3D SPATIAL OPERATIONS FOR GEO-DBMS: GEOMETRY VS. TOPOLOGY

T.K. Chen[a,] *, A. Abdul-Rahman[a], S. Zlatanov[b]

[a]Department of Geoinformatics, Faculty of Geoinformation Science and Engineering, Universiti Teknologi Malaysia, Skudai, Malaysia - {kenchen, alias}@fksg.utm.my
[b]Section GIS Technology, OTB Research Institute for Housing, Urban and Mobility Studies, Delft University of Technology, The Netherlands - s.zlatanova@tudelft.nl

Commission II, WgS II/4

KEY WORDS: Spatial, Database, Geometry, Query, Three-dimensional, GIS

ABSTRACT:

Geo-DBMS becomes very important medium for GIS as it can handle and manage (e.g. retrieve and update) large volume of spatial data. Providing 3D spatial database with appropriate operation tools such as 3D spatial operations would be very useful for next generation of GIS software (i.e. 3D GIS) since the software would highly depend on the Geo-DBMS in both modeling and analysis. One of the desired components in such future software or system is geometric modeling capability that works with 3D spatial operations. The literature reveals 3D spatial database would be greatly enhanced if analytical operations on the spatial data could be manipulated in real 3D domain. Fundamentally, it can be considered that the aspect of 3D spatial operations within GIS software are still not much been addressed and solved as expected (i.e. up to the level where an operational 3D system could be realized). The main problem from this aspect is the unavailability of 3D spatial data type within geo-DBMS environment. It is the aim of this paper to describe 3D spatial operations for geometrical and topological data types within geo-DBMS environment. In the experiment, we utilize an existing geo-DBMS, PostgreSQL, later known as PostGIS, which complied with the standard specifications from Open Geospatial Consortium (OGC), e.g. abstract and geometry specification. The second factor why we utilise the PostGIS is because its an open source based technology and suitable for academic and research purposes. In this paper, we discuss a suitable way of developing a new 3D data type, polyhedron, for both geometrical and topological data types and spatial operations using C language.

1. INTRODUCTION

1.1 General Introduction

The database management system, or DBMS, is a computer software program that is designed as the means of managing all databases that are currently installed on a system hard drive or network. Different types of database management systems exist, with some of them designed for the oversight and proper control of databases that are configured for specific purposes. In spatial database, spatial data types are usually defined as Abstract Data Types (ADT), i.e. encapsulated types together with spatial operations. At implementation level, one can define spatial indices on spatial ADTs (Cardelli and Wegner, 1985; Liskov and Zilles, 1974; Stonebraker, et al., 1983; Stonebraker, 1986). A spatial object is an instance of a spatial type; it can have 0 (point), 1 (line), 2 (polygon), and 3 (solid) dimensions. All data stored in a DBMS is ultimately in binary form. A spatial DBMS will have a pre-defined way of organizing binary data to represent geometry, and this pre-defined way of organizing binary data is built into the DBMS in the form of a data type, such as SDO_GEOMETRY in Oracle Spatial or ST_GEOMETRY in IBM's DB2 Spatial Extender. Because this data type is built into the DBMS it is called a native geometry type. However, the lack of 3D primitive, such as polyhedron, yield the absent of 3D spatial operations. Therefore, to solve this current problem, the paper is organized in the following order: first, short discussion for the 3D objects construction in three-dimension, i.e. polyhedron. Then, the rules to formulate spatial operations for geometry and topology data type are highlight based on the developed data types, i.e. denotes the third section of the paper. The rules to construct the spatial operations for the minimal set of topological operations are discussed extensively in the fourth section. The experiment and discussions are presented in section 5, which provide the comparison between geometry and topology data types and finally, the research is concluded in section 6.

1.2 Characteristic of polyhedron

Polyhedron is a 3D equivalent of a set of polygon that bounds a solid object. It is made up by connecting all faces, sharing a common edge between two adjacent polygons. The polygons that make up the polyhedron have to be flat. This means that all points that construct a polygon must be in the same plane. Figure 1 denotes a sample of a planar and non-planar polygon.

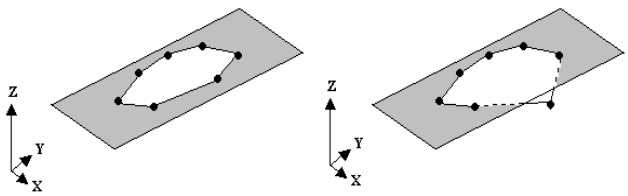


Figure 1: (a) Planar polygon, and (b) non-planar polygon

The characteristics of a valid polyhedron must include following rules (Aguilera & Ayala (1997), Aguilera (1998)):

* Corresponding author.

- Flatness – all polygons that bound a single volume of polyhedron must be flat. This means all vertices involve in constructing a polygon should be in the same plane. The flatness of a polygon can be verified by plane equation as follow:

$$Ax + By + Cz + D = 0 \quad (1)$$

  The Eq. (1) denotes the standard equation of a plane in 3D space. The normal to the plane is the vector (A,B,C).
- Polyhedron must be single volume object – a set of polygons that make up a polyhedron should be bounded as a single volume.
- Simplicity characteristic – this rule had been discussed by Arens (2005). However, this condition could be simplified by enforcing the construction of a polygon as follow:

  - Each edge has exactly 2 vertices only.
  - The starting and ending points of a polygon is same, and will only be stored once. E.g. a polygon consists 4 points (a,b,c,d), thus the polygon will be stored as (a,b,c,d,a), instead of (a,b,c,d,e), although a = e. Any point(s) with same location will be stored only once.
  - Polygon must have an area.
  - Lines from a polygon must not self-intersecting.
  - Singularity of polyhedron is not allowed, i.e. lower dimension object must not exist in the interior of higher dimension. E.g. point will not exist in the interior of line or polygon or polyhedron, line will not exist in the interior of polygon or polyhedron. However, lower dimension object may exist at the border of higher dimension object. This rule may directly avoid polygon intersects with other polygon(s) (see Figure 2). Any polygon that intersects other polygon(s) will not be stored as a part of polyhedron.

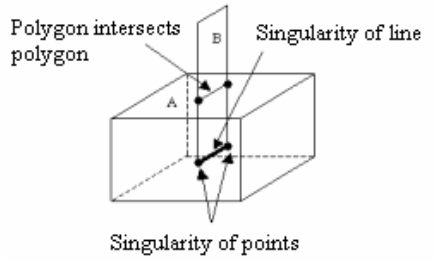


Figure 2: Polygon intersection causes the singularity of points and line

2. 3D DATA TYPES IN DBMS

2.1 3D Polyhedron

The existing spatial objects available in PostgreSQL are rather limited to 2D, but appear in three-dimensional space. The 3D primitive object is not available. Thus, 3D polyhedron will be discussed. Refer to the section 1.2, a polyhedron is constructed by a set of polygons that bounds a closed object. There are several important arrays that make up a complete data structure of a polyhedron. First, the array of coordinates should not be redundant. In most of the DBMS, they tend to store multi-polygon in such a way that coordinate-values are redundant. Each location of point consists of 3 coordinate-values, (x,y,z). If multi-polygon or polyhedron stores all coordinate-values, the data storage will become huge. However, Arens *et al.* (2005) had overcome this problem by representing each location with an ID. With this method, every triplet of coordinate-values (x,y,z) will be re-used if any shared vertex is found. This method directly reduces the storage of polyhedron, because each vertex will be shared by polygons that bound a closed object. Unlikely to multi-polygon, vertex may, somehow, not be shared with other polygons. Although polyhedron is similar to multi-polygon, it can be stored in a simpler and lesser-storage way. Thus, a set of coordinates array (with no redundant) will be recorded. Because of this, all vertices from each polygon will be referred as ID number. Thus, a set of array storing IDs that bound a polygon will be recorded as well. Besides, some extra information will be added, i.e. total polygon, vertices, etc.

2.2 Geometrical Data Type

The following data structure denote a sample of a polyhedron for geometrical data type:

```
SELECT * FROM GM_BODYTABLE WHERE PID = 1;
```
(For geometrical data type)

```
POLYHEDRON(PolygonInfo(6,24),SumVertexList(
8),SumPolygonList(4,4,4,4,4,4),VertexList(1
00.0,100.0,100.0,400.0,100.0,100.0,400.0,40
0.0,100.0,100.0,400.0,100.0,100.0,100.0,400
.0,400.0,100.0,400.0,400.0,400.0,400.0,100.
0,400.0,400.0),PolygonList(1,2,6,5,2,3,7,6,
3,4,8,7,4,1,5,8,5,6,7,8,1,4,3,2))
```

1). `PolygonInfo(6,24)` denotes 6 polygons and 24 IDs in `PolygonList`,
2). `SumVertexList(8)` denotes the total vertices,
3). `SumPolygonList (4, 4, 4, 4, 4, 4)` denotes total vertices for each of polygon (total polygon is 6, referred to (1)),
4). `VertexList()` denotes the list of coordinate-values for all vertices (with no redundant), and
5). `PolygonList()` denotes the information about each polygon from sets of ID.

The graphical representation of the sample polyhedron stated above is given as follows (Figure 3):

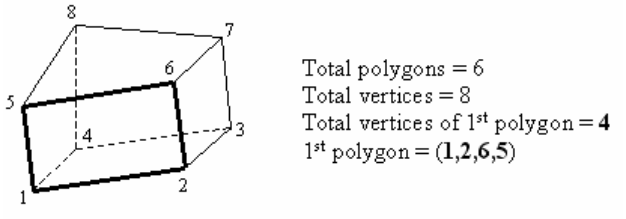


Figure 3: Sample structure of a polyhedron for geometrical data type

2.3 Topological Data Type

For topological data type, only object unique identifiers of lower dimension are stored for the polyhedron. For this case, face IDs are recorded as an input data set for a polyhedron, as shown in Figure 4.

```
SELECT * FROM TP_BODYTABLE WHERE PID = 1;
```
(For topological data type)

```
4,POLYHEDRON(FaceInfo(6,3),Face(1,2,3,4,5,6
), FaceSingularity(7,8,9))
```

1). `FaceInfo(6,3)` denotes 6 polygons and 3 singularities for the polyhedron,
2). `Face(1,2,3,4,5,6)` denotes all faces ID that construct a polyhedron,
3). `FaceSingularity(7,8,9)` denotes all singularities of polygon within polyhedron.

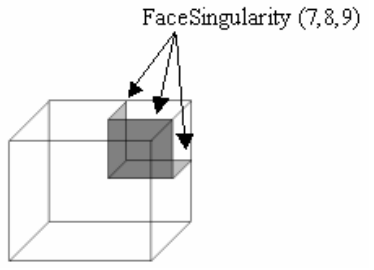


Figure 4: Sample structure of a polyhedron for topological data type

3. IMPLEMENTATION IN DBMS

Most of the commercial DBMS enable a user to create a new user-defined data type and functions. This user-defined datatype and functions can be written in C, C++ or Java. Data types can be started also using high-level language PL/SQL but usually these implementations have a bad performance. In this research, we have used C. In general, a user-defined type is defined as a class and must always have *input* and *output* functions. These functions determine how the type appears in strings (for input by the user and output to the user) and how the type is organized in memory. The input function takes a null-terminated character string as its argument and returns the internal (in memory) representation of the type. The output function takes the internal representation of the type as argument and returns a null-terminated character string. If users want to do anything more with the type than merely store it, they must provide additional functions to implement whatever operations they'd like to have for the type. The following three sections will illustrate how a new data type and a new function can be designed in C, compiled and used in PostgreSQL

3.1 Polyhedron Data Type

Suppose user wants to define a type complex that represents complex numbers. A natural way to represent a complex number in memory would be the following C structure:

```
typedef struct {
   char buf[200];
}POLYHEDRON;
```

As the external string representation of the type, a string of the form (POLYHEDRON) is chosen. The input and output functions are usually not hard to write especially the output function. But when defining the external string representation of the type, remember that users must eventually write a complete and robust parser for that representation as their input function. For instance:

```
PG_FUNCTION_INFO_V1(Polyhedron_in);
Datum
Polyhedron_in(PG_FUNCTION_ARGS)
{
    //== POLYHEDRON input class ==//
}
```

The output function can simply be:

```
PG_FUNCTION_INFO_V1(Polyhedron_out);
Datum
Polyhedron_out(PG_FUNCTION_ARGS)
{
    //== POLYHEDRON output class ==//
}
```

To define the complex data type, user needs to create the user-defined I/O functions within PostgreSQL environment before creating the type:

```
CREATE FUNCTION Polyhedron_in(cstring)
    RETURNS POLYHEDRON
    AS 'filename'
    LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION Polyhedron_out(POLYHEDRON)
    RETURNS cstring
    AS 'filename'
    LANGUAGE C IMMUTABLE STRICT;
```

Notice that the declarations of the input and output functions must reference the not-yet-defined type. Although this is allowed, but it will draw warning messages that could be be ignored. The input function must appear first. Finally, the data type will be declared:

```
CREATE TYPE POLYHEDRON (
   internallength = 100,
   input = Polyhedron_in,
   output = Polyhedron_out,
   alignment = double
);
```

3.2 User-defined Function/Operation

To create new user-defined functions/operations, C language is used within PostgreSQL. The PG_FUNCTION_INFO_V1( ) macro is used in calling for the function. Within the function, each actual argument is fetched using a PG_GETARG_xxx() macro that corresponds to the argument's data type, and the result is returned using a PG_RETURN_xxx() macro for the return type. PG_GETARG_xxx() takes as its argument the number of the function argument to fetch, where the count starts at 0. PG_RETURN_xxx() takes as its argument the actual value to return. The C function is give as follows:

```
PG_FUNCTION_INFO_V1(OVERLAP3D);
Datum OVERLAP3D(PG_FUNCTION_ARGS)
{
   int32 arg = PG_GETARG_xxx(0);
   //== 3D OVERLAP function class ==//
   PG_RETURN_xxx();
```

}

4. TOPOLOGICAL OPERATIONS

The topological operations presented here are based on the 4-intersection model and extends to 3D. The related operations include Overlap, Meet, Disjoint, Inside, Covers, CoveredBy, Contain, and Equal (see Figure 5). Some approaches will be considered in developing 3D spatial operation for DBMS:

- The 3D spatial operation will cover all necessary topological structures that define a complete solid object. In certain cases, not all primitives are needed, e.g. a polyhedron is defined by an ordered set of coordinate triplets for each polygon that bound a volumetric body, and line will not be used in the data structure.
- The implementing of the 3D spatial operations will be tested within the DBMS environment.
- The results from 3D topological operations return to a Boolean form (TRUE/FALSE). It involves two spatial objects, polyhedron and polyhedron.

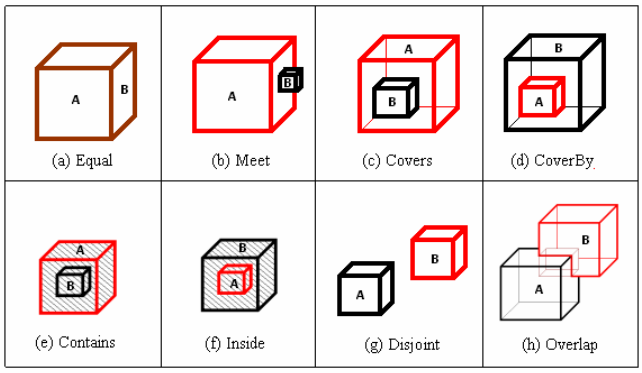


Figure 5: Body and body relation (after Zlatanova, 2000)

4.1 Spatial Operations for Geometrical Data Type

For topological operation in geometrical data type, coordinate triplet of vertex will be discussed. Similar to computational-geometry operation from previous section, the binary operation is divided into base and target object. However, the vertices from base object and polygons from target object will be discussed (see Figure 6).

This topological operation involves vertices (from base object) and polygon (from target object). Therefore, the relation between these 2 objects will be examined. The location of base vertices relative to target polygon will be either outside, touch, or inside. The implementation was discussed in Chen and Abdul-Rahman (2006). These relations will be used to determine how these 2 polyhedrons intersect each other as shown in Figure 5. For example (see Figure 7), vertices from base object are either touch the target polyhedron or located outside from target object.

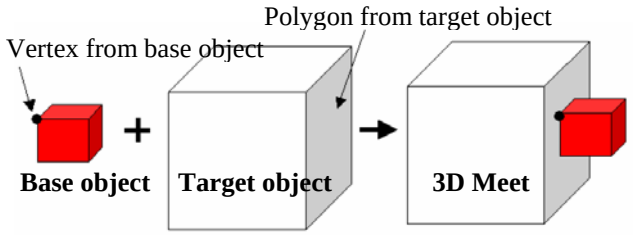
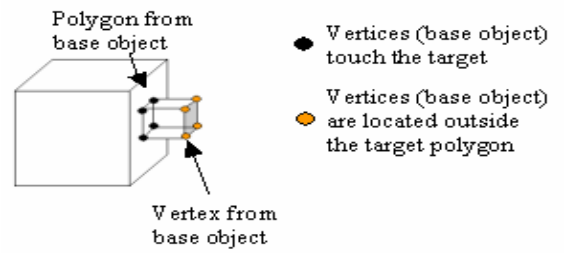


Figure 6: Base and target object for 3D operation



3D topological operations	Inside	Outside	Touch
Meet	NO	YES	YES

Figure 7: Vertices (base) are located and touch the target polygon

The following Table 1 denotes the complete relation between base and target object.

3D topological operations	Inside	Outside	Touch
Equal	X	X	✓
Meet	X	✓	✓
Covers	✓	X	✓
CoveredBy	✓	X	✓
Contains	✓	X	X
Inside	✓	X	X
Disjoint	X	✓	X
Overlap	✓	✓	✓

Table 1: Conditions for topological operations

The relationship of Covers and CoveredBy are different due to the role of base and target objects between these two relationships are different. For Covers, the base object covers the entire target object, whereas for CoveredBy, the target object covers the entire base object. The similar approach implemented between Contains and Inside.

4.2 Spatial Operations for Topological Data Type

Conventional topological data types involve the design of primitive's definition and object's construction. The main purposes of these implementations are to maintain 3D topology, which will be used to perform visualization and spatial analysis. These analyses are rather limited to spatial query, e.g. find any face that is shared by 2 objects, or select all vertices that construct a body. The strategy of implementing 3D topological operation for 3D topological structure is to define the similarity between 2 objects. Refer to Figure 5, the relationship between body A and B exists only if similar face stores both object A and B. If a spatial query is required to examined the relationship between these 2 bodies, i.e. find all faces that shared by Body A and B, the experiment will return to positive result due to the similarity of faces are found (see Figure 8).

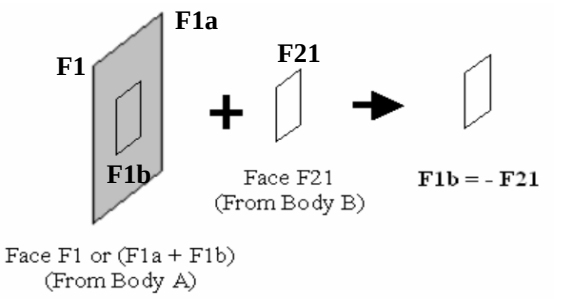


Figure 8: Determination of the similarity of face for 2 bodies

The research implements the Simplified Spatial Model (SSM) (see Figure 9) developed by Zlatanova (2000) as a spatial model for polyhedron. The main reason the research utilizes the SSM because it is an optimized spatial model, in term of data storage size. The comparison among other spatial models could be found in Zlatanova (2000). The model consists of two constructive objects, (*nodes* and *faces*) and four geometric objects (*point*, *line*, *surface* and *body*). A point is a spatial object that does not have shape or size but position is the space. A line is a type of a spatial object that has length and position. A surface is an abstraction of spatial object that has position and area. A body is a type of spatial object that has a position and a volume. Nodes constitute points and lines and faces constitute surfaces and bodies.

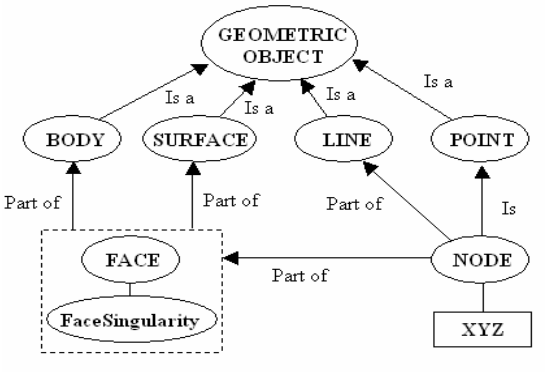


Figure 9: Modified SSM for topological operations

In order to implement the 3D topological operations for SSM, a minor modification for face-body relationship needs to be modified. Previously, SSM only relates the face and body relationship by implement the constructive object for the body itself. The intersection among other bodies is not stored in body table. However, this can be overcome by adding the relationship FaceSingularity-Body to the body table (see Figure 10b). With the new implementation of body table for SSM, 3D topological operations can be carried out based on following rules:

1). The new approach attempts to implement FaceSingularity as intersection result for body table.
2). Examine the similarity of Faces and FaceSingularity between 2 bodies. Faces with different orientation, i.e. 1245 and -1245 are considered as similar face but different orientation of vertices, will be selected as well.
3). The similarity and non-similarity of surfaces from 2 bodies will be used to determine in which relationship these bodies are. The rules for each of the topological operations are (see Table 2 and Figure 9):

Topological Operators	Face (base) = Face (target)	Face (base) = FaceSingularity (target)	FaceSingularity (base) = Face (target)	Face (base) = -Face (target)	Face (base) ≠ Face (target)
3D Meet	✗	✗	✗	✓	✓
3D Overlap	✗	✓	✓	✓	✓
3D Inside	✗	✓(All)	✗	✗	✗
3D Contains	✗	✗	✓(All)	✗	✗
3D Covers	✓	✗	✓	✗	✗
3D CoveredBy	✓	✓	✗	✗	✗
3D Equal	✓(All)	✗	✗	✗	✗
3D Disjoint	✗	✗	✗	✗	✓(All)

Table 2: Rules of 3D topological operations for modified SSM

5. ANALYSIS & DISCUSSIONS

5.1 Geometry Vs. Topology Data Types

The comparison could only be done (between geometrical and topological data types) is the 3D topological operations (see Figure 10). Since the computational-geometry and metric operations manipulate the coordinate triplets within the mathematical computations, these 3D operations are impossible to be implemented in topological data type. With the absent of these 3D operations, comparison could not be done for both geometrical and topological data types. Thus, the comparison is focused on the 3D topological operations, in terms of execution time consumed.

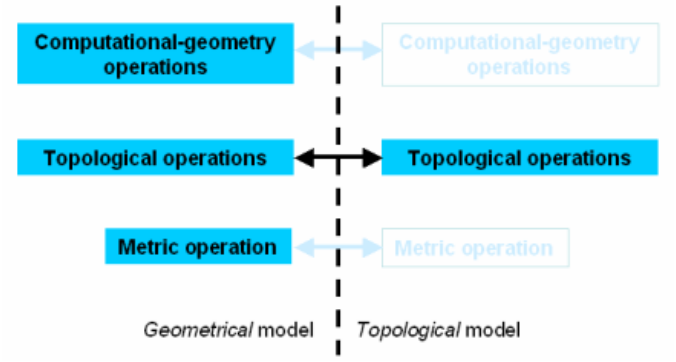
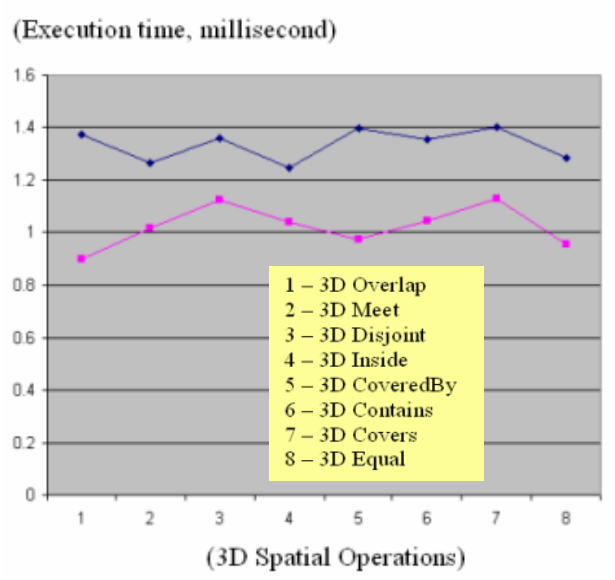


Figure 10: Possible comparison between geometrical and topological data types

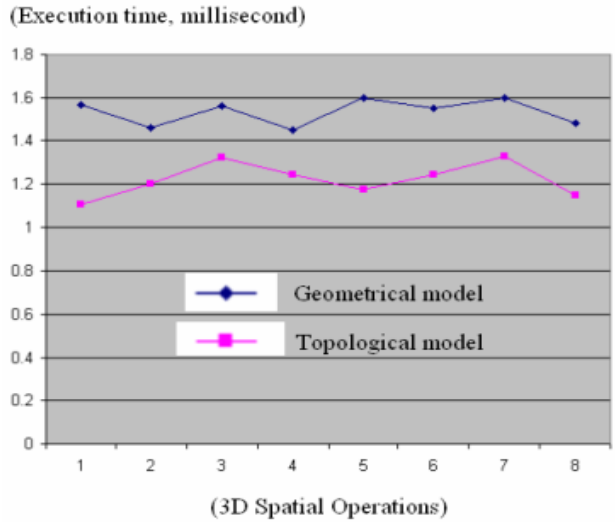
Refer to the Figure 11a, the execution time consumed for geometrical data type is higher than the topological data type. The similar results are appeared in the Figure 11b, which the datasets are getting bigger, the execution time of data storage size are gradually increased because the geometrical data type applied complex mathematical approaches within the Geo-DBMS environment. The topological data type only implements the spatial query between two spatial objects.

6. CONCLUDING REMARKS

We have implemented an approach for 3D topological operations of geometrical and topological data types in Geo-DBMS. The results have shown that implementation of a 3D data type and functions allowing 3D GIS analysis are possible.



(a)



(b)

Figure 11: The comparison between geometrical and topological data types

Our concept was tested within PostgreSQL computing environment and has provided a promising outcome with respect to the developed algorithms. The 3D topological operations implemented in this paper covers the relationship of 4-intersection model (8 kinds of relations), i.e. meet, overlap, contains, covers, coveredby, inside, equal, and disjoint. However, the 3D topological operations could be extended to 9-intersection model, which the exterior element is considered in the relationship together with the interior and boundary elements. More different kinds of relations could be implemented (in future) with the same approach from this research. The 3D topological operations for DBMS could be implemented using different approaches such as using other programming language, i.e. PL/PGSQL, PL/TCL, PL/Perl, and SQL within PostgreSQL environment. However, since the PostgreSQL was developed mostly using C language, an implementation using procedural languages could result in less efficiency and low performances. A very important issue still need to be addressed is visualization of the result of 3D queries. Appropriate graphical visualization is especially important for 3D in order to get a better perception of the result of the query. We believe this research effort towards realizing a fully 3D spatial analysis tools within Geo DBMS environment would be beneficial to 3D GIS research community. This is because major GIS task involves DBMS (except 3D visualization), i.e. dataset handling, spatial operations, etc. It is our aim to move further in addressing this issue of spatial data modeling and geometrical modeling for 3D GIS.

ACKNOWLEGEMENTS

The author would like to thank the Ministry of Science, Technology and Innovation (MOSTI), Malaysia for providing the financial support for the research.

REFERENCES

Aguilera, A, and Ayala, D. (1997). Orthogonal Polyhedra as Geometric Bounds In Constructive Solid Geometry. In C. Hoffman, and W. Bronsvort, (ed.), Fourth ACM Siggraph Symposium on Solid Modeling and Applications, Vol.4: 56-67.

Aguilera, A. (1998). Orthogonal Polyhedra: Study and Application. Ph.D. Thesis, LSI-Universitat Politècnica de Catalunya.

Arens, C., Stoter, J.E., and van Oosterom, P.J.M. 2005. Modelling 3D Spatial Objects In a geo-DBMS Using a 3D Primitive. In Computers & Geosciences, volume 31, 2. pp. 165-177

Cardelli, L. and Wegner, P. (1985). On Understanding Types, Data Abstactions, and Polymorphism. ACM Computing Survey. 17(4):471-522.

Chen, T.K., and Abdul-Rahman, A, (2006) “0-D Feature In 3D Planar Polygon Testing for 3D Spatial Analysis”, Geoinformation Science Journal, Vol. 6, No.(1), Faculty of Geoinformation Science & Engineering, UTM, Malaysia, 15 p.

Liskov, B., and Zilles, S. (1974). Programming With Abstract Data Types ACM SIGPLAN Notices.

Stonebraker, M., Rubenstein, B., and Guttman, A. (1983). Application and Abstract Data Types and Abstract Indices to CAD Database. In: Proc. Of the Annual Meeting Database Week. 107-113.

Stonebraker, M. (1986). Inclusion of New Types in Relational Database System. In: Proc. Intl. Conf.on Data Engineering. 262-269.

Zlatanova, S. (2000). “3D GIS for Urban Development.” PhD Thesis, ITC, The Netherlands, 222 p.