EXTENDING THE EXISTING GEO-DBMS TO THIRD DIMENSION: THE 3D DATA TYPE AND SPATIAL OPERATIONS

Chen Tet Khuan, and Alias Abdul-Rahman

Institute for Geospatial Science & Technology,
Faculty of Geoinformation Science and Engineering,
81310 UTM Skudai, Malaysia
{kenchen14, alias.fksg}@gmail.com

Abstract

Next generation of geo-DBMS would be given attention on extending its dimensionality and functionality in order to fill the demand in 3D applications, e.g. geosciences, subsurface, AEC, and etc. Current 3D GIS offer predominantly 2D functionality with 3D visualization and navigation capability. However, promising developments were observed in the DBMS domain where more spatial data types, functions and indexing mechanism were supported. In this aspect, DBMS are expected to become a critical component in developing of an operational 3D GIS. However, extended research and developments are needed to achieve native 3D support at DBMS level. One of the desired components in such future software or system is the geometric modeling that works with 3D spatial operations. The fundamental aspects of the 3D spatial operations are still not much been addressed 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 models within geo-DBMS environment. In the experiment, we utilize an existing geo-DBMS, PostgreSQL, later known as PostGIS, that compliant to the standard specifications from Open Geospatial Consortium (OGC), e.g. abstract and geometry specification. The main reason about utilizing the PostGIS is because it is an open source based technology and suitable for educational purposes. In this paper, we discuss a suitable way of developing a new 3D data type, polyhedron, using C language. This polyhedron is a 3D equivalent of a set of polygon that bounds a solid object, in which by connecting them in a proper manner and sharing a common edge between two adjacent polygons. These new data types will be used as inputs for 3D spatial operations. The research focuses two types of 3D spatial operations, i.e. computational-geometry, and metric operations. The computational-geometry operations manipulate the coordinate triplet using the geometrical modeling approach, e.g. 3D intersection, 3D difference, 3D XOR, and 3D union. The metric operation deals with the mathematical computation, e.g. volume calculation of a polyhedron. The approach works and we highlighted the results by using the simple data sets. The research shows that the essential research findings are applicable for real world objects and provides a solution towards a full 3D analytical operation in future.

Key Words: Geo-DBMS, 3D data type, spatial operation and 3D GIS

1.0 INTRODUCTION

An optimum way to manage with geographic objects and images of immense size with high performance is to use of spatial DBMS technology. When a DBMS offers a native geometry type together with supporting capabilities (such as spatial indexing), it is referred to as a spatial DBMS (Paredaens, 1995; and Schneider, 1997). GIS has become an important computerized application that major DBMS vendors have extended their DBMS products with native geometry types as well as with supporting capabilities. Oracle's dedicated spatial product, Oracle Spatial, provides Oracle's SDO_GEOMETRY in order to manage spatial object within DBMS. Other DBMS packages, such as IBM DB2 have spatial extenders, and open source DBMS packages such as PostgreSQL now include “spatial” capabilities as well. There are two kinds of elements, which will be stored into the spatial DBMS. One type of data is the geometric information that defines objects that represent the abstract of real world. This type of geometric information is often simply called the geometry of the spatial object. It specifies the shape and location of the points, lines and areas of which the object consists. The second type of information is the attributes information that may be attached to geometric objects. For example, an object (polygon) that consists of a set of coordinates triplet providing the name from a country, may also have attributes that give information about its area. In nature, spatial data types are not provided by DBMS. Hence, a DBMS is named as spatial DBMS if spatial data types are considered. A special column is used to stored spatial data types within spatial DBMS, namely spatial column. However, special program would be needed to visualize the geometry. There are other ways to store and retrieve spatial information than offered by DBMS (Ingvarsson, 2005). File format such as ESRI shape file (SHP) and Drawing eXchange Format (DXF) used in CAD systems implement file system to store geometry data. However, the concept of spatial DBMS offer several advantages batter than file based solution (Shekhar & Chawla 2003). These advantages come from the capability of DBMS system and architecture, which it manage to handle large amounts dataset with different data structures. Besides, DBMS able to perform complex spatial query on the data with near instant results, and provide concurrency control, e.g. locking mechanism and consistency checks, enabling multi-users.

Although DBMS, by nature, is not meant to be a core component of GIS architecture that used to maintain geographic data, it can be evolved so that it manages to perform spatial data handling and functionalities (Breuning and Zlatanova, 2006; Zlatanova and Stoter, 2006). Therefore we believe much GIS functionality can and have to be implemented within DBMS environment. The DBMS then becomes an important medium for spatial data maintenance, spatial operations and integration purposes (e.g. data access from different front-ends). The two major aspects of DBMS functionality that define a Geo-DBMS are then spatial data and spatial operations and functions built on these data types.

Many researches have been working for providing support to 2D spatial data types and operations in mainstream DBMS. Few years ago, almost all DBMS vendors implemented the geometry models as defined by Open Geospatial Consortium (OGC). For example, Informix (2006) supports three basic spatial data types: point, line and polygon; Ingres (2007) supports one more data type: circle, beside the three basic types; Oracle Spatial (2007) not only able to handle points, lines, polygons and circles, but also gives further support to arc strings and compound polygons. These DBMSs manage spatial objects, together with their 3D coordinates, i.e. 2D objects (2D polygon + z-coordinate) are embedded in 3D space (Zlatanova, 2000). Beside spatial data types, the spatial DBMS also implement operators and functions on the spatial data types, and thus spatial queries are possible at DBMS level. There are also functions, which can return list with coordinates, complete basic geometric transformations, maintain geometry validity, etc. Although coordinates in the data types can be 3D, the functions on spatial data types are still 2D, the z-values are hardly considered. Actually the lack of “true” 3D data type (i.e. 3D tetrahedron, 3D polyhedron) results in the unavailability of 3D operations in DBMS.

The first attempt of 3D spatial data type and corresponding operations in a spatial DBMS has been investigated quite successfully by Arens 2003, Arens et al 2005. The basic idea is that a 3D polyhedron could be defined as a bounded subset of 3D space enclosed by a finite set of flat polygons, such that every edge of a polygon is shared by exactly one other polygon. Here, the polygons are in 3D space because they are represented by vertices, which can be 3D points in a spatial DBMS. This research has been studied and some concepts are taken over in the new 3D data type implemented in Oracle Spatial 11.

Yet, another attempt to define 3D object is reported by Penninga, 2005. The 3D object, i.e. tetrahedron is used for representing volumetric shapes. The tetrahedron is the simplest possible geometry in 3D domain.

The conceptual design is intended for implementation for both geometry and topology model (Penninga et al, 2006).

Some other related research is reported by Pu (2005), who has created complex geometry types, i.e. freeform curves and surfaces. Although freeform shapes can be simulated by tiny line segments/triangles/polygons, it is quite unrealistic and inefficient to store all these line segments/triangles/polygons into a DBMS especially when shapes are rather huge or complex. Therefore, Pu and Zlatanova (2006) stored freeform shapes directly in DBMS. Corresponding spatial operators and functions that manage these shapes are also tested.

With these efforts, the possibility of developing 3D data types in DBMS is obvious. Currently missing are the spatial operations for 3D data type. Current DBMS only provides 2D operations, which the z-value is not considered. 3D spatial operations are also rather limited in typical GIS software. In this paper, we focus on simple but complete strategy in creating new datatype, i.e. polyhedron. Furthermore, the 3D datatype is used to test the developed spatial operations for 3D GIS. The algorithm fully covers the third dimension and able to be applied in 3D situation. The 3D spatial operations are tested for PostgreSQL and can theoretically become a part of the PostGIS.

The paper is organized in the following order: In Section 2, short discussion involves the criteria to select an appropriate DBMS for the research is given. The chosen DBMS module should be able to refered to the educational purposes. The method to create functions for new 3D datatype and spatial operation described in Section 3. The research proposes two kinds of spatial operations, i.e. computational-geometry and metric operations. The computational-geometry operations manipulate the coordinate triplet using the geometrical modeling approach, e.g. 3D intersection, 3D difference, 3D XOR, and 3D union. The metric operation deals with the mathematical computation, e.g. volume calculation of a polyhedron. The experiment and discussions are presented in Section 4 and the research is concluded with some future work remarks in Section 5.

2.0 DBMS SELECTION

Existing DBMS provides a SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial features. Most of the existing spatial databases support the object-oriented model for representing geometries. The benefits of this model is that it support for many geometry types, including arcs, circles, and different kinds of compound objects. Therefore, geometries could be modeled in a single row and single column. The model also able to create and maintain indexes, and later on, perform spatial queries efficiently.

Conventional DBMS offer spatial data types and spatial functions fully in two-dimension. Storing spatial data and performing spatial analysis can be completed with SQL queries. The spatial data types and spatial operations reflect only simple two-dimensional features. Lately, 2D spatial objects have been extended and embedded in 3D space. The support of 3D coordinates allows for alternatives in management of 3D features. With this spatial extend, DBMS have immediately been challenged by the third dimension. The volumetric 3D data type (e.g. polyhedron, and tetrahedron) is expected to be available in the coming Oracle 11g. However, certain DBMSs provide the support of creating new data type using native programming. For instances, Oracle and PostgreSQL support Java and C++ native programming. Consequently, function for validation and 3D operators remains self-responsible by the users. The conclusions, some DBMSs offer 2D data types (basically point, line, and polygon) but support 3D/4D coordinates and offer a large number of functions more or less compliant with the Open Geospatial Consortium (OGC, 2001) standards. Most of the functions are only 2D dimensional. Several geo-DBMS, e.g. MySQL, and PostGIS, however, supports very limited 3D operations, but only limited to certain regular 3D volumetric object, i.e. 3D box. In the next section, some commercial spatial database will be discussed, in term of their characteristics, capabilities and limitations in handling multi-dimensional datasets. The last section will provide reasons in order to decide which geo-DBMS was selected to be implemented into this study. The criteria to select one DBMS as testbed was considered in some aspects as follows:

1. Commercial aspect – As this research was aim to produce critical development that manage to fill the gaps of spatial DBMS in the context of 3D, the chosen DBMS should be able to provide an useful module for educational purposes. One of the important issues is that the open-source issue was the primary target for this research. The distribution terms of open-source tools / software must comply with free redistribution. The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. Besides, it shall not require a royalty or other fee for such sale. Other than licensing aspect, the program must include source code, and must allow distribution in source code as well as compiled form. If some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a user would modify the program. The benefit of using the open-source module is that it allows modifications and derived works, and allows them to be distributed under the same terms as the license of the original software.
2. OGC compliant aspect – Standardization in GIS is very important for the interoperability. The purpose of the OGC compliance module to be chosen is to permit any user to take advantage of the standards that OGC has created. This important aspect should be able to provide a research direction that follows the OGC's abstract and implementation specifications. The key to all of this is "interoperability", that is the ability for different kinds of software to successfully interact with one another. An applicable example is to build a GeoSpatial web, similar in scope to the World Wide Web, where anyone just needs a Web Server, such as Apache or IIS, and then others can start interacting with their information. The advantage over high interoperability is that data exchange effort become less and ensure same data format could be accessed easily to any user. One of the best things the OGC has done is define a standard specification for data type and spatial operations. And beyond that OGC have made every effort to be as easy as possible to set up a standards compliant module, with no additional configuration needed to meet the open standards for GIS development.
3. Native programming support – Several DBMSs were designed for non-spatial data only. Until very recent development of geospatial modelling, DBMS has been linked with spatial data, where a specific column is meant for managing the geographic object. Taking the advantages storing data logically in the form of tablespaces and physically in the form of data files, spatial data can be managed securely due to the available tool for checking the data integrity. However, several geo-DBMSs provide very limited geospatial data type, respectively to the dimensionality. The native programming support is the most appropriate way to create user-defined data types and functions within geo-DBMS environment. By extending the geo-DBMS with custom data type and functions, this study could design the new data type and spatial operations according to the proposed structure that follows the standard specifications given by OGC. Some example of native programming support are C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, etc.
4. Portability aspect - For the general usability purposes, portability gives an important mean of software installation with as low requirement as it can, and easy-to-use. It is also the ability to install to other platforms, i.e. different operating systems, with less data size consumed.

In this study, several DBMSs are considered, e.g. Oracle Spatial, MySQL, and PostgreSQL. These chosen DBMS are evaluated due to their capabilities in handling spatial data are greater than other conventional DBMSs. However, the final selection for the research is PostgreSQL. The reasons to support the chosen DBMS are based on four factors as mentioned in the previous section.

The first perceived advantage of open source models is the fact that open source software is made available without fee, as what PostgreSQL provided to user. The availability of the source code and the right to modify it is very important. The source-code of PostGIS could be found at PostGIS official web page (http://postgis.refractions.net/download/). It enables the unlimited tuning and improvement of a software tool. It also makes it possible to port the code to new hardware, to adapt it to changing conditions, and to reach a detailed understanding of how the system works. This is why many experts are reaching the conclusion that to really extend the lifetime of an application, it must be available in source form. In fact, no binary-only application more than 10 years old now survives in unmodified form. Source code availability also makes it much easier to isolate bugs, and (for a programmer) to fix them.

The PostGIS is released under the GNU General Public License (GPL), which is the directly referred to open-source license. The right to redistribute modifications and improvements to the code, and to reuse it, permits all the advantages due to the modifiability of the software to be shared by large communities. This is usually the point that differentiates open source software licenses from the commercial one. In substance, the fact that redistribution rights cannot be revoked, and that they are universal, is what attracts a substantial crowd of developers to work around open source software projects. This ensures a large population of users, which helps in turn to build up a research and development medium for support and customization of the software, which can only attract more and more developers to work in the project. This in turn helps to improve the quality of the product, and to improve its functionality. With this point, the utilization of PostgreSQL in educational purposes is more appropriate as students can benefit from it without any fee. Thus, the study was also aimed to produce an extension that part of PostGIS module, which add-on the 3D spatial operation for spatial DBMS.

Another advantages of utilizing PostgreSQL, following of the advantage of easy-access to source code is that it supports many languages, e.g. SQL, PL/pgsql, PL/perl, PL/python, PL/tcl, PL/PHP, PL/R, PL/Java, PL/J, C/C++, and etc. Most of the spatial data types found in the source code of PostGIS were created using C/C++ language. The native programming support in C/C++ language is the most appropriate way to create user-defined data types and functions within geo-DBMS environment. By extending the geo-DBMS with custom data type and functions, this study could design the new data type and spatial operations for 3D object, i.e. polyhedron.
The third factor related to the portability issue. The notion of portability is widely used, and it's often attached to software inappropriately. Software is only portable if it can actually be moved to a different platform. The operating system portability is somehow the most troublesome issues, because operating systems vary much more widely than compilers, CPU architectures and build environments. Sometimes, it's often simply not possible to work around with a common operating system, e.g. Windows or UNIX, in the same way that it can be resolved with another specific operating system, like Apple Mac OS. However, PostgreSQL work fine in majority of the operating system platforms, and its hardware requirement is rather low compared to the current computer technology.

The OGC standard specification for SQL schema is to insert, query, manipulate and delete spatial objects That is one of the reason of choosing PostGIS for this study, is that the PostGIS follows the OGC (1999) "Simple Features Specification for SQL" and has been certified as compliant with the data types and functions profile. The purpose of implementing this specification to PostGIS is to define a standard SQL schema for geospatial object. Simple geospatial feature collections will conceptually be stored as tables with geometry valued columns in DBMS environment, each feature will be stored as a row in a table. The non-spatial attributes of features will be mapped onto columns whose types are drawn from the set of standard ODBC/SQL92 data types. The spatial attributes of features will be mapped onto columns whose SQL data types are based on the underlying concept of additional geometric data types for SQL.

3.0 USER-DEFINED DATA TYPE AND OPERATION

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

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 most important constraint is all the polygons that make up the polyhedron have to be flat. This means that all points used to construct a polygon must be in the same plane. Figure 1 denotes a sample of a planar and non-planar polygon. The characteristics of a valid polyhedron are given in Aguilera & Ayala (1997), and Aguilera (1998), and simplified in Chen *et al* (2007).

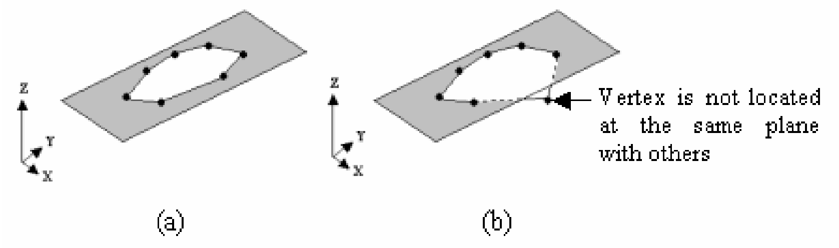

Figure 1: a) Planar polygon, and (b) non-planar polygon

In PostgreSQL environment, user-defined data type is developed using C language. Suppose user wants to define a type, *polyhedron*, 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 *polyhedron* 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 Functions/Operations

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();
  }
```

3.3 Compiling and Linking Dynamically-Loaded Functions

Before the implementation of PostgreSQL extension functions written in C, they must be compiled and linked in a special way to produce a file that can be dynamically loaded by the server. To be precised, a *shared library* needs to be created. First, the source files are compiled into object files, then the object files are linked together. The object files need to be created as *position-independent code* (PIC), which conceptually means that they can be placed at an arbitrary location in memory when they are loaded by the executable. The dynamic loading feature involves 2 processes:

- *Dynamic loading*: `gcc -fpic -c Polyhedron.c`
- *Loading and link editing:* `gcc -shared -o Polyhedron.so Polyhedron.o`

The methodology of creating user-defined datatype and function/operation are presented in flowchart as follows: (see Figure 2).

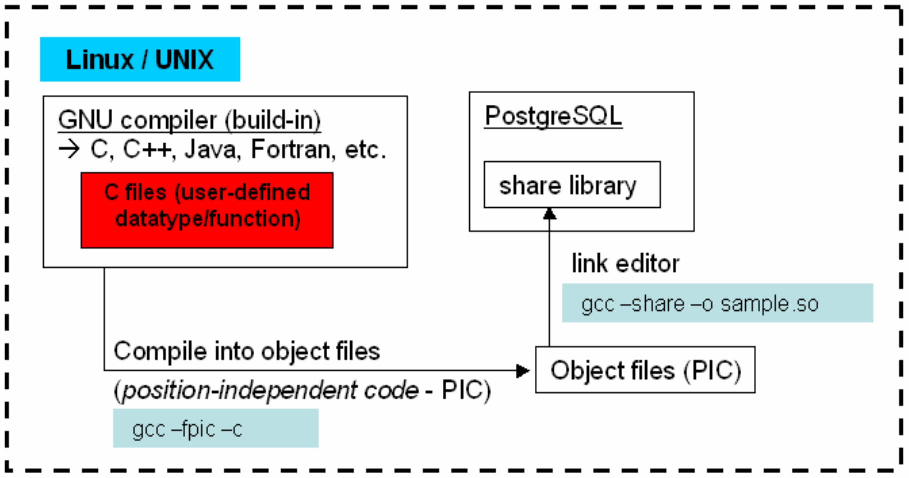
Linux / UNIX
GNU compiler (build-in)
→ C, C++, Java, Fortran, etc.
C files (user-defined datatype/function)
PostgreSQL
share library
link editor
gcc –share –o sample.so
Compile into object files
(position-independent code - PIC)
gcc –fpic –c
Object files (PIC)

Figure 2: Workflow of creating user-defined data type/function in PostgreSQL

4.0 EXPERIMENT AND DISCUSSIONS

The 3D spatial operations were tested within PostgreSQL environment. A 3D data type, i.e. POLYHEDRON was created and several 3D functions were implemented. The following SQL syntax denotes a sample of a polyhedron within PostgreSQL:

```
SELECT * FROM GM_BODYTABLE WHERE PID = 1;

1,POLYHEDRON(PolygonInfo(6,24),SumVertexList(8),SumPolygonList(4,4,4,4,
4,4),VertexList(100.0,100.0,100.0,400.0,100.0,100.0,400.0,400.0,100.0,1
00.0,400.0,100.0,100.0,100.0,400.0,400.0,100.0,400.0,400.0,400.0,400.0,
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 (see Figure 3):

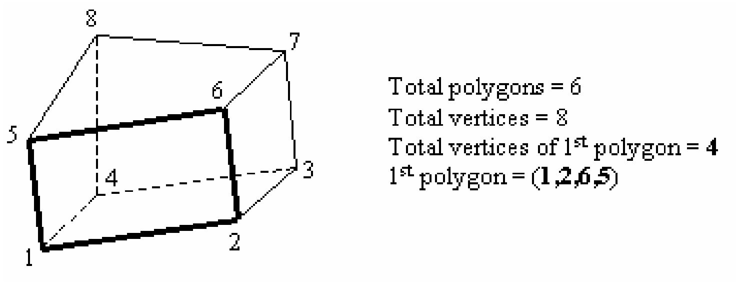

Figure 3: Sample structure of a polyhedron for geometrical data type

4.1 Computational-geometry Operations

For computational-geometry operations, the following SQL statement runs the 3D Difference (see Figure 4). The other computational-geometry operation results are given in Figure 5.

```
SELECT GMDIFFERENCE3D (a.POLYHEDRON, b.POLYHEDRON) AS GM_DIFFERENCE3D
FROM test a, test b where a.PID=1 and b.PID=2;
```

The result:

```
GM_DIFFERENCE3D
-----------------------------------------------------------------------
('POLYHEDRON(PolygonInfo(9,42),SumVertexList(14),SumPolygonList(4,6,6,4
,6,4,4,4,4),VertexList(100,100,100,400,100,100,400,100,400,100,100,400,
400,400,100,400,400,300,400,300,300,400,300,400,100,400,100,100,400,400
,300,400,400,300,400,300,300,300,400,300,300,300),PolygonList(1,2,3,4,2
,5,6,7,8,3,5,9,10,11,12,6,9,1,4,10,4,3,8,13,11,10,1,9,5,2,14,7,8,13,12,
14,13,11,14,12,6,7))')
```

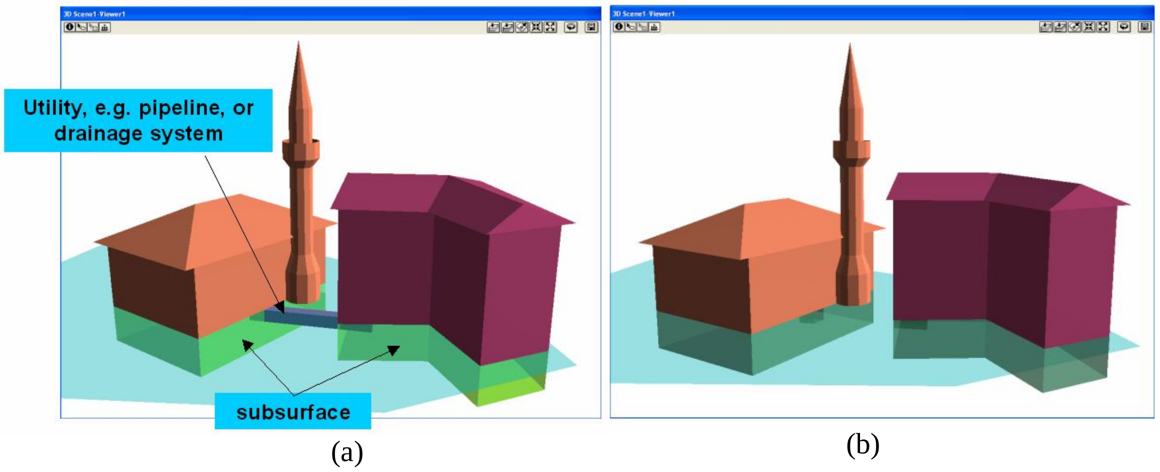

Figure 4: (a) Real input dataset, and (b) result from 3D Difference

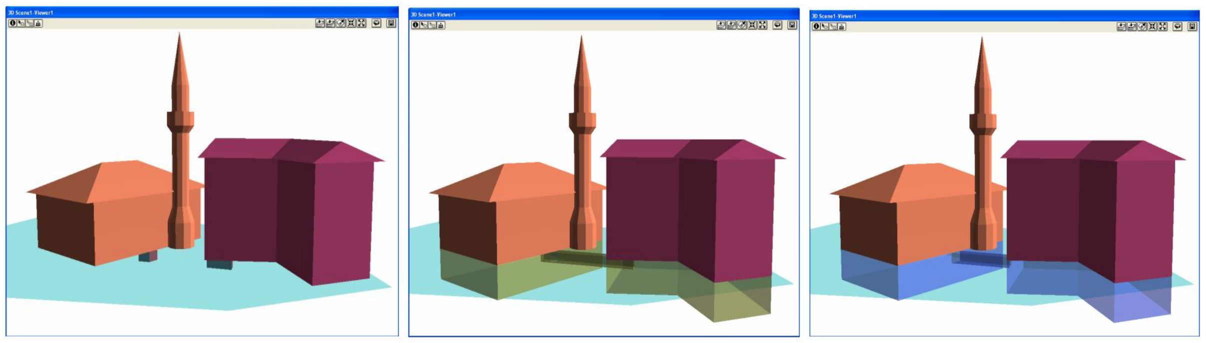
(a) (b) (c)

Figure 5: Result from (a) 3D Intersection, (b) 3D XOR, and (c) 3D Union

4.2 Metric Operations

For metric operation, the volume calculation of polyhedron is based on the polyhedron 1 from Figure 14. The SQL query is given as below:

```
SELECT VOLUME3D(POLYHEDRON) AS VOLUME_3D FROM test where PID=1;
```

The result:

```
Volume_3D
-----------------
27000000.0 meters
```

5.0 CONCLUDING REMARKS

The paper presents an approach for extending the geo-DBMS to the third dimension by adding the 3D spatial data type and operations. The approach is expected to be providing complete modeling for 3D GIS analysis. The results have shown that implementation of a 3D data type and functions allowing 3D GIS analysis are possible.

Our concept was tested within PostgreSQL computing environment and has provided a promising outcome with respect to the developed algorithms. Future research will concentrate on interfacing the geo-DBMS as the conventional DBMS is considered not as user-friendly as other software modules that provide interfaces. Utilizing the operating system terminal to operate the DBMS functionalities would affect many GIS tasks as the situation goes more complicated, e.g. data modelling and integration. However, the strength of DBMS remains unquestioned due to its high stability and security within the system architecture.

DBMS is a very important medium for GIS that able to connect many different components of GIS, e.g. visualization, web-GIS, etc. A very important issue still need to be addressed is visualization of the result of 3D operations. Appropriate graphical visualization is especially important for 3D in order to get a better perception of the result of the query. Some topics to be considered are: 1) direct access to the new data type from GIS, avoiding first export to a shape file, 2) direct connection with CAD/CAM software, e.g. Microstation and Autodesk Map 3D to be able not only to visualize but also edit, 3) user-defined environment, where user develops display tool that manage to retrieve and visualize data from DBMS, or 4) access via Internet, using e.g. WFS. 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.

REFERENCES

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

Aguilera, A. (1998). Orthogonal Polyhedra: Study and Application. PhD thesis, LSI-Universitat Politècnica de Catalunya.

Arens, C.A. (2003). Modelling 3D Spatial Objects In a Geo-DBMS Using a 3D Primitives. Msc thesis, TU Delft, The Netherlands.

Breuning, M., Zlatanova, S. (2005). 3D Geo-DBMS, Chapter 4. In: S. Zlatanova, & D. Prosperi (Ed.), Large-scale 3D data integration: challenges and opportunities, Taylor & Francis, A CRC press book. pp. 88-113.

Chen, T.K., Abdul-Rahman, A., and S. Zlatanova, (2007). New 3D Data Type and Topological Operations for Geo-DBMS. In: V. Coors, M. Rumor, E. Fendel, and S. Zlatanova (Eds.): Urban and Regional Data Management, UDMS Annual 2007, Taylor Francis Group, London. pp. 211–222.

Ingres Documentation. URL: *http://www.ingres.com/downloads/documentation.php*. Access since 2007.

Ingvarsson, T. M. (2005). CCDM and Open Source Applications. Master's Thesis, TU Delft. 147 p.

MySQL 6.0 Reference Manual. URL: *http://dev.mysql.com/doc/refman/6.0/en/index.html*. Access since 2007.

OGC. (1999). Simple Features - SQL, OGC document 99-049.

OGC. (2001). Abstract Specification Topic 1 – Feature Geometry (same as ISO19107), OGC document 01-101.

Oracle Spatial Documentation. URL: *http://www.oracle.com/technology/documentation/spatial.html*. Access since 2007.

Paredaens, J. (1995). Spatial Databases, The Final Frontier. Proc. 5th Intl. Conf. on Database Theory. 14-32.

Penninga, F. (2005). 3D Topographic Data Modelling: Why Rigidity Is Preferable To Pragmatism. In: Spatial Information Theory, Cosit'05. Lecture Notes on Computer Science, Springer. 3693: 409-425.

Penninga, F., van Oosterom, P.J.M., Kazar, B.M. (2006). A TEN-based DBMS Approach For 3D Topographic Data Modelling. In: Spatial Data Handling 2006.

Pu, S. (2005). Managing Freeform Curves and Surfaces In A Spatial DBMS. Msc Thesis, TU Delft.

Pu, S., and Zlatanova, S. (2006). Integration of GIS and CAD at DBMS Level. In: E. Fendel E, Rumor M (Ed.) Proceedings of UDMS'06. Aalborg, Denmark, 9.61-9.71.

PostGIS. PostGIS Manual, Refractions Research. URL: *http://postgis.refractions.net/docs/*. Access since 2007.

Schneider, M. (1997). Spatial Data Types for Database Systems. LNCS 1288. Springer-Verlag, Berlin-Heidelberg-New York.

Shekhar, S. and Chawla, S. (2003). Spatial Databases: A Tour. Pearson Education Inc., New Jersey. 262 p.

Zlatanova, S. (2000). 3D GIS for urban development. PhD thesis, ITC, The Netherlands.