street 75

predir (2) : North, South, East, West (STREET NAME PRE-DIRECTIONAL)

qual (3) : 3715 OLD HIGHWAY 99 OLD (STREET NAME PRE-MODIFIER)

pretype (4) : (STREET PREFIX TYPE)

name (5) : (STREET NAME)

suftype (6) : St, Ave, Cir (STREET POST TYPE) STREET

sufdir (7) : 3715 TENTH AVENUE WEST WEST (STREET POST-DIRECTIONAL)

ruralroute is text (token number 8): RURAL ROUTE . Example 7 in RR 7.

extra :

city (10) :

state (11) :

country (12) : USA

postcode (13) : 02109 (postal code, zip code)

box (14, 15) : 02109 (POSTAL BOX NUMBER)

unit (17) : APT 3B 3B

### 12.1.3

#### 12.1.3.1 rules table

rules table — -1(terminator), -1, .



**UNITH** (16). *APT UNIT*

**QUINT** (28). (Zip Code)

**QUAD** (29). ZIP4

**PCH** (27). , , 3 . FSA

**PCT** (26). , , 3 . LDU

**(不用語; stopword)**

**STOPWORD** WORD WORD STOPWORD WORD WORD

**STOPWORD** (7). *THE*

-1( ) , -1 . *stdaddr* the section called " " .

( ) 0 ( ) 17 .

**MACRO\_C**

( = "0"). *PLACE STATE ZIP* MACRO

**MACRO\_C output tokens** (excerpted from <http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--> .

**CITY** ( "10"). : "Albany"

**STATE** ( "11"). : "NY"

**NATION** ( "12"). : "USA"

**POSTAL** ( "13"). (SADS "ZIP CODE", "PLUS 4").

**MICRO\_C**

( = "1"). ( , , *sufdir*, *predir*, *pretyp*, *suftype*, *qualif* ) MICRO

**MICRO\_C output tokens** (excerpted from <http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--> .

**HOUSE** ( ) 1 : 75 State Street 75

**predir** ( ) 2 : North, South, East, West (STREET NAME PRE-DIRECTIONAL)

**qual** ( ) 3 : (STREET NAME PRE-MODIFIER) . : 3715 OLD HIGHWAY 99 *OLD*



**pretype** (4): (STREET PREFIX TYPE)

**street** (5): (STREET NAME)

**suftype** (6): St, Ave, Cir (STREET POST TYPE).  
 Example: 75 State Street STREET

**sufdir** (7): (STREET POST-DIRECTIONAL).  
 Example: 3715 TENTH AVENUE WEST WEST

**ARC\_C**

(= "2"). HOUSE MICRO HOUSE MICRO\_C

**CIVIC\_C**

(= "3"). HOUSE

**EXTRA\_C**

(= "4"). EXTRA - -

**EXTRA\_C output tokens** (excerpted from <http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ-->).

**BLDNG** (0):

**BOXH** (token number 14): The **BOX** in BOX 3B

**BOXT** (15): BOX 3B **3B**

**RR** (8): RR 7 **RR**

**UNITH** (16): APT 3B **APT**

**UNITT** (17): APT 3B **3B**

**UNKNWN** (9):

**12.1.3.2 lex table**

lex table — (lex) (1) (the section called " ") (2)

(lexicon) (1) (the section called " ") (2) ONE stdword 1

**id**

**seq**:

**word**:

**stdword**:

**token**: PAGC Tokens

### 12.1.3.3 gaz table

`gaz table` — (gaz) table, (1) (the section called “”) (2) .

A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called “”) and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

. .

**id** :

**seq** : -

**word** :

**stdword** :

**token** : . . **PAGC Tokens** .

### 12.1.4

#### 12.1.4.1 debug\_standardize\_address

`debug_standardize_address` — Returns a json formatted text listing the parse tokens and standardizations

#### Synopsis

`text debug_standardize_address(text lextab, text gaztab, text rultab, text micro, text macro=NULL);`

This is a function for debugging address standardizer rules and lex/gaz mappings. It returns a json formatted text that includes the matching rules, mapping of tokens, and best standardized address **stdaddr** form of an input address utilizing **lex table** table name, **gaz table**, and **rules table** table names and an address.

For single line addresses use just `micro`

For two line address A `micro` consisting of standard first line of postal address e.g. `house_num street`, and a `macro` consisting of standard postal second line of an address e.g. `city, state postal_code country`.

Elements returned in the json document are

**input\_tokens** For each word in the input address, returns the position of the word, token categorization of the word, and the standard word it is mapped to. Note that for some input words, you might get back multiple records because some inputs can be categorized as more than one thing.


**rules** The set of rules matching the input and the corresponding score for each. The first rule (highest scoring) is what is used for standardization

**stdaddr** The standardized address elements **stdaddr** that would be returned when running **standardize\_address**

Availability: 3.4.0

 This method needs address\_standardizer extension.



address\_standardizer\_data\_us 

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address and returning the input tokens

```
SELECT it->'pos' AS position, it->'word' AS word, it->'stdword' AS standardized_word,
 it->'token' AS token, it->'token-code' AS token_code
FROM jsonb(
 debug_standardize_address('us_lex',
 'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

position	word	standardized_word	token	token_code
0	ONE	1	NUMBER	0
0	ONE	1	WORD	1
1	DEVONSHIRE	DEVONSHIRE	WORD	1
2	PLACE	PLACE	TYPE	2
3	PH	PATH	TYPE	2
3	PH	PENTHOUSE	UNITT	17
4	301	301	NUMBER	0

(7 rows)

Variant 2: Multi line address and returning first rule input mappings and score

```
SELECT (s->'rules'->0->'score')::numeric AS score, it->'pos' AS position,
 it->'input-word' AS word, it->'input-token' AS input_token, it->'mapped-word' AS ↵
 standardized_word,
 it->'output-token' AS output_token
FROM jsonb(
 debug_standardize_address('us_lex',
 'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

score	position	word	input_token	standardized_word	output_token
0.876250	0	ONE	NUMBER	1	HOUSE
0.876250	1	DEVONSHIRE	WORD	DEVONSHIRE	STREET
0.876250	2	PLACE	TYPE	PLACE	SUFTYP
0.876250	3	PH	UNITT	PENTHOUSE	UNITT
0.876250	4	301	NUMBER	301	UNITT

(5 rows)



[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pagc\\_Normalize\\_Address](#)

### 12.1.4.2 parse\_address

parse\_address —

#### Synopsis

record parse\_address(text address);

Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

2.2.0

 This method needs address\_standardizer extension.

```
SELECT num, street, city, zip, zipplus
 FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

num	street	city	zip	zipplus
1	Devonshire Place	Boston	02109	1234

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

-- parse the addresses
-- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
 FROM places) AS p;
```

addid	num	street	city	state	zip	zipplus
1	529	Main Street	Boston	MA	02129	
2	77	Massachusetts Avenue	Cambridge	MA	02139	
3	25	Wizard of Oz	Walaford	KS	99912	323
4	26	Capen Street	Medford	MA		
5	124	Mount Auburn St	Cambridge	MA	02138	
6	950	Main Street	Worcester	MA	01610	

(6 rows)



key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

2: .

```
SELECT (each(hstore(p))).*
FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
 'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

key	value
box	
city	BOSTON
name	DEVONSHIRE
qual	
unit	# PENTHOUSE 301
extra	
state	MA
predir	
sufdir	
country	USA
pretype	
suftype	PL
building	
postcode	02109
house_num	1
ruralroute	

(16 rows)

[stdaddr](#), [rules table](#), [lex table](#), [gaz table](#), [Pagc\\_Normalize\\_Address](#)

## 12.2 TIGER

TIGER, PostGIS

- **Nominatim** OpenStreetMap. osm2pgsql, PostgreSQL 8.4, PostGIS 1.5. TIGER, Nominatim TIGER SQL.
- **GIS Graphy** PostGIS Nominatim OSM(OpenStreetMap). OSM, Nominatim, Java 1.5, Servlet apps, Solr. GIS Graphy.

### 12.2.1 Drop\_Indexes\_Generate\_Script

Drop Indexes Generate Script — TIGER tiger\_data

#### Synopsis

text Drop\_Indexes\_Generate\_Script(text param\_schema=tiger\_data);

TIGER tiger\_data. (bloat) Install Missing Indexes 2.0.0

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql

DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
```





### Synopsis

text **Drop\_State\_Tables\_Generate\_Script**(text param\_state, text param\_schema=tiger\_data);

Examples

Drop state tables for Pennsylvania (州) tiger\_data schema. (州) tiger\_data schema.

2.0.0 Examples.

Examples

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

Examples

### Loader\_Generate\_Script

## 12.2.4 Geocode

Geocode — (Geocode) NAD83 coordinates, return results. restrict\_region(NULL) restrict\_region(NULL).

### Synopsis

setof record geocode(varchar address, integer max\_results=10, geometry restrict\_region=NULL, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);
setof record geocode(norm\_addy in\_addy, integer max\_results=10, geometry restrict\_region=NULL, norm\_addy OUT addy, geometry OUT geomout, integer OUT rating);

Geocoding

Geocoding (geocoding) is the process of converting a human-readable address into a geographic coordinate system (NAD83). The normalized\_address (addy) function is used for this purpose. It takes an address string and returns a geometry object. The function uses the TIGER (edge, face, addr), PostgreSQL (soundex, levenshtein), and PostGIS (TIGER) data to perform the geocoding. The function is available in PostGIS 2.0.0 and later.

Example: 2.0.0 uses TIGER 2010 data. The function is available in PostGIS 2.0.0 and later. The max\_results parameter is used to limit the number of results returned.

Example

Example (MA), Example (MN), Example (CA), Example (RI) uses TIGER data and PostgreSQL 9.1rc1/PostGIS 2.0. The function is available in PostGIS 2.0 and later.

Example (61 results).

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,(←
 addy).zip
FROM geocode('75 State Street, Boston MA 02109', 1) As g;
```

rating	lon	lat	stno	street	styp	city	st	zip
0	-71.0557505845646	42.35897920691	75	State	St	Boston	MA	02109

Example (122 ~ 150 results).

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,(←
 addy).zip
FROM geocode('226 Hanover Street, Boston, MA',1) As g;
```

rating	wktlonlat	stno	street	styp	city	st	zip
1	POINT(-71.05528 42.36316)	226	Hanover	St	Boston	MA	02113

Example (500 results).

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,(←
 addy).zip
FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
```

rating	wktlonlat	stno	street	styp	city	st	zip
70	POINT(-71.06466 42.35114)	31	Stuart	St	Boston	MA	02116

Example (batch) max\_results = 1. The function is available in PostGIS 2.0 and later.

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
 lon numeric, lat numeric, new_address text, rating integer);

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129');
```

```

('77 Massachusetts Avenue, Cambridge, MA 02139'),
('25 Wizard of Oz, Waford, KS 99912323'),
('26 Capen Street, Medford, MA'),
('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms - there are caching and shared memory ←
 effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
 SET (rating, new_address, lon, lat)
 = (COALESCE(g.rating, -1), pprint_addy(g.addy),
 ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5))
FROM (SELECT addid, address
 FROM addresses_to_geocode
 WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
 LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;

```

result  
 -----  
 Query returned successfully: 3 rows affected, 480 ms execution time.

```
SELECT * FROM addresses_to_geocode WHERE rating is not null;
```

addid	new_address	rating	lon	lat	
1	529 Main Street, Boston MA, 02129 Boston, MA 02129	0	-71.07177	42.38357	529 Main St, ←
2	77 Massachusetts Avenue, Cambridge, MA 02139 Massachusetts Ave, Cambridge, MA 02139	0	-71.09396	42.35961	77 ←
3	25 Wizard of Oz, Waford, KS 99912323 KS 67502	108	-97.92913	38.12717	Willowbrook, ←

(3 rows)

☒☒: ☒☒☒☒☒☒

```

SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
 (addy).address As stno, (addy).streetname As street,
 (addy).streettypeabbrev As styp,
 (addy).location As city, (addy).stateabbrev As st,(addy).zip
FROM geocode('100 Federal Street, MA',
 3,
 (SELECT ST_Union(the_geom)
 FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
) As g;

```

rating	wktlonlat	stno	street	styp	city	st	zip
7	POINT(-70.96796 42.4659)	100	Federal	St	Lynn	MA	01905
16	POINT(-70.96786 42.46853)	NULL	Federal	St	Lynn	MA	01905

(2 rows)

Time: 622.939 ms





☐☐

[Set\\_Geocode\\_Setting](#)

### 12.2.7 Get\_Tract

Get Tract — `get_tract(geometry loc_geom, text output_field=name)` (tract) `get_tract(geometry loc_geom, text output_field=name)` (field) `get_tract(geometry loc_geom, text output_field=name)`. `get_tract(geometry loc_geom, text output_field=name)`.

#### Synopsis

```
text get_tract(geometry loc_geom, text output_field=name);
```

☐☐

`get_tract(geometry loc_geom, text output_field=name)`. `get_tract(geometry loc_geom, text output_field=name)` NAD83 `get_tract(geometry loc_geom, text output_field=name)`.

**Note**

This function uses the census tract which is not loaded by default. If you have already loaded your state table, you can load tract as well as bg, and tabblock using the [Loader\\_Generate\\_Census\\_Script](#) script.



If you have not loaded your state data yet and want these additional tables loaded, do the following

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name IN('tract', 'bg', 'tabblock');
```

then they will be included by the [Loader\\_Generate\\_Script](#).

2.0.0 `get_tract(geometry loc_geom, text output_field=name)`.

☐☐☐☐

```
SELECT get_tract(ST_Point(-71.101375, 42.31376)) As tract_name;
tract_name

1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id') As tract_id;
tract_id

25025120301
```

☐☐

[Geocode](#)>

### 12.2.8 Install\_Missing\_Indexes

Install\_Missing\_Indexes — 安装缺失索引 (join) 键索引的函数。该函数接受一个表名、一个键名、一个索引名、一个索引类型、一个索引选项、一个索引表空间和索引表名。该函数返回一个布尔值，表示是否成功安装了索引。

#### Synopsis

boolean **Install\_Missing\_Indexes**(

表名

键名 tiger 表名 tiger\_data 索引名 索引类型 索引选项 Missing\_Indexes\_Generate\_Script 索引表空间 索引表名 update\_geocode.sql 2.0.0

表名

```
SELECT install_missing_indexes(
 install_missing_indexes

t
```

表名

Loader\_Generate\_Script, Missing\_Indexes\_Generate\_Script

### 12.2.9 Loader\_Generate\_Census\_Script

Loader\_Generate\_Census\_Script — 生成 Census 脚本的函数。该函数接受一个州名、一个 Tiger 表名、一个 tract 表名、一个 bg 表名、一个 tabblock 表名、一个 tiger\_data 表名、一个 staging 表名、一个 temp 表名、一个 OS 表名。

#### Synopsis

setof text **loader\_generate\_census\_script**(text[] param\_states, text os);

表名

州名 Tiger 表名 (州) tract, bg, tabblocks tiger\_data 表名 (州) staging temp OS

1. loader\_variables - 设置, 选项, 选项 (staging) 选项
2. loader\_platform - 选项. 选项. 选项.
3. loader\_lookuptables - 选项 (州, 县), 选项, 选项, 选项, 选项. 选项, 选项, 选项, 选项, 选项. 选项 (州名) 选项, TIGER 选项, 选项. 选项: tiger.faces 选项 tiger\_data.ma\_faces 选项.

2.0.0 选项.



**Note**

**Loader Generate Script** 选项, PostGIS 2.0.0 alpha5 选项 TIGER 选项, 选项 (州) 选项.

选项

选项.

```

SELECT loader_generate_census_script(ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- ←
 relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%*. * /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id)) ←
 INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom -W "latin1" tl_2010_25_tract10.dbf tiger_staging. ←
 ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id; SELECT ←
 loader_load_staged_data(lower('MA_tract10'), lower('MA_tract'));"
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ←
 (the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp = ←
 '25');"
:

```













3. streetName varchar.
4. streetTypeAbbrev varchar, St, Ave, Cir. street\_type\_lookup.
5. postdirAbbrev varchar, N, S, E, W. direction\_lookup.
6. internal varchar.
7. location varchar, .
8. stateAbbrev varchar, MA, NY, MI (州名). state\_lookup.
9. zip varchar. 02109.
10. parsed (boolean). normalize\_address.
11. zip4 last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. address\_alphanumeric Full street number even if it has alpha characters like 17R. Parsing of this is better using [Pgcn Normalize Address](#) function. Availability: PostGIS 2.4.0.

. [Pprint\\_Addy](#).

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
FROM (SELECT address, normalize_address(address) As na
 FROM addresses_to_geocode) As g;
```

orig	streetname	streettypeabbrev
28 Capen Street, Medford, MA	Capen	St
124 Mount Auburn St, Cambridge, Massachusetts 02138	Mount Auburn	St
950 Main Street, Worcester, MA 01610	Main	St
529 Main Street, Boston MA, 02129	Main	St
77 Massachusetts Avenue, Cambridge, MA 02139	Massachusetts	Ave
25 Wizard of Oz, Walaford, KS 99912323	Wizard of Oz	

[Geocode, Pprint\\_Addy](#)

### 12.2.14 Pgcn Normalize Address

**Pgcn Normalize Address** — , , , , norm\_addy , tiger\_geocoder (TIGER ) address\_standardizer .

#### Synopsis

```
norm_addy pgcn_normalize_address(varchar in_address);
```









include strnum range .

TIGER , , 26 Court Sq. 26 Court St. 26 Court Sq. .

: TIGER . NULL

:

- 1. intpt : .
2. addy norm\_addy(addy) : , .
3. street varchar : (1 ) ( )

Enhanced: 2.4.1 if optional zcta5 dataset is loaded, the reverse\_geocode function can resolve to state and zip even if the specific state data is not loaded. Refer to Loader\_Generate\_Nation\_Script for details on loading zcta5 data.

2.0.0

MIT - 77 Massachusetts Ave, Cambridge, MA 02139 - 3 , PostgreSQL (上限; upper bound) NULL

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_streets FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ;
```

result table with columns st1, st2, st3, cross\_streets. Row 1: 67 Massachusetts Ave, Cambridge, MA 02139 | | | 67 - 127 Massachusetts Ave,32 - 88 Vassar St

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;
```

result

st1	st2	st3	cross_str
5 Bradford St, Boston, MA 02118	49 Waltham St, Boston, MA 02118		Waltham St

**Geocode** 2

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
 (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
 reverse_geocode(ST_SetSRID(ST_Point(lon,lat),4326)) As rg
 FROM addresses_to_geocode WHERE rating
 > -1) As foo;
```

actual_addr	int_addr1	lon	lat	cross1	cross2
529 Main Street, Boston MA, 02129 Boston, MA 02129	Medford St	-71.07181	42.38359	527 Main St,	
77 Massachusetts Avenue, Cambridge, MA 02139 Massachusetts Ave, Cambridge, MA 02139	Vassar St	-71.09428	42.35988	77	
26 Capen Street, Medford, MA Medford, MA 02155	Capen St Tesla Ave	-71.12377	42.41101	9 Edison Ave,	
124 Mount Auburn St, Cambridge, Massachusetts 02138 Rd, Cambridge, MA 02138	Mount Auburn St	-71.12304	42.37328	3 University	
950 Main Street, Worcester, MA 01610 Worcester, MA 01603	Main St	-71.82368	42.24956	3 Maywood St,	Maywood Pl

**Pprint\_Addy, Pprint\_Addy, ST\_AsText**

### 12.2.17 Topology\_Load\_Tiger

Topology\_Load\_Tiger — PostGIS TIGER

#### Synopsis

```
text Topology_Load_Tiger(varchar topo_name, varchar region_type, varchar region_id);
```

PostGIS TIGER. , , TIGER , ID , TIGER .

, , ,



**Note**

TIGER PostGIS Chapter 9 Section 2.2.3. ,



**Note**

:

- 1. topo\_name - PostGIS
2. region\_type - place county
3. region\_id - TIGER ID(geoid) place tiger.place tiger.county

2.0.0

:

(2249) 0.25 TIGER , ,

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology

15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added. 20576 nodes added.
19962 nodes contained in a face. 0 edge start end corrected. 31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
-- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
topology.ValidateTopology('topo_boston');

error | id1 | id2
-----+-----+-----
```

Example:

Topology (26986) with precision 0.25, loaded from TIGER, county, 25025.

```

SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
24761 nodes added. 24075 nodes contained in a face. 0 edge start end corrected. 38175 ←
edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
-- topologysummary--
Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
 topology.ValidateTopology('topo_suffolk');

```

error	id1	id2
coincident nodes	81045651	81064553
edge crosses node	81045651	85737793
edge crosses node	81045651	85742215
edge crosses node	81045651	620628939
edge crosses node	81064553	85697815
edge crosses node	81064553	85728168
edge crosses node	81064553	85733413

Example

[CreateTopology](#), [CreateTopoGeom](#), [TopologySummary](#), [ValidateTopology](#)

### 12.2.18 Set\_Geocode\_Setting

Set\_Geocode\_Setting — Set geocode settings for a topology.

#### Synopsis

text **Set\_Geocode\_Setting**(text setting\_name, text setting\_value);

Example

```

tiger.geocode_settings topology_suffolk.
tiger.geocode_settings topology_suffolk. Get_Geocode_Setting topology_suffolk.

```

2.1.0 tiger.geocode\_settings.

繁體中文: 繁體中文

繁體中文 **Geocode** 繁體中文, NOTICE 繁體中文。

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result

true
```

繁體中文

[Get\\_Geocode\\_Setting](#)

## Chapter 13

# PostGIS Special Functions Index

### 13.1 PostGIS Aggregate Functions

The functions below are spatial aggregate functions that are used in the same way as SQL aggregate function such as sum and average.

- **CG\_3DUnion** - Perform 3D union.
  - **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
  - **ST\_3DUnion** - Perform 3D union.
  - **ST\_AsFlatGeobuf** - Return a FlatGeobuf representation of a set of rows.
  - **ST\_AsGeobuf** - Return a Geobuf representation of a set of rows.
  - **ST\_AsMVT** - Aggregate function returning a MVT representation of a set of rows.
  - **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
  - **ST\_ClusterIntersecting** - Aggregate function that clusters input geometries into connected sets.
  - **ST\_ClusterIntersectingWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
  - **ST\_ClusterKMeans** - Window function that returns a cluster id for each input geometry using the K-means algorithm.
  - **ST\_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
  - **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
  - **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
  - **ST\_CoverageInvalidEdges** - Window function that finds locations where polygons fail to form a valid coverage.
  - **ST\_CoverageSimplify** - Window function that simplifies the edges of a polygonal coverage.
  - **ST\_CoverageUnion** - Computes the union of a set of polygons forming a coverage by removing shared edges.
  - **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
-

- **ST\_MakeLine** - Returns a line from a set of points.
- **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way
- **ST\_Polygonize** - Computes a collection of polygons formed from the linework of a set of geometries.
- **ST\_SameAlignment** - Returns true if two lines are parallel, false otherwise. (Returns true if two lines are parallel, false otherwise.)
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST\_Union** - Returns the union of two geometries.
- **TopoElementArray\_Agg** - Returns a topoelementarray for a set of element\_id, type arrays (topoelements).

### 13.2 PostGIS Window Functions

The functions below are spatial window functions that are used in the same way as SQL window functions such as row\_number(), lead(), and lag(). They must be followed by an OVER() clause.

- **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterIntersectingWin** - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- **ST\_ClusterKMeans** - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST\_CoverageInvalidEdges** - Window function that finds locations where polygons fail to form a valid coverage.
- **ST\_CoverageSimplify** - Window function that simplifies the edges of a polygonal coverage.

### 13.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- **CG\_3DArea** - 3D area of a geometry.
- **CG\_3DDifference** - 3D difference of two geometries.
- **CG\_3DIntersection** - 3D intersection of two geometries.
- **CG\_3DUnion** - Perform 3D union.
- **CG\_Volume** - 3D volume of a geometry.
- **ST\_3DArea** - 3D area of a geometry.
- **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
- **ST\_3DDifference** - 3D difference of two geometries.



- **ST\_3DDistance** - Returns the 3D distance between two geometries in a 3D SRS (SRID 31466).
- **ST\_3DIntersection** - Returns the 3D intersection of two geometries.
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_3DLength** - Returns the 3D length of a geometry.
- **ST\_3DPerimeter** - Returns the 3D perimeter of a geometry.
- **ST\_3DUnion** - Perform 3D union.
- **ST\_AddEdgeModFace** - Returns the geometry with the specified edge added to the specified face of the geometry.
- **ST\_AddEdgeNewFaces** - Returns the geometry with the specified edge added to the specified faces of the geometry.
- **ST\_AddIsoEdge** - Returns the geometry with the specified edge added to the specified node of the geometry.
- **ST\_AddIsoNode** - Returns the geometry with the specified node added to the specified edge of the geometry.
- **ST\_Area** - Returns the area of a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsGML** - Returns the GML 2 or GML 3 representation of the geometry.
- **ST\_AsText** - Returns the WKT(Well-Known Text) representation of the geometry/geography with SRID meta data.
- **ST\_Boundary** - Returns the boundary of a geometry.
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Centroid** - Returns the centroid of a geometry.
- **ST\_ChangeEdgeGeom** - Returns the geometry with the specified edge replaced by the specified geometry.
- **ST\_Contains** - Tests if every point of B lies in A, and their interiors have a point in common
- **ST\_ConvexHull** - Computes the convex hull of a geometry.
- **ST\_CoordDim** - Returns the dimension of a geometry.
- **ST\_CreateTopoGeo** - Returns the TopoGeo representation of a geometry.
- **ST\_Crosses** - Tests if two geometries have some, but not all, interior points in common
- **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_Dimension** - Returns the dimension of a geometry.
- **ST\_Disjoint** - Tests if two geometries have no points in common
- **ST\_Distance** - Returns the 3D distance between two geometries (longest).

- **ST\_EndPoint** - ST\_LineString | ST\_CircularString
- **ST\_Envelope** - (double precision; float8)
- **ST\_Equals** - Tests if two geometries include the same set of points
- **ST\_ExteriorRing**
- **ST\_GMLToSQL** - GML | ST\_Geometry | ST\_GeomFromGML
- **ST\_GeomCollFromText** - Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_GeomFromText** - WKT | ST\_Geometry
- **ST\_GeomFromWKB** - WKB(Well-Known Binary) | SRID
- **ST\_GeometryFromText** - WKT(Well-Known Text) | ST\_Geometry | ST\_GeomFromText
- **ST\_GeometryN** - ST\_Geometry
- **ST\_GeometryType** - ST\_Geometry
- **ST\_GetFaceEdges** - aface
- **ST\_GetFaceGeometry** - ID
- **ST\_InitTopoGeo** - Creates a new topology schema and registers it in the topology.topology table.
- **ST\_InteriorRingN**
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common)
- **ST\_IsClosed** - LINESTRING | TRUE
- **ST\_IsEmpty** - Tests if a geometry is empty.
- **ST\_IsRing** - Tests if a LineString is closed and simple.
- **ST\_IsSimple** - TRUE
- **ST\_IsValid** - Tests if a geometry is well-formed in 2D.
- **ST\_Length**
- **ST\_LineFromText** - SRID | WKT | SRID | 0
- **ST\_LineFromWKB** - SRID | WKB | LINESTRING
- **ST\_LinestringFromWKB** - SRID | WKB
- **ST\_LocateAlong** - Returns the point(s) on a geometry that match a measure value.
- **ST\_LocateBetween** - Returns the portions of a geometry that match a measure range.
- **ST\_M** - Returns the M coordinate of a Point.
- **ST\_MLineFromText** - WKT | ST\_MultiLineString

- **ST\_MPointFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_MPolyFromText** - Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_ModEdgeHeal** - Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.
- **ST\_ModEdgeSplit** - `ST_ModEdgeSplit(geom, edge_id, new_geom)`, `ST_ModEdgeSplit(geom, edge_id, new_geom, id)`.
- **ST\_MoveIsoNode** - Moves an isolated node in a topology from one point to another. If new a point geometry exists as a node an error is thrown. Returns description of move.
- **ST\_NewEdgeHeal** - Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.
- **ST\_NewEdgesSplit** - `ST_NewEdgesSplit(geom, edge_id, new_geom, 2, new_geom)`, `ST_NewEdgesSplit(geom, edge_id, new_geom, id)`.
- **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST\_NumGeometries** - `ST_NumGeometries(geom)`. `ST_NumGeometries(geom, id)`.
- **ST\_NumInteriorRings** - `ST_NumInteriorRings(geom)`.
- **ST\_NumPatches** - `ST_NumPatches(geom)`. `ST_NumPatches(geom, id)` NULL `ST_NumPatches(geom, id, id)`.
- **ST\_NumPoints** - `ST_NumPoints(ST_LineString geom)` `ST_NumPoints(ST_CircularString geom)`.
- **ST\_OrderingEquals** - Tests if two geometries represent the same geometry and have points in the same directional order
- **ST\_Overlaps** - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- **ST\_PatchN** - `ST_PatchN(ST_Geometry geom, n)`.
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Point** - Creates a Point with X, Y and SRID values.
- **ST\_PointFromText** - `ST_PointFromText(SRID, WKT)` `ST_PointFromText(SRID, WKT, SRID)`, `ST_PointFromText(WKT, SRID)`, `ST_PointFromText(WKT, SRID, SRID)`.
- **ST\_PointFromWKB** - `ST_PointFromWKB(SRID, WKB)`.
- **ST\_PointN** - `ST_PointN(ST_LineString geom, n)` `ST_PointN(ST_CircularString geom, n)`.
- **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
- **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID.
- **ST\_PolygonFromText** - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.
- **ST\_Relate** - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST\_RemEdgeModFace** - Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.
- **ST\_RemEdgeNewFace** - `ST_RemEdgeNewFace(geom, edge_id, new_geom)`, `ST_RemEdgeNewFace(geom, edge_id, new_geom, id)`.

- **ST\_RemoveIsoEdge** - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.
- **ST\_RemoveIsoNode** - Removes an isolated node and returns description of action. If the node is not isolated, then an exception is thrown.
- **ST\_SRID** - Returns the spatial reference identifier for a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST\_Touches** - Tests if two geometries have at least one point in common, but their interiors do not intersect.
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST\_Volume** - Returns the volume of a 3D geometry.
- **ST\_WKBToSQL** - WKB(Well-Known Binary) representation of a ST\_Geometry object. Returns the SRID of the geometry.
- **ST\_WKTToSQL** - WKT(Well-Known Text) representation of a ST\_Geometry object. Returns the SRID of the geometry.
- **ST\_Within** - Tests if every point of A lies in B, and their interiors have a point in common.
- **ST\_X** - Returns the X coordinate of a Point.
- **ST\_Y** - Returns the Y coordinate of a Point.
- **ST\_Z** - Returns the Z coordinate of a Point.
- **TG\_ST\_SRID** - Returns the spatial reference identifier for a topogeometry.

## 13.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a **geography** data type object.



### Note

Functions with a (T) are not native geodetic functions, and use a ST\_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- **ST\_Area** - Returns the area of a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKT** - Returns the WKT(Well-Known Text) representation of a geometry/geography with SRID meta data.
- **ST\_AsGML** - Returns the GML 2 or GML 3 representation of a geometry/geography.

- **ST\_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST\_AsKML** - GML 2 GML 3
- **ST\_AsSVG** - Returns SVG path data for a geometry.
- **ST\_AsText** - WKT(Well-Known Text) SRID
- **ST\_Azimuth** - 2
- **ST\_Buffer** - Computes a geometry covering all points within a given distance from a geometry.
- **ST\_Centroid**
- **ST\_ClosestPoint** - Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST\_CoveredBy** - Tests if every point of A lies in B
- **ST\_Covers** - Tests if every point of B lies in A
- **ST\_DWithin** - Tests if two geometries are within a given distance
- **ST\_Distance** - 3 (longest)
- **ST\_GeogFromText** - WKT ( )
- **ST\_GeogFromWKB** - WKB EWKB( WKB)
- **ST\_GeographyFromText** - WKT ( )
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common)
- **ST\_Length**
- **ST\_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
- **ST\_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
- **ST\_LineLocatePoint** - Returns the fractional location of the closest point on a line to a point.
- **ST\_LineSubstring** - Returns the part of a line between two fractional locations.
- **ST\_Perimeter** - Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Project** - Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_Segmentize** - Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_ShortestLine** - 2
- **ST\_Summary**
- **<->** - A B 2
- **&&** - A 2D B 2D TRUE





- **ST\_HillShade** - Returns a raster of hillshade values for a given raster. The raster is converted to a 3D surface and the hillshade is calculated based on the slope and aspect of the surface.
- **ST\_Histogram** - Returns a histogram of the values in a raster. The histogram is a table with columns for the value range and the count of pixels in that range.
- **ST\_InterpolateRaster** - Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- **ST\_Intersection** - Returns the intersection of two rasters. The resulting raster has the same extent as the input rasters and contains the minimum value of the two input rasters at each pixel.
- **ST\_Intersects** - Returns true if two rasters intersect. The rasters are identified by their raster IDs (rastA and rastB).
- **ST\_IsEmpty** - Returns true if a raster is empty (width = 0, height = 0).
- **ST\_MakeEmptyCoverage** - Cover georeferenced area with a grid of empty raster tiles.
- **ST\_MakeEmptyRaster** - Returns an empty raster with the same extent as the input raster. The raster is defined by its extent (xmin, xmax, ymax, ymin), its resolution (scalex, scaley, skewx & skewy), its SRID, and its extenttype (INTERSECTION, UNION, FIRST, SECOND).
- **ST\_MapAlgebra (callback function version)** - Returns a raster where each pixel is the result of a callback function applied to the input rasters. The callback function is a SQL expression that takes the values of the input rasters as arguments.
- **ST\_MapAlgebraExpr** - Returns a raster where each pixel is the result of a PostgreSQL expression applied to the input rasters. The expression is a SQL expression that takes the values of the input rasters as arguments.
- **ST\_MapAlgebraExpr** - Returns a raster where each pixel is the result of a PostgreSQL expression applied to the input rasters. The expression is a SQL expression that takes the values of the input rasters as arguments. The extenttype is INTERSECTION, UNION, FIRST, or SECOND.
- **ST\_MapAlgebraFct** - Returns a raster where each pixel is the result of a PostgreSQL function applied to the input rasters. The function is a SQL function that takes the values of the input rasters as arguments.
- **ST\_MapAlgebraFct** - Returns a raster where each pixel is the result of a PostgreSQL function applied to the input rasters. The function is a SQL function that takes the values of the input rasters as arguments. The extenttype is INTERSECTION.
- **ST\_MapAlgebraFctNgb** - Returns a raster where each pixel is the result of a PostgreSQL function applied to the input rasters. The function is a SQL function that takes the values of the input rasters and their neighbors as arguments. The neighborhood is defined by the neighborhood parameter.
- **ST\_MapAlgebra (expression version)** - Returns a raster where each pixel is the result of a PostgreSQL expression applied to the input rasters. The expression is a SQL expression that takes the values of the input rasters as arguments.
- **ST\_MemSize** - Returns the memory size of a raster in bytes.
- **ST\_MetaData** - Returns the metadata of a raster, including its extent, resolution, SRID, and extenttype.
- **ST\_MinConvexHull** - Returns the minimum convex hull of a raster.
- **ST\_NearestValue** - Returns the value of the nearest non-null pixel to a given pixel. The input raster is identified by its raster ID.
- **ST\_Neighborhood** - Returns a raster where each pixel is the result of a PostgreSQL expression applied to the input rasters and their neighbors. The neighborhood is defined by the neighborhood parameter.



- **ST\_NotSameAlignmentReason** - Returns the reason why two rasters do not have the same alignment.
- **ST\_NumBands** - Returns the number of bands in a raster.
- **ST\_Overlaps** - Returns true if two rasters overlap. `rastA` and `rastB` are the rasters to be compared.
- **ST\_PixelAsCentroid** - Returns the centroid of a pixel in a raster.
- **ST\_PixelAsCentroids** - Returns a table of centroids for each pixel in a raster. Columns: `x`, `y`.
- **ST\_PixelAsPoint** - Returns a point for each pixel in a raster.
- **ST\_PixelAsPoints** - Returns a table of points for each pixel in a raster. Columns: `x`, `y`.
- **ST\_PixelAsPolygon** - Returns a polygon for each pixel in a raster.
- **ST\_PixelAsPolygons** - Returns a table of polygons for each pixel in a raster. Columns: `x`, `y`.
- **ST\_PixelHeight** - Returns the height of a pixel in a raster.
- **ST\_PixelOfValue** - Returns the column and row of a pixel with a specific value. Columns: `columnx`, `rowy`.
- **ST\_PixelWidth** - Returns the width of a pixel in a raster.
- **ST\_Polygon** - Returns a polygon for a NODATA value in a raster.
- **ST\_Quantile** - Returns the quantile of a raster. Parameters: `population`, `quantile`. Example: `25%`, `50%`, `75%` (percentile).
- **ST\_RastFromHexWKB** - Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- **ST\_RastFromWKB** - Return a raster value from a Well-Known Binary (WKB) raster.
- **ST\_RasterToWorldCoord** - Returns the world coordinates (X, Y) for a pixel in a raster. Column: `1`.
- **ST\_RasterToWorldCoordX** - Returns the X world coordinate for a pixel in a raster. Column: `1`.
- **ST\_RasterToWorldCoordY** - Returns the Y world coordinate for a pixel in a raster. Column: `1`.
- **ST\_Reclass** - Reclassify a raster. Parameters: `nband`, `1`. Example: `16BUI`, `8BUI`.
- **ST\_Resample** - Resample a raster. Parameters: `method`, `width`, `height`.
- **ST\_Rescale** - Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- **ST\_Resize** - Resize a raster.
- **ST\_Reskew** - Reskew a raster. Parameters: `method`, `width`, `height`. Example: `NearestNeighbor`, `Bilinear`, `Cubic`, `CubicSpline`, `Lanczos`.



- **ST\_TPI** - (Topographic Position Index)
- **ST\_TRI** - (Terrain Ruggedness Index)
- **ST\_Tile** -
- **ST\_Touches** - rastA rastB, TRUE
- **ST\_Transform** - NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos. NearestNeighbor
- **ST\_Union** - 1
- **ST\_UpperLeftX** - X
- **ST\_UpperLeftY** - Y
- **ST\_Value** - columnx, rowy, exclude\_nodata\_value, nodata
- **ST\_ValueCount** - (frequency) 1 NODATA
- **ST\_Width** -
- **ST\_Within** - rastB rastA, rastA rastB
- **ST\_WorldToRasterCoord** - X, Y(xw, yw)
- **ST\_WorldToRasterCoordX** - (pt) X, Y(xw, yw)
- **ST\_WorldToRasterCoordY** - (pt) X, Y(xw, yw)
- **UpdateRasterSRID** - SRID

### 13.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single **geometry\_dump** or **geomval** data type object.

- **ST\_DumpAsPolygons** - geomval(geom, val)
- **ST\_Intersection** -
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** -
- **ST\_DumpRings** - Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.
- **ST\_DumpSegments** -

## 13.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box\* family of PostGIS spatial types. The box family of types consists of **box2d**, and **box3d**

- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **Box3D** - `BOX3D`
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DMakeBox** - Creates a BOX3D defined by two 3D point geometries.
- **ST\_AsMVTGeom** - Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_AsTWKB** - `TWKB`(Tiny Well-Known Binary)
- **ST\_Box2dFromGeoHash** - GeoHash `BOX2D`
- **ST\_ClipByBox2D** - Computes the portion of a geometry falling within a rectangle.
- **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST\_Expand** - Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_Extent** - Aggregate function that returns the bounding box of geometries.
- **ST\_MakeBox2D** - Creates a BOX2D defined by two 2D point geometries.
- **ST\_RemoveIrrelevantPointsForView** - Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **RemoveUnusedPrimitives** - Removes topology primitives which not needed to define existing Topo-Geometry objects.
- **ValidateTopology** - Returns a set of `validatetopology_returntype` objects detailing issues with topology.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (BOX2DF).
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

## 13.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- **AddGeometryColumn** - Add a geometry column to a table.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **CG\_3DArea** - 3D area of a geometry. Returns 0 for non-polygons.
- **CG\_3DConvexHull** - 3D convex hull of a geometry.
- **CG\_3DDifference** - 3D difference of two geometries.
- **CG\_3DIntersection** - 3D intersection of two geometries.
- **CG\_3DUnion** - Perform 3D union.
- **CG\_ApproximateMedialAxis** - Approximate medial axis of a geometry.
- **CG\_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **CG\_Extrude** - Extrude a 2D geometry into 3D.
- **CG\_ForceLHR** - LHR(Left Hand Reverse; `force_lhr`) flag for `CG_3DUnion`.
- **CG\_IsPlanar** - Check if a 3D geometry is planar.
- **CG\_IsSolid** - Check if a 3D geometry is a solid.
- **CG\_MakeSolid** - Make a 3D geometry solid. Returns `NULL` if the input is not a TIN.
- **CG\_Orientation** - Orientation (orientation) of a 3D geometry.
- **CG\_StraightSkeleton** - Straight skeleton (straight skeleton) of a 3D geometry.
- **CG\_Tessellate** - Tessellation (tessellation) of a 3D geometry into TINs.
- **CG\_Visibility** - Compute a visibility polygon from a point or a segment in a polygon geometry.
- **CG\_Volume** - 3D volume of a geometry. Returns 0 for non-polyhedrons.
- **DropGeometryColumn** - Drop a geometry column from a table.
- **GeometryType** - ST\_GeometryType of a 3D geometry.

- **ST\_3DArea** - 3D area of a geometry. Returns 0 for non-3D geometries.
- **ST\_3DClosestPoint** - g2 3D closest point to g1 3D geometry. Returns 3D geometry.
- **ST\_3DConvexHull** - 3D convex hull of a geometry.
- **ST\_3DDFullyWithin** - Tests if two 3D geometries are entirely within a given 3D distance
- **ST\_3DDWithin** - Tests if two 3D geometries are within a given 3D distance
- **ST\_3DDifference** - 3D difference of two geometries.
- **ST\_3DDistance** - 3D distance between two geometries (SRS dependent) 3D distance.
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DIntersection** - 3D intersection of two geometries.
- **ST\_3DIntersects** - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_3DLength** - 3D length of a geometry.
- **ST\_3DLineInterpolatePoint** - Returns a point interpolated along a 3D line at a fractional location.
- **ST\_3DLongestLine** - 3D longest line (longest) of a geometry.
- **ST\_3DMaxDistance** - 3D maximum distance between two geometries (SRS dependent) 3D distance.
- **ST\_3DPerimeter** - 3D perimeter of a geometry.
- **ST\_3DShortestLine** - 3D shortest line (shortest) of a geometry.
- **ST\_3DUnion** - Perform 3D union.
- **ST\_AddMeasure** - Interpolates measures along a linear geometry.
- **ST\_AddPoint** - 3D add point to a geometry.
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.
- **ST\_ApproximateMedialAxis** - 3D approximate medial axis of a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST\_AsEWKT** - 3D WKT(Well-Known Text) with SRID meta data.
- **ST\_AsGML** - 3D GML 2 or GML 3 representation.
- **ST\_AsGeoJSON** - Return a geometry or feature in GeoJSON format.
- **ST\_AsHEXEWKB** - 3D (NDR) (XDR) HEXEWKB (HEX) representation.
- **ST\_AsKML** - 3D GML 2 or GML 3 representation.
- **ST\_AsX3D** - 3D X3D XML representation: ISO-IEC-19776-1.2-X3DEncodings-XML representation.
- **ST\_Boundary** - 3D boundary of a geometry.

- **ST\_BoundingDiagonal** - Returns the bounding diagonal of a geometry.
- **ST\_CPAWithin** - Tests if the closest point of approach of two trajectories is within the specified distance.
- **ST\_ChaikinSmoothing** - Returns a smoothed version of a geometry, using the Chaikin algorithm.
- **ST\_ClosestPointOfApproach** - Returns a measure at the closest point of approach of two trajectories.
- **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_ConstrainedDelaunayTriangles** - Return a constrained Delaunay triangulation around the given input geometry.
- **ST\_ConvexHull** - Computes the convex hull of a geometry.
- **ST\_CoordDim** - ST\_Geometry dimension.
- **ST\_CurveN** - Returns the Nth component curve geometry of a CompoundCurve.
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_DelaunayTriangles** - Returns the Delaunay triangulation of the vertices of a geometry.
- **ST\_Difference** - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_DistanceCPA** - Returns the distance between the closest point of approach of two trajectories.
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - Returns a set of geometry\_dump rows for the points of a geometry.
- **ST\_DumpRings** - Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.
- **ST\_DumpSegments** - Returns a set of geometry\_dump rows for the segments of a geometry.
- **ST\_EndPoint** - ST\_LineString or ST\_CircularString endpoint.
- **ST\_ExteriorRing** - Returns the exterior ring of a Polygon.
- **ST\_Extrude** - Returns a 3D geometry from a 2D geometry.
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.
- **ST\_Force2D** - Returns a 2D geometry from a 3D geometry.
- **ST\_ForceCurve** - Returns a Curve geometry from a Geometry.
- **ST\_ForceLHR** - LHR(Left Hand Reverse; Orientation) Returns a Polygon.
- **ST\_ForcePolygonCCW** - Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST\_ForcePolygonCW** - Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST\_ForceRHR** - Returns a Polygon (orientation) Returns a Polygon (Right-Hand Rule).
- **ST\_ForceSFS** - Returns SFS 1.1 Geometry.
- **ST\_Force\_3D** - Returns XYZ Geometry. ST\_Force3DZ Returns XYZ Geometry.
- **ST\_Force\_3DZ** - Returns XYZ Geometry.
- **ST\_Force\_4D** - Returns XYZM Geometry.
- **ST\_Force\_Collection** - Returns GeometryCollection.

- **ST\_GeomFromEWKB** - EWKB(Extended Well-Known Binary) ST\_Geometry.
- **ST\_GeomFromEWKT** - EWKT(Extended Well-Known Text) ST\_Geometry.
- **ST\_GeomFromGML** - GML PostGIS.
- **ST\_GeomFromGeoJSON** - GeoJSON PostGIS.
- **ST\_GeomFromKML** - KML PostGIS.
- **ST\_GeometricMedian** - (median).
- **ST\_GeometryN** - ST\_Geometry.
- **ST\_GeometryType** - ST\_Geometry.
- **ST\_HasArc** - Tests if a geometry contains a circular arc
- **ST\_HasM** - Checks if a geometry has an M (measure) dimension.
- **ST\_HasZ** - Checks if a geometry has a Z dimension.
- **ST\_InteriorRingN** -.
- **ST\_InterpolatePoint** - (M) .
- **ST\_Intersection** - Computes a geometry representing the shared portion of geometries A and B.
- **ST\_IsClosed** - LINESTRING TRUE. ( ) TRUE.
- **ST\_IsCollection** - , , TRUE.
- **ST\_IsPlanar** -.
- **ST\_IsPolygonCCW** - Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST\_IsPolygonCW** - Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- **ST\_IsSimple** - TRUE.
- **ST\_IsSolid** -.
- **ST\_IsValidTrajectory** - Tests if the geometry is a valid trajectory.
- **ST\_Length\_Spheroid** -.
- **ST\_LineFromMultiPoint** -.
- **ST\_LineInterpolatePoint** - Returns a point interpolated along a line at a fractional location.
- **ST\_LineInterpolatePoints** - Returns points interpolated along a line at a fractional interval.
- **ST\_LineSubstring** - Returns the part of a line between two fractional locations.
- **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST\_LocateBetweenElevations** - Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_M** - Returns the M coordinate of a Point.
- **ST\_MakeLine** - , .
- **ST\_MakePoint** - Creates a 2D, 3DZ or 4D Point.



- **ST\_MakePolygon** - Creates a Polygon from a shell and optional list of holes.
- **ST\_MakeSolid** - `ST_MakeSolid(geometry, hole_list)`. `ST_MakeSolid(geometry, hole_list, TIN)`.
- **ST\_MakeValid** - Attempts to make an invalid geometry valid without losing vertices.
- **ST\_MemSize** - `ST_MemSize(geometry)`.
- **ST\_MemUnion** - Aggregate function which unions geometries in a memory-efficient but slower way.
- **ST\_NDims** - `ST_NDims(geometry)`.
- **ST\_NPoints** - `ST_NPoints(geometry)` (`ST_NPoints(geometry, hole_list)`).
- **ST\_NRings** - `ST_NRings(geometry)`.
- **ST\_Node** - Nodes a collection of lines.
- **ST\_NumCurves** - Return the number of component curves in a CompoundCurve.
- **ST\_NumGeometries** - `ST_NumGeometries(geometry)`. `ST_NumGeometries(geometry, hole_list)`.
- **ST\_NumPatches** - `ST_NumPatches(geometry)`. `ST_NumPatches(geometry, hole_list)` NULL `ST_NumPatches(geometry, hole_list, TIN)`.
- **ST\_Orientation** - `ST_Orientation(geometry)` (orientation) `ST_Orientation(geometry, hole_list)`.
- **ST\_PatchN** - `ST_PatchN(geometry, n)`.
- **ST\_PointFromWKB** - `ST_PointFromWKB(SRID, WKB)`.
- **ST\_PointN** - `ST_PointN(geometry, n)` `ST_PointN(geometry, n, hole_list)`.
- **ST\_PointOnSurface** - Computes a point guaranteed to lie in a polygon, or on a geometry.
- **ST\_Points** - `ST_Points(geometry)`.
- **ST\_Polygon** - Creates a Polygon from a LineString with a specified SRID.
- **ST\_RemovePoint** - Remove a point from a linestring.
- **ST\_RemoveRepeatedPoints** - Returns a version of a geometry with duplicate points removed.
- **ST\_Reverse** - `ST_Reverse(geometry)`.
- **ST\_Rotate** - Rotates a geometry about an origin point.
- **ST\_RotateX** - Rotates a geometry about the X axis.
- **ST\_RotateY** - Rotates a geometry about the Y axis.
- **ST\_RotateZ** - Rotates a geometry about the Z axis.
- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_Scroll** - Change start point of a closed LineString.
- **ST\_SetPoint** - `ST_SetPoint(geometry, point, index)`.
- **ST\_ShiftLongitude** - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST\_SnapToGrid** - `ST_SnapToGrid(geometry, snap)`.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_StraightSkeleton** - `ST_StraightSkeleton(geometry)` (straight skeleton) `ST_StraightSkeleton(geometry, hole_list)`.
- **ST\_SwapOrdinates** - `ST_SwapOrdinates(geometry)`.

- **ST\_SymDifference** - Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST\_Tessellate** - (tessellation) TIN TIN.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_UnaryUnion** - Computes the union of the components of a single geometry.
- **ST\_Union** - Computes a geometry representing the point-set union of the input geometries.
- **ST\_Volume** - 3 (0).
- **ST\_WrapX** - X.
- **ST\_X** - Returns the X coordinate of a Point.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_Y** - Returns the Y coordinate of a Point.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_Z** - Returns the Z coordinate of a Point.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - ST\_Geometry.
- **TG\_Equals** - TopoGeometry.
- **TG\_Intersects** - TopoGeometry.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **geometry\_overlaps\_nd** - A n B n TRUE.
- **overlaps\_nd\_geometry\_gidx** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **overlaps\_nd\_gidx\_geometry** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **overlaps\_nd\_gidx\_gidx** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

## 13.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- **AddGeometryColumn** - `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name, integer srid, integer typecode, integer flags`.
- **Box2D** - Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - Returns a BOX3D representing the 3D extent of a geometry.
- **DropGeometryColumn** - `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name, integer srid`.
- **GeometryType** - ST\_Geometry `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name`.
- **PostGIS\_AddBBox** - `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name, integer srid`.
- **PostGIS\_DropBBox** - `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name, integer srid`.
- **PostGIS\_HasBBox** - `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name, integer srid, boolean has_bbox`.
- **ST\_3DExtent** - Aggregate function that returns the 3D bounding box of geometries.
- **ST\_Affine** - Apply a 3D affine transformation to a geometry.
- **ST\_AsBinary** - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKB** - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST\_AsEWKT** - `text` WKT(Well-Known Text) `integer` SRID `text` `integer` flags
- **ST\_AsHEXEWKB** - `integer` (NDR) `integer` (XDR) `text` HEXEWKB (`integer`) `integer` flags
- **ST\_AsSVG** - Returns SVG path data for a geometry.
- **ST\_AsText** - `text`/`text` WKT(Well-Known Text) `integer` SRID `text` `integer` flags
- **ST\_ClusterDBSCAN** - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterWithin** - Aggregate function that clusters geometries by separation distance.
- **ST\_ClusterWithinWin** - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST\_Collect** - Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_CoordDim** - ST\_Geometry `varchar(64) schema_name, varchar(64) table_name, varchar(64) column_name`.
- **ST\_CurveToLine** - Converts a geometry containing curves to a linear geometry.
- **ST\_Distance** - `integer` 3 `integer` (longest) `text`.
- **ST\_Dump** - Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - `text`/`text` `integer` `integer` `text`.
- **ST\_EndPoint** - ST\_LineString `text` ST\_CircularString `text`/`text` `integer` `integer` `text`.
- **ST\_EstimatedExtent** - Returns the estimated extent of a spatial table.
- **ST\_FlipCoordinates** - Returns a version of a geometry with X and Y axis flipped.

- **ST\_Force2D** - Forces 2D geometry.
- **ST\_ForceCurve** - Forces geometry to be a curve (upcast).
- **ST\_ForceSFS** - Forces geometry to SFS 1.1.
- **ST\_Force3D** - Forces XYZ geometry. **ST\_Force3DZ** - Forces XYZ geometry.
- **ST\_Force3DM** - Forces XYM geometry.
- **ST\_Force3DZ** - Forces XYZ geometry.
- **ST\_Force4D** - Forces XYZM geometry.
- **ST\_ForceCollection** - Forces geometry to be a collection.
- **ST\_GeoHash** - Returns GeoHash for geometry.
- **ST\_GeogFromWKB** - WKB to EWKB (Extended Well-Known Binary).
- **ST\_GeomFromEWKB** - EWKB(Extended Well-Known Binary) to ST\_Geometry.
- **ST\_GeomFromEWKT** - EWKT(Extended Well-Known Text) to ST\_Geometry.
- **ST\_GeomFromText** - WKT to ST\_Geometry.
- **ST\_GeomFromWKB** - WKB(Well-Known Binary) to ST\_Geometry with SRID.
- **ST\_GeometryN** - Returns N-th geometry in a collection.
- **=** - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **&<|** - Returns TRUE if geometry A is contained by geometry B.
- **ST\_HasArc** - Tests if a geometry contains a circular arc.
- **ST\_Intersects** - Tests if two geometries intersect (they have at least one point in common).
- **ST\_IsClosed** - Tests if a LINESTRING is closed.
- **ST\_IsCollection** - Tests if a geometry is a collection.
- **ST\_IsEmpty** - Tests if a geometry is empty.
- **ST\_LineToCurve** - Converts a linear geometry to a curved geometry.
- **ST\_MemSize** - Returns memory size of a geometry.
- **ST\_NPoints** - Returns number of points in a geometry.
- **ST\_NRings** - Returns number of rings in a geometry.
- **ST\_PointFromWKB** - WKB to ST\_Point.
- **ST\_PointN** - Returns N-th point in a geometry.
- **ST\_Points** - Returns array of points in a geometry.
- **ST\_Rotate** - Rotates a geometry about an origin point.
- **ST\_RotateZ** - Rotates a geometry about the Z axis.
- **ST\_SRID** - Returns the spatial reference identifier for a geometry.

- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_SetSRID** - Set the SRID on a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_Summary** - `ST_Summary(geometry)`.
- **ST\_SwapOrdinates** - `ST_SwapOrdinates(geometry)`.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - `ST_Geometry(geometry, flags)`.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table meta-data.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- **&&** - `A && B` - A 2D BOX2DF B 2D BOX2DF TRUE BOX2DF.
- **&&&** - `A &&& B` - A n BOX2DF B n BOX2DF TRUE BOX2DF.
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).









- **ST\_Points** - Returns a set of points from a geometry.
- **ST\_Rotate** - Rotates a geometry about an origin point.
- **ST\_RotateZ** - Rotates a geometry about the Z axis.
- **ST\_SRID** - Returns the spatial reference identifier for a geometry.
- **ST\_Scale** - Scales a geometry by given factors.
- **ST\_SetSRID** - Set the SRID on a geometry.
- **ST\_StartPoint** - Returns the first point of a LineString.
- **ST\_Summary** - Returns a summary of a geometry.
- **ST\_SwapOrdinates** - Swaps the X and Y coordinates of a geometry.
- **ST\_TransScale** - Translates and scales a geometry by given offsets and factors.
- **ST\_Transform** - Return a new geometry with coordinates transformed to a different spatial reference system.
- **ST\_Translate** - Translates a geometry by given offsets.
- **ST\_XMax** - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- **ST\_XMin** - Returns the X minima of a 2D or 3D bounding box or a geometry.
- **ST\_YMax** - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- **ST\_YMin** - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMax** - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- **ST\_ZMin** - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- **ST\_Zmflag** - ST\_Geometry returns TRUE if the geometry is 3D.
- **UpdateGeometrySRID** - Updates the SRID of all features in a geometry column, and the table metadata.
- **~(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- **~(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bounding box.
- **~(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box contains a 2D float precision bounding box (BOX2DF).
- **&&** - A 2D bounding box B contains A 2D bounding box TRUE.
- **&&&** - A n-dimensional bounding box B contains A n-dimensional bounding box TRUE.
- **@(box2df,box2df)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **&&(box2df,box2df)** - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.



- **&&(box2df,geometry)** - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **&&&(geometry,gidx)** - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **&&&(gidx,geometry)** - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **&&&(gidx,gidx)** - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

### 13.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A  means the function works with the type or subtype natively.
- A  means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A  means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A  means the function only available if PostGIS compiled with SFCGAL support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Collect	✓		✓	✓			
ST_LineFromMPoint	✓		✓				
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint	✓		✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_Point	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointZ	✓						
ST_PointM	✓						
ST_PointZM	✓						
ST_Polygon	✓		✓		✓		
ST_TileEnvelope	✓						
ST_HexagonGrid	✓						
ST_Hexagon	✓						
ST_SquareGrid	✓						
ST_Square	✓						
ST_Letters	✓						
GeometryType	✓		✓	✓		✓	✓
ST_Boundary	✓		✓		✓		
ST_BoundingDiagonal	✓		✓				
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_InteriorRing	✓		✓		✓		
ST_NumCurves	✓		✓		✓		
ST_CurveN	✓		✓		✓		
ST_IsClosed	✓		✓	✓	✓	✓	
ST_IsCollection	✓		✓	✓			
ST_IsEmpty	✓			✓	✓		
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing	✓				✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_IsSimple	✓		✓		✓		
ST_M	✓		✓		✓		
ST_MemSize	✓		✓	✓		✓	✓
ST_NDims	✓		✓				
ST_NPoints	✓		✓	✓		✓	
ST_NRings	✓		✓	✓			
ST_NumGeometries	✓		✓		✓	✓	✓
ST_NumInteriorRings	✓				✓		
ST_NumInteriorRing	✓						
ST_NumPatches	✓		✓		✓	✓	
ST_NumPoints	✓				✓		
ST_PatchN	✓		✓		✓	✓	
ST_PointN	✓		✓	✓	✓		
ST_Points	✓		✓	✓			
ST_StartPoint	✓		✓	✓	✓		
ST_Summary	✓	✓		✓		✓	✓
ST_X	✓		✓		✓		
ST_Y	✓		✓		✓		
ST_Z	✓		✓		✓		
ST_Zmflag	✓		✓	✓			
ST_HasZ	✓		✓				
ST_HasM	✓		✓				
ST_AddPoint	✓		✓				
ST_CollectionExtract	✓						
ST_CollectionHomogenize	✓						
ST_CurveToLine	✓		✓	✓	✓		
ST_Scroll	✓		✓				
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_Force3D	✓		✓	✓		✓	
ST_Force3DZ	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_ForceCurve	✓		✓	✓			
ST_ForcePolygonCW	✓		✓				
ST_ForcePolygonW	✓		✓				
ST_ForceSFS	✓		✓	✓		✓	✓
ST_ForceRHR	✓		✓			✓	
ST_LineExtend	✓						
ST_LineToCurve	✓		✓	✓			
ST_Multi	✓						
ST_Normalize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordinates	✓						
ST_RemovePoint	✓		✓				
ST_RemoveRepeatedPoints	✓		✓			✓	
ST_RemoveIrrelevantPointsForView	✓						
ST_RemoveSmallParts	✓						
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_InverseTransformPipeline	✓						
ST_SetSRID	✓			✓			
ST_SRID	✓			✓	✓		
ST_Transform	✓			✓	✓	✓	
ST_TransformPipeline	✓						
postgis_srs_codes							
postgis_srs							
postgis_srs_all							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
postgis_srs_seal	✓						
ST_BdPolyFromText	✓						
ST_BdMPolyFromText	✓						
ST_GeogFromText		✓					
ST_GeographyFromText		✓					
ST_GeomCollFromText	✓				✓		
ST_GeomFromEWKT	✓		✓	✓		✓	✓
ST_GeomFromMRC21	✓						
ST_GeometryFromText	✓				✓		
ST_GeomFromI	✓			✓	✓		
ST_LineFromText	✓				✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_PointFromText	✓				✓		
ST_PolygonFromText	✓				✓		
ST_WKTToSQL	✓				✓		
ST_GeogFromWKB		✓		✓			
ST_GeomFromEWKB	✓		✓	✓		✓	✓
ST_GeomFromV3	✓			✓	✓		
ST_LineFromWKB	✓				✓		
ST_LinestringFromWKB	✓				✓		
ST_PointFromWKB	✓		✓	✓	✓		
ST_WKBToSQL	✓				✓		
ST_Box2dFromGeoHash	✓						
ST_GeomFromGeoHash	✓						
ST_GeomFromC	✓		✓			✓	✓
ST_GeomFromCJSON	✓		✓				
ST_GeomFromkml	✓		✓				
ST_GeomFromI3B	✓						
ST_GMLToSQL	✓				✓		
ST_LineFromEncodedPolyline	✓						
ST_PointFromGeoHash							
ST_FromFlatGeobufToTable							
ST_FromFlatGeobuf							

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsText	✓	✓		✓	✓		
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsHEXEWKB	✓		✓	✓			
ST_AsEncodedPolyline	✓ line						
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,box2df)	✓		✓	✓		✓	✓
&&&(gidx,geometry)	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
&<	✓						
&<	✓			✓		✓	
&>	✓						
<<	✓						
<<	✓						
=	✓	✓		✓		✓	
>>	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
@	✓						
@(geometry,box) ✓	✓			✓		✓	
@(box2df,geom) ✓	✓			✓		✓	
@(box2df,box2df) ✓	✓			✓		✓	
&>	✓						
>>	✓						
~	✓						
~(geometry,box) ✓	✓			✓		✓	
~(box2df,geom) ✓	✓			✓		✓	
~(box2df,box2df) ✓	✓			✓		✓	
~=	✓					✓	
<->	✓	✓					
=	✓						
<#>	✓						
<<->>	✓						
ST_3DIntersects	✓		✓		✓	✓	✓
ST_Contains	✓				✓		
ST_ContainsProperly	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_Disjoint	✓				✓		
ST_Equals	✓				✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_LineCrossingDirection	✓						
ST_OrderingEquivalent	✓				✓		
ST_Overlaps	✓				✓		
ST_Relate	✓				✓		
ST_RelateMatch	✓						
ST_Touches	✓				✓		
ST_Within	✓				✓		
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDFullyWithin	✓		✓			✓	
ST_DFullyWithin	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_DWithin	✓	✓					
ST_PointInsideCircle	✓						
ST_Area	✓	✓			✓	✓	
ST_Azimuth	✓	✓					
ST_Angle	✓						
ST_ClosestPoint	✓	✓					
ST_3DClosestPoint	✓		✓			✓	
ST_Distance	✓	✓		✓	✓		
ST_3DDistance	✓		✓		✓	✓	
ST_DistanceSphere	✓						
ST_DistanceSphereoid	✓						
ST_FrechetDistance	✓						
ST_HausdorffDistance	✓						
ST_Length	✓	✓			✓		
ST_Length2D	✓						
ST_3DLength	✓		✓		✓		
ST_LengthSphereoid	✓		✓				
ST_LongestLine	✓						
ST_3DLongestLine	✓		✓			✓	
ST_MaxDistance	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_MinimumClearance	✓						
ST_MinimumClearanceLine	✓						
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_3DPerimeter	✓		✓		✓		
ST_ShortestLine	✓	✓					
ST_3DShortestLine	✓		✓			✓	
ST_ClipByBox2D	✓						
ST_Difference	✓		✓		✓		
ST_Intersection	✓	☺	✓		✓		
ST_MemUnion	✓		✓				
ST_Node	✓		✓				
ST_Split	✓						



Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Subdivide	✓						
ST_SymDifferen	✓		✓		✓		
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Buffer	✓	😄			✓		
ST_BuildArea	✓						
ST_Centroid	✓	✓			✓		
ST_ChaikinSmo	✓		✓				
ST_ConcaveHul	✓						
ST_ConvexHull	✓		✓		✓		
ST_DelaunayTri	✓		✓				✓
ST_FilterByM	✓						
ST_GeneratePoi	✓						
ST_GeometricM	✓		✓				
ST_LineMerge	✓						
ST_MaximumIn	✓						
ST_LargestEmp	✓						
ST_MinimumBo	✓						
ST_MinimumBo	✓						
ST_OrientedEnv	✓						
ST_OffsetCurve	✓						
ST_PointOnSurf	✓		✓		✓		
ST_Polygonize	✓						
ST_ReducePrec	✓						
ST_SharedPaths	✓						
ST_Simplify	✓						
ST_SimplifyPres	✓						
ST_SimplifyPoly	✓						
ST_SimplifyVW	✓						
ST_SetEffective	✓						
ST_Triangulate	✓						
ST_VoronoiLine	✓						
ST_VoronoiPoly	✓						
ST_CoverageInv	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_CoverageSimplify	✓						
ST_CoverageUnion	✓						
ST_Affine	✓		✓	✓		✓	✓
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_Scale	✓		✓	✓		✓	✓
ST_Translate	✓		✓	✓			
ST_TransScale	✓		✓	✓			
ST_ClusterDBSCAN	✓			✓			
ST_ClusterIntersecting	✓						
ST_ClusterIntersectingWin	✓						
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithin/n	✓			✓			
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
ST_EstimatedExtent	✗			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_3DExtent	✓		✓	✓		✓	✓
ST_MakeBox2D	✓						
ST_3DMakeBox	✓						
ST_XMax	✗		✓	✓			
ST_XMin	✗		✓	✓			
ST_YMax	✗		✓	✓			
ST_YMin	✗		✓	✓			
ST_ZMax	✗		✓	✓			
ST_ZMin	✗		✓	✓			
ST_LineInterpolatePoint	✓	✓	✓				
ST_3DLineInterpolatePoint	✓		✓				
ST_LineInterpolatePoints	✓	✓	✓				
ST_LineLocatePoint	✓	✓					

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_LineSubstring	✓	✓	✓				
ST_LocateAlong	✓				✓		
ST_LocateBetween	✓				✓		
ST_LocateBetweenElevations	✓		✓				
ST_InterpolateFunction	✓		✓				
ST_AddMeasure	✓		✓				
ST_IsValidTrajectory	✓		✓				
ST_ClosestPointApproach	✓		✓				
ST_DistanceCPA	✓		✓				
ST_CPASWithin	✓		✓				
postgis.backend							
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.enable_outdb_rasters							
postgis.gdal_vsi_options							
PostGIS_AddBB	✓			✓			
PostGIS_DropBB	✓			✓			
PostGIS_HasBB	✓			✓			

## 13.12 New, Enhanced or changed PostGIS Functions

### 13.12.1 PostGIS Functions new or enhanced in 3.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.5

- **ST\_HasM** - Availability: 3.5.0 Checks if a geometry has an M (measure) dimension.
- **ST\_HasZ** - Availability: 3.5.0 Checks if a geometry has a Z dimension.
- **ST\_RemoveIrrelevantPointsForView** - Availability: 3.5.0 Removes points that are irrelevant for rendering a specific rectangular view of a geometry.
- **ST\_RemoveSmallParts** - Availability: 3.5.0 Removes small parts (polygon rings or linestrings) of a geometry.

Functions changed in PostGIS 3.5

- **ST\_AsGeoJSON** - Changed: 3.5.0 allow specifying the column containing the feature id Return a geometry or feature in GeoJSON format.
- **ST\_DFullyWithin** - Changed: 3.5.0 : the logic behind the function now uses a test of containment within a buffer, rather than the ST\_MaxDistance algorithm. Results will differ from prior versions, but should be closer to user expectations. Tests if a geometry is entirely inside a distance of another

### 13.12.2 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- **PostGIS\_GEOS\_Compiled\_Version** - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- **PostGIS\_PROJ\_Compiled\_Version** - Availability: 3.5.0 Returns the version number of the PROJ library against which PostGIS was built.
- **ST\_ClusterIntersectingWin** - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- **ST\_ClusterWithinWin** - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- **ST\_CoverageInvalidEdges** - Availability: 3.4.0 Window function that finds locations where polygons fail to form a valid coverage.
- **ST\_CoverageSimplify** - Availability: 3.4.0 Window function that simplifies the edges of a polygonal coverage.
- **ST\_CoverageUnion** - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- **ST\_InverseTransformPipeline** - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- **ST\_LargestEmptyCircle** - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- **ST\_LineExtend** - Availability: 3.4.0 Returns a line extended forwards and backwards by specified distances.
- **ST\_TransformPipeline** - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- **postgis\_srs** - Availability: 3.4.0 Return a metadata record for the requested authority and srid.
- **postgis\_srs\_all** - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.
- **postgis\_srs\_codes** - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- **postgis\_srs\_search** - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- **PostGIS\_Full\_Version** - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK\_ENABLED, URL\_ENDPOINT and DATABASE\_PATH of proj.db location Reports full PostGIS version and build configuration infos.
  - **PostGIS\_PROJ\_Version** - Enhanced: 3.4.0 now includes NETWORK\_ENABLED, URL\_ENDPOINT and DATABASE\_PATH of proj.db location Returns the version number of the PROJ4 library.
  - **ST\_AsSVG** - Enhanced: 3.4.0 to support all curve types Returns SVG path data for a geometry.
-



### 13.12.4 PostGIS Functions new or enhanced in 3.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.2

- **ST\_AsFlatGeobuf** - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
- **ST\_DumpSegments** - Availability: 3.2.0
- **ST\_FromFlatGeobuf** - Availability: 3.2.0 Reads FlatGeobuf data.
- **ST\_FromFlatGeobufToTable** - Availability: 3.2.0 Creates a table based on the structure of FlatGeobuf data.
- **ST\_Scroll** - Availability: 3.2.0 Change start point of a closed LineString.
- **postgis.gdal\_vsi\_options** - Availability: 3.2.0 DB

Functions enhanced in PostGIS 3.2

- **ST\_ClusterKMeans** - Enhanced: 3.2.0 Support for max\_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_MakeValid** - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Point** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- **ST\_PointM** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- **ST\_PointZ** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- **ST\_PointZM** - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST\_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- **ST\_RemovePoint** - Enhanced: 3.2.0 Remove a point from a linestring.
- **ST\_RemoveRepeatedPoints** - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- **ST\_StartPoint** - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.

Functions changed in PostGIS 3.2

- **ST\_Boundary** - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

### 13.12.5 PostGIS Functions new or enhanced in 3.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.1

- **ST\_Hexagon** - 2.1.0. Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
- **ST\_HexagonGrid** - 2.1.0. Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.
- **ST\_MaximumInscribedCircle** - Availability: 3.1.0. Returns the maximum inscribed circle of a geometry.
- **ST\_ReducePrecision** - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- **ST\_Square** - 2.1.0. Returns a single square, using the provided edge size and cell coordinate within the square grid space.
- **ST\_SquareGrid** - 2.1.0. Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Functions enhanced in PostGIS 3.1

- **ST\_AsEWKT** - Enhanced: 3.1.0 support for optional precision parameter. Returns WKT(Well-Known Text) with SRID.
- **ST\_ClusterKMeans** - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_Difference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the part of geometry A that does not intersect geometry B.
- **ST\_Intersection** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the shared portion of geometries A and B.
- **ST\_MakeValid** - Enhanced: 3.1.0, added removal of Coordinates with NaN values. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Subdivide** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a rectilinear subdivision of a geometry.
- **ST\_SymDifference** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the portions of geometries A and B that do not intersect.
- **ST\_TileEnvelope** - 2.0.0. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- **ST\_UnaryUnion** - Enhanced: 3.1.0 accept a gridSize parameter. Computes the union of the components of a single geometry.
- **ST\_Union** - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the point-set union of the input geometries.

Functions changed in PostGIS 3.1

- **ST\_Force3D** - Changed: 3.1.0. Added support for supplying a non-zero Z value. ST\_Force3DZ.
- **ST\_Force3DM** - Changed: 3.1.0. Added support for supplying a non-zero M value. ST\_Force3DM.

- **ST\_Force3DZ** - Changed: 3.1.0. Added support for supplying a non-zero Z value. `ST_Force3DZ(geometry, z)`.
- **ST\_Force4D** - Changed: 3.1.0. Added support for supplying non-zero Z and M values. `ST_Force4D(geometry, z, m)`.

### 13.12.6 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.0

- **ST\_3DLineInterpolatePoint** - 2.1.0 `ST_3DLineInterpolatePoint(geometry, fraction)`. Returns a point interpolated along a 3D line at a fractional location.
- **ST\_TileEnvelope** - 2.1.0 `ST_TileEnvelope(xmin, xmax, ymin, ymax, z)`. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

Functions enhanced in PostGIS 3.0

- **ST\_AsMVT** - Enhanced: 3.0 - added support for Feature ID. Aggregate function returning a MVT representation of a set of rows.
- **ST\_Contains** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A, and their interiors have a point in common
- **ST\_ContainsProperly** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in the interior of A
- **ST\_CoveredBy** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B
- **ST\_Covers** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A
- **ST\_Crosses** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have some, but not all, interior points in common
- **ST\_CurveToLine** - Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. Converts a geometry containing curves to a linear geometry.
- **ST\_Disjoint** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have no points in common
- **ST\_Equals** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries include the same set of points
- **ST\_GeneratePoints** - Enhanced: 3.0.0, added seed parameter Generates a multipoint of random points contained in a Polygon or MultiPolygon.
- **ST\_GeomFromGeoJSON** - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. `ST_GeomFromGeoJSON(geojson)` PostGIS `ST_GeomFromGeoJSON(geojson, srid)`.
- **ST\_LocateBetween** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- **ST\_LocateBetweenElevations** - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_Overlaps** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other



- **ST\_Relate** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST\_Segmentize** - Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_Touches** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect
- **ST\_Within** - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 3.0

- **PostGIS\_Extensions\_Upgrade** - Changed: 3.0.0 to repackaging loose extensions and support postgis\_raster. Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.
- **ST\_3DDistance** - Changed: 3.0.0 - SFCGAL version removed, GEOS backend (SRS 3D) 3D longest distance.
- **ST\_3DIntersects** - Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- **ST\_Area** - Changed: 3.0.0 - does not depend on SFCGAL anymore.
- **ST\_AsGeoJSON** - Changed: 3.0.0 support records as input Return a geometry or feature in GeoJSON format.
- **ST\_AsGeoJSON** - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry or feature in GeoJSON format.
- **ST\_AsKML** - Changed: 3.0.0 - Removed the "versioned" variant signature GML 2 GML 3
- **ST\_Distance** - Changed: 3.0.0 - does not depend on SFCGAL anymore. 3D longest distance.
- **ST\_Intersection** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINs added. Tests if two geometries intersect (they have at least one point in common)
- **ST\_Union** - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

**13.12.7 PostGIS Functions new or enhanced in 2.5**

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- **PostGIS\_Extensions\_Upgrade** - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis\_raster, postgis\_topology, postgis\_sfcgal) to given or latest version.
- **ST\_Angle** - Availability: 2.5.0 3D longest distance.

- **ST\_ChaikinSmoothing** - Availability: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm
- **ST\_FilterByM** - Availability: 2.5.0 Removes vertices based on their M value
- **ST\_LineInterpolatePoints** - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- **ST\_OrientedEnvelope** - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- **ST\_QuantizeCoordinates** - Availability: 2.5.0 Sets least significant bits of coordinates to zero

Functions enhanced in PostGIS 2.5

- **ST\_AsMVT** - Enhanced: 2.5.0 - added support parallel query. Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsText** - Enhanced: 2.5 - optional parameter precision introduced. `WKT(Well-Known Text) SRID`.
- **ST\_Buffer** - Enhanced: 2.5.0 - `ST_Buffer` geometry support was enhanced to allow for side buffering specification `side=both|left|right`. Computes a geometry covering all points within a given distance from a geometry.
- **ST\_GeomFromGeoJSON** - Enhanced: 2.5.0 can now accept json and jsonb as inputs. GeoJSON
- **ST\_GeometricMedian** - Enhanced: 2.5.0 Added support for M as weight of points. (median)
- **ST\_Intersects** - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common)
- **ST\_OffsetCurve** - Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING Returns an offset line at a given distance and side from an input line.
- **ST\_Scale** - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- **ST\_Split** - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_Subdivide** - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

### 13.12.8 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- **ST\_AsGeobuf** - 2.2.0. Return a Geobuf representation of a set of rows.
- **ST\_AsMVT** - 2.2.0. Aggregate function returning a MVT representation of a set of rows.
- **ST\_AsMVTGeom** - 2.2.0. Transforms a geometry into the coordinate space of a MVT tile.
- **ST\_Centroid** - Availability: 2.4.0 support for geography was introduced.

- **ST\_ForcePolygonCCW** - 2.2.0. Orients all exterior rings counter-clockwise and all interior rings clockwise.
- **ST\_ForcePolygonCW** - 2.2.0. Orients all exterior rings clockwise and all interior rings counter-clockwise.
- **ST\_FrechetDistance** - Availability: 2.4.0 - requires GEOS >= 3.7.0 (shortest).
- **ST\_IsPolygonCCW** - 2.2.0. Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- **ST\_IsPolygonCW** - 2.2.0. Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

#### Functions enhanced in PostGIS 2.4

- **ST\_AsTWKB** - Enhanced: 2.4.0 memory and speed improvements. TWKB(Tiny Well-Known Binary).
- **ST\_Covers** - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- **ST\_CurveToLine** - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- **ST\_Project** - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_Reverse** - Enhanced: 2.4.0 support for curves was introduced.

#### Functions changed in PostGIS 2.4

- **=** - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use `ST_Equals` instead. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- **ST\_Node** - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

### 13.12.9 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.

#### Functions new in PostGIS 2.3

- **ST\_ContainsProperly(geometry,gidx)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- **ST\_ContainsProperly(gidx,geometry)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- **ST\_ContainsProperly(gidx,gidx)** - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

- **&&(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- **&&(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- **&&(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- **@(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- **@(box2df,geometry)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- **@(geometry,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- **ST\_ClusterDBSCAN** - 2.3.0 `ST_ClusterDBSCAN(geometry, float, boolean)`. Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- **ST\_ClusterKMeans** - 2.3.0 `ST_ClusterKMeans(geometry, float, boolean)`. Window function that returns a cluster id for each input geometry using the K-means algorithm.
- **ST\_GeneratePoints** - 2.3.0 `ST_GeneratePoints(geometry, integer)`. Generates a multipoint of random points contained in a Polygon or MultiPolygon.
- **ST\_GeometricMedian** - 2.3.0 `ST_GeometricMedian(geometry)`. `ST_GeometricMedian(geometry, float)` (median) `ST_GeometricMedian(geometry, float, float)`.
- **ST\_MakeLine** - 2.0.0 `ST_MakeLine(geometry, geometry)`. `ST_MakeLine(geometry, geometry, float)`. `ST_MakeLine(geometry, geometry, float, float)`.
- **ST\_MinimumBoundingRadius** - 2.3.0 `ST_MinimumBoundingRadius(geometry)`. Returns the center point and radius of the smallest circle that contains a geometry.
- **ST\_MinimumClearance** - 2.3.0 `ST_MinimumClearance(geometry)`. `ST_MinimumClearance(geometry, float)` (robustness) `ST_MinimumClearance(geometry, float, float)` (clearance) `ST_MinimumClearance(geometry, float, float, float)`.
- **ST\_MinimumClearanceLine** - 2.3.0 `ST_MinimumClearanceLine(geometry)`. GEOS 3.6.0 `ST_MinimumClearanceLine(geometry, float)`. `ST_MinimumClearanceLine(geometry, float, float)`.
- **ST\_Normalize** - 2.3.0 `ST_Normalize(geometry)`. `ST_Normalize(geometry, float)`.
- **ST\_Points** - 2.3.0 `ST_Points(geometry)`. `ST_Points(geometry, float)`.
- **ST\_VoronoiLines** - 2.3.0 `ST_VoronoiLines(geometry)`. Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- **ST\_VoronoiPolygons** - 2.3.0 `ST_VoronoiPolygons(geometry)`. Returns the cells of the Voronoi diagram of the vertices of a geometry.
- **ST\_WrapX** - Availability: 2.3.0 requires GEOS X `ST_WrapX(geometry, float)`.
- **~(box2df,box2df)** - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).

- `~(box2df,geometry)` - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- `~(geometry,box2df)` - Availability: 2.3.0 support for Block Range INdices (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

Functions enhanced in PostGIS 2.3

- `ST_Contains` - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A, and their interiors have a point in common
- `ST_Covers` - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A
- `ST_Expand` - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions. Returns a bounding box expanded from another bounding box or a geometry.
- `ST_Intersects` - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if two geometries intersect (they have at least one point in common)
- `ST_Segmentize` - Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- `ST_Transform` - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- `ST_Within` - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 2.3

- `ST_PointN` - `ST_LineString` `ST_CircularString`

### 13.12.10 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.2

- `<<->` - 2.2.0 PostgreSQL 9.1 KNN
- `ST_AsEncodedPolyline` - 2.2.0
- `ST_AsTWKB` - 2.2.0 TWKB(Tiny Well-Known Binary)
- `ST_BoundingDiagonal` - 2.2.0
- `ST_CPAWithin` - 2.2.0 Tests if the closest point of approach of two trajectories is within the specified distance.

- **ST\_ClipByBox2D** - 2.2.0. Computes the portion of a geometry falling within a rectangle.
- **ST\_ClosestPointOfApproach** - 2.2.0. Returns a measure at the closest point of approach of two trajectories.
- **ST\_ClusterIntersecting** - 2.2.0. Aggregate function that clusters input geometries into connected sets.
- **ST\_ClusterWithin** - 2.2.0. Aggregate function that clusters geometries by separation distance.
- **ST\_DistanceCPA** - 2.2.0. Returns the distance between the closest point of approach of two trajectories.
- **ST\_ForceCurve** - 2.2.0. (upcast).
- **ST\_IsValidTrajectory** - 2.2.0. Tests if the geometry is a valid trajectory.
- **ST\_LineFromEncodedPolyline** - 2.2.0. (polyline).
- **ST\_RemoveRepeatedPoints** - 2.2.0. Returns a version of a geometry with duplicate points removed.
- **ST\_SetEffectiveArea** - 2.2.0. Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- **ST\_SimplifyVW** - 2.2.0. Returns a simplified representation of a geometry, using the Visvalingam-Whyatt algorithm.
- **ST\_Subdivide** - 2.2.0. Computes a rectilinear subdivision of a geometry.
- **ST\_SwapOrdinates** - 2.2.0.
- **postgis.enable\_outdb\_rasters** - 2.2.0. DB.
- **postgis.gdal\_datapath** - 2.2.0. GDAL GDAL\_DATA.
- **postgis.gdal\_enabled\_drivers** - 2.2.0. PostGIS GDAL.
- **|=** - 2.2.0. PostgreSQL 9.5 (index-supported) A B (closest point of approach) (trajectory).

Functions enhanced in PostGIS 2.2

- **<->** - 2.2.0 -- PostgreSQL 9.5 KNN("K nearest neighbor"). PostgreSQL 9.4 A B 2.
- **ST Area** - 2.2.0 GeographicLib Proj 4.9.0.
- **ST AsX3D** - 2.2.0 (x/y, z/r) X3D XML ISO-IEC-19776-1.2-X3DEncodings-XML.

- **ST\_Azimuth** - **Enhanced**: 2.2.0 support for GeographicLib 2.2.0. Proj 4.9.0 support for 2D (longest).
- **ST\_Distance** - **Enhanced**: 2.2.0 support for GeographicLib 2.2.0. Proj 4.9.0 support for 3D (longest).
- **ST\_Scale** - **Enhanced**: 2.2.0 support for scaling all dimension (factor parameter) was introduced. Scales a geometry by given factors.
- **ST\_Split** - **Enhanced**: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- **ST\_Summary** - **Enhanced**: 2.2.0 support for TIN (curve) support.

Functions changed in PostGIS 2.2

- **<->** - **Enhanced**: 2.2.0 support for PostgreSQL 9.5 Hybrid syntax.
- **ST\_3DClosestPoint** - **Enhanced**: 2.2.0 support for 2D, (Z=0) 2D to 3D, g2 to g1 3D to 3D.
- **ST\_3DDistance** - **Enhanced**: 2.2.0 support for 2D to 3D, (Z=0) 2D to 3D, (SRS) 3D to 3D.
- **ST\_3DLongestLine** - **Enhanced**: 2.2.0 support for 2D, (Z=0) 2D to 3D, (SRS) 3D to 3D (longest).
- **ST\_3DMaxDistance** - **Enhanced**: 2.2.0 support for 2D to 3D, (Z=0) 2D to 3D, (SRS) 3D to 3D.
- **ST\_3DShortestLine** - **Enhanced**: 2.2.0 support for 2D, (Z=0) 2D to 3D, (SRS) 3D to 3D (shortest).
- **ST\_DistanceSphere** - **Enhanced**: 2.2.0 support for ST\_Distance\_Sphere.
- **ST\_DistanceSpheroid** - **Enhanced**: 2.2.0 support for ST\_Distance\_Spheroid.
- **ST\_Equals** - **Changed**: 2.2.0 Returns true even for invalid geometries if they are binary equal Tests if two geometries include the same set of points
- **ST\_LengthSpheroid** - **Enhanced**: 2.2.0 support for ST\_Length\_Spheroid, ST\_3DLength\_Spheroid.
- **ST\_MemSize** - **Changed**: 2.2.0 name changed to ST\_MemSize to follow naming convention. ST\_Geometry
- **ST\_PointInsideCircle** - **Changed**: 2.2.0 In prior versions this was called ST\_Point\_Inside\_Circle Tests if a point geometry is inside a circle defined by a center and radius



### 13.12.11 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.1

- **ST\_Box2dFromGeoHash** - 2.1.0. GeoHash BOX2D.
- **ST\_DelaunayTriangles** - 2.1.0. Returns the Delaunay triangulation of the vertices of a geometry.
- **ST\_GeomFromGeoHash** - 2.1.0. GeoHash.
- **ST\_PointFromGeoHash** - 2.1.0. GeoHash.
- **postgis.backend** - 2.1.0. GEOS SFCGAL. geos sfcgal, geos.

Functions enhanced in PostGIS 2.1

- **ST\_AsGML** - 2.1.0. GML 3 ID. GML 2 GML 3.
- **ST\_Boundary** - 2.1.0.
- **ST\_DWithin** - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance.
- **ST\_DWithin** - Enhanced: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance.
- **ST\_Distance** - 2.1.0. Making Geography faster. 3 (longest).
- **ST\_Distance** - 2.1.0. 3 (longest).
- **ST\_DumpPoints** - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C.
- **ST\_MakeValid** - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Segmentize** - 2.1.0. Returns a modified geometry/geography having no segment longer than a given distance.
- **ST\_Summary** - 2.1.0. S.

Functions changed in PostGIS 2.1

- **ST\_EstimatedExtent** - Changed: 2.1.0. Up to 2.0.x this was called ST\_Estimated\_Extent. Returns the estimated extent of a spatial table.
- **ST\_Force2D** - 2.1.0, 2.0.x ST\_Force\_2D. "2".
- **ST\_Force3D** - 2.1.0, 2.0.x ST\_Force\_3D. XYZ. ST\_Force3DZ.





- **ST\_3DMaxDistance** - 2.0.0. Returns the maximum distance between any two vertices in a 3D geometry. `geometry, geometry (SRS optional) 3D geometry`.
- **ST\_3DShortestLine** - 2.0.0. Returns the shortest line segment between two 3D geometries. `geometry, geometry 3D geometry (shortest) geometry`.
- **ST\_AsLatLonText** - 2.0. Returns a text representation of a geometry in latitude/longitude coordinates. `geometry, text, geometry`.
- **ST\_AsX3D** - 2.0.0. Returns an X3D XML representation of a geometry. `geometry ISO-IEC-19776-1.2-X3DEncodings-XML X3D XML: ISO-IEC-19776-1.2-X3DEncodings-XML`.
- **ST\_CollectionHomogenize** - 2.0.0. Returns the simplest representation of a geometry collection.
- **ST\_ConcaveHull** - 2.0.0. Computes a possibly concave geometry that contains all input geometry vertices.
- **ST\_FlipCoordinates** - 2.0.0. Returns a version of a geometry with X and Y axis flipped.
- **ST\_GeomFromGeoJSON** - 2.0.0. Returns a geometry from a GeoJSON object. `text JSON-C 0.9 GeoJSON PostGIS`.
- **ST\_InterpolatePoint** - 2.0.0. Returns a point interpolated along a line. `geometry, float, geometry (M) geometry`.
- **ST\_IsValidDetail** - 2.0.0. Returns a valid\_detail row stating if a geometry is valid or if not a reason and a location.
- **ST\_IsValidReason** - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- **ST\_MakeLine** - 2.0.0. Returns a line from a set of points. `geometry, geometry geometry`.
- **ST\_MakeValid** - 2.0.0. Attempts to make an invalid geometry valid without losing vertices.
- **ST\_Node** - 2.0.0. Nodes a collection of lines.
- **ST\_NumPatches** - 2.0.0. Returns the number of patches in a geometry. `geometry NULL geometry`.
- **ST\_OffsetCurve** - 2.0. Returns an offset line at a given distance and side from an input line.
- **ST\_PatchN** - 2.0.0. Returns the Nth patch of a geometry. `geometry ST_Geometry geometry`.
- **ST\_Perimeter** - Availability: 2.0.0. Returns the length of the boundary of a polygonal geometry or geography.
- **ST\_Project** - 2.0.0. Returns a point projected from a start point by a distance and bearing (azimuth).
- **ST\_RelateMatch** - 2.0.0. Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern.
- **ST\_SharedPaths** - 2.0.0. Returns the shared paths between two geometries. `geometry, geometry geometry/geometry geometry`.
- **ST\_Snap** - 2.0.0. Returns a geometry snapped to another geometry.
- **ST\_Split** - Availability: 2.0.0 requires GEOS. Returns a collection of geometries created by splitting a geometry by another geometry.

- **ST\_UnaryUnion** - 2.0.0. Computes the union of the components of a single geometry.

Functions enhanced in PostGIS 2.0

- **&&** - 2.0.0 (polyhedral surface). A 2D B 2D TRUE.
- **AddGeometryColumn** - 2.0.0. use typmod.
- **Box2D** - 2.0.0, TIN. Returns a BOX2D representing the 2D extent of a geometry.
- **Box3D** - 2.0.0, TIN. Returns a BOX3D representing the 3D extent of a geometry.
- **GeometryType** - 2.0.0, TIN. ST\_Geometry.
- **Populate\_Geometry\_Columns** - 2.0.0. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST\_3DExtent** - 2.0.0, TIN. Aggregate function that returns the 3D bounding box of geometries.
- **ST\_Affine** - 2.0.0, TIN. Apply a 3D affine transformation to a geometry.
- **ST\_Area** - 2.0.0 2 (polyhedral surface).
- **ST\_AsBinary** - 2.0.0, TIN. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsBinary** - 2.0.0. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsBinary** - 2.0.0. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsEWKB** - 2.0.0, TIN. Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- **ST\_AsEWKT** - 2.0.0, TIN. WKT(Well-Known Text) SRID.
- **ST\_AsGML** - 2.0.0. GML 3 '4' TIN. GML 2 GML 3.
- **ST\_AsKML** - 2.0.0. GML 2 GML 3.
- **ST\_Azimuth** - 2.0.0 2.
- **ST\_Dimension** - 2.0.0 (polyhedral surface) TIN.

- **ST\_Dump** - 函数: 2.0.0 返回 TIN 的几何分量。Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_DumpPoints** - 函数: 2.0.0 返回 TIN 的几何分量。Returns a set of geometry\_dump rows for the components of a geometry.
- **ST\_Expand** - 函数: 2.0.0 返回 TIN 的几何分量。Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_Extent** - 函数: 2.0.0 返回 TIN 的几何分量。Aggregate function that returns the bounding box of geometries.
- **ST\_Force2D** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 2D。强制为 2D 强制为 2D。
- **ST\_Force3D** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 3D。强制为 XYZ 强制为 XYZ。ST\_Force3DZ 强制为 XYZ。
- **ST\_Force3DZ** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 3D。强制为 XYZ 强制为 XYZ。
- **ST\_ForceCollection** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为集合。强制为集合强制为集合。
- **ST\_ForceRHR** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为右手规则。强制为右手规则 (orientation) 强制为右手规则 (Right-Hand Rule) 强制为右手规则。
- **ST\_GMLToSQL** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 SQL。GML 强制为 ST\_Geometry 强制为 ST\_Geometry。强制为 ST\_GeomFromGML 强制为 ST\_GeomFromGML。
- **ST\_GMLToSQL** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 SQL。GML 强制为 ST\_Geometry 强制为 ST\_Geometry。强制为 ST\_GeomFromGML 强制为 ST\_GeomFromGML。
- **ST\_GeomFromEWKB** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 EWKB。EWKB(Extended Well-Known Binary) 强制为 ST\_Geometry 强制为 ST\_Geometry。
- **ST\_GeomFromEWKT** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 EWKT。EWKT(Extended Well-Known Text) 强制为 ST\_Geometry 强制为 ST\_Geometry。
- **ST\_GeomFromGML** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 GML。强制为 GML 强制为 PostGIS 强制为 PostGIS。
- **ST\_GeomFromGML** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 GML。强制为 GML 强制为 PostGIS 强制为 PostGIS。
- **ST\_GeometryN** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 N。ST\_Geometry 强制为 ST\_Geometry。
- **ST\_GeometryType** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为类型。ST\_Geometry 强制为 ST\_Geometry。
- **ST\_IsClosed** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为是否闭合。LINESTRING 强制为 TRUE 强制为 TRUE。强制为 TRUE (是否闭合) 强制为 TRUE 强制为 TRUE。
- **ST\_MakeEnvelope** - 函数: 2.0 返回 SRID 强制为 envelope 强制为 envelope。强制为 SRID 强制为 SRS 强制为 SRS。
- **ST\_MakeValid** - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- **ST\_NPoints** - 函数: 2.0.0 返回 TIN 的几何分量 (polyhedral surface) 强制为 N 点。强制为 N 点 (强制为 N) 强制为 N 点。

- **ST\_NumGeometries** - Enhanced: 2.0.0 - added support for specifying boundary node rule. Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- **ST\_Rotate** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Rotates a geometry about an origin point.
- **ST\_RotateX** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Rotates a geometry about the X axis.
- **ST\_RotateY** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Rotates a geometry about the Y axis.
- **ST\_RotateZ** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Rotates a geometry about the Z axis.
- **ST\_Scale** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Scales a geometry by given factors.
- **ST\_ShiftLongitude** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- **ST\_Summary** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added.
- **ST\_Transform** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Return a new geometry with coordinates transformed to a different spatial reference system.

Functions changed in PostGIS 2.0

- **AddGeometryColumn** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`
- **AddGeometryColumn** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`
- **AddGeometryColumn** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`
- **DropGeometryColumn** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. `ALTER TABLE some_table DROP COLUMN geom;`
- **DropGeometryTable** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. `DROP TABLE geometry_columns;`
- **Populate Geometry Columns** - Enhanced: 2.0.0 - added support for specifying the origin of rotation were added. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.

- **ST\_3DExtent** - Changed: 2.0.0 In prior versions this used to be called ST\_Extent3D Aggregate function that returns the 3D bounding box of geometries.
- **ST\_3DLength** - `geometry`: 2.0.0 `geometry` ST\_Length3D `geometry`. `geometry`.
- **ST\_3DMakeBox** - Changed: 2.0.0 In prior versions this used to be called ST\_MakeBox3D Creates a BOX3D defined by two 3D point geometries.
- **ST\_3DPerimeter** - `geometry`: 2.0.0 `geometry` ST\_Perimeter3D `geometry`. `geometry`.
- **ST\_AsBinary** - `geometry`: 2.0.0 `geometry` ST\_AsBinary(`geometry`, `n`) `bytea`. ST\_AsBinary('POINT(1 2)') `bytea`, n st\_asbinary(unknown) is not unique error `bytea`. `geometry` ST\_AsBinary('POINT(1 2)::geometry'); `bytea`. `geometry`, legacy.sql `bytea`. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsGML** - `geometry`: 2.0.0 `geometry` (named arg) `bytea`. `geometry` GML 2 `geometry` GML 3 `bytea`.
- **ST\_AsGeoJSON** - `geometry`: 2.0.0 `geometry` (default arg) `bytea` (named arg) `bytea`. Return a geometry or feature in GeoJSON format.
- **ST\_AsSVG** - `geometry`: 2.0.0 `geometry` (default arg) `bytea` (named arg) `bytea`. Returns SVG path data for a geometry.
- **ST\_EndPoint** - `geometry`: 2.0.0 `geometry` ST\_EndPoint(`geometry`) `geometry`. PostGIS `geometry` ST\_EndPoint(`geometry`) `geometry`. 2.0.0 `geometry` ST\_EndPoint(`geometry`) `geometry` NULL `geometry`. `geometry`, `geometry` ST\_EndPoint(`geometry`) `geometry` 2.0 `geometry` NULL `geometry`. ST\_LineString `geometry` ST\_CircularString `geometry`.
- **ST\_GeomFromText** - `geometry`: PostGIS 2.0.0 `geometry` ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY') `geometry`. PostGIS 2.0.0 `geometry`, SQL/MM `geometry` ST\_GeomFromText('GEOMETRYCOLLECTION EMPTY') `geometry`. WKT `geometry` ST\_Geometry `geometry`.
- **ST\_GeometryN** - `geometry`: 2.0.0 `geometry` ST\_GeometryN(`geometry`, `n`) `geometry`. 2.0.0 `geometry` ST\_GeometryN(`geometry`, `n`) `geometry`. ST\_Geometry `geometry`.
- **ST\_IsEmpty** - `geometry`: PostGIS 2.0.0 `geometry` ST\_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') `boolean`. PostGIS 2.0.0 `geometry`, SQL/MM `geometry` ST\_IsEmpty(`geometry`) `boolean`. Tests if a geometry is empty.
- **ST\_Length** - `geometry`: 2.0.0 `geometry` ST\_Length(`geometry`) `float`. 2.0.0 `geometry` ST\_Length(`geometry`) `float` / `float` ST\_Length(`geometry`) `float` / `float`. 2.0.0 `geometry` ST\_Length(`geometry`) `float` / `float` 0 `float`. `geometry` ST\_Perimeter `float`. `geometry`.
- **ST\_LocateAlong** - `geometry`: 2.0.0 `geometry` ST\_Locate Along\_Measure `geometry`. `geometry`, `float`. Returns the point(s) on a geometry that match a measure value.
- **ST\_LocateBetween** - `geometry`: 2.0.0 `geometry` ST\_Locate Along\_Measure `geometry`. `geometry`, `float`, `float`. Returns the portions of a geometry that match a measure range.
- **ST\_NumGeometries** - `geometry`: 2.0.0 `geometry` ST\_NumGeometries(`geometry`) `integer` NULL `integer`. 2.0.0 `geometry`, `integer`, `integer` ST\_NumGeometries(`geometry`) `integer` 1 `integer`. `geometry`.

- **ST\_NumInteriorRings** - 2.0.0: Returns the number of interior rings of a polygon. Returns NULL for non-polygonal geometries.
- **ST\_PointN** - 2.0.0: Returns the Nth point of a LineString. Returns NULL for non-LineString geometries. ST\_LineString and ST\_CircularString are supported.
- **ST\_StartPoint** - 2.0.0: Returns the first point of a LineString. Returns NULL for non-LineString geometries.

### 13.12.13 PostGIS Functions new or enhanced in 1.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.5

- **&&** - 1.5.0: Returns TRUE if two 2D geometries are disjoint.
- **PostGIS\_LibXML\_Version** - 1.5: Returns the version number of the libxml2 library.
- **ST\_AddMeasure** - 1.5.0: Interpolates measures along a linear geometry.
- **ST\_AsBinary** - 1.5.0: Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- **ST\_AsGML** - 1.5.0: Returns GML 2 or GML 3 representation.
- **ST\_AsGeoJSON** - 1.5.0: Return a geometry or feature in GeoJSON format.
- **ST\_AsText** - 1.5.0: Returns WKT(Well-Known Text) with SRID.
- **ST\_Buffer** - Availability: 1.5 - ST\_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.
- **ST\_ClosestPoint** - 1.5.0: Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- **ST\_CollectionExtract** - 1.5.0: Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
- **ST\_Covers** - 1.5.0: Tests if every point of B lies in A
- **ST\_DFullyWithin** - 1.5.0: Tests if a geometry is entirely inside a distance of another
- **ST\_DWithin** - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
- **ST\_Distance** - 1.5.0: Returns the shortest distance between two geometries. Supports 3 types (longest).
- **ST\_DistanceSphere** - 1.5: Returns the shortest distance between two geometries on a sphere. PostGIS 1.5 support for geography was introduced.



- **ST\_DistanceSpheroid** - 1.5.0. Returns the distance between two geometries in spheroid coordinates. 1.5.0. Returns the distance between two geometries in spheroid coordinates. PostGIS 1.5.
- **ST\_DumpPoints** - 1.5.0. Returns the points of a geometry.
- **ST\_Envelope** - 1.5.0. Returns the bounding box of a geometry. float4 (single precision; float8) (double precision; float8).
- **ST\_Expand** - Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates. Returns a bounding box expanded from another bounding box or a geometry.
- **ST\_GMLToSQL** - 1.5.0. LibXML2 1.6. GML ST\_Geometry ST\_GeomFromGML.
- **ST\_GeomFromGML** - 1.5.0. LibXML2 1.6. GML PostGIS.
- **ST\_GeomFromKML** - Availability: 1.5, requires libxml2 2.6+ KML PostGIS.
- **ST\_HausdorffDistance** - 1.5.0. Returns the maximum distance between any point on the shortest path between two geometries.
- **ST\_Intersection** - Availability: 1.5 support for geography data type was introduced. Computes a geometry representing the shared portion of geometries A and B.
- **ST\_Intersects** - Availability: 1.5 support for geography was introduced. Tests if two geometries intersect (they have at least one point in common).
- **ST\_Length** - 1.5.0. Returns the length of a geometry.
- **ST\_LongestLine** - 1.5.0. Returns the longest line segment between two geometries.
- **ST\_MakeEnvelope** - 1.5.0. Returns the bounding box of a geometry. SRID SRS.
- **ST\_MaxDistance** - 1.5.0. Returns the maximum distance between any point on the shortest path between two geometries.
- **ST\_ShortestLine** - 1.5.0. Returns the shortest line segment between two geometries.
- **~=** - 1.5.0. A B TRUE.

### 13.12.14 PostGIS Functions new or enhanced in 1.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.4

- **Populate\_Geometry\_Columns** - 1.4.0. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- **ST\_Collect** - 1.4.0. ST\_MakeLine ST\_MakeLine. Creates a GeometryCollection or Multi\* geometry from a set of geometries.
- **ST\_ContainsProperly** - 1.4.0. Tests if every point of B lies in the interior of A.
- **ST\_GeoHash** - 1.4.0. GeoHash.
- **ST\_IsValidReason** - Availability: 1.4 Returns text stating if a geometry is valid, or a reason for invalidity.



- **ST\_LineCrossingDirection** - Availability: 1.4 Returns a number indicating the crossing behavior of two LineStrings
- **ST\_LocateBetweenElevations** - 1.4.0. Returns the portions of a geometry that lie in an elevation (Z) range.
- **ST\_MakeLine** - 1.4.0. `ST_MakeLine` returns a line from a set of points. `ST_MakeLine` also takes a `geomarray` as input.
- **ST\_MinimumBoundingCircle** - 1.4.0. Returns the smallest circle polygon that contains a geometry.
- **ST\_Union** - Availability: 1.4.0 - `ST_Union` was enhanced. `ST_Union(geomarray)` was introduced and also faster aggregate collection in PostgreSQL. Computes a geometry representing the point-set union of the input geometries.

### 13.12.15 PostGIS Functions new or enhanced in 1.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.3

- **ST\_AsGML** - 1.3.2. Returns a geometry in GML 2 or GML 3 format.
- **ST\_AsGeoJSON** - 1.3.4. Return a geometry or feature in GeoJSON format.
- **ST\_CurveToLine** - Availability: 1.3.0 Converts a geometry containing curves to a linear geometry.
- **ST\_LineToCurve** - Availability: 1.3.0 Converts a linear geometry to a curved geometry.
- **ST\_SimplifyPreserveTopology** - 1.3.3. Returns a simplified and valid representation of a geometry, using the Douglas-Peucker algorithm.

## Chapter 14

# Reporting Problems

### 14.1 Reporting Software Bugs

Reporting bugs effectively is a fundamental way to help PostGIS development. The most effective bug report is that enabling PostGIS developers to reproduce it, so it would ideally contain a script triggering it and every information regarding the environment in which it was detected. Good enough info can be extracted running `SELECT postgis_full_version()` [for PostGIS] and `SELECT version()` [for postgresql].

If you aren't using the latest release, it's worth taking a look at its [release changelog](#) first, to find out if your bug has already been fixed.

Using the [PostGIS bug tracker](#) will ensure your reports are not discarded, and will keep you informed on its handling process. Before reporting a new bug please query the database to see if it is a known one, and if it is please add any new information you have about it.

You might want to read Simon Tatham's paper about [How to Report Bugs Effectively](#) before filing a new report.

### 14.2 Reporting Documentation Issues

The documentation should accurately reflect the features and behavior of the software. If it doesn't, it could be because of a software bug or because the documentation is in error or deficient.

Documentation issues can also be reported to the [PostGIS bug tracker](#).

If your revision is trivial, just describe it in a new bug tracker issue, being specific about its location in the documentation.

If your changes are more extensive, a patch is definitely preferred. This is a four step process on Unix (assuming you already have [git](#) installed):

1. Clone the PostGIS' git repository. On Unix, type:

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git
```

This will be stored in the directory `postgis`

2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):

```
vim doc/postgis.xml
```

Note that the documentation is written in DocBook XML rather than HTML, so if you are not familiar with it please follow the example of the rest of the documentation.

---

3. Make a patch file containing the differences from the master copy of the documentation. On Unix, type:

```
git diff doc/postgis.xml > doc.patch
```

4. Attach the patch to a new issue in bug tracker.
-

# Appendix A

## Appendix

### A.1 PostGIS 3.4.0

2023/08/15

This version requires PostgreSQL 12-16, GEOS 3.6 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL features, SFCGAL 1.4.1+ is needed.

NOTE: GEOS 3.12.0 details at [GEOS 3.12.0 release notes](#)

Many thanks to our translation teams, in particular:

Teramoto Ikuhiro (Japanese Team)

Vincent Bre (French Team)

There are 2 new ./configure switches:

- `--disable-extension-upgrades-install`, will skip installing all the extension upgrade scripts except for the ANY-currentversion. If you use this, you can install select upgrades using the postgis commandline tool
- `--without-pgconfig`, will build just the commandline tools raster2pgsql and shp2pgsql even if PostgreSQL is not installed

#### A.1.1 New features

[5055](#), complete manual internationalization (Sandro Santilli)

[5052](#), target version support in postgis\_extensions\_upgrade (Sandro Santilli)

[5306](#), expose version of GEOS at compile time (Sandro Santilli)

New install-extension-upgrades command in postgis script (Sandro Santilli)

[5257](#), [5261](#), [5277](#), Support changes for PostgreSQL 16 (Regina Obe)

[5006](#), [705](#), ST\_Transform: Support PROJ pipelines (Robert Coup, Koordinates)

[5283](#), [postgis\_topology] RenameTopology (Sandro Santilli)

[5286](#), [postgis\_topology] RenameTopoGeometryColumn (Sandro Santilli)

[703](#), [postgis\_raster] Add min/max resampling as options (Christian Schroeder)

[5336](#), [postgis\_topology] topogeometry cast to topoelement support (Regina Obe)

---

Allow singleton geometry to be inserted into Geometry(Multi\*) columns (Paul Ramsey)

721, New window-based ST\_ClusterWithinWin and ST\_ClusterIntersectingWin (Paul Ramsey)

5397, [address\_standardizer] debug\_standardize\_address function (Regina Obe)

5373 ST\_LargestEmptyCircle, exposes extra semantics on circle finding. Geos 3.9+ required (Martin Davis)

5267, ST\_Project signature for geometry, and two-point signature (Paul Ramsey)

5267, ST\_LineExtend for extending linestrings (Paul Ramsey)

New coverage functions ST\_CoverageInvalidEdges, ST\_CoverageSimplify, ST\_CoverageUnion (Paul Ramsey)

### A.1.2 Enhancements

5194, do not update system catalogs from postgis\_extensions\_upgrade (Sandro Santilli)

5092, reduce number of upgrade paths installed on system (Sandro Santilli)

635, honour --bindir (and --prefix) configure switch for executables (Sandro Santilli)

Honour --mandir (and --prefix) configure switch for man pages install path (Sandro Santilli)

Honour --htmldir (and --docdir and --prefix) configure switch for html pages install path (Sandro Santilli)

5447 Manual pages added for postgis and postgis\_restore utilities (Sandro Santilli)

[postgis\_topology] Speed up check of topology faces without edges (Sandro Santilli)

[postgis\_topology] Speed up coincident nodes check in topology validation (Sandro Santilli)

718, ST\_QuantizeCoordinates(): speed-up implementation (Even Rouault)

Repair spatial planner stats to use computed selectivity for contains/within queries (Paul Ramsey)

734, Additional metadata on Proj installation in postgis\_proj\_version (Paul Ramsey)

5177, Allow building tools without PostgreSQL server headers. Respect prefix/bin for tools install (Sandro Santilli)

ST\_Project signature for geometry, and two-point signature (Paul Ramsey)

4913, ST\_AsSVG support for curve types CircularString, CompoundCurve, MultiCurve, and MultiSurface (Regina Obe)

5266, ST\_ClosestPoint, ST\_ShortestLine, ST\_LineSubString support for geography type (MobilityDB Esteban Zimanyi, Maxime Schoemans, Paul Ramsey)

### A.1.3 Breaking Changes

5229, Drop support for Proj < 6.1 and PG 11 (Regina Obe)

5306, 734, postgis\_full\_version() and postgis\_proj\_version() now output more information about proj network configuration and data paths. GEOS compile-time version also shown if different from runtime (Paul Ramsey, Sandro Santilli)

5447, postgis\_restore.pl renamed to postgis\_restore (Sandro Santilli)

Utilities now installed in OS bin or user specified --bindir and --prefix instead of postgresql bin and extension stripped except on windows (postgis, postgis\_restore, shp2pgsql, raster2pgsql, pgsql2shp, pgtopo\_import, pgtopo\_export)