3D Queries Practical - Table of Contents

Part 1 – Exploring 3D Queries 2
Exercise 1 – Connect FME Data Inspector to PostgreSQL/PostGIS 2
Exercise 2- Exploring 3D Queries 2
Exercise 3 – Constructing the Flying Freehold using SQL 4
Part 2 – UCL Facilities Management 5
Part 1 – DDL- Create Table 5
Part 2 – DDL- Create Constraints 5
Part 3 – DML – Insert Data 5
Part 4 – SQL Queries 5

Part 1 – 3D Queries

For this part of the exercise, keep using the practical4 schema you used last week, and also the practical4.txt file

Exercise 1 – Connect FME Data Inspector to PostgreSQL/PostGIS

Note that you need to be on the UCL VPN for this to work (or working on the UCL Desktop or Desktop Anywhere systems)

(See separate instruction sheet in Moodle for how to do this)

Exercise 2 - Exploring 3D Queries

This exercise assumes that you have created the 3D objects as per Practical 4. Write the SQL statements to answer the following questions. You can find a list of helpful 3D functions here:

http://postgis.net/docs/PostGIS_Special_Functions_Index.html#PostGIS_3D_Functions

1. Depending on how you wrote your insert statements las week, the geometryname column in your threedbuildings table might not be populated yet (i.e. the values in this column are null) – update this with the geometry type for each geometry using st_geometrytype

2. Calculate the 2D and 3D length of all the geometry in the threedbuildings table and list these with the geometry type (using st_geometrytype). Note which types of geometry have length values.

3. How far is the 3D point from the line in 2D and 3D.

4. How far is the 3D point from the 3D polygon in 2D and 3D?

5. How far is the point from the polyhedron?

6. What is the area of the polygon?

7. What is the volume of the polyhedron (use st_makesolid if necessary)?

8. Does the point intersect the polyhedron in 2D or in 3D?

9. Use ST_ASTEXT to examine the coordinates of the polyhedral surface and find a corner of this surface. Insert a new point with coordinates of this corner (you can select any of the corners). Call the new point 'Point 2'. Remember that we used coordinate system 27700 for this dataset.

10. Does Point 2 intersect the polyhedral surface in 2D or 3D. Do you get an error?

11. Modify the above query to only look at 3D intersection.

12. Write the SQL to generate a collection of the different component parts that make up the polyhedral surface using st_forcecollection.

13. Use the above statement in a WITH clause and force this collection into 2D using st_force2d

14. Take the 2D data from the above query, and embed it into the WITH statement. Using this combined WITH statement, check whether a 2D intersection query works now?

15. You probably got this error in the above query: *Relate Operation called with a LWGEOMCOLLECTION type. This is unsupported*. This basically means that you can't do an ST_INTERSECT3query (2D) on a geometry collection (3D intersection works fine!). What needs to happen is that the geometry collection needs to be split into individual components. You do this by changing the first line of your query from[1]:


```
WITH collection as (select st_forcecollection(location) as location
from practical4.threedbuildings
where geometryname = 'ST_PolyhedralSurface')
```

to

```
select (ST_Dump(st_forcecollection(location))).geom as location
from practical4.threedbuildings
where geometryname = 'ST_PolyhedralSurface'
```

This will return 6 rows – one for each face in the polyhedral surface. Now repeat the 2D intersection query with the modified statement.

[1] ST_Dump creates a list of the different parts of the geometry, along with a number for each part. To get just the geometry, you need to use the .geom statement

16. Write a query that
    a. Creates a geometrycollection and then collapses the polyhedral surface to 2D
    b. Buffers the result by 5m
    c. Cookie cuts the buffered data with the 2D collapsed polyhedral surface

    For each stage, make use of a WITH statement. The in class example is very similar so you can use that as a guide.

17. Visualise the result of the above query by using it in an INSERT statement - first create a table called practical4.buffer with columns ID and geometryname. Add a column called location that will hold a 3D geometry.

18. Insert the geometry is 3D from #16 - you will need to use ST_FORCE3D to insert the geometry as the cookie cut result is 2D. Visualise the result in FME

19. Extend the above SQL to shift the cookie cut surface to the original lower height of 14m (as per the original INSERT statement for the polyhedral surface). You can use ST_TRANSLATE for this st_translate(st_force3d(location), 0,0,14). Visualise the result in FME.

20. Extend the above query to:
    a. Extrude the cookie cut surface to the original height of the 3D polyhedral surface by extruding the cookie cut surface by 8m (see the original geometry creation script from the previous practical – the lower height is 14m and the upper height is 22m)

Now use FME Data Inspector to visualise this new geometry

21. Use st_area to find out the area of the polyhedral surface – collapse the surface into a 2D object (st_force2d) first

Note that you will get a double-count on the st_area value (128 instead of 64). This is because when the 3D geometry is collapsed into 2D the roof surface is shifted to the floor. st_area(st_envelope(st_force2d(location))) will fix this problem.

22. Calculate the volume of the new geometry and compare it to the area of the 2D cookie cut buffer * the height of the polyhedral surface (use a UNION ALL statement so that you can compare the results of two queries at the same time). Remember that you will need both st_zmax and st_zmin to work out the height as the polyhedron is off the ground (i.e. lowest height is 14m)

Exercise 3 – Constructing the Flying Freehold using SQL

Last week, we created the 3 objects making up the flying freehold example as a three separate objects. This week, we will first use UNION and DIFFERNECE operations to see if we can end up with the 2 buildings that we require (one with the extended room, one with the gap that is the space that the room from the neighbour takes up. The SQL for this has been demonstrated in class – run the required SQL and then visualise the results in FME to make sure that you got the expected objects.

Part 2 – UCL Facilities Management

Create a schema called *assets* for this part of the practical, and use a script file called assets.txt

** Note – as the example data is referenced to a LOCAL coordinate system, you should use SRID = 0 **

Part 1 – DDL- Create Table

Use the logical diagram for the UCL facilities management to write the CREATE TABLE scripts. The logical diagram does not provide field lengths for any *character varying* fields, so make best assumptions based on the information to be stored in the field – e.g. for a name or surname 25 characters might be suitable, a room number might only require 10.

Remember to add the spatial columns (type geometry) to the tables SEPARATELY.

Part 2 – DDL- Create Constraints

Using the diagram, write the primary key, foreign key, unique and check constraints for the Facilities Management system.

Part 3 – DML – Insert Data

Use the sketch diagram to write the SQL and insert the data into each table in the system

Part 4 – SQL Queries

Write the SQL queries that will answer the functional requirements set out for the UCL Facilities Management system