1 3D Geometric Types

3D geometric data types represent three-dimensional spatial objects. Table 1 shows the 3D geometric types available in PostgreSQL.

Table 1: 3D Geometric Types

Name	Storage Size	Description	Representation
point3d	24 bytes	Point on 3D space	(x,y,z)
lseg3d	48 bytes	Finite 3D line segment	((x1,y1,z1),(x2,y2,z2))
line3d	48 bytes	Infinite 3D line	((x1,y1,z1),(x2,y2,z2))
box3d	48 bytes	Rectangular 3D box	((x1,y1,z1),(x2,y2,z2))
path3d	64+24n bytes	Closed 3D path	((x1,y1,z1),...)
path3d	64+24n bytes	Open 3D path	[(x1,y1,z1),...]
polygon3d	56+24n bytes	3D polygon	((x1,y1,z1),...)
sphere	32 bytes	3D sphere	<(x,y,z),r>

A rich set of functions and operators is available to perform various geometric operations such as scaling, translation, and determining intersections. They are explained in Section 2.

1.1 3D Points

3D points are the fundamental three-dimensional building block for 3D geometric types. Values of type `point3d` are specified using either of the following syntaxes:

```
( x , y , z )
  x , y , z
```

where `x`, `y`, and `z` are the respective coordinates, as floating-point numbers. 3D points are output using the first syntax.

1.2 3D Line Segments

3D line segments are represented by pairs of 3D points that are the endpoints of the segment. Values of type `lseg3d` are specified using any of the following syntaxes:

```
[ ( x1 , y1 , z1 ) , ( x2 , y2 , z2 ) ]
( ( x1 , y1 , z1 ) , ( x2 , y2 , z2 ) )
  [ x1 , y1 , z1   ,   x2 , y2 , z2 ]
  ( x1 , y1 , z1   ,   x2 , y2 , z2 )
  ( x1 , y1 , z1 ) , ( x2 , y2 , z2 )
    x1 , y1 , z1   ,   x2 , y2 , z2 &
```

where `(x1,y1,z1)` and `(x2,y2,z2)` are the end points of the line segment. 3D line segments are output using the first syntax.

The two end points of a 3D line segment may be equal, in which case it represents a 3D point.

1.3 3D Lines

3D lines are specified by two of its points `(x1,y1,z1)` and `(x2,y2,z2)`, which must not be equal. The direction vector of this line is `(a,b,c) = (x2-x1,y2-y1,z2-z1)` and the parametric equations of the line are as follows

```
x = x1 + at
y = y1 + bt
z = z1 + ct
```

where `t` is any real number from $-\infty$ to ∞. Notice that the above equations also applies for 3D line segments, but in that case `t` is any real number in $[0, 1]$.

Values of type `line3D` are input and output in the following form:

```
[ ( x1 , y1 , z1 ) , ( x2 , y2 , z2 ) ]
( ( x1 , y1 , z1 ) , ( x2 , y2 , z2 ) )
  [ x1 , y1 , z1   ,   x2 , y2 , z2 ]
  ( x1 , y1 , z1   ,   x2 , y2 , z2 )
  ( x1 , y1 , z1 ) , ( x2 , y2 , z2 )
    x1 , y1 , z1   ,   x2 , y2 , z2 &
```

3D lines are output using the first syntax.

1.4 3D Boxes

3D boxes are represented by pairs of points that are opposite corners of the box. Values of type `box3d` are specified using any of the following syntaxes:

```
( ( x1 , y1 , z1 ) , ( x2 , y2 , z2 ) )
  ( x1 , y1 , z1 ) , ( x2 , y2 , z2 )
    x1 , y1 , z1   ,   x2 , y2 , z2
```

where `(x1,y1,z1)` and `(x2,y2,z2)` are any two opposite corners of the box. Boxes are output using the second syntax. Any two opposite corners can be supplied on input, but the values will be reordered as needed to store the upper right and lower left corners, in that order.

The two end points of a 3D box may be equal, in which case it represents a 3D point. The two end points of a 3D box may have be equal in one of the axes (`x`, `y`, or `z`), in which case it represents a 2D rectangle perpendicular to the axis with equal values.

1.5 3D Paths

3D paths are represented by lists of connected points. Paths can be open, where the first and last points in the list are considered not connected, or closed, where the first and last points are considered connected. Connected paths are "hollow", that is, they do not contain their interior.

Values of type `path3d` are specified using any of the following syntaxes:

```
[ ( x1 , y1 , z1 ) , ... , ( xn , yn , zn ) ]
( ( x1 , y1 , z1 ) , ... , ( xn , yn , zn ) )
  ( x1 , y1 , z1 ) , ... , ( xn , yn , zn )
  ( x1 , y1 , z1   , ... ,   xn , yn , zn )
    x1 , y1 , z1   , ... ,   xn , yn , zn
```

where the points are the end points of the line segments comprising the path. Square brackets (`[]`) indicate an open path, while parentheses (`()`) indicate a closed path. When the outermost parentheses are omitted, as in the third through fifth syntaxes, a closed path is assumed. Paths are output using the first or second syntax, as appropriate.

3d paths must contain at least one point. Furthermore, closed 3D paths must contain at least 3 non collinear points.

1.6 3D Polygons

3D polygons are represented by lists of points (the vertexes of the polygon). Polygons are similar to closed paths, but unlike them, they include their interior.

Values of type `polygon3d` are specified using any of the following syntaxes:

```
( ( x1 , y1 , z1 ) , ... , ( xn , yn , zn ) )
  ( x1 , y1 , z1 ) , ... , ( xn , yn , zn )
  ( x1 , y1 , z1   , ... ,   xn , yn , zn )
    x1 , y1 , z1   , ... ,   xn , yn , zn
```

where the points are the end points of the line segments comprising the boundary of the polygon. Polygons are output using the first syntax.

As for closed 3D paths, 3D polygons must contain at least 3 non collinear points. 3D polygons must be planar so that are "well behaved" with respect to their operations, such as determining whether a point is contained in the polygon. Although non planar polygons are valid values of the type `polygon3D`, many of the operations return a null value for non planar polygons.

1.7 Spheres

Spheres are represented by a center point and radius. Values of type `sphere` are specified using any of the following syntaxes:

```
< ( x , y, z ) , r >
( ( x , y, z ) , r )
  ( x , y, z ) , r
    x , y, z   , r
```

where `(x,y,z)` is the center point and `r` is the radius of the sphere. Circles are output using the first syntax.

2 3D Geometric Functions and Operators

The geometric types `point3d`, `box3d`, `lseg3d`, `line3d`, `path3d`, `polygon3d`, and `sphere` have a large set of native support functions and operators, shown in Table 2, Table 3, and Table 4.

The "same as" operator, `~=`, represents the usual notion of equality for 3D geometries. In the case of `point3d`, `box3d`, and `sphere` types the operator test whether the values are identical, while for the other types the operator tests whether the values represent the same set of points in 3D space. For example, in the case of the `line3d` type the operator tests whether the two lines coincide, while for the `path3d` type the operator tests whether the two paths have the same set of vertices (but not necessarily in the same order). These types also have an `=` operator that test identical values. The other scalar comparison operators (`<=` and so on) are rather arbitrary and they are used internally for indexing purposes, they are not useful in the real world.

It is possible to access the three component numbers of a `point3d` as though the point were an array with indexes 0, 1, and 2. For example, if `t.p` is a `point` column then `SELECT p[0] FROM t` retrieves the `x` coordinate and `UPDATE t SET p[1] = ...` changes the `y` coordinate. In the same way, a value of type `lseg3d`, `line3d` or `box3d` can be treated as an array of two point values.

The `area` function works for the types `box3d`, `path3d`, `polygon3d`, and `sphere`. In the case of the `path3d` data type, the `area` function only works if the path is closed, is planar, and is not self-crossing. For example, the function applied to the path

```
'((0,0,0),(0,1,1),(2,1,1),(2,2,2),(1,2,2),(1,0,0))'::path3d
```

will not work. However, the following the function applied to the visually identical path

```
'((0,0,0),(0,1,1),(1,1,1),(1,2,2),(2,2,2),(2,1,1),(1,1,1),(1,0,0),(0,0,0))'::path3d
```

will work. Similarly, the `area` function only works if the polygon is planar and is not self-crossing.

The `box3d` and `sphere` functions computing the bounding box and bounding sphere work for all types excepted `line3d`. When applied to a `point3d` they return an empty box or sphere representing the point.

The `center` function and the corresponding operator `@@` work for all types excepted `point3d`.

The `coplanar` function works for various combinations of types `point3d`, `lseg3d`, `line3d`, `path3d`, and `polygon3d` that involve more than three points. For example, it works for `point3d` and `polygon3d` but neither for `point3d` and `lseg3d` (since it is trivially true) nor for `point3d` and `point3d` (since it does not make sense).

The `length` function works for the types `lseg3d`, `path3d`, and `polygon3d`.

The function `points` work for all types excepted `point3d`.

The `point3d` functions computing the center work for all types excepted `line3d`.

3 Indexing

GiST and SP-GiST indexes can be created for table columns of some of the 3D geometry types. The GiST index implements an R-tree for the types `point3d`, `box3d`, and `sphere`. The SP-GiST index implements an Oct-tree for the type `point3d`. An example of creation of a GiST and an SP-GiST indexes is as follows:

```
CREATE TABLE geo3d_tbl(k int, p point3d, b box3d, s sphere);
CREATE INDEX geo3d_tbl_idx_p ON geo3d_tbl USING spgist (p);
CREATE INDEX geo3d_tbl_idx_b ON geo3d_tbl USING gist (b);
```

A GiST or SP-GiST index can accelerate queries involving the following operators: `<<`, `&<`, `&&`, `&>`, `>>`, `~=`, `@>`, `<@`, `&<|`, `<<|`, `|>>`, `|&>`, `<->`, `&</`, `<</`, `/>>`, and `/&>` (see Table 2 for more information).

In addition, B-tree indexes can be created for table columns of all 3D geometries. For this index type, basically the only useful operation is equality. There is a B-tree sort ordering defined for 3D geometry values, with corresponding `<` and `>` operators, but the ordering is rather arbitrary and not usually useful in the real world. B-tree support for 3D geometries is primarily meant to allow sorting internally in queries, rather than creation of actual indexes.

Table 2: 3D Geometric Operators

<table>
<tr><th>Operator</th><th>Description</th><th>Example</th></tr>
<tr><td>+</td><td>Translation</td><td>box3d '((0,0,0),(1,1,1))' + point3d '(2.0,0,0)'</td></tr>
<tr><td>-</td><td>Translation</td><td>box3d '((0,0,0),(1,1,1))' - point3d '(2.0,0,0)'</td></tr>
<tr><td>*</td><td>Scaling</td><td>box3d '((0,0,0),(1,1,1))' * 2.0</td></tr>
<tr><td>/</td><td>Scaling</td><td>box3d '((0,0,0),(2,2,2))' / 2.0</td></tr>
<tr><td>+</td><td>Concatenation</td><td>path3d '[(0,0,0),(1,1,1)]' + path3d '[(2,0,0),(2,2,2)]'</td></tr>
<tr><td>#</td><td>Point or box of intersection</td><td>box3d '((1,-1,0),(-1,1,1))' # box3d '((1,1,1),(-2,-2,-2))'</td></tr>
<tr><td>#</td><td>Number of points in path or polygon</td><td># path3d '((1,0,0),(0,1,1),(-1,0,2))'</td></tr>
<tr><td>@-@</td><td>Length of segment, path, or polygon</td><td>@-@ path3d '((0,0,0),(1,0,0))'</td></tr>
<tr><td>@@</td><td>Center</td><td>@@ sphere '((0,0,0),10)'</td></tr>
<tr><td>##</td><td>Closest point to first operand on second operand</td><td>point3d '(0,0,0)' ## lseg3d '((2,0,0),(0,2,1))'</td></tr>
<tr><td><-></td><td>Distance between</td><td>sphere '((0,0,0),1)' <-> sphere '((5,0,1),1)'</td></tr>
<tr><td>&&</td><td>Overlaps? (One point in common makes this true)</td><td>box3d '((0,0,0),(1,1,1))' && box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td><<</td><td>Is strictly left of?</td><td>sphere '((0,0,0),1)' << sphere '((5,0,0),1)'</td></tr>
<tr><td>>></td><td>Is strictly right of?</td><td>sphere '((5,0,0),1)' >> sphere '((0,0,0),1)'</td></tr>
<tr><td>&<</td><td>Does not extend to the right of?</td><td>box3d '((0,0,0),(1,1,1))' &< box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td>&></td><td>Does not extend to the left of?</td><td>box3d '((0,0,0),(3,3,3))' &> box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td><<|</td><td>Is strictly below?</td><td>box3d '((0,0,0),(3,3,3))' <<| box3d '((3,4,4),(5,5,5))'</td></tr>
<tr><td>|>></td><td>Is strictly above?</td><td>box3d '((3,4,4),(5,5,5))' |>> box3d '((0,0,0),(3,3,3))'</td></tr>
<tr><td>&<|</td><td>Does not extend above?</td><td>box3d '((0,0,0),(1,1,1))' &<| box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td>|&></td><td>Does not extend below?</td><td>box3d '((0,0,0),(3,3,3))' |&> box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td><</</td><td>Is strictly in front?</td><td>box3d '((0,0,0),(3,3,3))' <<| box3d '((3,4,4),(5,5,5))'</td></tr>
<tr><td>/>></td><td>Is strictly behind?</td><td>box3d '((3,4,4),(5,5,5))' |>> box3d '((0,0,0),(3,3,3))'</td></tr>
<tr><td>&</</td><td>Does not extend in front?</td><td>box3d '((0,0,0),(1,1,1))' &<| box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td>/&></td><td>Does not extend below?</td><td>box3d '((0,0,0),(3,3,3))' |&> box3d '((0,0,0),(2,2,2))'</td></tr>
<tr><td>?#</td><td>Intersects?</td><td>lseg3d '((-1,0),(1,0))' ?# box3d '((-2,-2),(2,2))'</td></tr>
<tr><td>?-</td><td>Is horizontal?</td><td>?- lseg3d '((-1,0,0),(1,0,0))'</td></tr>
<tr><td>?-</td><td>Are horizontally aligned?</td><td>point3d '(1,0,0)' ?- point3d '(0,0,0)'</td></tr>
<tr><td>?|</td><td>Is vertical?</td><td>?| lseg3d '((-1,0,0),(1,0,1))'</td></tr>
<tr><td>?|</td><td>Are vertically aligned?</td><td>point3d '(0,1)' ?| point3d '(0,0,0)'</td></tr>
<tr><td>?/</td><td>Is perpendicular?</td><td>?/ lseg3d '((-1,0,1),(1,0,1))'</td></tr>
<tr><td>?/</td><td>Are perpendicularly aligned?</td><td>point3d '(0,1,1)' ?/ point3d '(0,0,1)'</td></tr>
<tr><td>?-|</td><td>Are orthogonal?</td><td>lseg3d '((0,0,0),(0,1,1))' ?-| lseg3d '((0,0,0),(1,0,1))'</td></tr>
<tr><td>?||</td><td>Are parallel?</td><td>lseg3d '((-1,0,0),(1,0,1))' ?|| lseg3d '((-1,2,2),(1,2,3))'</td></tr>
<tr><td>?-/</td><td>Are skew?</td><td>lseg3d '((-1,0,0),(1,0,1))' ?-/ lseg3d '((-1,2,2),(1,2,3))'</td></tr>
<tr><td>@></td><td>Contains?</td><td>sphere '((0,0,0),2)' @> point3d '(1,1,1)'</td></tr>
<tr><td><@</td><td>Contained in or on?</td><td>point3d '(1,1,1)' <@ sphere '((0,0,0),2)'</td></tr>
</table>

Continued on next page

Table 2 3D Geometric Types – continued from previous page

Operator	Description	Example
`~=`	Same as?	`polygon3D '((0,0,0),(1,1,1))' ~= polygon3D '((1,1,1),(0,0,0))'`

Table 3: 3D Geometric Functions

Function	Return Type	Description	Example
`area(object3d)`	`double precision`	Area of object	`area(sphere '((0,0,0),1)')`
`center(object3d)`	`point3d`	Center of object	`center(box3d '((0,0,0),(2,2,2))')`
`coincide(line3d, line3d)`	`boolean`	Are the same line?	`coincide(line3d '((0,0,0),(1,1,1))', line3d '((2,2,2),(3,3,3))')`
`collinear(point3d, point3d, point3d)`	`boolean`	Are the points collinear?	`collinear(point3d '0,0,0', point3d '1,1,1', point3d '2,2,2')`
`coplanar(object3d, object3d)`	`boolean`	Are the objects coplanar?	`coplanar(point3d '1,1,1', polygon3d '0,0,0,2,2,2,2,0,0')`
`depth(box3d)`	`double precision`	Perpendicular size of box	`depth(box3d '((0,0,0),(1,2,3))')`
`diameter(sphere)`	`double precision`	Diameter of sphere	`diameter(sphere '((0,0,0),2.0)')`
`height(box3d)`	`double precision`	Vertical size of box	`height(box3d '((0,0,0),(1,2,3))')`
`high(box3d)`	`point3d`	High corner of box	`high(box3d '((0,0,0),(1,2,3))')`
`horizontal(point3d, point3d)`	`boolean`	Are the y values equal?	`horizontal(point3d '(0,0,0)', point3d '(1,0,1)')`
`horizontal(lseg3d)`	`boolean`	Are the y values equal?	`horizontal(lseg3d '((0,0,0),(1,0,1))')`
`horizontal(line3d)`	`boolean`	Are the y values equal?	`horizontal(line3d '((0,0,0),(1,0,1))')`
`isclosed(path3d)`	`boolean`	Is the path closed?	`isclosed(path3d '((0,0,0),(1,1,1),(2,0))')`
`isopen(path3d)`	`boolean`	Is the path open?	`isopen(path3d '[(0,0,0),(1,1,1),(2,0,0)]')`
`isplanar(path3d)`	`boolean`	Is the path planar?	`isplanar(path3d '[(0,0,0),(1,1,1),(2,0,0)]')`
`isplanar(polygon3d)`	`boolean`	Is the polygon planar?	`isplanar(polygon3d '((0,0,0),(0,1,1),(1,1,1),(1,0,0))')`
`length(object3d)`	`double precision`	Length of object	`length(path3d '((-1,0,0),(1,0,1))')`
`low(box3d)`	`point3d`	Low corner of box	`low(box3d '((0,0,0),(1,2,3))')`
`npoints(path3d)`	`int`	Number of points of path	`npoints(path3d '[(0,0,0),(1,1,1),(2,0,0)]')`
`npoints(polygon3d)`	`int`	Number of points of polygon	`npoints(polygon3d '((0,0,0),(0,1,1),(1,1,1),(1,0,0))')`
`orthogonal(lseg3d, lseg3d)`	`boolean`	Are orthogonal?	`orthogonal(lseg3d '((0,0,0),(1,1,1))', lseg3d '((0,1,1),(1,0,1))')`
`orthogonal(line3d, line3d)`	`boolean`	Are orthogonal?	`orthogonal(line3d '((0,0,0),(1,1,1))', line3d '((0,1,1),(1,0,1))')`
`parallel(lseg3d, lseg3d)`	`boolean`	Are orthogonal?	`parallel(lseg3d '((0,0,0),(1,1,1))', lseg3d '((2,1,1),(3,2,2))')`

Continued on next page

Table 3 3D Geometric Functions – continued from previous page

Function	Return Type	Description	Example
parallel(line3d, line3d)	boolean	Are orthogonal?	parallel(line3d '((0,0,0),(1,1,1))', line3d '((0,1,1),(1,0,1))')
pclose(path3d)	path3d	Convert path to closed	pclose(path3d '[(0,0,1),(1,1,1),(2,0,0)]')
perpendicular(point3d, point3d)	boolean	Are the z values equal?	perpendicular(point3d '(0,0,0)', point3d '(1,1,0)')
perpendicular(lseg3d)	boolean	Are the z values equal?	perpendicular(lseg3d '((0,0,0),(1,1,0))')
perpendicular(line3d)	boolean	Are the z values equal?	perpendicular(line3d '((0,0,0),(1,1,0))')
points(object3d)	point3D[]	Points of an object	points(polygon3d '((0,0,0),(0,1,1),(1,1,1),(1,0,0))')
popen(path3d)	path3d	Convert path to open	popen(path3d '((0,0,0),(1,1,1),(2,0,0))')
radius(sphere)	double precision	Radius of sphere	radius(sphere '((0,0,0),2.0)')
segments(path3d)	lseg3D[]	Array of edges of path	segments(path3d '[(0,0,0),(1,1,1),(2,0,0)]')
segments(polygon3d)	lseg3D[]	Array of edges of polygon	segments(polygon3d '((0,0,0),(0,1,1),(1,1,1),(1,0,0))')
skew(lseg3d, lseg3d)	boolean	Are skew?	skew(lseg3d '((0,0,0),(1,1,1))', lseg3d '((0,1,1),(1,0,2))')
skew(line3d, line3d)	boolean	Are skew?	skew(line3d '((0,0,0),(1,1,1))', line3d '((0,1,1),(1,0,2))')
vertical(point3d point3d)	boolean	Are the x values equal?	vertical(point3d '(0,0,0)', point3d '(0,1,1)')
vertical(lseg3d)	boolean	Are the x values equal?	vertical(lseg3d '((0,0,0),(0,1,1))')
vertical(line3d)	boolean	Are the x values equal?	vertical(line3d '((0,0,0),(0,1,1))')
volume(box3d)	double precision	Volume of box	volume(box3d '((0,0,0),(1,1,1))')
volume(sphere)	double precision	Volume of sphere	volume(sphere '((0,0,0),1))')
width(box3d)	double precision	Horizontal size of box	width(box3d '((0,0,0),(1,2,3))')

Table 4: 3D Geometric Type Conversion Functions

Function	Return Type	Description	Example
box3d(point3d, point3d)	box3d	points to box	box3d(point3d '(0,0,0)', point3d '(1,1,1)')
box3d(object3d)	box3d	bounding box	box3d(sphere '((0,0,0),2.0)')
box3d(box3d, box3d)	box3d	boxes to bounding box	box3d(box3d '((0,0,0),(1,1,1))', box3d '((3,3,3),(4,4,4))')
line3d(point3d, point3d)	line3d	points to line	line3d(point3d '(-1,0,0)', point3d '(1,0,1)')
lseg3d(box3d)	lseg3d	box diagonal to line segment	lseg3d(box3d '((-1,0,0),(1,0,1))')
lseg3d(point3d, point3d)	lseg3d	points to line segment	lseg3d(point3d '(-1,0,0)', point3d '(1,0,1)')
path3d(polygon3d)	path3d	polygon to path	path3d(polygon3d '((0,0,0),(1,1,1),(2,0,1))')
point3d(double precision, double precision, double precision)	point	construct point	point3d(23.4, -44.5, 66.1)

Continued on next page

Table 4 3D Geometric Type Conversion Functions – continued from previous page

Operator	Description	Example	
`point3d(object3d)`	`point3d`	center of object	`point3d(box3d '((-1,0,0),(1,0,1))')`
`polygon3d(path3d)`	`polygon3d`	path to polygon	`polygon3d(path3d '((0,0,0),(1,1,1),(2,0,0))')`
`sphere(point3d, double precision)`	`sphere`	center and radius to sphere	`sphere(point3d '(0,0,0)', 2.0)`
`sphere(object3d)`	`sphere`	bounding sphere	`sphere(lseg3d '((0,0,0),(1,1,1))')`
`sphere(sphere, sphere)`	`sphere`	spheres to bounding sphere	`sphere(sphere '((0,0,0),1)', sphere '((2,0,0),1)')`