Spatial Databases

Dr Claire Ellul

c.ellul@ucl.ac.uk

Assignment Progress

- By now you should have:
  - Created your system specification
  - Created your conceptual and logical diagrams and written the documentation
  - Written the DDL, DML and the non-spatial queries
  - Made good progress on your 500 word assignment
- You can now refine the above to add spatial queries and 3D geometry creation (both this week) after which you can almost complete the assignment

Overview

- **Operations on Spatial Data**
  - Metric Operations
  - Topological Operations
- Spatial SQL
  - Examples of spatial queries
- Introduction to 3D Data
- 3D – DDL and DML
- Bonus Information

Spatial Functionality

- To solve real world problems, GIS (the software used for spatial data) offer a wide range of different analysis that you can do on your spatial data
  - Euclidean (Metric) Operations
  - Topological Operations
  - Operations Returning a Geometry

Spatial Functionality

- You can do these operations in a standard GIS software package (e.g. QGIS, ArcMap)
- However, for this module we will be using SQL for any spatial functionality we need

Euclidean (Metric) Operations

- Euclidean (or metric) operations are those that relate to properties of an object that are closely tied in with the coordinate system and projection used for its representation
- They always return a numerical answer
- Use local or national coordinate systems –
  - NB: you can’t measure area in degrees/minutes (global projection, WGS84, 4326)

Euclidean (Metric) Operations

- Distance
  - Input: Two objects, having 2D or 3D coordinates
  - Output: A real number (units will depend on the projection and map units set in the GIS)
    - (Distance here is crow-fly distance – i.e. straight lines. If we get time we may cover routing and navigation as an advanced topics)
- Perimeter:
  - Input: An object, usually represented as an area
  - Output: A real number (units depend on projection and map units)

Euclidean (Metric) Operations

- Length
  - Input: An object, usually represented as a line
  - Output: A real number (again units depend on projection and map units)
- Area
  - Input: An object, usually represented as an area
  - Output: A real number (units depend on projection and map units)

Euclidean (Metric) Operations

- For some of the operations, there are multiple ways of using the operation, for example:
  - What is the *distance* between Object A and Object B?
  - Find me all the objects within *distance* X of Object A
  - Find me all the objects having area larger than X.
  - What is the area of Object A?
  - Is Object A larger than Object B?

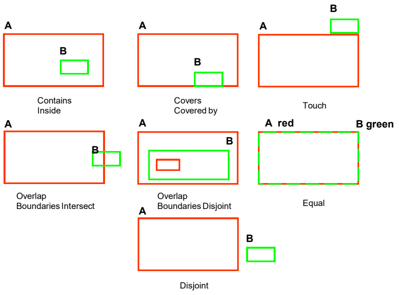
Topological Operations

- In contrast to Euclidean Operations, topological operations are those whose result does not change no matter what units or projection is used – i.e. no matter how the map is distorted
- Topology is defined as the identification of spatial relationships between adjacent or neighbouring objects

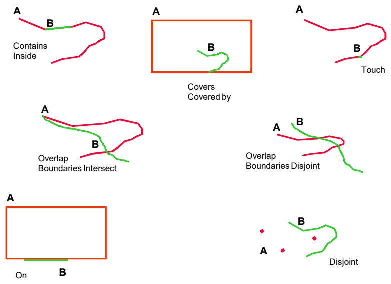
Topological Operations

- Topos = place, logos = speech, science
- Deals with spatial relationships between features in (a) space independently from geometry
- Answer to the operation is *true* or *false*

Topological Operations



Topological Operations

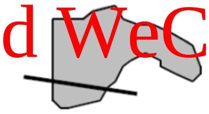


Topological Operations

- Types of questions that can be asked:
  - What topological relationship exists between Object A and Object B?
  - Find all the objects having relationship X with Object A
  - Find all the pairs of objects having relationship X
  - Find all the objects *intersecting* with Object A (where intersection represents any non-disjoint relationship)

Topological Operations

- Which statement best matches the picture (park is grey, road is the black line)?
  - A - "The road goes through the park"
  - B - "The road crosses the park"
  - C - "The road goes across the park"
  - D - "The road traverses the park"
  - E - "The road intersects the park"
  - F - "The road overlaps the park"



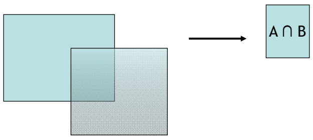
(Example adapted from Mark, D and Egenhofer M, 1994, CALIBRATING THE MEANINGS OF SPATIAL PREDICATES FROM NATURAL LANGUAGE: LINE-REGION RELATIONS, Proceedings, Spatial Data Handling, Vol. 1, pp. 538-553)

Operations returning a Geometry

- Rather than the real numbers (Euclidean) or true/false (topological) these return a geometry (spatial object) that represents the result of the operation.

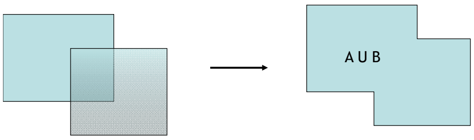
Operations returning a Geometry

- Intersection
  - Returns the geometry of the intersection of two objects



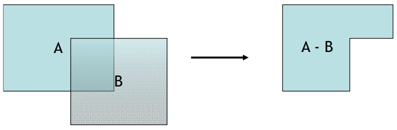
Operations returning a Geometry

- Union
  - Returns the geometry of the combination of two objects



Operations returning a Geometry

- Difference
  - Returns the geometry of the difference between A and B – i.e. A minus B (the geometry of A taking away that which is shared with B, sometimes written as A \ B)



Operations Returning a Geometry

- Centroid (the geometric centre of an object):
  - Input: An object, can be a point, line or area
  - Output: A pair of coordinates (units depend on projection and map units)

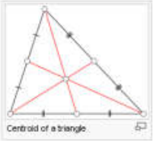
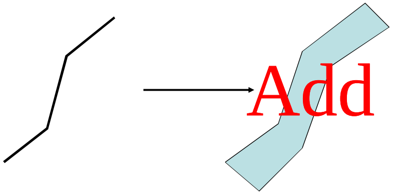


Image from: http://en.wikipedia.org/wiki/Centroid

Operations Returning a Geometry

- Buffering
  - Take an object and 'extend' the boundary



Overview

- Operations on Spatial Data
  - Metric Operations
  - Topological Operations
- **Spatial SQL**
  - Examples of spatial queries
- Introduction to 3D Data
- 3D – DDL and DML
- Bonus Information

Spatial SQL Queries - PostGIS

- Spatial SQL in the Database
  - Geometry is stored as Well Known Binary, and can be converted into Well Known Text using ST_ASTEXT

```
select st_astext(geom) from public.london_poi
```

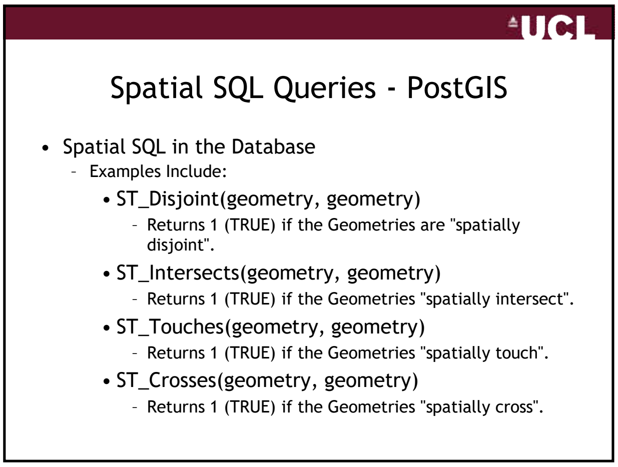
- Find the length of a line

```
SELECT ST_LENGTH(geom)
FROM public.london_highway;
```

--Note: as geom is not projected but is in WGS84, then the 'length' is in degrees/minutes/seconds (not useful) – we need to transform it to get a useful length in m

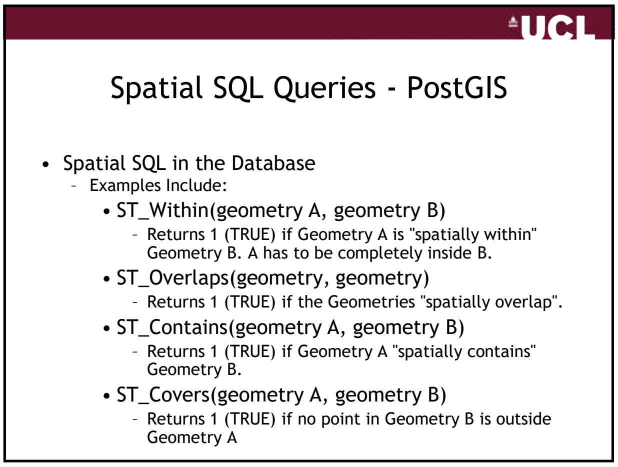
Spatial SQL Queries - PostGIS

- Spatial SQL in the Database
  - PostGIS prefixes all its spatial queries with ST_
  - Examples include:
    - ST_distance(geometry, geometry)
      - Returns the smaller distance between two geometries.
    - ST_max_distance(linestring,linestring)
      - Returns the largest distance between two line strings.
    - ST_perimeter(geometry)
      - Returns the 2-dimensional perimeter of the geometry, if it is a polygon or multi-polygon
    - ST_Area(geometry)
      - Returns the area of the geometry if it is a polygon or multi-polygon.



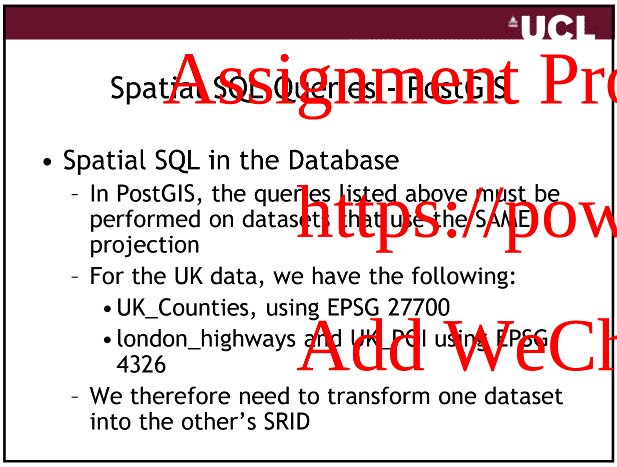
Spatial SQL Queries - PostGIS

- Spatial SQL in the Database
  - Examples Include:
    - ST_Disjoint(geometry, geometry)
      - Returns 1 (TRUE) if the Geometries are "spatially disjoint".
    - ST_Intersects(geometry, geometry)
      - Returns 1 (TRUE) if the Geometries "spatially intersect".
    - ST_Touches(geometry, geometry)
      - Returns 1 (TRUE) if the Geometries "spatially touch".
    - ST_Crosses(geometry, geometry)
      - Returns 1 (TRUE) if the Geometries "spatially cross".



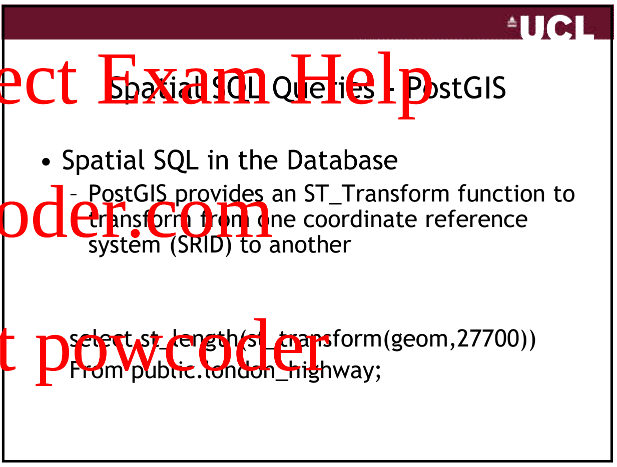
Spatial SQL Queries - PostGIS

- Spatial SQL in the Database
  - Examples Include:
    - ST_Within(geometry A, geometry B)
      - Returns 1 (TRUE) if Geometry A is "spatially within" Geometry B. A has to be completely inside B.
    - ST_Overlaps(geometry, geometry)
      - Returns 1 (TRUE) if the Geometries "spatially overlap".
    - ST_Contains(geometry A, geometry B)
      - Returns 1 (TRUE) if Geometry A "spatially contains" Geometry B.
    - ST_Covers(geometry A, geometry B)
      - Returns 1 (TRUE) if no point in Geometry B is outside Geometry A



Spatial SQL Queries - PostGIS

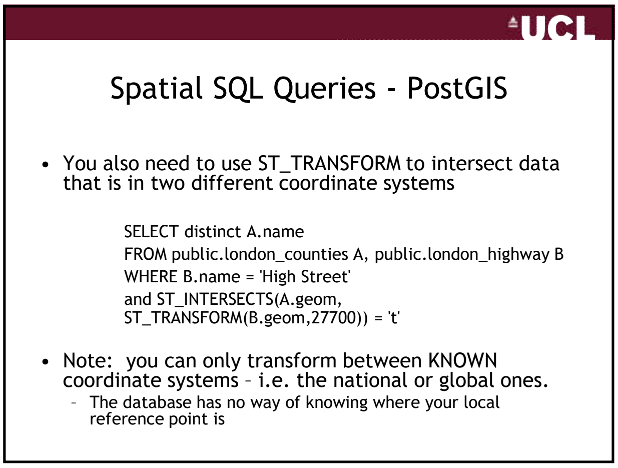
- Spatial SQL in the Database
  - In PostGIS, the queries listed above must be performed on datasets that use the SAME projection
  - For the UK data, we have the following:
    - UK_Counties, using EPSG 27700
    - london_highways and UK_POI using EPSG 4326
  - We therefore need to transform one dataset into the other's SRID



Spatial SQL Queries - PostGIS

- Spatial SQL in the Database
  - PostGIS provides an ST_Transform function to transform from one coordinate reference system (SRID) to another

```
select st_length(st_transform(geom,27700))
From public.london_highway;
```

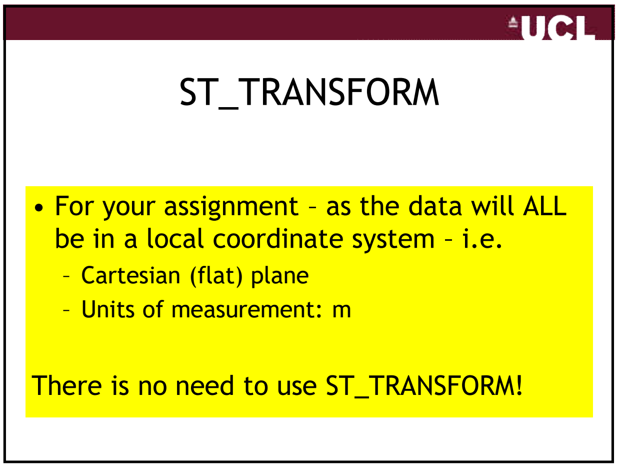


Spatial SQL Queries - PostGIS

- You also need to use ST_TRANSFORM to intersect data that is in two different coordinate systems

```
SELECT distinct A.name
FROM public.london_counties A, public.london_highway B
WHERE B.name = 'High Street'
and ST_INTERSECTS(A.geom,
ST_TRANSFORM(B.geom,27700)) = 't'
```

- Note: you can only transform between KNOWN coordinate systems - i.e. the national or global ones.
  - The database has no way of knowing where your local reference point is



ST_TRANSFORM

- For your assignment – as the data will ALL be in a local coordinate system – i.e.
  - Cartesian (flat) plane
  - Units of measurement: m

There is no need to use ST_TRANSFORM!

Combining Query Types

- Spatial and Non-Spatial SQL
  - Each spatial database has been extended to include spatial SQL queries as described
  - However, as you have just seen, there is limited requirement to run spatial SQL in isolation – in fact most queries require both spatial and non-spatial SQL
  - It is possible to combine the spatial query types we have seen just now with non-spatial queries such as aggregates, joins, filters and so forth
  - It is this combination that is the great strength of a spatial database, as it allows the map data to be combined with other corporate data such as staff information, purchasing and sales, payroll, customer information and so forth

Combining Query Types

- Examples of questions spatial databases can answer:
  - How many people who live within 25 minutes walk from this store also purchase fresh bread daily (this question assumes that a store-card system is in operation which records customer purchases)?
  - How many people travel over 10 miles to get to work?
  - How many customers could we target if we placed an advertising hoarding on Main Road X?
  - What mileage has consultant X travelled this month, and how much time has he billed (to identify whether travel time has been eating into billing hours)?
  - How many pharmacies are located within 2 miles of this potential site for a new pharmacy, and who owns them?
  - I am a salesman and want to target the highest-spending clients first, whilst at the same time minimising the distance I have to travel. Which route should I take?
  - How many houses in this area take their broadband service from us, what revenue does this generate and how much more revenue could we gain if we add a second fibre-optic cable?

Overview

- Operations on Spatial Data
  - Metric Operations
  - Topological Operations
- Spatial SQL
  - **Examples of spatial queries**
- Introduction to 3D Data
- 3D – DDL and DML
- Bonus Information

Spatial Query Examples

alter table public.london_highway add column length numeric(10,2);

update public.london_highway set length = st_length(st_transform(geom,27700));

Note: in reality you would not store the length in a column – you would calculate it when you needed it. That way if the geometry changes – i.e. the line is edited – you always get an up-to-date value

Spatial Query Examples

- Some examples – Calculating Length and Area

alter table public.London_counties add column area numeric(10,2);

update public.london_counties set area = st_area(geom);

In this case, we don't need ST_TRANSFORM as the data is already projected in British National Grid – and in British National Grid the units are "m" not degress/minutes/seconds

Spatial Query Examples

- If you want to work out what coordinate system your data is in, you can use this query:

select * from geometry_columns
where f_table_name = 'london_counties';

UCL

Spatial Query Examples

- Find the distance of all points of interest to all highways – this is a CROSS JOIN between
  - 96k highways and 37k points of interest = 3533664000 distance calculations – which can take a lot of time – so use a limit to limit to the first 1000 results

UCL

Spatial Query Examples

```
select st_distance(a.geom,b.geom), a.id as poi_id,
b.id as highway_id
from public.london_poi a, public.london_highway b
limit 1000;
```

UCL

Spatial Query Examples

- Find the CLOSEST highway to each POI

```
select distinct on (b.id) b.id as poi_id,
st_distance(b.geom_27700, s.geom_27700) as distance,
s.id as highway_id
from (select * from public.london_poi limit 10) b,
public.london_highway s
order by b.id, st_distance(b.geom_27700, s.geom_27700)
limit 10;
```

UCL

Spatial Query Examples

- Use Order By to list the distance measurements from shortest to longest
- Then DISTINCT ON to pick the first time each POI ID occurs (which will be the ID that corresponds to the shortest distance due to the order by)
- In this case using LIMIT 1000 to keep the query short doesn't work, as the distance between every POI and every highway has to be worked out in order to find out which is the shortest one ..
- So we limit the number of POIs being input into the query to 10

UCL

Spatial Query Examples

- Find out which highways are in a particular county – non-spatial option

```
select * from public.london_highway
where county_id = (select id from
public.london_counties where name = 'Bromley and Chislehurst Boro Const');
```

- 2085 rows, 968ms
- This only works as we have populated the COUNTY_ID foreign key in the london_highway table

UCL

Spatial Query Examples

- Now use a spatial query

```
select * from public.london_highway a
where st_intersects(a.geom_27700,(select geom from
public.london_counties where name = 'Bromley and Chislehurst Boro
Const'));
```

- 2135 rows, 4.32s
- Note that we are using a column called geom_27700 which is a British National Grid version of the highway geometry
  - This would cause problems if the highway geometry is edited

Spatial Query Examples

- Counties/Highways using ST_TRANSFORM

```
select * from public.london_highway a
where st_intersects(st_transform(a.geom,27700),(select geom from
public.london_counties where name = 'Bromley and Chislehurst Boro
Const'));
```

- 2135 rows, 4.321s
- Best option of the three, as if the road is moved (i.e. geom changes) the answer will reflect the updated data
- Also 2135 is more correct than 2085 – there is a many:many relationship between highways and counties which is not modelled correctly using county_id PK/FK

Spatial Query Examples

- Find out length of segments in a county

```
select sum(st_length(st_transform(a.geom,27700))), b.name
from
public.london_highway a, public.london_counties b
where st_intersects(st_transform(a.geom,27700),b.geom)
group by b.name
```

Spatial Query Advanced Example

- Can we get an indication of the temperature in rooms that don't have a sensor, based on the values measured by closest sensors:

```
select st_distance(b.location, s.location) as distance, b.room_id as
room_id, s.sensor_id as sensor_id
from assets.temperature_sensor s, assets.rooms b;
```

- This is an example of a Cartesian join, using spatial relationships to join the data and gives us all the distances from all the rooms to all the sensors
- This doesn't take into account the different floors

Spatial Query Advanced Example

- Find sensors on the same floor as the rooms
  - We can use st_z() to find the height of the sensor point

```
select st_z(s.location), b.floor, st_distance(b.location,
s.location) as distance, b.room_id as room_id,
s.sensor_id as sensor_id
from assets.temperature_sensor s, assets.rooms b;
```

Spatial Query Advanced Example

- We want the sensor height to be 8.5 and floor =2 or the height 2.5 and the floor = 1 (see the briefing document for where this is explained)

```
select st_z(s.location) as height, b.floor, st_distance(b.location,
s.location) as distance, b.room_id as room_id, s.sensor_id as
sensor_id
from assets.temperature_sensor s, assets.rooms b
where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor = 2 and
st_z(s.location) = 8.5);
```

Spatial Query Advanced Example

- Remove the rooms that actually have sensors in them
  - Option a – repeat the query as a nested query
  - SQL now becomes very complex!

Spatial Query Advanced Example

```
select * from (select st_z(s.location) as height, b.floor,
st_distance(b.location, s.location) as distance,
b.room_id as room_id, s.sensor_id as sensor_id from
assets.temperature_sensor s,
assets.rooms b where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor =
2 and
st_z(s.location) = 8.5)) f where f.room_id not in
(select room_id from (select st_z(s.location) as height, b.floor,
st_distance(b.location, s.location) as distance,
b.room_id as room_id, s.sensor_id as sensor_id from
assets.temperature_sensor s, assets.rooms b
where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor = 2 and
st_z(s.location) = 8.5)) g where g.distance = 0);
```

Spatial Query Advanced Example

- Option b – use the WITH statement
  - This assigns a temporary name to an SQL statement
  - Allows you to predefine a query and then use it as if it were a table
  - The WITH statement is part of the main SQL script – so you only write one SQL statement

Spatial Query Advanced Example

```
WITH roomsensors as (select st_z(s.location) as height,
b.floor, st_distance(b.location, s.location) as distance,
b.room_id as room_id, s.sensor_id as sensor_id
from assets.temperature_sensor s, assets.rooms b
where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor = 2
and st_z(s.location) = 8.5) order by room_id, distance)

select * from roomsensors where room_id not in (select
room_id from roomsensors where distance = 0);
```

Spatial Query Advanced Example

- Use DISTINCT ON to just get the closest sensor

```
WITH roomsensors as (select st_z(s.location) as height, b.floor,
st_distance(b.location, s.location) as distance, b.room_id as room_id,
s.sensor_id as sensor_id
from assets.temperature_sensor s, assets.rooms b
where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor = 2 and
st_z(s.location) = 8.5) order by room_id, distance)
select distinct on(room_id) room_id, sensor_id from roomsensors where
room_id not in (select room_id from roomsensors where distance = 0)
order by room_id, distance;
```

- Use WITH again to get the average temperature readings for these rooms

```
WITH allocatedsensors as (WITH roomsensors as (select st_z(s.location) as height, b.floor,
st_distance(b.location, s.location) as distance, b.room_id as room_id, s.sensor_id as
sensor_id
from assets.temperature_sensor s, assets.rooms b
where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor = 2 and st_z(s.location) = 8.5)
order by room_id, distance)
select distinct on(room_id) room_id, sensor_id from roomsensors where room_id not in
(select room_id from roomsensors where distance = 0)
order by room_id, distance)

select avg(h.value_degrees_c), h.temperature_sensor_id from
assets.temperature_values h where temperature_sensor_id in (select sensor_id from
allocatedsensors)
group by h.temperature_sensor_id;
```

- And finally a join to link the room_id to the temperature averages

```
WITH allocatedsensors as (WITH roomsensors as (select st_z(s.location)
as height, b.floor, st_distance(b.location, s.location) as distance,
b.room_id as room_id, s.sensor_id as sensor_id
from assets.temperature_sensor s, assets.rooms b
where (b.floor = 1 and st_z(s.location) =2.5) or (b.floor = 2 and
st_z(s.location) = 8.5) order by room_id, distance)
select distinct on(room_id) room_id, sensor_id from roomsensors where
room_id not in (select room_id from roomsensors where distance = 0)
order by room_id, distance)
select z.room_id, p.temperature_sensor_id, p.avg_c from
allocatedsensors z inner join (select avg(h.value_degrees_c) as avg_c,
h.temperature_sensor_id from
assets.temperature_values h where temperature_sensor_id in (select
sensor_id from allocatedsensors)
group by h.temperature_sensor_id) p on z.sensor_id =
p.temperature_sensor_id;
```

Spatial Query Examples

- For your assignment
  - This is a very complex query and took me about 1 hour to write
  - However, it does show you how to build up a query bit by bit ..
  - If you write something similar to this and it works you'll get some marks for advanced work!

Spatial Query Examples

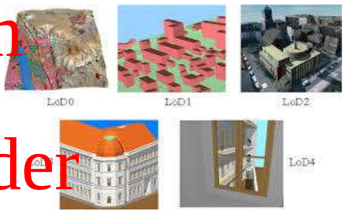
- For your assignment
  - Remember to make at least some of your functional requirements quite simple so that you don't have to spend time writing very complex queries
    - But not too simple so as to be unrealistic
  - Do the simple queries first ..

Overview

- Operations on Spatial Data
  - Metric Operations
  - Topological Operations
- Spatial SQL
  - Examples of spatial queries
- **Introduction to 3D Data**
- 3D – DDL and DML
- Bonus Information

Describing 3D Data

- CityGML
  - Defines 5 levels of detail (new version will change this slightly)

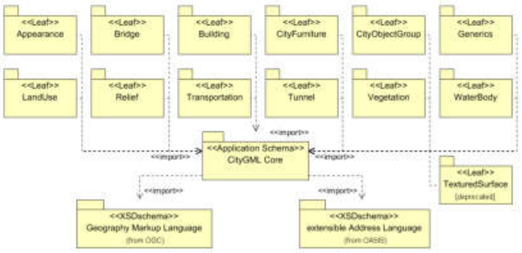


http://www.directionsmag.com/images/newsletter/2006/06_22/LoD_lg.jpg

Levels of Detail and Proposed Accuracy

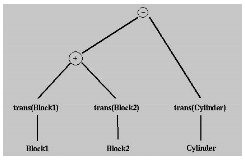
	LOD0	LOD1	LOD2	LOD3	LOD4
Model scale description	regional, landscape	city, region	city, city districts, projects	city districts, architectural models (exterior), landmark	architectural models (interior), landmark
Class of accuracy	lowest	low	middle	high	very high
Absolute 3D point accuracy (position / height)	lower than LOD1	5/5m	2/2m	0.5/0.5m	0.2/0.2m
Generalisation	maximal generalisation	object blocks as generalised features; > 6*6m/3m	objects as generalised features; > 4*4m/2m	object as real features; > 2*2m/1m	constructive elements and openings are represented
Building installations	no	no	yes	representative exterior features	real object form
Roof structure/representation	yes	flat	differentiated roof structures	real object form	real object form
Roof overhanging parts	yes	no	yes, if known	yes	yes
CityFurniture	no	important objects	prototypes, generalized objects	real object form	real object form
SolitaryVegetationObject	no	important objects	prototypes, higher 6m	prototypes, higher 2m	prototypes, real object form
PlantCover	no	>50*50m	>5*5m	< LOD2	<LOD2
…to be continued for the other feature themes					

CityGML Modules



Modelling 3D Data

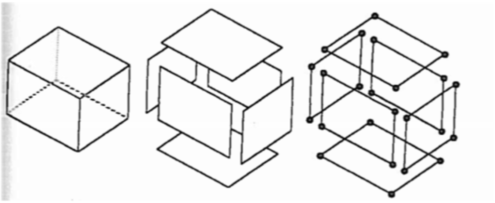
- In many 3D modelling packages constructive solid geometry is used:



https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/model/csg.html

Modelling 3D Data

- In PostGIS, 3D data is stored using a Boundary-Representation Structure
  - i.e. only the ‘shell’ of the 3D object is stored

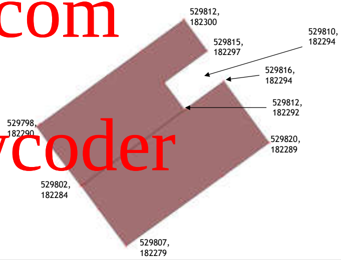


Modelling 3D Data

- That means you need to work out the coordinates of each face (side) of your 3D object (including base and roof)
- These are then stitched together to make the object
  - For your assignment – a simple 3D box is enough

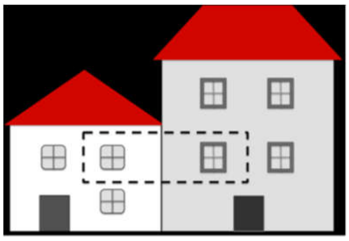
Exercise - Draw Some 3D Objects

- Draw these two buildings in 3D – you can assume that the building has a height of 10m



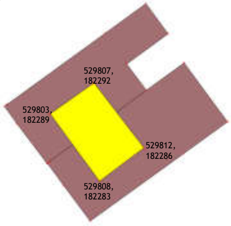
Exercise - Draw Some 3D Objects

- This building has an internal flying freehold



Exercise - Draw Some 3D Objects

- The 2D version of the flying freehold has the following coordinates



Exercise - Draw Some 3D Objects

- Now add the internal flying freehold to the object (you can ignore the roof structure)
  - Assume that the overlapping space has a lower height of 3m and an upper height of 7m

Modelling 3D Data – other sources of data

- Drawing objects manually is relatively OK for simple objects but isn’t easy for real data
- Other sources of 3D data include:
  - Extrusion
  - Modelling tools e.g. city engine, sketch-up, blender, rhino
  - BIM
  - LiDAR and photogrammetry

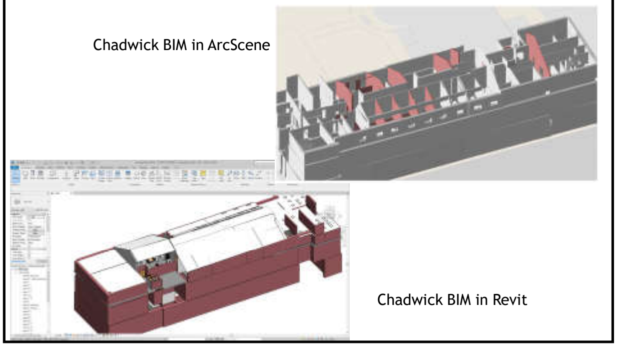
3D Data – Extrusion

3D Data - Esri City Engine

3D Data – Manual Capture - Sketch-Up and Google Earth

3D Data - BIM

Chadwick BIM in ArcScene



Chadwick BIM in Revit

UCL

3D Data – LiDAR and Photogrammetry

Data from Ordnance Survey

UCL

Overview

- Operations on Spatial Data
  - Metric Operations
  - Topological Operations
- Spatial SQL
  - Examples of spatial queries
- Introduction to 3D Data
- **3D - DDL and DML**
- Bonus Information

UCL

3D - DDL

- This is identical to 2D DDL
  - Use AddGeometryColumn
  - Note the number of dimensions is 3!

alter table assetsclass.buildings drop column if exists location;

select AddGeometryColumn('assetsclass','buildings','location',0, 'geometry',3);

UCL

3D - DDL

- You can constrain the data type if you like
  - This will prevent any invalid surfaces being inserted

alter table assetsclass.buildings drop column if exists location;

select AddGeometryColumn('assetsclass','buildings','location',0, 'polyhedralsurface',3);

UCL

DDL

- Or you can just create a table with a geometry column type
  - Not such a good idea as no constraints at all on SRID or dimension
  - Also defaults to 2D so 3D won't work

create table assetsclass.OSBuildings
(id serial,
location geometry);

UCL

Creating 3D Data In SQL

- insert into assetsclass.osbuildings (location) values (ST_GEOMFROMTEXT('POINT(0 0 3)',27700));
- insert into assetsclass.osbuildings (location) values (ST_GEOMFROMTEXT('LINESTRING(0 0 0,1 0 0,1 1 2)',27700));

Creating 3D Data In SQL

- insert into assetsclass.osbuildings (location) values (ST_GEOMFROMTEXT('POLYGON((1 1 3, 1 2 3,2 2 3, 2 1 3, 1 1 3))',27700));
- insert into assetsclass.osbuildings (location) values (ST_GEOMFROMTEXT('MULTIPOLYGON(((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 0 1 0, 0 1 1, 0 0 1, 0 0 0)))',27700));

Creating 3D Data In SQL

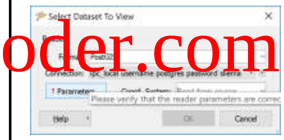
- insert into assetsclass.osbuildings (location) values (ST_GEOMFROMTEXT('POLYHEDRALSURFACE(((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)),((1 0 0, 1 1 0, 1 1 1, 1 0 1, 1 0 0)),((1 1 0, 0 1 0, 0 1 1, 1 1 1, 1 1 0)))',27700));

Creating 3D Data In SQL

SELECT ST_ASTEXT(location) FROM assetsclass.osbuildings;

- "POINT Z (0 0 3)"
  - "LINESTRING Z (0 0 0,1 0 0,1 1 2)"
  - "MULTIPOLYGON Z (((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,0 1 0,0 1 1,0 0 1,0 0 0)))"
  - "POLYGON Z ((1 1 3,1 2 3,2 2 3,2 1 3,1 1 3))"
  - "POLYHEDRALSURFACE Z (((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)),((1 0 0,1 1 0,1 1 1,1 0 1,1 0 0)),((1 1 0,0 1 0,0 0 1,1 1 1,1 1 0)))"

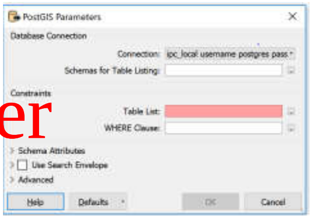
Viewing the Results in FME – Data Inspector



Select Dataset To View

Connection: ipc_local username postgres password sferra

Please verify that the reader parameters are correc



PostGIS Parameters

Database Connection

Connection: ipc_local username postgres pass

Schemas for Table Listing:

Constraints

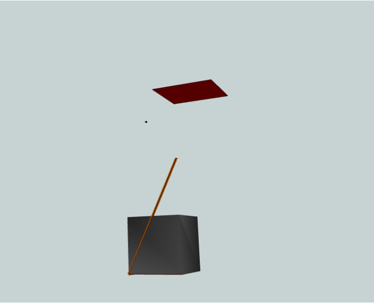
Table List:

WHERE Clause:

□ Use Search Envelope

Detailed instructions in the practical session

Viewing the Results in FME



Creating 3D Data in SQL

- Important – the order you list the nodes for each of the faces matters for the polyhedral surfaces!
  - If these are wrong, then they won't form a closed volume

Creating 3D Data in SQL

- -- 1. LEFT SIDE FACE - LOWER LEFT FRONT, UPPER LEFT FRONT, UPPER LEFT BACK, LOWER LEFT BACK, LOWER LEFT FRONT
- -- 2. BOTTOM FACE - LOWER LEFT FRONT, LOWER LEFT BACK, LOWER RIGHT BACK, LOWER RIGHT FRONT, LOWER LEFT FRONT
- -- 3. FRONT FACE - LOWER LEFT FRONT, LOWER RIGHT FRONT, UPPER RIGHT FRONT, UPPER LEFT FRONT, LOWER LEFT FRONT
- -- 4. RIGHT FACE - LOWER RIGHT BACK, UPPER RIGHT BACK, UPPER RIGHT FRONT, LOWER RIGHT FRONT, LOWER RIGHT BACK
- -- 5. BACK FACE - LOWER LEFT BACK, UPPER LEFT BACK, UPPER RIGHT BACK, LOWER RIGHT BACK, LOWER LEFT BACK
- -- 6. TOP FACE - TOP LEFT FRONT, TOP RIGHT FRONT, TOP RIGHT BACK, TOP LEFT BACK, TOP LEFT FRONT

Creating 3D Data in SQL

```
SELECT ST_Volume(location) As cube_surface_vol,
        ST_Volume(ST_MakeSolid(location)) As solid_surface_vol
  FROM (SELECT 'POLYHEDRALSURFACE(
    ((2 2 0, 2 2 12, 2 4 12, 2 4 0, 2 2 0)),
    ((2 2 0, 2 4 0, 4 4 0, 4 2 0, 2 2 0)),
    ((2 2 0, 4 2 0, 4 2 12, 2 2 12, 2 2 0)),
    ((4 4 0, 4 4 12, 4 2 12, 4 2 0, 4 4 0)),
    ((2 4 0, 2 4 12, 4 4 12, 4 4 0, 2 4 0)),
    ((2 2 12, 4 2 12, 4 4 12, 2 4 12, 2 2 12)) )'::geometry) As
f(location);
```

Creating 3D Data in SQL

```
insert into assetsclass.buildings (building_name,
university_id,location)
values
('Chadwick', (select university_id from assetsclass.university
where                university_name =
'UCL'),st_geomfromtext('POLYHEDRALSURFACE(
    ((3 2 0, 3 2 12, 3 22 12, 3 22 0, 3 2 0)),
    ((3 2 0, 3 22 0, 16 22 0, 16 2 0, 3 2 0)),
    ((3 2 0, 16 2 0, 16 2 12, 3 2 12, 3 2 0)),
    ((16 22 0, 16 22 12, 16 2 12, 16 2 0, 16 22 0)),
    ((3 22 0, 3 22 12, 16 22 12, 16 22 0, 3 22 0)),
    ((3 2 12, 16 2 12, 16 22 12, 3 22 12, 3 2 12)))'));
```

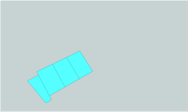
Creating 3D Data in SQL

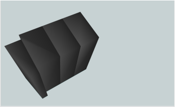
- Check the buildings data

```
select st_volume(st_makesolid(location)),
building_name from assetsclass.buildings
where location is not null;
```

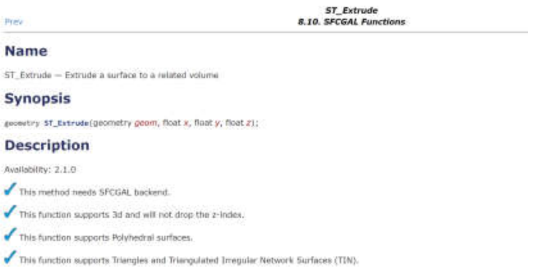
Creating 3D Data in PostGIS

- PostGIS (as of 2.1.0) now offers the ST_Extrude function
  - Extrude a surface to a related volume
  - Powerful as you can extrude along the X, Y, Z axis





Creating 3D Data in PostGIS



ST_Extrude
8.10. SFCGAL Functions

Name

ST_Extrude — Extrude a surface to a related volume

Synopsis

`geometry ST_Extrude(geometry geom, float x, float y, float z);`

Description

Availability: 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).

Creating 3D Data in PostGIS

- Creating a 3D object using extrude

```
insert into assetsclass.buildings
(building_name, university_id, location)
values
('Chadwick Extrude',(select university_id from
assetsclass.university where university_name = 'UCL'),
 st_extrude(st_geomfromtext
('POLYGON((3 2 0, 3 22 0, 16 22 0, 16 2 0, 3 2 0))'),0,0,12)
);
```

Creating 3D Data in PostGIS

```
select st_volume(st_makesolid(location)),
building_name from assetsclass.buildings
where location is not null;
```

Overview

- Operations on Spatial Data
  - Metric Operations
  - Topological Operations
- Spatial SQL
  - Examples of spatial queries
- Introduction to 3D Data
- 3D – DDL and DML
- Bonus Information

Bonus Information

- The following slides are not required for this module, but might be of interest in particular to the geospatial students

PostGIS – Geometry Versus Geography

- PostGIS – Geometry Versus Geography
  - When working with PostGIS you will see that two data types are offered for storing spatial data – Geometry and Geography
  - We will be working with Geometry

SOURCE: http://gis.stackexchange.com/questions/6681/postgis-what-are-the-pros-and-cons-of-geography-and-geometry-types

PostGIS – Geometry Versus Geography

- PostGIS – Geometry Versus Geography
  - Geography features are always stored in WGS84.
  - Measurements based on geography features will be in meters instead of CRS units and PostGIS will use geodetic calculations instead of planar geometry.
  - There is only a limited list of functions for manipulating/analyzing geography features, including:
    - measuring functions, ST_Intersects, ST_Intersection, ST_Buffer, ST_Covers and ST_CoversBy.

SOURCE: http://gis.stackexchange.com/questions/6681/postgis-what-are-the-pros-and-cons-of-geography-and-geometry-types

PostGIS – Geometry Versus Geography

- PostGIS – Geometry Versus Geography
  - Geometry features can be stored as projected data – i.e. in a Cartesian coordinate system, if required.
  - Measurements are in the units of the Coordinate Reference System chosen
  - As the data is represented on a 2D plane, there are many more spatial functions than for the Geography data type

SOURCE: http://gis.stackexchange.com/questions/6681/postgis-what-are-the-pros-and-cons-of-geography-and-geometry-types

Spatial Standards

- The Open Geospatial Consortium is the body responsible for setting standards for spatial data
- They provide documents describing:
  - how data should be modelled
  - How data can be shared
  - What functionality should be available
- Vendors can then certify their products against the standards

Databases - Spatial Queries - OGC

- A number of comparison operations are defined by the OGC on Geometry
- Each operation compares two geometries, A and B
- Operations defined include:
  - Equals
    - Returns true if two geometries are spatially equal – I.e. all coordinate values are identical and ordered in the same way
  - Intersects
    - Returns true if two geometries are adjacent or overlap each other, no matter what the dimension of their intersection

Databases - Spatial Queries - OGC

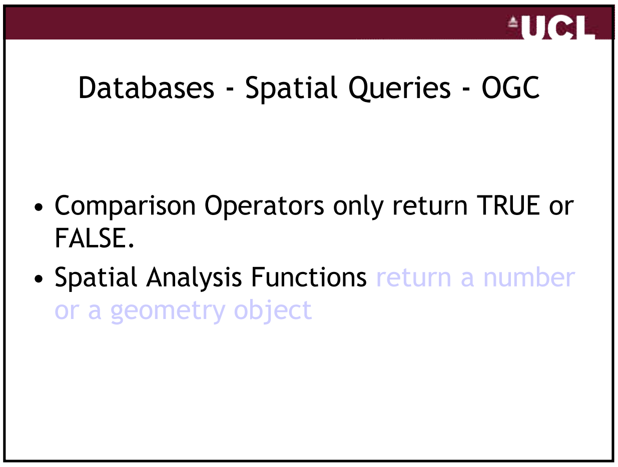
- OGC Comparison Operations
  - Crosses
    - Returns true if two geometries overlap each other and the dimension of the overlap is less than the dimension of the maximum dimension of the geometries
    - For example, the crossing of two lines (1-dimensional) will return a point (0-dimensional)
  - Contains
    - Returns true if geometry A is completely inside geometry B
  - Within
    - Returns true if geometry B is completely inside geometry A

Databases - Spatial Queries - OGC

- OGC Comparison Operations
  - Relates
    - Returns true if the two geometries being tested are related in any way (i.e. if ANY of the other relationships are true)
  - Overlaps
    - Returns true if the intersection of the geometries is of the SAME dimension as the geometries
    - For example, the overlap of two polygons (two-dimensional) returns another polygon (two dimensional)

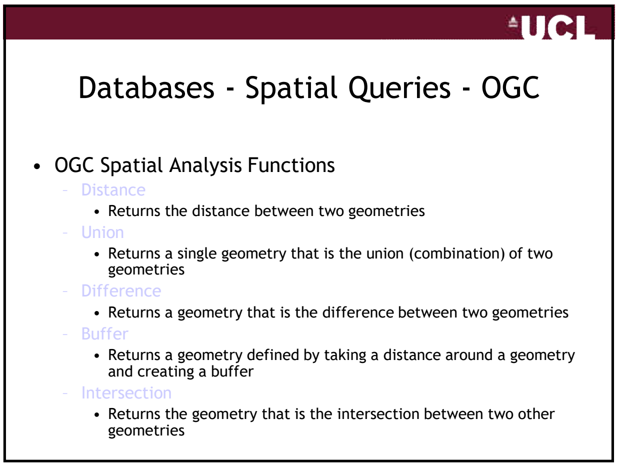
Databases - Spatial Queries - OGC

- OGC Comparison Operations
  - Disjoint
    - Returns true if two geometries are not connected at all
  - Touches
    - Returns true if two geometries are adjacent to each other (I.e. if the points in common are boundary points)



Databases - Spatial Queries - OGC

- Comparison Operators only return TRUE or FALSE.
- Spatial Analysis Functions return a number or a geometry object



Databases - Spatial Queries - OGC

- OGC Spatial Analysis Functions
  - Distance
    - Returns the distance between two geometries
  - Union
    - Returns a single geometry that is the union (combination) of two geometries
  - Difference
    - Returns a geometry that is the difference between two geometries
  - Buffer
    - Returns a geometry defined by taking a distance around a geometry and creating a buffer
  - Intersection
    - Returns the geometry that is the intersection between two other geometries