INFO-H415 ADVANCED DATABASES
TERM PROJECT

3D Spatial Databases and PostGIS

Tzu-Jou Hsiao **000424157**
Bahadır Han Akyüz **000423438**

31.12.2015

Tzu-Jou Hsiao
Bahadır Han Akyüz

Contents

Tzu-Jou Hsiao
Bahadır Han Akyüz

Introduction

In the last five years DBMS made a large step toward maintenance of geometries as GIS used to manage them. The support of 2D (two-dimensional) objects with 3D (three-dimensional) coordinates is adopted by all mainstream DBMS. The offered functions and operations are predominantly in the 2D domain. The DBMS spatial schemas have to be extended to fully represent the third dimension (first with simple volumetric object and later with more complex 3D data types).

Management of texture and mechanism for texture mapping and texture draping is critical for management of realistic 3D City models. 3D objects usually need more attributes for visualization compared to 2D objects.

In the geoinformation domain, 2D and 3D spatial data are commonly available. There is no doubt that 2D data are utilized much more than 3D. This situation is attributable to several factors including difficulties in 3D data structuring, particularly topological data structuring *(Raper, 1992; Li, 1994)*. These problems need to be investigated so that the feasibility of having a system capable of handling both 2D and 3D data types can be assessed. We will focus on the subject of spatial data representation in an attempt to contribute to an understanding of how spatial data could be utilized for a geoinformation system. We aim to review some of the pertinent spatial data representations and adopt suitable structures for a geoinformation system capable of handling 3D spatial data.

PostGIS is the well-known spatial database extension to PostgreSQL. PostGIS features a wide range of vector spatial data processing features as well as raster data management features. PostGIS is the spatial database of choice. It allows you to store and manipulate your geographical data with ease and speed. It is full of features and let you leverage the power of spatial analysis to transform your data into information.

Data Types

ESRI Shapefiles(SHP)

The shapefile format is a digital vector storage format for storing geometric location and associated attribute information. This format lacks the capacity to store topological information. The shapefile format was introduced with ArcView GIS version 2 in the early 1990s. It is now possible to read and write geographical datasets using the shapefile format with a wide variety of software.

The shapefile format is simple because it can store the primitive geometric data types of points, lines, and polygons. Shapes (points/lines/polygons) together with data attributes can create infinitely many representations about geographic data. Representation provides the ability for powerful and accurate computations.

The term "shapefile" is quite common, but is misleading since the format consists of a collection of files with a common filename prefix, stored in the same directory. The three mandatory files have filename extensions .shp, .shx, and .dbf. The actual shapefile relates specifically to the .shp file, but alone is incomplete for distribution as the other supporting files are required. Legacy GIS software may expect that the filename prefix be limited to eight characters to conform to the DOS 8.3 filename convention, though modern software applications accept files with longer names.

The Esri shapefile is one of the most common formats for exchanging vector data. It actually consists of several files with the same root name, but with different suffixes. At a minimum, you must include the .shp, .shx, and .dbf files. Other files may be included in addition to these three when extra spatial index or projection information is included with the file. Because a shapefile requires multiple files, it is often expected that you will zip them all together in a single file when downloading, uploading, and emailing them.

Keyhole Markup Language (KML)

KML is a file format for storing spatial information. It is most commonly associated with Google Earth, but there are lots of other programs that can read/write KML files.

Since KML information is XML-based, it can be stored in any database that is capable of storing text information.

The following is an example of a simple KML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
    <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

KML gained widespread use as the simple spatial data format used to place geographic data on top of Google Earth. It is also supported in Google Maps and various non-Google products.

KML stands for Keyhole Markup Language, and was developed by Keyhole, Inc., before the company's acquisition by Google. KML became an Open Geospatial Consortium (OGC) standard data format in 2008, having been voluntarily submitted by Google.

KML is a form of XML, wherein data is maintained in a series of structured tags. KML is unique and versatile in that it can contain styling information and it can hold either vector or raster formats ("overlays", in KML-speak). The rasters themselves are not written in the KML, but are included with it in a zipped file called a KMZ. Large vector datasets are also commonly compressed into KMZs.

The City Geography Markup Language (CityGML)

CityGML is a common information model and XML-based encoding for the representation, storage, and exchange of virtual 3D city and landscape models. CityGML provides a standard model and mechanism for describing 3D objects with respect to their geometry, topology, semantics and appearance, and defines five different levels of detail. Included are also generalization hierarchies between thematic classes, aggregations, relations between objects, and spatial properties. CityGML is highly scalable and datasets can include different urban entities supporting the general trend toward modeling not only individual buildings but also whole sites, districts, cities, regions, and countries.

CityGML provides much more than 3D content for visualization by diverse applications. It allows users to share virtual 3D city and landscape models for sophisticated analysis and display tasks in application domains such as environmental simulations, energy demand estimations, city lifecycle management, urban facility management, real estate appraisal, disaster management, pedestrian navigation, robotics, urban data mining, and location based marketing.

CityGML has been implemented in many software solutions and is in use in many projects around the world. In National Spatial Data Infrastructure programs in the Netherlands, Germany, France, Malaysia, Abu Dhabi and

other countries, CityGML provides an important platform for the transition from 2D to 3D data. It also plays an important role in bridging Urban Information Models with Building Information Models (BIM) to improve interoperability among information systems used in the design, construction, ownership and operation of buildings and capital projects.

CityGML is realized as an open data model is implemented as an application schema for the Geography Markup Language 3 (GML3), the extendible international standard for spatial data exchange issued by the Open Geospatial Consortium (OGC) and the ISO TC211. Because CityGML is based on GML, it can be used with the whole family of GML compatible OGC web services for data accessing, processing, and cataloging like the Web Feature Service, Web Processing Service, and the Catalog Service. CityGML is an open standard that can be used free of charge.

GeoJSON and YopoJSON

JavaScript Object Notation (JSON) is a structured means of arranging data in a hierarchical series of key-value pairs that a program can read. (It's not required for the program to be written in JavaScript.) JSON is less verbose than XML and ultimately results in less of a "payload," or data size, being transferred across the wire in web applications.

Following this pattern, GeoJSON is a form of JSON developed for representing vector features.
You might choose to save GeoJSON features into a .js (JavaScript) file that can be referenced by your web map. Other times, you may encounter web services that return GeoJSON.

A variation on GeoJSON is TopoJSON, which stores each line segment as a single arc that can be "called" multiple times by different polygons. In other words, when two features share a border, the vertices are only stored once. This results in a more compact file, which can pay performance dividends when the data needs to be transferred from server to client.

Well-known text (WKT)

Well-known text (WKT) is a text markup language for representing vector geometry objects on a map, spatial reference systems of spatial objects and transformations between spatial reference systems. A binary equivalent, known as well-known binary (WKB), is used to transfer and store the same information on databases, such as PostGIS, Microsoft SQL Server and DB2. Both WKT and WKB include information about the type of the object and the coordinates which form the object.

The formats were originally defined by the Open Geospatial Consortium (OGC) and described in their Simple Feature Access and Coordinate Transformation Service specifications.

The following is an example of a simple WKT file:
Show a 3D compound coordinate system, which is made by combining a projected coordinate system and a vertical coordinate system.

```
COMPD_CS["OSGB36 / British National Grid + ODN",
PROJCS["OSGB 1936 / British National Grid",
    GEOGCS["OSGB 1936",
        DATUM["OSGB_1936",
            SPHEROID["Airy 1830",6377563.396,299.3249646,AUTHORITY["EPSG","7001"]],
            TOWGS84[375,-111,431,0,0,0,0],
            AUTHORITY["EPSG","6277"]],
        PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
        UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG","9108"]],
        AXIS["Lat",NORTH],
```

```
            AXIS["Long",EAST],
            AUTHORITY["EPSG","4277"]],
        PROJECTION["Transverse_Mercator"],
        PARAMETER["latitude_of_origin",49],
        PARAMETER["central_meridian",-2],
        PARAMETER["scale_factor",0.999601272],
        PARAMETER["false_easting",400000],
        PARAMETER["false_northing",-100000],
        UNIT["metre",1,AUTHORITY["EPSG","9001"]],
        AXIS["E",EAST],
        AXIS["N",NORTH],
        AUTHORITY["EPSG","27700"]],
    VERT_CS["Newlyn",
        VERT_DATUM["Ordnance Datum Newlyn",2005,AUTHORITY["EPSG","5101"]],
        UNIT["metre",1,AUTHORITY["EPSG","9001"]],
        AXIS["Up",UP],
        AUTHORITY["EPSG","5701"]],
    AUTHORITY["EPSG","7405"]]
```

Extended Well-known Text / Binary (EWKT/EWKB)

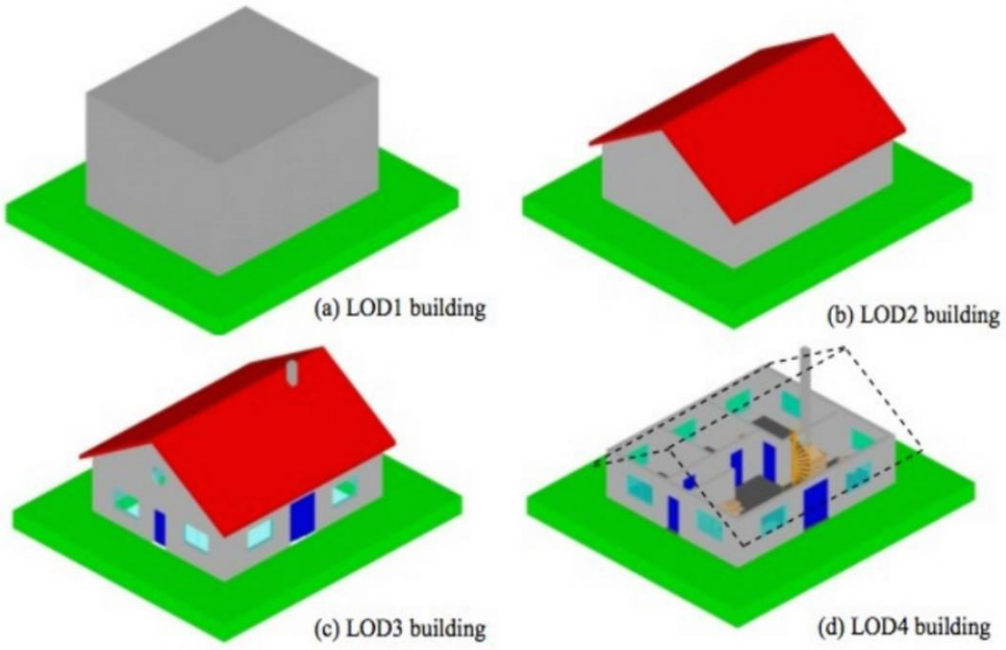
Since WKT and WKB were defined in the *SFSQL* specification, they do not handle 3- or 4-dimensional geometries. For these cases PostGIS has defined EWKT and EWKB formats. These provide the same formatting capabilities of WKT and WKB with the added dimensionality.

A not-at-all-standard WKT variant introduced by PostGIS. EWKT offers real support for 3D [XYZ, XYM and XYZM], and is quite widely supported, most notably by open source sw. EWKT can often offer better cross platform portability than pure WKT. PostGIS EWKT/EWKB add 3DM,3DZ,4D coordinates support and embedded SRID information.

LOD(Level of Details) in CityGML

CityGML has 5 Levels of Detail. Which can be basic differences in given image:[1]

[1] Source:Research Center Karlsruhe



LOD 0 - Regional model

- 2.5D Digital Terrain Model
- 11g: GeoRaster/TIN/PointCloud

LOD 1 - City / Site model

- block model without roof structures
- 11g: 3D Polygons/Surfaces

LOD 2 - City / Site model

- textured roof structures
- 11g: 3D Surfaces, GIF images

LOD 3 - City / Site model

- detailed architecture model
- 11g: 3D surfaces, GIF images

LOD 4 - Interior model

- architecture models
- 11g: 3D Lines/Polygons/Solids/Surfaces

Maintenance of multiple representations to be used as Level-Of-Details is another critical aspect of large three-dimensional models. Still the management of multiple representations is far from formalized. A very promising initiative is CityGML, which concepts can further be incorporated in the Spatial Schema and later incorporated in Implementation Specifications.

Comparison of DB solutions for 3D spatial data

In this part we aimed to give a general idea about different DB solutions for spatial data. We benefit from websites of relevant solutions.

Database	3D Support	Customized functions for 3D	Open Source	Visual Support[2]	Documentation for help	Extensions Support[3]	Support for Web applications
PostGIS	YES	YES	YES	YES	YES	YES	YES
Oracle Spatial	YES	YES	NO	YES	YES	YES	YES
IBM DB2	YES	YES	NO	YES	YES	YES	YES
Teradata Geospatial	YES	YES	NO	YES	YES	YES	YES
SpatiaLite	YES	YES	YES	YES	YES	YES	YES
MySQL	NO	YES	YES	NO	YES	YES	YES
GeoMesa	NO	NO	YES	NO	YES	NO	YES

Details of PostGIS

PostGIS is an open source project maintained by Refractions Research and is licensed under the GNU General Public License (GPL). First version of PostGIS was released in 2001, 4 years later than first version it has reached stability in 2005 after building of 6 middle version. It is used by many individuals and companies that includes UBER as well[4].

It is one of the widely used Spatial DBs since it provides various properties for spatial calculation.
It supports 2D data, 3D data as well. In additionally it enables user to keep extra information such as time, road-miles, distance which is known as “M” dimension. Shortly it can be said that it provides support for 4 dimensional spatial data.

PostGIS supports Spatial Reference System Identifier(SRID), thanks to that it is possible to work on different SRID values which is quite common for especially global-scale spatial data. To add, to drop or to change SRID is done easily by particular function. To choose the correct SRID is important part of projects on which calculations on and it’s mostly related locations in sphere.[5]

In PostGIS, geometrical information is kept in Well-Known Binary(WKT) representation. As well as other tools in PostGIS, It develops functions and usability according to ISO SQL/MM specification. It provides functions to convert data between common formats such as WKT, EWKT, GML, KML, GeoJSON, Text. For each conversion there are predefined functions that get input format generally as string then gives output such following examples:

- ST_AsGML,
- ST_GeogFromWKB,
- ST_GeomFromKML,
- ST_AsBinary.

Since PostGIS has two main data type for spatial data, one should decide which one to use before starting project. The data types are called as “geometry” and “geography”. While geography keeps records as latitude, longitude

[2] Easily and directly visual support with common applications such as QGIS, ARCGIS
[3] Open Source or paid 3rd party extension support
[4] https://en.wikipedia.org/wiki/PostGIS retrieved on 14.12.2015
[5] http://spatialreference.org/

unit, on the other hand geometry type keeps records as degrees which is not equivalent in whole world.[6] Although it seems more accurate to use geometry type in order to get in real units (generally in meters), it does make more complicated calculations on the backside (So it is costly, takes more time) and geography type has less function than geometry has in PostGIS. Geometry type uses Cartesian projection which provides much faster results with big number of available functions.

Application of PostGIS

At this part, we would like to show properties of PostGIS with sample 3D spatial DB. In order to get efficient and more realistic results we obtained dataset -that is provided by City of Berlin[7]- at official CityGML webpage[8]. This data consists of 1123 LOD2(mostly) and LOD1 buildings. This data was provided in CityGML version 1.0.0. In order to get dataset into PostGIS, we used Geospatial Data Abstraction Library (commonly known as GDAL)[9] which provides conversion between more than 100 formats and released under open license.

Several PostGIS functions will be used and we will try to give example cases for their using. But before going through calculations and after importing data to PostGIS, we have to define SRID in order to get correct results when we calculate values. To do so, PostGIS has function what we use.[10]

```
/*table name berlin_alexander*/
/*new SRID value 3068 that is mentioned by data-source*/
SELECT UpdateGeometrySRID('berlin_alexander','geom',3068);
```

Now, we changed SRID as 3068 and we can continue with calculations and what operations are possible in PostGIS.

Simple Distance Calculation

There are couple of functions to find the distance between 3d objects. Since objects are 3D, there are more than one distance between two objects. Two get the shortest ‘3D distance’ that doesn’t have to be on ground’ between two objects, we use ST_3DDistance() function.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3D
  FROM berlin_alexander AS a, berlin_alexander AS b
  WHERE a.gid AND b.gid<21
```

Since query for all pairs of buildings takes more than 10 minutes, we get the results for first 20 building which provides 190 pairs. For 190 lines, so calculation Postgres makes calculations in almost 1 second.

Simple output is as following:[11]

building1'	building2'	distance3d'
1'	2'	21.2507665335417'

[6] Marquez, Angel, PostGIS Essentials, Packt Publishing Ltd, April 2015
[7] http://www.berlin.de/en/
[8] http://www.citygml.org/index.php?id=1539
[9] http://www.gdal.org/
[10] http://postgis.net/docs/UpdateGeometrySRID.html
[11] Single quotation marks are seen at the end of each cell which is because of representation.

1'	3'	447.561771821768'
1'	4'	457.948229941049'

As it is seen in 3rd column, results are in meters which makes much sense. In the older versions of PostGIS, measurements were in degrees then user needed to convert into desired units as in that distance in km's changes according to location (ex. ~111 km in equator and getting smaller by going to North/ South) it was used conversion of the closest one to Berlin which equals to 78.71 km per degree.[12]

As we mentioned before, PostGIS provides multiple functions for distance calculations. On above, we used ST_3DDistance function which gives us the shortest distance between any possible 3D point-pairs.

We will continue with others and at the try to represent differences on basic images.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_Distance(a.geom, b.geom) AS Distance2Dmeter
  FROM berlin_alexander AS a, berlin_alexander AS b
  WHERE a.gid < b.gid AND b.gid<21
```

Above formula gives shortest distance for 3D objects or we can call it as bird's-eye distance.

In order to compare the values, we would prefer to represent 3D and 2D distances together.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_Distance(a.geom, b.geom) AS Distance2Dmeter
  FROM berlin_alexander AS a, berlin_alexander AS b
  WHERE a.gid < b.gid AND b.gid<21
```

As you can see on sample below, 2D and 3D distances have tiny differences. For some pairs, difference is much better and for some they are totally same.

building1'	**building2'**	**distance3dmeter'**	**distance2dmeter'**
1'	2'	21.2507665335417'	21.2508473055211'
1'	3'	447.561771821768'	447.560712265538'
1'	4'	457.948229941049'	457.947194416054'
1'	5'	38.8304840537922'	38.8304840537922'
1'	6'	54.0526883495609'	54.0526883495609'
1'	7'	48.274294612083'	48.274294612083'

We might get the same results as forcing our 3D data into 2D as following:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_Distance(a.geom, b.geom) AS Distance2Dmeter,
  ST_Distance(ST_Force2D(a.geom), ST_Force2D(b.geom)) AS ForcedDistance2Dmeter
  FROM berlin_alexander AS a, berlin_alexander AS b
  WHERE a.gid < b.gid AND b.gid<21
```

The results for last two distance will be same:

[12] https://en.wikipedia.org/wiki/Decimal_degrees

building1'	building2'	distance3dmeter'	distance2dmeter'	forceddistance2dmeter'
1'	2'	21.2507665335417'	21.2508473055211'	21.2508473055211'
1'	3'	447.561771821768'	447.560712265538'	447.560712265538'
1'	4'	457.948229941049'	457.947194416054'	457.947194416054'
1'	5'	38.8304840537922'	38.8304840537922'	38.8304840537922'
1'	6'	54.0526883495609'	54.0526883495609'	54.0526883495609'
1'	7'	48.274294612083'	48.274294612083'	48.274294612083'

With PostGIS, it is also possible to find the maximum distances between 3D objects:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_3DMaxDistance(a.geom, b.geom) AS MaxDistance3Dmeter,
  ST_Distance(a.geom, b.geom) AS Distance2Dmeter,
  ST_MaxDistance(a.geom, b.geom) AS MaxDistance2Dmeter
  FROM berlin_alexander AS a, berlin_alexander AS b
  WHERE a.gid < b.gid AND b.gid<21
```

As it can be seen on values varies for maximum and minimum cases:

building1'	building2'	distance3dmeter'	maxdistance3dmeter'	distance2dmeter'	maxdistance2dmeter'
1'	2'	21.2507665335417'	31.4235499470059'	21.2508473055211'	26.0802948210733'
1'	3'	447.561771821768'	455.917910632118'	447.560712265538'	455.599751283345'
1'	4'	457.948229941049'	489.59128609918'	457.947194416054'	489.295893743153'
1'	5'	38.8304840537922'	58.2947287019993'	38.8304840537922'	53.3481526806664'
1'	6'	54.0526883495609'	74.3719316891463'	54.0526883495609'	71.3455270018734'
1'	7'	48.274294612083'	70.9184138362599'	48.274294612083'	66.9116687958911'

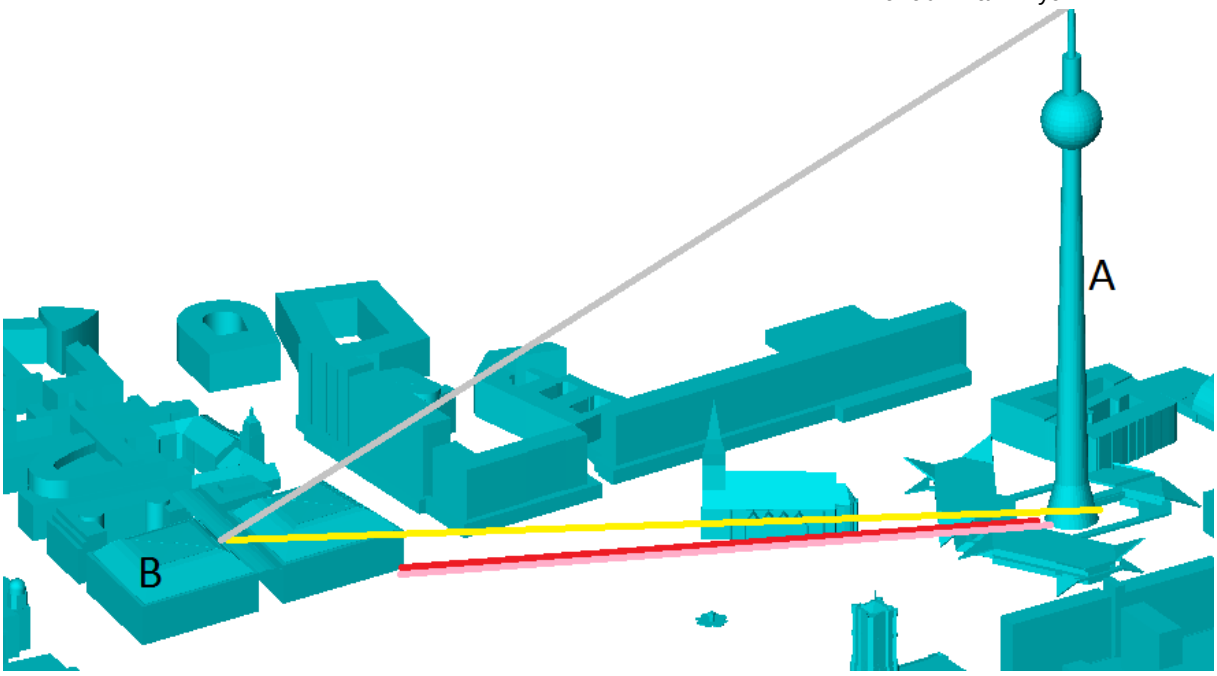
When we represent our data in 3D viewer, it could be more understandable to describe what we calculated. to represent our data we used FZK Viewer which has Open Source License by Karlsruhe Institute of Technology.[13] It is also preferable to use one of the most common product, QGIS which provides much more than representation.[14] QGIS provides supports for many useful needs with different extension. For our case - representing 3D data- we have met 2 main extensions (or Plug-in):

- Globe[15],
- Qgis2threejs[16]

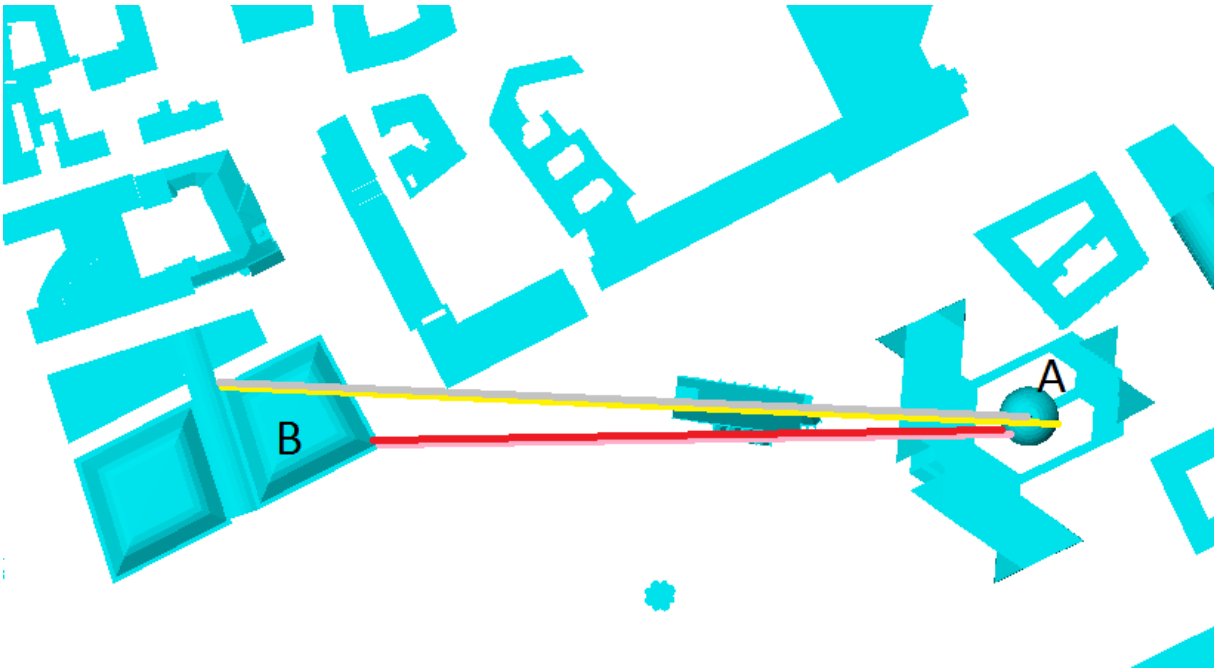
We will continue with first option because of its simplicity.

[13] http://www.iai.fzk.de/www-extern/index.php?id=1134
[14] http://www.qgis.org/en/site/
[15] https://hub.qgis.org/wiki/17/Globe_Plugin
[16] https://plugins.qgis.org/plugins/Qgis2threejs/

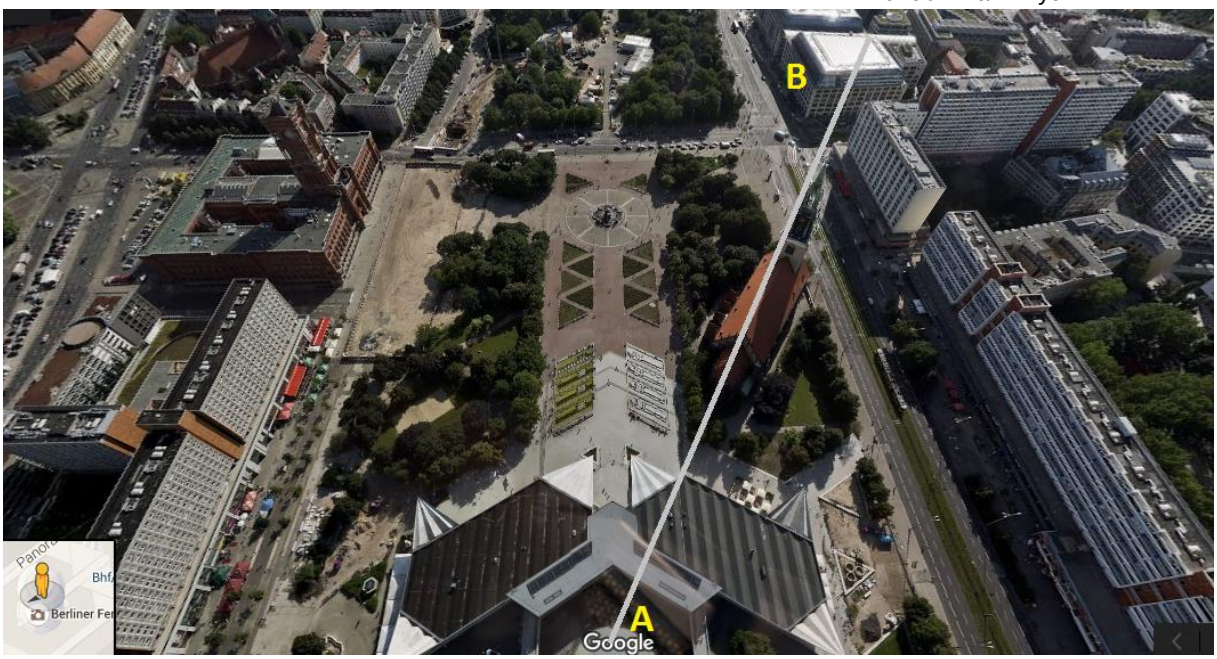
Tzu-Jou Hsiao
Bahadır Han Akyüz



In image “A” represents “Berliner Fernsehturm” in Alexanderplatz and B represents nearby building. Grey line (on top) stands for 3d maximum distance while the pink (under red) is for 3D minimum distance. On the other hand, red represents 2D minimum distance that is slightly less than 3D min distance for this case.



Just for real images, we can see 3D maximum distance from top of Berliner Fernsehturm with help of Google Street View.



More Detailed Calculation for Amazon Drones

Problem Definition

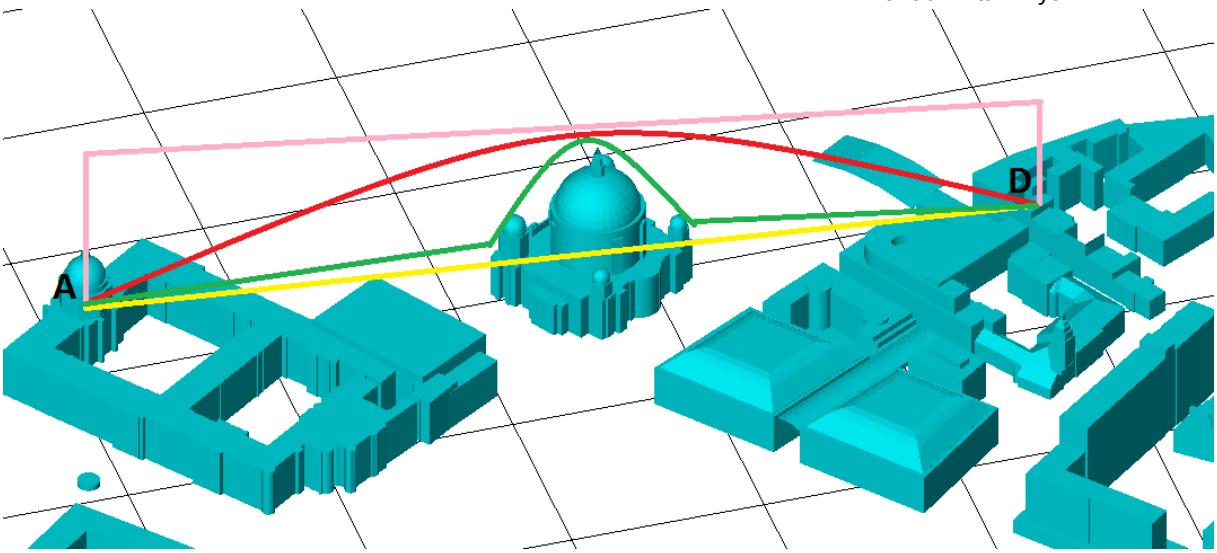
We already discussed about simple usage of 3D Spatial data, but what is more? What is it going to change? Why companies/people should use them instead of 2D while it is so costly to obtain and process 3D data?

Since there is not lot much application of 3D DBs, we will ask some questions by our imaginations and try to answer them with the help of PostGIS DBs.

We will get the case of Amazon Prime Air. As it is mentioned in Amazon's futuristic delivery plan, they aim to ship orders by "Octocopter" drones.[17] Studies are still going on this drones but further to drone's capabilities Amazon needs to consider about also roots. Since there is no driver or responsible to lead these drones, they should calculate the best (maybe not the shortest) roots for their travels.[18]

We will get two arbitrary building in our data set and try to calculate root between those building that we can call one of them Amazon's warehouse and the other delivery point.

[17] http://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone/
[18] This videos explain very well flying delivery process:
https://www.youtube.com/watch?v=98BIu9dpwHU
http://www.amazon.com/b?node=8037720011



As it can be seen on given image, there are different possibilities routes and some of them unfortunately are not feasible. Amazon (represented A) should send customer package to point D (Delivery point) through one of pink, red or green route since yellow route has meets another building. In reality there are more than possible routes and it won't be enough just to calculate route because there are also dynamic movements which are needed to be taken care of drone's intelligence. We will try to calculate initial feasible route for drone.

Queries

Firstly, we can assume that Amazon is sending orders from "the closest point to its facility to customer" to the closest point of customer building. This arbitrary point can be on earth at 0 level height or roof of the building. To get this we benefit PostGIs functions:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_3DClosestPoint(a.geom, b.geom) AS PointA,
  ST_3DClosestPoint(b.geom, a.geom) AS PointB
  FROM berlin_alexander AS a, berlin_alexander AS b
  WHERE a.gid < b.gid AND b.gid<21
```

building1'	building2'	distance3dmeter'	pointa'	pointb'
1'	2'	21.2507665335417'	01010000A0FC0B00003AE12684F7918240249DAC592D647C40F91C6F69D9E43440'	01010000A0FC0B000000000000042F824000000000A8787D40040000CCCCCC3440'
1'	3'	447.561771821768'	01010000A0FC0B0000000000003835824000000000488E7D40FBFFFFEB7C533240'	01010000A0FC0B00000000000004C38824000000000A0C38C40040000CCCC4C3340'
1'	4'	457.948229941049'	01010000A0FC0B0000000000003835824000000000488E7D40FBFFFFEB7C533240'	01010000A0FC0B000000000000785E844000000000B4EC8C40040000CCCC4C3340'
1'	5'	38.8304840537922'	01010000A0FC0B000000000000F4F4804000000000A8867B40F7FFFF5F66660240'	01010000A0FC0B00000000000080E5814000000000C80F7D40F7FFFF5F66660240'
1'	6'	54.05268	01010000A0FC0B00002DE165D77	01010000A0FC0B000000000000E0828

		83495609'	CE2814055215A7932187D40BA85E9B0D1563240'	04000000000C8207B40BA85E9B0D1563240'
1'	7'	48.274294612083'	01010000A0FC0B00000000000078E2814000000000040187D400C00002CDB563240'	01010000A0FC0B0000000000002CA5804000000000F05F7B400C00002CDB563240'

On above we can see closest point of two 3D object we are going to send packages between this points. As defined in PostGIS documentation these points are the start and end point of minimum 3D distance.

Here the points are represented in WKT (Well Known Text) format as usual in PostGIS. If needed, we can convert them to text which is more readable:

b1'	b2'	distance3dmeter'	pointa'	pointb'
1'	2'	21.2507665335417'	POINT Z (594.245857528447 454.261071848176 20.8939424415848)'	POINT Z (581.876953125 471.541015625 20.7999999523163)'
1'	3'	447.561771821768'	POINT Z (582.65234375 472.892578125 18.3261249065399)'	POINT Z (583.037109375 920.453125 19.2999999523163)'
1'	4'	457.948229941049'	POINT Z (582.65234375 472.892578125 18.3261249065399)'	POINT Z (651.80859375 925.587890625 19.2999999523163)'
1'	5'	38.8304840537922'	POINT Z (542.619140625 440.416015625 2.29999995231628)'	POINT Z (572.6875 464.986328125 2.29999995231628)'
1'	6'	54.0526883495609'	POINT Z (572.310957714018 465.512322761603 18.3391371317027)'	POINT Z (528.359375 434.048828125 18.3391371317027)'
1'	7'	48.274294612083'	POINT Z (572.30859375 465.515625 18.3392817974091)'	POINT Z (532.646484375 437.99609375 18.3392817974091)'

Not surprisingly, our points have 3 measurement and Point Z definition that is used in PostGIS for 3D points. Values give respectively dimensions of X, Y and Z coordinates.

We need to find 3D line between these two points check whether there is a building in route. Instead of finding points we will continue discovering routes. We can also calculate the length of route with functions but in this case it will be same as minimum distance.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
ST_3DShortestLine(a.geom, b.geom) AS ShortestRoute,
ST_3DLength(ST_3DShortestLine(a.geom, b.geom)) AS ShortestRouteLength
FROM berlin_alexander AS a, berlin_alexander AS b
WHERE a.gid < b.gid AND b.gid<21
```

Now, we have to check whether there is conjunction in our route.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  /*ST_3DShortestLine(a.geom, b.geom) AS ShortestRoute,*/
```

```
  /*ST_3DLength(ST_3DShortestLine(a.geom, b.geom)) AS ShortestRouteLength,*/
  c.gid AS Building3,
  ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5) AS WithinRoute
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid < b.gid AND b.gid<21 AND c.gid<10
  AND c.gid <> a.gid
  AND c.gid <> b.gid
```

Above SQL code check whether there is any “c” building between “a” and “b”. Obviously if there is a building within route we cannot use that route. As you can see we compare relations of 20 building (190 pair) with 10 building (for some cases less) to understand it is within our route. This query takes more than 10 seconds. We can say that when you include big numbers of buildings for analysis, it will take much more time. Additionally, we used ST_3DWithin function with 5 meters of range.

Some part of the results are given:

building1'	building2'	distance3dmeter'	building3'	withinroute'
1'	10'	40.1706616379044'	3'	f'
1'	10'	40.1706616379044'	4'	f'
1'	10'	40.1706616379044'	5'	t'
1'	10'	40.1706616379044'	6'	f'
1'	10'	40.1706616379044'	7'	f'
1'	10'	40.1706616379044'	8'	f'
1'	10'	40.1706616379044'	9'	f'

With given results, we can say that it is not possible to send order from 1 to 10 with defined route since 5th building is within route.

Just to see difference between 2D-within and 3D within function we can call given SQL query:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  /*ST_3DShortestLine(a.geom, b.geom) AS ShortestRoute,*/
  /*ST_3DLength(ST_3DShortestLine(a.geom, b.geom)) AS ShortestRouteLength,*/
  c.gid AS Building3,
  ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5) AS WithinRoute3D,
  ST_DWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5) AS WithinRoute2D
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid < b.gid AND b.gid<21 AND c.gid<10
  AND c.gid <> a.gid
  AND c.gid <> b.gid
```

Sometimes footprint of our route can conjunct with some other building but that doesn't mean that we cannot use that route. We can imagine that particular conjunct building has less height than our departure and arrival points.

building1'	building2'	distance3dmeter'	building3'	withinroute3d'	withinroute2d'
1'	7'	48.274294612083'	4'	f'	f'
1'	7'	48.274294612083'	5'	t'	t'
1'	7'	48.274294612083'	6'	f'	f'
1'	7'	48.274294612083'	8'	f'	f'
1'	7'	48.274294612083'	9'	f'	f'

1'	8'	447.572567154663'	2'	f'	f'
1'	8'	447.572567154663'	3'	f'	t'
1'	8'	447.572567154663'	4'	f'	f'
1'	8'	447.572567154663'	5'	f'	f'
1'	8'	447.572567154663'	6'	f'	f'

Building 3 is within footprint of our route but it is not within our original route. Since Amazon's droid flies that doesn't make any problem. But we can say that a pedestrian cannot use footprint of our route since there are at least one building on his/her way. Above query takes approximately 15 seconds.

To get faster result and more realistic case, we will 2nd and 3rd building to send orders in between and check all other buildings for conjunctions.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  /*ST_3DShortestLine(a.geom, b.geom) AS ShortestRoute,*/
  /*ST_3DLength(ST_3DShortestLine(a.geom, b.geom)) AS ShortestRouteLength,*/
  COUNT(c.gid) AS ConjuctedBuildingNumber
  /*ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5) AS WithinRoute3D*/
  /*ST_DWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5) AS WithinRoute2D*/
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid=2 AND b.gid=3 AND b.gid<21 AND c.gid<1124
  AND c.gid <> a.gid
  AND c.gid <> b.gid
  AND ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom, 5)=TRUE
  GROUP BY Building1,  Building2, Distance3Dmeter
```

There are 30 building which might cause a problem for our shipping:

building1'	**building2'**	**distance3dmeter'**	**conjuctedbuildingnumber'**
2'	3'	465.333259967083'	30'

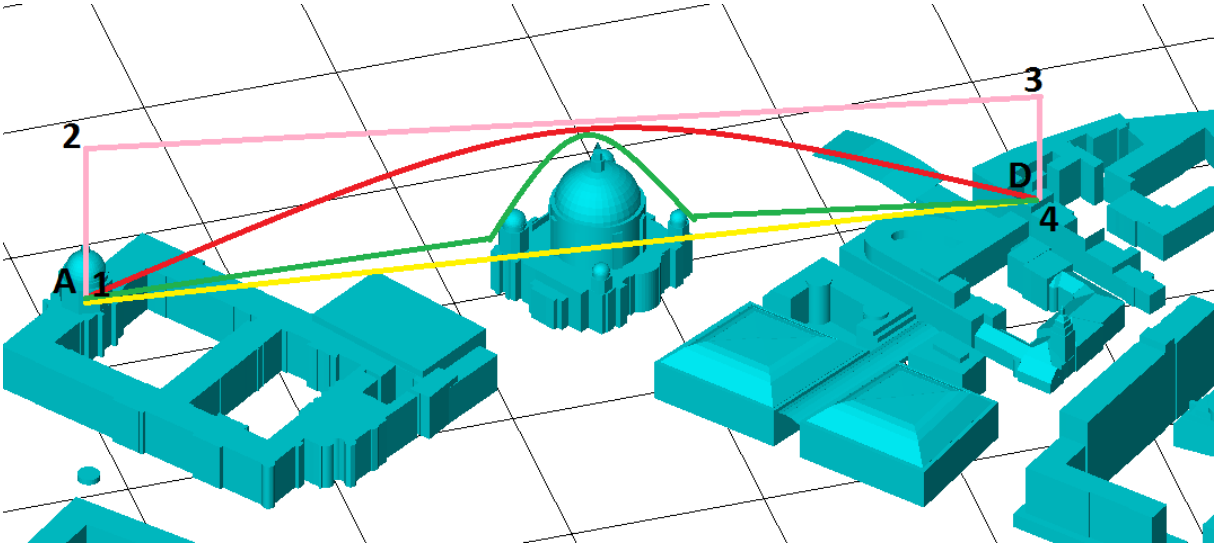
Now, it is not easy to decide how can the problem optimized since there are lots of possible ways. We will continue with reverse-u route which is based on maximum height of conjunctions and tried to be represented with pink color in the possible routes images.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_ZMax(a.geom) AS B1MaxHeight,
  ST_ZMax(b.geom) AS B2MaxHeight,
  MAX(ST_ZMax(c.geom)) AS MaxHeightConjuction
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid=2 AND b.gid=3 AND b.gid<21 AND c.gid<1124
  AND c.gid <> a.gid
  AND c.gid <> b.gid
  AND ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5)=TRUE
  GROUP BY Building1, Building2,Distance3Dmeter, B1MaxHeight, B2MaxHeight
```

building1'	building2'	distance3dmeter'	b1maxheight'	b2maxheight'	maxheightconjuction'
2'	3'	465.333259967083'	22.1128203868866'	19.3379504680634'	36.1127364635468'

As we can see, between a and b there is one building has more height than a and also b. That situation causes a problem. In some cases, it a or b might be taller than max height, so we should get the max of these three height to find height of our route.

Firstly, our drone will go up in vertical direction till maximum height of conjunction then will direct to target building at horizontal then will land target building. So we have to define 4 different points for departure, turning horizontal, turning vertical then arrival. It can be more understandable image on below:



We are going to use ST_MakePoint function to create new points:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_3DClosestPoint(a.geom,b.geom) AS Point1,

ST_MakePoint(ST_X(ST_3DClosestPoint(a.geom,b.geom)),ST_Y(ST_3DClosestPoint(a.geom,b.geom)
),36.1127364635468) AS Point2,

ST_MakePoint(ST_X(ST_3DClosestPoint(b.geom,a.geom)),ST_Y(ST_3DClosestPoint(b.geom,a.geom)
),36.1127364635468) AS Point3,
  ST_3DClosestPoint(b.geom,a.geom) AS Point4
  /*MAX(ST_ZMax(c.geom)) AS MaxHeightConjuction*/
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid=2 AND b.gid=3 AND b.gid<21 AND c.gid<1124
  AND c.gid <> a.gid
  AND c.gid <> b.gid
  AND ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5)=TRUE
  GROUP BY Building1, Building2,Distance3Dmeter, Point1, Point2,Point3, Point4
```

Unfortunately, PostgreSQL doesn't allow users to make an ORDER BY statement with function, so we had to use real values of maxHeight in ST_MakePoint function. This process might be time consuming and not practical for bigger cases.

building1'	building2'	distance3dmeter'	point1'	point2'	point3'	point4'
2'	3'	465.333259967083'	01010000A0 FC0B000000 0000008C9D 8240000000 00B0747C40 030000AC83 543240'	0101000080 000000008C 9D82400000 0000B0747C 4007000026 6E0E4240'	010100008 000000000 503882400 0000000A0 C38C4007 0000266E0 E4240'	01010000A0 FC0B000000 0000005038 8240000000 00A0C38C4 0040000CC CC4C3340'

Above SQL code and results seem like correct but indeed they are not. Since we create new points in our query, new points are not in “our berlin_alexander” table and they don’t have any SRID value. Not to get errors in further operations we need to define SRID and update our query as following:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_3DClosestPoint(a.geom,b.geom) AS Point1,

ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(a.geom,b.geom)),ST_Y(ST_3DClosestPoint(a.g
eom,b.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')) AS
Point2,

ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(b.geom,a.geom)),ST_Y(ST_3DClosestPoint(b.g
eom,a.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')) AS
Point3,
  ST_3DClosestPoint(b.geom,a.geom) AS Point4
  /*MAX(ST_ZMax(c.geom)) AS MaxHeightConjuction*/
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid=2 AND b.gid=3 AND b.gid<21 AND c.gid<1124
  AND c.gid <> a.gid
  AND c.gid <> b.gid
  AND ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5)=TRUE
  GROUP BY Building1, Building2,Distance3Dmeter, Point1, Point2, Point3, Point4
```

Just to be more flexible in our model, we get the SRID value from existing table with Find_SRID function which takes table and column as input.

In order to convert to points into route, we benefit from ST_MakeLine functions. This function takes 2 3D-Points or array of 3D points as input to create a line.

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  /*ST_3DClosestPoint(a.geom,b.geom) AS Point1,*/

/*ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(a.geom,b.geom)),ST_Y(ST_3DClosestPoint(a
.geom,b.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')) AS
Point2,*/

/*ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(b.geom,a.geom)),ST_Y(ST_3DClosestPoint(b
.geom,a.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')) AS
Point3,*/
  /*ST_3DClosestPoint(b.geom,a.geom) AS Point4*/
  ST_MakeLine(ARRAY[
  ST_3DClosestPoint(a.geom,b.geom),

ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(a.geom,b.geom)),ST_Y(ST_3DClosestPoint(a.g
eom,b.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')),

ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(b.geom,a.geom)),ST_Y(ST_3DClosestPoint(b.g
eom,a.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')),
```

```
  ST_3DClosestPoint(b.geom,a.geom)
  ]) AS Route
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid=2 AND b.gid=3 AND b.gid<21 AND c.gid<1124
  AND c.gid <> a.gid
  AND c.gid <> b.gid
  AND ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5)=TRUE
  GROUP BY Building1, Building2,Distance3Dmeter, Route
```

Our route will be as given:

b1'	b2'	distance3dmeter'	route'
2'	3'	465.333259967083'	01020000A0FC0B000004000000000000008C9D824000000000B0747C40030000AC83543240000000008C9D824000000000B0747C40070000266E0E4240000000005038824000000000A0C38C40070000266E0E42400000000050388240000000000A0C38C40040000CCCC4C3340'

If we want to compare minimum distance and our route, we can use such query:

```
SELECT  a.gid AS Building1, b.gid AS Building2,
  ST_3DDistance(a.geom, b.geom) AS Distance3Dmeter,
  ST_3DMaxDistance(a.geom, b.geom) AS MaxDistance3Dmeter,
  ST_3DLength(
  ST_MakeLine(ARRAY[
  ST_3DClosestPoint(a.geom,b.geom),

ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(a.geom,b.geom)),ST_Y(ST_3DClosestPoint(a.geom,b.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')),

ST_SetSRID(ST_MakePoint(ST_X(ST_3DClosestPoint(b.geom,a.geom)),ST_Y(ST_3DClosestPoint(b.geom,a.geom)),36.1127364635468), Find_SRID('public','berlin_alexander', 'geom')),
  ST_3DClosestPoint(b.geom,a.geom)
  ])
  ) AS RouteLength
  FROM berlin_alexander AS a, berlin_alexander AS b, berlin_alexander AS c
  WHERE a.gid=2 AND b.gid=3 AND b.gid<21 AND c.gid<1124
  AND c.gid <> a.gid
  AND c.gid <> b.gid
  AND ST_3DDWithin(ST_3DShortestLine(a.geom, b.geom),c.geom,5)=TRUE
  GROUP BY Building1, Building2,Distance3Dmeter, RouteLength
```

Result looks very close to minimum distance between buildings with only a 35 meters difference.

building1'	building2'	distance3dmeter'	maxdistance3dmeter'	routelength'
2'	3'	465.333259967083'	473.368103891547'	499.927588071262'

Results and thoughts

As we tried to explain above part, we can use PostGIS for such problems that aim to find a solution of 3D-related objects. However, even in such a simple case which includes almost 1000 buildings, calculations take a lot of time to compute. With more joins between tables and some harder tasks it might take days to find solutions for real problem which is probably quite time consuming for them.

In our simple case, we presented a possible route calculation for Amazon Droids. We should mention that these calculations purely depend on our assumption. We choose the way which is going through maximizing the value of height as much as possible. In real life that might not give feasible solutions since it's known that Amazon Drones can fly at maximum height of nearly 120 meters (400 feet) according to news. 3D route calculations might

be much harder and time consuming than 2D route calculations because they have more possible solutions and need more calculations.

Conclusion

As we can see the details in above part with application, PostGIS provides many functions for 3D data but it has also some absences. We can conclude that in order to easily get the results of given questions, we need additional functions:

- Is a 3D object covered by another 3D object?
- Does a 3D object cover another 3D object?
- Difference between two 3D object?
- Volume of a 3D object? (It already gives length of 3D lines)
- Surface area of a 3D object or area of a 3D object in a specific height?

As we can understand, PostGIS helps us to use 3D objects for many purposes since it has a limit for scale and applicability. When we are deciding about 3D spatial data and databases, we should firstly know that “Do we really need 3rd dimension”. If we don’t need 3rd dimension at all, there is no point to work on 3D spatial data since it takes more time to query and gives less flexibility. Then, the next question is that “which DB solution suits well for our particular project?” These steps look so basic, however, in order to work on 3D data, we should firstly answer those since needs vary a lot for different projects.

Sources

1. Towards a True 3D GIS, Ellul Claire, University College London, UK 3DGIS Special Interest Group presentation.
2. 3D Geo-Information Sciences, Di Jiyeong Lee,Siyka Zlatanova
3. http://en.wiki.quality.sig3d.org/index.php/Modeling_Guide_for_3D_Objects_-_Part_2:_Modeling_of_Buildings_%28LoD1,_LoD2,_LoD3%29
4. http://www.diva-gis.org/Data
5. http://www.oslandia.com/pages/postgis-mise-en-oeuvre-en.html
6. http://revenant.ca/www/postgis/workshop/index.html
7. http://www.citygmlwiki.org/index.php/Open_Data_Initiatives
8. https://en.wikipedia.org/wiki/Shapefile
9. https://www.e-education.psu.edu/geog585/node/691
10. http://www.citygml.org/index.php?id=1523
11. https://docs.oracle.com/cd/E17952_01/refman-5.5-en/spatial-extensions.html
12. http://www.oracle.com/technetwork/database/enterprise-edition/spatial-opensource-097629.html
13. https://docs.oracle.com/cd/B28359_01/appdev.111/b28400/sdo_webserv_intro.htm
14. http://www-03.ibm.com/software/products/en/db2-connect-family
15. https://3d.bk.tudelft.nl/szlatanova/pdfs/3D_geometries_in_spatial_DBMS.pdf
16. https://en.wikipedia.org/wiki/IBM_DB2
17. http://www-01.ibm.com/support/docview.wss?uid=swg27009474
18. http://www-01.ibm.com/support/docview.wss?uid=swg21168270
19. https://www.gaia-gis.it/spatialite-2.1/SpatiaLite-manual.html
20. http://www.spatialdbadvisor.com/
21. http://boundlessgeo.com/press-release/boundless-ccri-launch-opengeo-suite-geomesa/
22. http://www.geomesa.org/
23. https://en.wikipedia.org/wiki/Well-known_text
24. http://www.geoapi.org/apidocs/org/opengis/referencing/doc-files/WKT.html
25. https://www.gaia-gis.it/spatialite-3.0.0-BETA/GeoNotations.pdf