# UNIVERSITI TEKNOLOGI MALAYSIA

## BORANG PENGESAHAN
## LAPORAN AKHIR PENYELIDIKAN

TAJUK PROJEK : **Mathematical Set and Euler theories for formalizing new concept of spatial data modelling for hybrid 3D spatial objects.**

Saya **Assoc. Prof. Dr. Alias Abdul Rahman**
**(HURUF BESAR)**

Mengaku membenarkan **Laporan Akhir Penyelidikan** ini disimpan di Perpustakaan Universiti Teknologi Malaysia dengan syarat-syarat kegunaan seperti berikut :

1. Laporan Akhir Penyelidikan ini adalah hakmilik Universiti Teknologi Malaysia.
2. Perpustakaan Universiti Teknologi Malaysia dibenarkan membuat salinan untuk tujuan rujukan sahaja.
3. Perpustakaan dibenarkan membuat penjualan salinan Laporan Akhir Penyelidikan ini bagi kategori TIDAK TERHAD.
4. * Sila tandakan ( / )

| | | |
|---|---|---|
| ☐ | SULIT | (Mengandungi maklumat yang berdarjah keselamatan atau Kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972). |
| ☐ | TERHAD | (Mengandungi maklumat TERHAD yang telah ditentukan oleh Organisasi/badan di mana penyelidikan dijalankan). |
| ☑ | TIDAK TERHAD | |

Ab. A. Rahman
TANDATANGAN KETUA PENYELIDIK

PROF. MADYA DR ALIAS ABDUL RAHMAN
TIMBALAN DEKAN (AKADEMIK)
Fakulti Kejuruteraan dan Sains Geoinformasi
Universiti Teknologi Malaysia
81310 UTM, Skudai
TEL : 07-5530806

Nama & Cop Ketua Penyelidik

Tarikh : 29/5/2008

**CATATAN :** **Jika Laporan Akhir Penyelidikan ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali sebab dan tempoh laporan ini perlu dikelaskan sebagai SULIT dan TERHAD.*

## ABSTRACT

### FUNDAMENTAL ASPECT OF 3D SPATIAL OBJECTS VALIDATION VIA SET AND EULER THEORIES

*(Key words: 3D spatial objects, DBMS, set theory, and Euler theory)*

This research attempts to explore two mathematical theories, namely, set and Euler theories for validation of 3D spatial objects which were generated or constructed from triangular irregular network. The relationship between the theories and spatial database development of the objects also investigated. Validating of 3D objects was restricted to 3D solid only, other primitives of validation such as solid and surface, solid and line, etc are not part of the study. The validation process of the 3D objects was carried out by developing a computer program based on C++ and couple with open source DBMS such as PostgreSQL. The results show that the theories could be utilised for validating 3D spatial objects.



**Researchers:**

Assoc Professor Dr Alias Abdul Rahman

Chen Tet Khuan

**Email:** {alias, kenchen}@fksg.utm.my

**Tel**: 07-5530806



**Vote No**. 78062

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

## 1.1 General Introduction

Nowadays, 2D GISs are common. 2D dataset involves in the geometrical modeling and spatial relationships are also well defined. In most of the tasks related to any 2D GIS applications are quite straightforward and can be solved in a very efficient manner (Abdul-Rahman, *et al.* 1998). However, some applications which involve the third dimension, such as geologic & geo-science explorations, meteorology, underground construction or even seabed profiling need 3D GIS. The available 2D GIS tools are no longer sufficient to represent the real world. If one insists to apply 2D tools into 3D situations, this definitely will limits the scientific work either in term of visualization of the volumetric object or even data manipulation for 3D analysis. The limitations of current 2D GIS for 3D situations have been discussed by several authors such as Smith & Paradis (1989), Verbree *et al.* (1999), Taylor (2000), and Zlatanova *et al.*(2004). For example, any 2D display tools are unable to make use of 3D datasets and provide 3D environment view due to the absent of the 3$^{rd}$ dimension. Besides, 3D datasets only are treated as 2D data in terms data manipulation. With all these issues, the third dimension should be added into 2D GIS in order to solve these problems. Yet, Zlatanova (2000) stated that the

3D conceptual spatial model, topological relationships, data collection, and spatial analysis might comprise a wide spectrum of questions.

With these efforts, the possibility of implementing set and Euler theories in 3D GIS is obvious and could be done by extending the existing 2D spatial data type to 3D. Therefore, some important questions are raised:

1). **What are the fundamental set and Euler theories that can be implemented for 3D spatial modeling?**
2). **How to implement the set and Euler theories within DBMS?**

Therefore, from the foregoing discussion, it can be seen that many issues need to be investigated. In this project, only part of the problems attempt to be investigated, that is to investigate the possibility of set and Euler theories for 3D GIS within DBMS environment. First, a new 3D data type, i.e. polyhedron, will be defined, which implements the set theory for object primitives and features. Later on, the 3D topological data structure will be designed within geo-DBMS (in this context, PostGIS (2006) is used). The 3D spatial object will be validated using the Euler theory. Recall the ultimate goal of this research is to develop 3D spatial spatial modelling using set and Euler theories, 3D visualization is out of the scope of this research.

## 1.2 Problem Statement and Motivation

Topology deals with object's semantics and relationship, either within feature itself or with other feature objects. Topological properties are those that are invariant to topological transformations, i.e. those properties, which do not change after transformations like rotation, translation, scaling, and rubber sheeting. Topology can

be considered the most primitive kind of spatial information, since a change in topology implies a change in other geometric aspects, while the opposite is not true. The strength of topology comes from its mechanism, which able to reduce redundant data, provide object relationship for topological analysis (spatial query). Besides, other useful application is to accelerate/skip the computational geometry process. This could be accomplished as feature instances and geometric object instances are associated explicitly. This method manages to produce fast topological query because the geometrical calculations such containment (point-in-polygon), which is computationally intensive process is not involved. The combinatorial structures, which known as topological complexes are implemented to convert computational geometry algorithms into combinatorial algorithms. The methodology of combining topological primitives, i.e. nodes, edges, or faces, to construct a topological complex, e.g. solid, will be involved into combinatorial algorithms. In GIS, topology is commonly used in DBMS in order to enhance the functionalities of topological analysis, e.g. object's construction and validation, relationship, etc.

The 3D spatial data modeling that implement set and Euler theories for topological structure are not available in DBMS. Two researches had been done in creating new 3D data types in DBMS: 3D polyhedron (Arens, 2003) and 3D freeform object (Pu, 2005). Both user-defined data types were designed and mapped into DBMS using external program. However, the 3D spatial data modeling does not implement the set and Euler theories. The implementations of both theories are important for topological data structure and data type validation.

## 1.3 Research Objectives

To answer the questions raised from section 1, the research direction should be referred to the 3D spatial data modeling implementing set and Euler theories. Therefore, the objectives of the research are:

1. Since the 3D spatial data modeling implementing the set theory is rather limited within DBMS, the research is to investigate the possibility of developing 3D topological data structure within the DBMS environment;
2. Since validation function that implements Euler theory is important to validate the correct input data, the research also investigates the Euler theory to validate the defined 3D spatial data type from the topological structure.

## 1.4 Research Methodology

The methodologies of the research are to:

1. Implement the C language to create new user-defined data type for 3D object, 3D polyhedron for topological data structure. The data structure will implement the set theory that will ensure the produced 3D spatial data type could be used for an input validation function.
2. Validate the produced 3D spatial object from the 3D topological data structure using the Euler theory. The rule for the validation function should clearly define the valid 3D spatial object within DBMS environment.
3. The design is mapped using the C language (3D validation function) into DBMS. The Linux operating system and PostgreSQL will be implemented.
4. Visualize the 3D spatial object using ESRI ArcGIS module.

## 1.5 Research Scope

The research concentrates on 3D spatial operations for DBMS. Several fundamental considerations outline the area of research as follows:

1. Since the research is focused on geometrical and topological models, the data structure (in defining a new 3D data type) of both models will be discussed extensively.
2. Since the absent of 3D data type in DBMS, the research aims to create a new user-defined data type for 3D spatial object, 3D polyhedron. With this effort, C language is used to define the new data type, whereas PostgreSQL will be used to provide the DBMS environment.
3. Since the absent of validation function that implement Euler theory for 3D spatial data type in DBMS, the research will attempt to create a validation function to validate the data input for 3D spatial object.
4. Questions related to data collection, dataset preparation, and 3D visualization are not treated explicitly in the research.

## 1.6 Issues to be Considered

1. How to implement the set theory for 3D topological data structure for 3D spatial data modelling?
2. How to implement the Euler theory for validating the designed 3D spatial data type, i.e. polyhedron?

## 1.7 Organization of Thesis

This report may be divided into four major parts. Part one includes the introduction, the elaboration of the status and prospects of 3D spatial data modeling for DBMS. This part comprises chapters 1, 2, and 3. The second part reports the

background theories of set and Euler theories. This part focuses on the supported theories for 3D spatial data modeling. This part is included in the chapters 4. The third part is the implementation and testing phase. It demonstrates how the design of 3D data type came into practice and explains the validation function for 3D spatial data object. This part includes chapter 5. Finally chapter 6, the concluding part, summarizes the most important achievements of the report.

Chapter 1 discusses the introduction for research. The chapter also discussed in brief the current DBMS status with respect to the defined scope is given, leading to the identification of the remaining problems and the objectives of the research.

Chapter 2 discusses the existing DBMS in general. A detailed review of current DBMS status is carried out, in term of spatial aspect, and data model. The research also selects one existing module as a DBMS platform.

Chapter 3 relates some reviews of OGC standard for 3D spatial data modeling. The standards include abstract specification and implementation issues for DBMS.

Chapter 4 discusses the background theory relating to the set and Euler theories. The theories will be used to extend the existing 2D spatial data modeling to 3D. The validation function that implements the Euler theory also being included in this chapter.

Chapter 5 discusses the aspects of the approach in developing 3D spatial data and topological structure implementing the set theory. The data structure for user-defined 3D data type in DBMS is also mentioned in this chapter.

Chapter 6 concludes with the major findings of the research and recommendations of issues for future research.

## 1.8 Expected Findings and Contributions

The contributions to knowledge as a result of this research are:

- Investigation of 3D spatial data modelling implementing set for topological data structure, and Euler theory for validation function within geo-DBMS environment.
- Documentation of a methodology for the 3D spatial data modeling for 3D GIS.

The findings and contributions of this thesis are applicable and useful to:

- Commercial DBMS for 3D GIS

## 1.9 Conclusion

This study was motivated by the need to address issue such as: What are the important 3D spatial data modelling issues should be discussed? Does that apply in any 3D topological data structure? Hence, this study creates new user-defined data type for 3D spatial object, polyhedron, in DBMS environment. Comprehensively, the approach of developing 3D spatial data modelling implementing set and Euler theories within DBMS environment will be described. The entire procedures related to the databasing, validation function, and the experiment will be presented.

# CHAPTER II

# DBMS

## 2.1 Introduction

The second stage of the research started with a review existing DBMSs, in terms of their characteristics and functionalities in managing data. A database management system (DBMS), or database manager, is a program that lets one or more computer users create and access data in a database. The DBMS manages user requests (and requests from other programs) so that users and other programs are free from having to understand where the data is physically located on storage. In handling user requests, the DBMS ensures the integrity of the data (that is, making sure it continues to be accessible and is consistently organized as intended) and security (making sure only those with access privileges can access the data). The most typical DBMS is a relational database management system (RDBMS). A newer kind of DBMS is the object-oriented database management system (ODBMS). Common DBMS implements a standard user and program interface, called Structured Query Language (SQL) to execute users request.

## 2.2 DBMS in General Terminology

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. A DBMS can be thought of as a file manager that manages data in databases rather than files in file systems. In IBM's mainframe operating systems, the non-relational data managers were (and are, because these legacy application systems are still used) known as access methods. IBM's Information Management System (IMS) was one of the first DBMSs. A DBMS may be used by or combined with transaction managers, such as IBM's Customer Information Control System (CICS).

There are four essential elements that are found with just about every existing DBMS. The first is the implementation of a modeling language that serves to define the language of each database that is hosted via the DBMS. There are several approaches currently in use, with hierarchical, network, relational, and object examples. Essentially, the modeling language ensures the ability of the databases to communicate with the DBMS and thus operate on the system.

Second, data structures also are administered by the DBMS. Examples of data that are organized by this function are individual profiles or records, files, fields and their definitions, and objects such as visual media. Data structures are what allow DBMS to interact with the data without causing and damage to the integrity of the data itself. Some related works could be found in Hadzilacos and Tryfona (1996), Lipeck and Neumann (1987) and, Manola and Orenstein, (1986).

A third component of DBMS software is the data query language. This element is involved in maintaining the security of the database, by monitoring the use of login data, the assignment of access rights and privileges, and the definition of the criteria that must be employed to add data to the system. The data query language works with the data structures to make sure it is harder to input irrelevant data into any of the databases in use on the system.

Last, a mechanism that allows for transactions is an essential basic for any DBMS. This helps to allow multiple and concurrent access to the database by multiple users, prevents the manipulation of one record by two users at the same time, and preventing the creation of duplicate records. Related work could be found in Abel *et al.* (1995).

Various applications of DBMSs have brought to the great evolutions to the computer revolution. It can store a huge quantity of data at one place and queried with simple methods. DBMS are furthermore ideally thought of facilitating several processes, such as (Rigaux *et al*., 2002):

- Defining a database - that is specifying data types, structures and constraints.
- Develop a database - create database and store dataset.
- Manipulating a database.
- Querying a database to retrieve specific data.
- Updating a database.

To execute any command within DBMS environment, the most common language is 'Structured Query Language' (SQL), which was initially developed by IBM. RDBMS are regarded as effective tool to store and manipulate simple data that is organized in tabular form using unique keys to join or relate different tables together.

However, as RDBMS only stores simple data types, alternative solution had to be invented to handle complex data types like spatial data. The solution is to implement the object-oriented approach in the DBMS (OO-DBMS). This technology has however not gained as much popularity as first expected, but guided the development of object-relational DBMS (OR-DBMS) that somewhat combines the functionalities of the two approaches (Shekhar and Chawla, 2003).

### 2.2.1 SQL (Structured Query Language)

SQL stands for Structured Query Language. SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. Unfortunately, there are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

While the SQL Standard is often perceived as established technology, rather than the innovative, cutting edge technology it was when the standards process started in early 1980's, it is still an expanding, evolving, relevant standard.

The original SQL standard was completed as a USA ANSI (American National Standards Institute) standard in 1986, and adopted as an ISO (International Standards Organization) standard in 1987. To allow relevant pieces to progress at different rates, the SQL standard has been divided into multiple parts. Two of these parts were

completed in the 1990's, as additions to SQL-1992. SQL/CLI (Call Language Interface) was completed in 1995 and SQL/PSM (Persistent Stored Modules) was completed in 1996. Following the completion of SQL:1999, there has been significant work on SQL with Java (a Sun trademark) and XML, as well as the use of SQL to manage data external to an SQL database.

Another revision to all of the parts was completed as SQL:2003. Since SQL:2003, the SQL standards committees have expanded XML support and corrected some errors. The expanded SQL/XML standard will be completed in late 2005 or early 2006. In addition to the SQL standards, there is a separate set of specifications called SQL/MM that is a multi-media expansion of the SQL Standard.

From SQL standard language, there are several separate components in SQL structure. Two of the most important are 'Data Definition Language' (DDL) and 'Data Manipulation Language' (DML) (Shekhar and Chawla, 2003; Rigaux *et al.*, 2002).

The Data Definition Language (DDL) part of SQL permits database tables to define the data, tables, constrains and association. User can also define indexes (keys), specify links between tables, and impose constraints between database tables. The most important DDL statements in SQL are:

- CREATE TABLE - creates a new database table
- ALTER TABLE - alters (changes) a database table
- DROP TABLE - deletes a database table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

In Contrary, DML is apply SQL syntax for executing queries or used to access and edit data in a database and perform operation like INSERT, UPDATE, DELETE, and SELECT. These query and update commands together form the DML part of SQL:

- SELECT - extracts data from a database table
- UPDATE - updates data in a database table
- DELETE - deletes data from a database table
- INSERT INTO - inserts new data into a database table

### 2.2.2 Conventional DBMS data types

All DBMSs provide multiple choices of data types for the information that can be stored in their database table fields. However, the set of data types made available (typically numeric and alpha-numeric) varies from DBMS to DBMS.

The *Object ID* data type usually utilizes as primary key type for most of the DBMSs. The primary key of a relational table uniquely identifies each record in the table. The major difference is that for unique keys the implicit NOT NULL constraint is not automatically enforced, while for primary keys it is. It can either be a normal attribute that is guaranteed to be unique or it can be generated by the DBMS (such as a globally unique identifier, or GUID, in Microsoft SQL Server). Primary keys may consist of a single attribute or multiple attributes in combination. The common data type for Object ID is usually referred as NUMBER (without decimal fraction) or INTEGER.

The *Integer* data type may store integer values as large as each DBMS may handle. Fields of this type may be created optionally as signed or unsigned integers, depending on different DBMS that support it. The type of integer can be divided into INTEGER (-2,147,483,647 to 2,147,483,647) SHORT INTEGER (-32,768 to 32,767), LONG INTEGER (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

The decimal data type may store decimal numbers accurately with a fixed number of decimal places. This data type is suitable for representing accurate values like currency amounts. Some DBMS drivers may emulate the decimal data type using integers. Such drivers need to know in advance how many decimal places that should be used to perform eventual scale conversion when storing and retrieving values from a database. There are several kinds of decimal data type available in DBMS, e.g. *Float*/*Real*, and *Double*. The *Float*/*Real* data type may store floating-point decimal numbers. This data type is suitable for representing numbers within a large-scale range that do not require high accuracy. The scale and the precision limits of the values that may be stored in a database depends on the DBMS that it is used (usually referred to 32 bit floating point).The *Double* data type stores same as *Float*/*Real* data type if the data type is suitable for representing numbers that require high accuracy. The scale and the precision limits of the values that may be stored in a database depends on the DBMS that it is used (usually referred to 64 bit floating point).

The text data type is available with several options for the length. For instances, *Character*, and *Varying Character*. The fields of this type should be able to handle 8 bit characters (variable-length character string, 0-255). Some DBMS able to provide 16 bits character (variable-length character string, 0- 65,535) for specific storage purposes, e.g. *Character String*.

The *Date* data type may represent dates with year, month and day. DBMS independent representation of dates is accomplished by using text strings formatted according to the IS0 8601 standard. The format defined by the ISO 8601 standard for dates is YYYY-MM-DD where YYYY is the number of the year (Gregorian calendar), MM is the number of the month from 1 to 12 and DD is the number of the day from 1 to 31. Months or days numbered below 10 should be padded on the left with 0. Some DBMS have native support for date formats, but for others the DBMS driver may have to represent them as integers or text values. In any case, it is always possible to make comparisons between date values as well sort query results by fields of this type. Another data type related to *Date* is the time data type that may represent the time of a given moment of the day. DBMS independent representation of the time of the day is also accomplished by using text strings formatted according to the IS0 8601 standard. The format defined by the ISO 8601 standard for the time of the day is HH:MI:SS where HH is the number of hour the day from 0 to 23 and MI and SS are respectively the number of the minute and of the second from 0 to 59. Hours, minutes and seconds numbered below 10 should be padded on the left with 0. Some DBMS have native support for time of the day formats, but for others the DBMS driver may have to represent them as integers or text values. In any case, it is always possible to make comparisons between time values as well sort query results by fields of this type.

The large object data types are meant to store data of undefined length that may be to large to store in text fields, like data that is usually stored in files. Certain DBMSs support two types of large object fields: Character Large OBject (*CLOB*) and Binary Large OBject (*BLOB*). *CLOB* fields are meant to store only data made of printable ASCII characters. *BLOB* fields are meant to store all types of data in binary format. Table 2.1 provides overview of database data types, its description and naming.

| Data Type Name | Oracle | SQL Server | DB2 | Informix | Access | PostgreSQL | MySQL |
|---|---|---|---|---|---|---|---|
| Object ID | NUMBER | INT | INTEGER | INT | LONG INTEGER | NUMBER | NUMBER |
| Short Integer | NUMBER | SMALLINT | SMALLINT | SMALLINT | INTEGER | INT2 | SMALLINT |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Long Integer | NUMBER | INT | INTEGER | INT | LONG INTEGER | INT8 | BIGINT |
| Float / Real | NUMBER | REAL | DECIMAL | DECIMAL | SINGLE | FLOAT | FLOAT |
| Double | NUMBER | DOUBLE | DECIMAL | DECIMAL | DOUBLE | FLOAT8 | DOUBLE |
| Varying Character | VARCHAR2 | VARCHAR | VARCHAR | VARCHAR | VARCHAR | VARCHAR | VARCHAR |
| Character | CHAR | CHAR | CHAR | CHAR | CHAR | CHAR | CHAR |
| Date | DATE | DATETIME | TIMESTAMP | DATETIME | DATE/TIME | DATE | DATE |
| BLOB | BLOB | IMAGE | BLOB | BLOB | OLE Object | Bytes | BLOB |

Table 2.1: List of the default data types in DBMS

## 2.3 Spatial DBMS

A great 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.

### 2.3.1 DBMS Advantages over File System

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;
- Perform complex spatial query on the data with near instant results (e.g. routing);
- Provide concurrency control, like e.g. locking mechanism and consistency checks, enabling multi-users.

No doubt, there are several advantages to storing attribute data externally outside the DBMS. Even though geometry continues to be managed exclusively within the file system, some limitations may be appeared.

The first limitation is that GIS packages usually do not have the sheer capacity and cluster scalability of mainstream DBMS packages like Oracle or SQL Server. As a result, the number of objects in geometry will normally be limited by the performance of the file system running on a single machine.

A second limitation is often a restriction to single users or single processes working with the geometry data stored by the file system. Mainstream DBMS packages have evolved to meet intensely multi-user, multi-process needs but most file systems have not been built with the multi-process, transaction-oriented architecture required for intensively multi-user operations on geometry.

A third limitation is limited dynamic interoperability between different file system packages and other applications. File system data can be interchanged more or less successfully in a static, limited way using well known formats such as shapefiles, but cannot in general be interchanged dynamically as is taken for granted with DBMS servers.

A dataset that can store geometry in a spatial DBMS can escape the above limitations. Storing geometry in a DBMS can take advantage of the capacity and scalability of the DBMS, resulting in drawings that can be terabytes in size. If a DBMS stores geometry, a GIS can operate as a client to the DBMS server in multi-user settings, taking advantage of this extensive module that will provide the simultaneous use of data by many different users. Storing geometry within the DBMS using data types native to

the DBMS allows interchange with any application that understands those data types, which will improve the interoperability in GIS.

### 2.3.2 Native Geometry and Non-Native Geometry Types

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. When storing geometry within a DBMS, the question is what internal format the DBMS should use to order the binary data used to store that geometry. There are two approaches.

One approach is used in DBMS as spatial DBMS. 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. The data is still binary data, of course, but it has been organized in accordance with a format expected for geometry data by the DBMS. When a DBMS has its own data types, native geometry type it is also usually being supported within the DBMS environment with additional infrastructure, such as the automatic creation of spatial indices or the provision of DBMS server commands that understand that data type.

The other approach is used with DBMS that do not specify a pre-defined way of organizing binary data to represent geometry but allow applications to utilize a generic binary data type. Almost all existing DBMS provide a generic binary data type that can be used to store binary data, which is unstructured by the DBMS. Often referred to as a BLOB (Binary Large OBjects), such generic binary storage can be employed by applications if they can fit. When such generic binary storage is used to store geometry in a form not built into the DBMS it is called a non-native geometry type. Although the use of non-native geometry types allows storing geometry within general-purpose DBMS without requiring a special spatial form of a DBMS, it does require a GIS application that supports the geometry formats to be used.

The main advantages of using a native geometry type within a spatial DBMS is, first, to provide interoperability with any application that uses the native type, and second, using a native type automatically takes advantage of the infrastructure within the spatial DBMS that supports that native data type.

The main advantages of non-native binary storage are, first, to make use of spatial DBMS functionality utilizing non-native types within virtually every DBMS and second, using a choice of non-native geometry types can provide greater flexibility rather than committing to a single data type with specific format. A possible disadvantage is that choosing from a variety of geometry types can make interoperability with other applications more difficult. However, if a generic geometry type that is well-defined and accepted by many applications, such as Well-Known Binary (WKB), is used, then interoperability might well be preserved.

### 2.3.3 Spatial Queries

Spatial query is a method of data searching within DBMS that satisfy a given condition. There are two types of queries; both queries are not much different compared to each other:

1. Query of attribute data - A spatial distribution or an area will be searched with respect to a given attribute of interest.
2. Query of geometric data - With a given geometric condition for example location, shape or intersection, all data that satisfy the condition will be searched. In the case of a vector data form, to search an area, which includes a given point, and to find all line segments that intersect a given line would be a typical query of geometric data. In the case of raster form of data, it will be easier to search any attribute and geometric data based on a given grid.

From the query of geometric data (also called spatial query), it is divided into two types: static and dynamic query. A static query only observes the spatial objects and returns a result without affecting the objects queried, e.g. measure area of a polygon. Dynamic queries are different from static in the way that they affect the data itself, e.g. merge, split, rotate, resize and copy (Shekhar and Chawla, 2003).

To perform spatial, some query languages are used. Query languages for geographic databases and geographic information systems are either complex macro languages, or extensions of SQL (Egenhofer, 1996). There are a large variety of Spatial SQL dialects, and such SQL provides the means for accessing geographic databases and retrieving data from a database. Most critical is the support for spatial relations. The spatial relations are defined in order to become the logical condition for spatial query. Certain DBMS, e.g. Oracle Spatial, uses a two-tier query model to resolve spatial

queries, which relates spatial relations. The term two-tier is used to indicate that two distinct operations are performed in order to resolve queries. If both operations are performed, the exact result set is returned. The two operations are referred to as primary filter and secondary filter operations (Oracle Spatial 10g, 2007).

- The primary filter permits fast selection of candidate records to pass along to the secondary filter. The primary filter uses geometry approximations (or index tiles) to reduce computational complexity and is considered a lower-cost filter.
- The secondary filter applies exact computational geometry to the result set of the primary filter. These exact computations yield the exact answer to a query. The secondary filter operations are computationally more intensive, but they are applied only to the relatively small result set returned from the primary filter.

### 2.3.4 Spatial Indexing

Another important aspect of data management within DBMS is spatial indexing. Spatial indexes are used in DBMS for fast search especially when spatial functions are applied. The problem with querying spatial data is that a common query, like querying a point, would need to compare and check the point location with the geometry of every object in the database, which is both time and memory consuming if the database is large. Spatial indexing was developed to resolve this. Without indexing, any searches for a feature would require a sequential scan of every record in the database. Indexing speeds up searching by organizing the data into a search tree that could be quickly traversed to find a particular record. There are few kinds of indexes within DBMS, i.e. PostGIS and Oracle Spatial: they are B-Tree indexes, R-Tree indexes, and GiST indexes.

- B-Trees are used for data, which can be sorted along one axis; for example, numbers, letters, dates. B-tree is a fast data-indexing method that organizes the index into a multi-level set of nodes. Each node contains a sorted array of key values (the indexed data). Two important properties of a B-tree are that all nodes are at least half-full and that the tree is always balanced (that is, an identical number of nodes must be read in order to locate all keys at any given level in the tree). A detail description of B-tree could be found in ITTIA (2005).

Another spatial indexing methods that store the approximation of geometry is in the form of minimum-bounding box. This kind of spatial query is considered as a two-step process, which involves the filter step and refinement step. The filter step compares and eleminate the candidates that do not intersect with the query condition. This is less time consumed as it is much simpler to compare and filter the geometry of the envelops than the object geometry. The refinement step drops out the geometry that does not intersect
with the spatial query though its envelope did. Finally the correct objects are returned as a result of the spatial query.

The refinement step is rather uniform in all spatial indexing methods. It is conventionally categorised into either space-driven or data-driven approaches (Rigaux *et al.* 2002).In a space-driven approach, the 2-dimensional planar space is partitioned into number of rectangles that are independent of the objects they serve. The objects are then mapped to cells according to geometric criteria that differ somewhat considering what method is used. Most popular space-driven methods are named "grid-file", "linear-quadtree" and "z-ordering tree". A data-driven approach on the other hand, focuses on the objects and in partitioning them into appropriate/logical groups considering number and distribution in space. Most popular data-driven methods is the 'R-tree' and its reformed versions 'R+tree' and 'R* tree' (Rigaux *et al.*, 2002; Guttman, 1984).

- R-Trees break up data into rectangles, and sub-rectangles, and sub-sub rectangles, etc. R-Trees are used by some spatial databases to index GIS data, but the PostGIS R-Tree implementation is not as robust as the GiST implementation. Oracle Spatial will implement the 3D R-Trees in the coming version 11g.

The concept of sample R-tree structure is given in Figure 2.1, Figure 2.2, and Figure 2.3 in two-dimension and three-dimension. The impact of z-coordinate in the 3D spatial indexing will influence the execution time due to the indexing mechanism will search each of the (x, y) elements that relates to its z-coordinate. For example, 7 (x, y, z) points will search 7 times greater than 7 (x, y) elements. Note that the Oracle Spatial R-Tree indexing provides the spatial index up to 4D, and the dimensionality should be defined in the syntax.

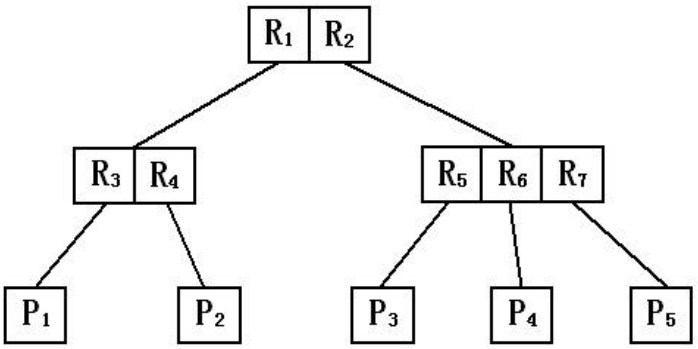



Figure 2.1: Directory of R-Tree indexing

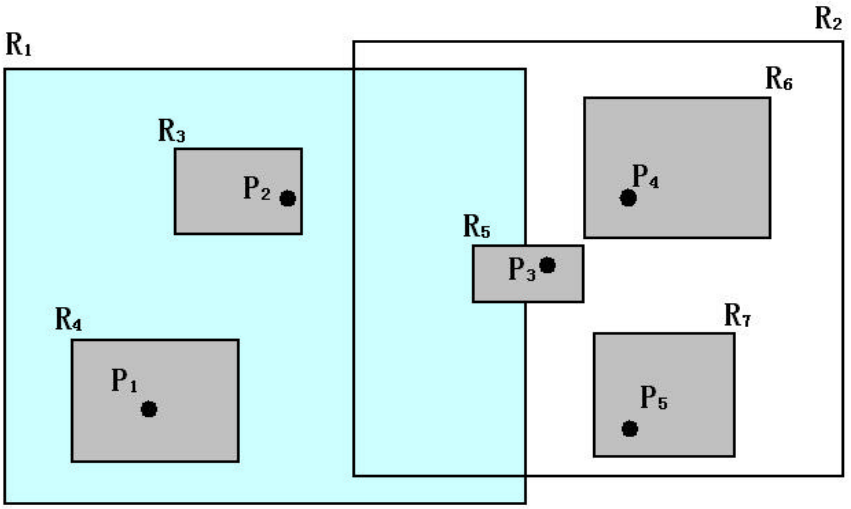



Figure 2.2: A planar representation of an R-tree

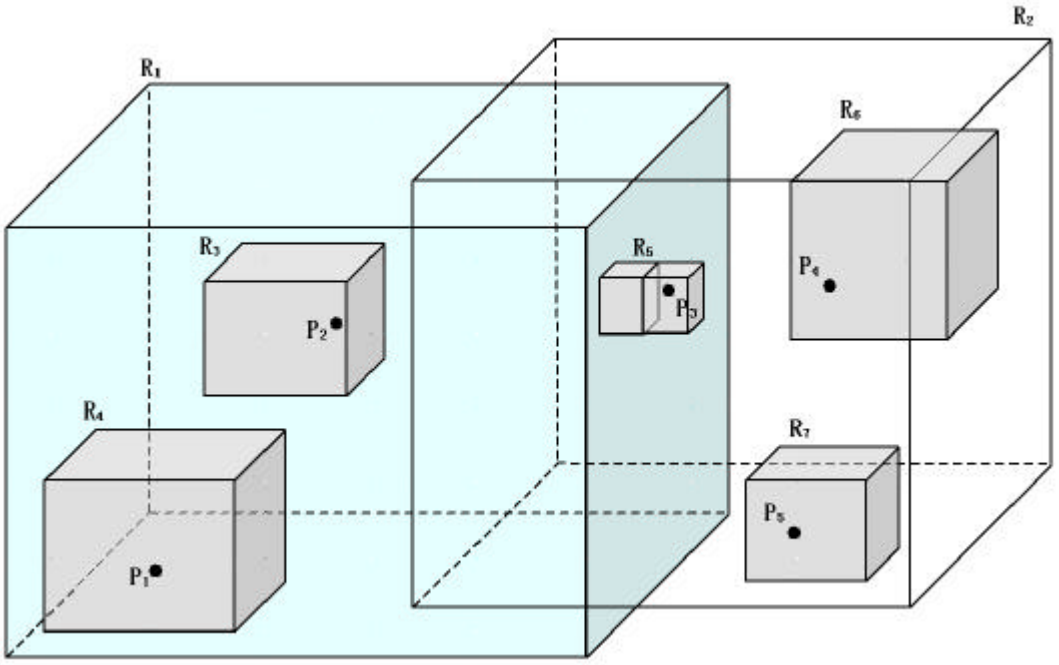



Figure 2.3: A 3D representation of an R-tree

Another complex spatial query developed by PostgreSQL (2007) in order to be implemented as part of PostGIS module is GiST.

- GiST (Generalized Search Trees) indexes break up data into “things to one side”, “things which overlap”, “things which are inside” and can be used on a wide range of data-types, including GIS data. PostGIS (2006) uses an R-Tree index implemented on top of GiST to index GIS data.

GiST indexes have two advantages over R-Tree indexes in PostGIS. Firstly, GiST indexes allow null value in the index columns. Secondly, GiST indexes could easily deal with GIS objects larger than the PostGIS 8K page size. The important part of an object in an index will only be considered within DBMS, e.g. in the case of GIS objects, just the bounding box. GIS objects larger than 8K will cause R-Tree indexes to fail in the process of being built. It could take a long time to create a GiST index if there is a significant huge amount of data in a table. However, The GiST index is widely used for 2D data. The implementation of GiST is rather limited for 3D data. The research and application on 3D GiST is expected in near future.

Other DBMS, e.g. Oracle Spatial, are able to provide 3D indexing for 3D object (MULTIPOLYGON). For Spatial, the metadata that maintains the lower and upper bounds and tolerance of 3D object needs to be created. Later, a spatial index (R-tree in 3D) could be created on the tables to speed up spatial queries. The following example denotes the sample in creating a 3D spatial index within Oracle Spatial.

```
-- Inserting metadata for 3D object: MUTIPOLYGON

INSERT INTO user_sdo_geom_metadata VALUES
   ('Solid3D', 'shape', mdsys.sdo_dim_array(
   mdsys.sdo_dim_element('X', 0, 100, 0.1),
   mdsys.sdo_dim_element('Y', 0, 100, 0.1),
   mdsys.sdo_dim_element('Z', 0, 100, 0.1)
   ), NULL);

-- Creating 3D Spatial Index
```

```
CREATE INDEX Solid3D_I on Solid3D(shape)
   INDEXTYPE IS mdsys.spatial_index
   PARAMETERS('sdo_indx_dims=3');          -- Dimension = 3

ANALYZE TABLE SOlid3D COMPUTE STATISTICS;
```

## 2.4 Spatial Data Model

Tsichritzis and Lochovsky (1977) define a data model as a set of guidelines for the representation of the logical organization of the data in a database consisting of named logical units of data and the relationships between them. While the concept of the data model is used in a variety of ways by numerous disciplines, a digital geographic data model is generally defined as an information structure, which allows the user to store specific phenomena as distinct representations, and enables the user to manipulate the phenomena when held in the system as data (Raper and Maguire, 1992).

The data model represents a set of guidelines to convert the real world (called entity) to the digitally and logically represented spatial objects consisting of the attributes and geometry. The attributes are managed by thematic or semantic structure while the geometry is represented by geometric-topological structure (Shunji 1999). The ability to take the geographic location of objects into account during search, retrieval, manipulation and analysis lies at the core of a GIS data model (Smith *et al.* 1987). How well these tasks can be accomplished is determined by the spatial data model, apart from other factors such as the data structures and database management systems selected for the DBMS (Berry 1993). The theory of spatial data models currently attracts the most active research and development with in the GIS community (Clarke 1986, Van Roessel 1987, Mounsey and Tomlinson 1988, Goodchild and Gopal 1989). The following

section discusses two types of data models, which they were reflected to this study, in term of geometry and topology data types.

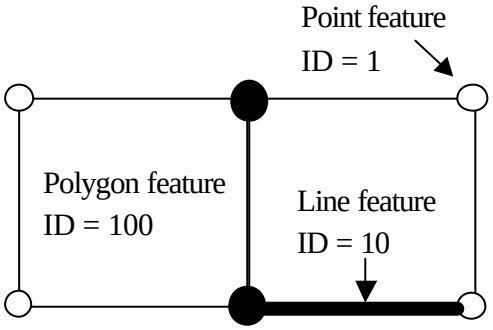
### 2.4.1 Spaghetti Model

Among many of the commonly used vector based data structure, the spaghetti data model has the most simple data structure (Aronoff 1989). In the spaghetti data model each entity on a map becomes one logical record in the digital file, and is defined as a string of x, y coordinates. It indicates no explicit structure and it is used if the geometry of spatial features in spatial DBMS is described completely independent and irrespective of other features in the database (Ingvarsson, 2005). Although all entities are spatially defined, no spatial relationships are encoded. This represents a significant deficiency since, to perform any type of spatial analysis, the spatial relationship between such entities must be derived through computation. Relationships like adjacency, within, outside etc., between separate geometries are therefore calculated on demand. But the spaghetti data model can efficiently reproduce maps digitally because information unconnected to the plotting process is not stored (Peuquet 1984).

This model is also referred to raw data. The main characteristics of such data are possible overlapping geometries and dangling lines. This is evident when representing land parcels as spaghetti polygons, whereas each boundary has to be stored twice, and the same corner monument stored at least three times, in different polygons (Ingvarsson, 2005). This creates problem of tracking boundary measurements. Also if geometric data is of different quality, adjacent land parcels can either overlap or be disjoint, but not touching as would be correct.

The properties of spaghetti data model are given as follows:

- Point is enclosed as single XY co-ordinate pair;
- Line is encoded as a string of XY co-ordinate pairs;
- Polygon is encoded as a closed loop of XY co-ordinates that define its boundary. The common boundary between adjacent polygons must be recorded twice, once for each polygon;
- The Spaghetti model is a file of spatial data constructed in this manner is essentially a collection of co-ordinate strings with no inherent structure, and hence the term spaghetti model is named; and
- Although all the spatial features are recorded, the spatial relationships between these features are not encoded.




| Feature | ID | Location |
|---|---|---|
| Point | 1 | $x_1,y_1$ |
| Line | 10 | $x_1,y_1,x_2,y_2$ |
| Polygon | 100 | $x_1,y_1,x_2,y_2,x_3,y_3, x_4,y_4$ |

Figure 2.4: Spaghetti data model

### 2.4.2 Topological Model

In this section, the discussion is focused on a model, which involves properties of the database that are topological in nature. In this data model, concepts such as adjacency, connectivity, and containment are important. Queries like "what is next to polygon A?" or "List all the lines that constructs the polygon C" are typical in this respect. Characteristic of topological properties is that they do not distinguish between two databases that can be obtained from each other by a topological deformation. Such data model is called topologically equivalent.

In applications in which only topological properties are under consideration, it may be desirable to be able to work with a representation of the database, which is topologically invariant, meaning that two topologically equivalent databases will be represented identically. The idea of topological property in 2D spatial databases is consisting of points in the plane $R_2$, lines between these points, and areas formed by these lines. This model is commonly referred to as the topological data model (Güting, 1994a; Güting, 1994b; Guting, 1989; Guting, and Schneider, 1993; Thompson and Laurini, 1992). An example application is a subway or railway map in which only relative positions of spatial objects such as stations and tracks are depicted without, for instance, taking the actual length of the trajectory into account.

The Census Bureau of the United States introduced this data model in 1979 (Corbett, 1979) to model topological information on what they called zero-cells (points), one-cells (lines) and two-cells (areas) (Thompson and Laurini, 1992). The information in the two-dimensional plane is described by a number of cells, and each has an identifier. Furthermore, the following topological relations $R_1$, $R_2$, $R_3$ and $R_4$ are given:

- $R_1$: every one-cell has two zero-cells (indicating that every line has exactly two endpoints);
- $R_2$: every one-cell has two two-cells (indicating that every line is the border between exactly two areas);
- $R_3$: every two-cell is surrounded by a (ordered) cycle of one-cells and zero-cells (indicating the border of an area);
- $R_4$: every zero-cell is surrounded by a (ordered) cycle of one-cells and two-cells (indicating the neighborhood of a point).

For relation $R_3$ a clockwise order is agreed upon for outer borders of areas and a counter-clockwise order is used for holes in areas. For relation $R_4$ a clockwise order is used. To settle the planarity of the model there is the additional condition that all intersections of one-cells are zero-cells and all intersections of two-cells are one-cells (see Figure 2.5).

| R1 | | |
|---|---|---|
| A | p | q |
| A | q | p |
| B | q | r |
| B | r | q |
| C | r | p |
| C | p | r |
| . . . | | |

| R2 | | |
|---|---|---|
| A | a | ß |
| A | ß | a |
| B | a | ? |
| B | ? | a |
| C | a | d |
| C | d | a |
| . . . | | |

| R3 | | | |
|---|---|---|---|
| a | p | A | 1 |
| a | q | B | 2 |
| a | r | C | 3 |
| ß | p | D | 1 |
| ß | s | E | 2 |
| ß | t | F | 3 |
| . . . | | | |

| R4 | | | |
|---|---|---|---|
| p | A | a | 1 |
| p | C | d | 2 |
| p | D | ß | 3 |
| q | B | a | 1 |
| q | A | ß | 2 |
| q | F | ? | 3 |
| . . . | | | |

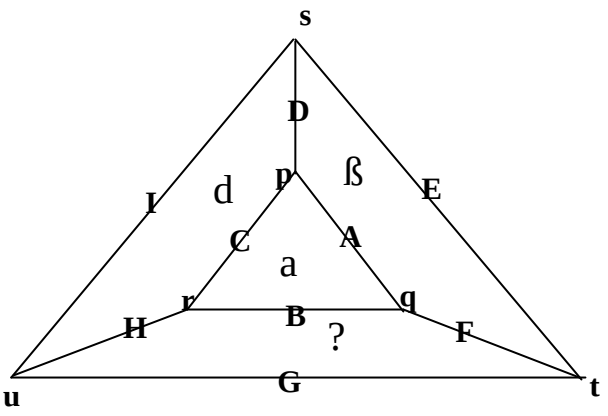



Figure 2.5: The relations $R_1$, $R_2$, $R_3$ and $R_4$ (Paredaens, and Kuijpers, 1998)

## 2.5 The Existing Geo-DBMS

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 database support the object-oriented model for representing geomeries. 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) 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 many 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.

### 2.5.1 Oracle Spatial

Oracle Spatial technology was first introduced in Oracle 7.2 under the name Oracle MultiDimension (MD). Later, the product name was changed to Oracle Spatial Data Option (or SDO) and to Spatial Data Cartridge in Oracle 8. Since objects were not supported in these releases, the coordinates of a geometry were stored as multiple rows in an associated table. Managing spatial (geometry) data in these prior versions was inefficient and cumbersome.

Starting with Oracle 8i, the SDO_GEOMETRY data type was introduced to store spatial data. Even in Oracle 10g, the same SDO_GEOMETRY model is used to store spatial data in Oracle. In Oracle 9i (and Oracle 10g), the geometry data also included support for coordinate systems information specified using the SRID attribute in the

SDO_GEOMETRY data type. In Oracle 10g, additional functionality (that exists in the Advanced Spatial Engine) such as the Network Data Model is introduced in the Spatial option of Oracle. Since the prior versions are named MultiDimension (MD) and Spatial Data Option (SDO), users will see the prefixes MD and SDO for the files and schemas that install Spatial technology. The name of the spatial install schema is MDSYS in all versions of Oracle.

Oracle Spatial technology is automatically installed with the Standard or Enterprise Edition of an Oracle database server, since Oracle Database 10g. Oracle Spatial supports the object-relational model for representing geometries. The object-relational model uses a table with a single column of SDO_GEOMETRY and a single row per geometry instance. It supports the data storage of three-dimensional and four-dimensional geometric types, where three or four coordinates are used to define each vertex of the object being defined. The SDO_GEOMETRY data type captures the location and shape information of data rows in a table. This data type is internally represented as an Oracle object data type. It can model different shapes such as points, lines, polygons, and appropriate combinations of each of these. In short, it can model spatial data occurring in most spatial applications and is conformant with the OpenGIS Consortium (OGC) Geometry model. The Oracle Spatial data model is a hierarchical structure consisting of elements, geometries, and layers. Layers are composed of geometries, which in turn are made up of elements.

An element is the basic building block of a geometry. The supported spatial element types are points, line strings, and polygons. Each coordinate in an element is stored as an X,Y pair, and it can be stored up to four-dimension. However, spatial functions can work with only the first two dimensions, and all spatial operators are disabled if the spatial index has been created on more than two dimensions. The primitive data types are:

- Point data consists of one coordinate.
- Line data consists of two coordinates representing a line segment of the element.
- Polygon data consists of coordinate pair values, one vertex pair for each line segment of the polygon. Coordinates are defined in order around the polygon (counterclockwise for an exterior polygon ring, clockwise for an interior polygon ring).

A geometry (or geometry object) is the representation of a spatial feature, modeled as an ordered set of primitive elements. A geometry can consist of a single element, which is an instance of one of the supported primitive types, or a homogeneous or heterogeneous collection of elements. A multipolygon, such as one used to represent a set of islands, is a homogeneous collection. A heterogeneous collection is one in which the elements are of different types, for example, a point and a polygon. With the approach of homogeneous collection of elements, the coming Oracle 11g is expected to provide 3D primitive, i.e. Polyhedron. However, the 3D spatial function remains uncertainty.

A layer is a collection of geometries having the same attribute set. For example, one layer in a GIS might include topographical features, while another describes population density, and a third describes the network of roads and bridges in the area (lines and points). Each layer's geometries and associated spatial index are stored in the database in standard tables.

In the installation, Oracle Spatial requires user to verify system requirements. For Oracle 10g, the minimum RAM required is 1024MB, and the minimum required swap space is 1GB. Swap space should be twice the amount of RAM for systems with 2GB of RAM or less and between one and two times the amount of RAM for systems with more than 2GB. Besides, users also need 2.5GB of available disk space for the Oracle Database 10g Release 2 software and another 1.2GB for the database. The /tmp directory

needs at least 400MB of free space. Although the development of current computer technology able to fulfill such requirements, for educational and research purposes, the research development must be synchronized with the latest computer technology if more research applications intended to be extended. Thus, the portability of using Oracle Spatial is considered low, as more expenses would be used on setting up a moderate type of database server.

As the Oracle Database is developed for high commercial applications, the Oracle Spatial is not an open-source module. Some of the commercial offerings like Oracle Spatial are not cost-sensible. This is because in order to use Oracle Spatial you need to buy Oracle Enterprise. Plus if you are going to use it for Web Applications then you will have to pay the per CPU license instead of the per user license. Therefore, the educational aspect for Oracle Spatial is rather low compared to other open-source module like PostgreSQL.

The native programming supported by Oracle Spatial are PL/SQL, C/C++, Java, and ect. These supports are the most appropriate way to create user-defined data types and functions within geo-DBMS environment.

### 2.5.2 PostGIS

Refractions Research Inc. develops PostGIS (first released in 2001) as a spatial database technology research project for PostgreSQL. It is a GIS and database consulting company, specializing in data integration and custom software development. The company develops PostGIS to support a range of important GIS functionality,

advanced topological constructs (coverages, surfaces, networks), desktop user interface tools for viewing and editing GIS data, and web-based access tools.

In terms of spatial databases, PostGIS is the most capable open source spatial database extender for the PostgreSQL Database Management System. Built as an object extension to PostgreSQL, PostGIS has been certified as "Simple Features for SQL" compliant by the Open Geospatial Consortium.

Although OGC standard only support 2D geometries, PostGIS extended formats are currently superset of OGC one (every valid WKB/WKT is a valid EWKB/EWKT). PostGIS EWKB/EWKT add 3DM, 3DZ, 4D coordinates support and embedded SRID information. Examples of the text representations (EWKT) of the extended spatial objects of the features are as follows (PostGIS, 2007):

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- XY with SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM with SRID
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9),LINESTRINGM((2 3 4,3 4 5)))

PostGIS also implements the extended of simple features for SQL specifications by defining a number of circularly interpolated curves. It includes 3DM, 3DZ and 4D coordinates, but do not allow the embedding of SRID information (different compared to

linear object, e.g. polygon). The well-known text extensions are not yet fully supported. Besides, PostGIS cannot support the use of Compound Curves in a Curve Polygon. Examples of some simple curved geometries are shown below (PostGIS, 2007):

- CIRCULARSTRING(0 0, 1 1, 1 0)
- COMPOUNDCURVE((0 0, 1 1, 1 0),(1 0, 0 1)
- CURVEPOLYGON(CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),(1 1, 3 3, 3 1, 1 1))
- MULTICURVE((0 0, 5 5),CIRCULARSTRING(4 0, 4 4, 8 4))
- MULTISURFACE(CURVEPOLYGON(CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),(1 1, 3 3, 3 1, 1 1)),((10 10, 14 12, 11 10, 10 10),(11 11, 11.5 11, 11 11.5, 11 11)))

Same as other spatial databases such as Oracle Spatial, it is used for high-performance multi-user access to large seamless data sets. In a nutshell it adds spatial functions such as distance, area, and specialty geometry data types to the database. PostGIS is very similar in functionality to ESRI ArcSDE, Oracle Spatial, and DB II spatial extender. The latest release version, it comes packaged with the PostgreSQL DBMS installs as an optional add-on. This spatial DBMS works in windows environment although most of the implementation applies to other supported platforms such as Linux, Unix, BSD, Mac, and etc.

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 1.3.2 or above requires full PostgreSQL server headers access in order to compile. PostGIS 1.3.2 can be built against PostgreSQL versions 7.2.0 or higher. The installation does not require complex setting. The minimum memory of physical RAM is 256 MB, 2 GB of hard drive space, and minimum of 250-MHz processor are required. It was easy to start and stop the server (same for client side), and user could do this without logging in as root.

Within the PostgreSQL environment, it allows user-defined functions to be specified in a number of languages, including its own procedural version of SQL called PL/PGSQL, and C/C++, Java.

When compared with commercial spatial databases, PostGIS has most of the core functions users will see in the commercial databases such as Oracle Spatial, has comparable speed, fewer deployment headaches, but lacks some of the advanced add-ons modules, such as Oracle Spatial network topology model, Raster Support and Geodetic support. Often, the advanced spatial features are add-ons, on top of the standard price of the database software. In certain application that requires wide-ranging of database functions, Oracle Enterprise version with its myriad of features maybe able to fulfill the requirements of a project better than PostgreSQL/PostGIS. However, as far as spatial databases are concerned PostGIS does make economic sense more than Oracle Spatial.

### 2.5.3 MySQL

MySQL (2007) is one of open source SQL database management system is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers. It is a second-generation open source company that unites open source values and methodology with business model. MySQL is a relational database management system. MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). MySQL implements a subset of the SQL with Geometry Types environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry

type. The specification describe a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

MySQL has data types that correspond to OGC classes. Some of these types hold single geometry values:

- GEOMETRY
- POINT
- LINESTRING
- POLYGON

GEOMETRY can store geometry values of any type. The other single-value types (POINT, LINESTRING, and POLYGON) restrict their values to a particular geometry type. The other data types hold collections of values:

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

GEOMETRYCOLLECTION can store a collection of objects of any type. The other collection types (MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, and GEOMETRYCOLLECTION) restrict collection members to those having a particular geometry type.

Similar to PostGIS, MySQL enable spatial object being created using WKT , and WKB functions, which are defined in the OGC standard. Besides, MySQL also provides MySQL-specific functions in create spatial object.

MySQL requires average time from software download to installation completion. It remains true whether the platform is Microsoft Windows, Linux, Macintosh, or UNIX. The self-management features like automatic space expansion, auto-restart, and dynamic configuration changes are ready once installed. The portability level is more or less similar to PostGIS. Recommended hardware requirements for MySQL installation are Pentium V processor and 128 MB RAM. Note that MySQL can be installed on a platform with as little as 32 MB. However, for better performance it is recommended to have at least 128MB memory.

Although MySQL is part of LAMP (Linux, Apache, MySQL, PHP / Perl / Python) environment, it only supports user-defined functions if written in C, and a privileged account is then needed to link the compiled version with the MySQL executable. The native programming language support aspect is rather low, compared to Oracle and PostgreSQL.

### 2.5.4 Some Reasons Why the PostgreSQL Was Chosen

In this study, PostgreSQL was chosen as a DBMS testbed to perform 3D spatial operations. 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 OpenGIS "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.

## 2.6 Summary and Conclusion

DBMS attempts to make an effort in handling and upgrading geometries for GIS. The support of 2D objects with 3D coordinates is adopted by all mainstream DBMS. The general discussion related to the spatial DBMS and its functionalities were given in this chapter. The selected spatial DBMS for this study is clear with the practical explanation given in the previous section. However, most of 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. In the next chapter, the discussion will focus on the OGC standards for geospatial modelling. It involves the specification for data type schema together with the spatial operations. The discussion will provide a clear direction in order assist the study to extend the current spatial data type to the third dimension and perform spatial operations for 3D GIS.

# CHAPTER III

# OGC STANDARDS FOR GEOSPATIAL MODELING

## 3.1 Introduction

GIS introduce methods and environments to visualize, manipulate, analyze and display geographic data. These methods and environments have some interoperability problems. Different organizations and commercial vendors develop their own data models and storage structures. If GIS services are not interoperable, GIS services cannot interact with each other even though they are in the same organization or they belong to same commercial vendor. The nature of the geographical applications requires seamless integration and sharing of spatial data from a variety of providers. To solve the interoperability problems, the OGC has introduced some standards by publishing specifications for the GIS services. OGC is a nonprofit, international standards organization that is leading the development of standards for geographic data related operations and services. OGC has variety of contributors from different areas such as private industry and academia to create open and extensible software application programming interfaces for GIS (OGC, 2007). Therefore, the third stage of the research started with a review of Open Geospatial Consortium (OGC) standards for GIS.

The OGC is a voluntary consensus standards organization. Founded in 1994, the OGC produces many kinds of technical documents, including standards, working drafts, technical reports, discussion papers, and XML schemas. A number of OGC standards and application schemas of OGC standards are now used and/or referenced by standards specifications from other standards organizations, including OASIS, the IETF, IEEE, ISO, and OMA. The OGC core mission is to develop spatial interface and encoding specifications that are openly available and royalty free. Products and services that conform to OGC interface specifications enable users to freely exchange and process spatial information across networks, computing platforms, and products. Interoperability in such an environment is facilitated by the use of a system of persistent identifiers that are global in scope. The OGC is the only standards organization whose mission is specifically focused in interfaces and encoding for geospatial content and services. This chapter discussed the abstract specification for feature geometry, together with the implementation specification in DBMS.

## 3.2 OGC Abstract Specification for Feature Geometry

This OGC standard presents the conceptual schemas to describe and manipulate the spatial characteristics of geographic features (OGC, 2001). A feature is an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth. There are two kinds of data types: vector data consists of geometric and topological primitives used, separately or in combination, to construct objects that express the spatial characteristics of geographic features; raster data is based on the division of the extent covered into small units (normally defined as pixel size) according to a tessellation of the space and each unit is assigned to an attribute value. This abstract standard deals only with vector data. In this standard model, the attributes to describe spatial characteristics are given by a geometric object (GM_Object) or a

topological object (TP_Object). Geometry provides the means for the quantitative description (in the form of coordinates and mathematical functions), including dimension, position, size, shape, and orientation. The mathematical functions used for describing the geometry of an object depend on the type of coordinate reference system used to define the spatial position. Geometry is the only aspect of geographic information that changes when the information is transformed from one geodetic reference system or coordinate system to another.

Topology deals with the characteristics of geometric figures that remain invariant if the space is deformed elastically and continuously – for example, when geographic data is transformed from one coordinate system to another. Within the context of geographic information, topology is commonly used to describe the connectivity of an *n*-dimensional graph, a property that is invariant under continuous transformation of the graph. Computational topology provides information about the connectivity of geometric primitives that can be derived from the underlying geometry.

Standardized conceptual schemas for spatial characteristics will increase the ability to share geographic information among applications. These schemas will be reviewed and extended into third dimension for spatial operation implemented in spatial DBMS. In the next section, the discussions will focused on the concept of Unified Modeling Language as it give a general idea to describe the architecture of spatial data and model. The study will implement UML method to develop the standard notation for new 3D spatial data type and spatial operations.

### 3.2.1 Unified Modeling Language (UML) Concept

The architecture of spatial data type and modeling for GIS are commonly presented using UML. Instead of entities, it models object classes. It can be used it to specify, visualize, construct and document designs. The language is also used to denote the design structure of software systems, as a part of the software development process. UML become a standard notation for representing the structure of data in the object-oriented community. It was developed when the object-oriented method adopted the standard notation for modeling purposes (Embley, *et al.* 1992). The UML was published by the Object Management Group in 1997 (OMG, 1998). The Unified Modeling Language includes:

- Elements that involve the fundamental concepts and semantics;
- Notation for visual rendering; and
- Rules and condition involve in the architecture.

UML also provides extensibility and specialization mechanisms to extend the core concepts. It has a semantics model that maps well to a family of OO languages, but in itself does not require the use of a specific programming language. Besides, UML does not define a standard process, and it has intentionally been made process independent.

As a system of notation for representing the structure of data, the UML static diagram is functionally the exact equivalent of any other data modeling, entity/relationship modeling, or object modeling technique. Its classes of entity objects are really entities, and its associations are relationships. It has specialized symbols for some things that are already represented by the main symbols in other notations, and it lacks some symbols used in E/R diagrams. It does, however, have a more extensive

ability to describe inter-relationship constraints. Several modeling add the ability to describe the behavior of each object class/entity, but the technique for data structure is similar to the UML data modeling technique. The components of UML include:

*Entities* (Object Classes) and *Attributes*: As stated above, in object models, entities are called object classes. A class in the UML static model is a square cornered rectangle with three divisions. The top part contains the class name. The middle section contains a list of attributes. The bottom, if included, contains descriptions of behavior. Since the UML is mostly used for design, these behavior descriptions are usually in the form of pseudo-code.

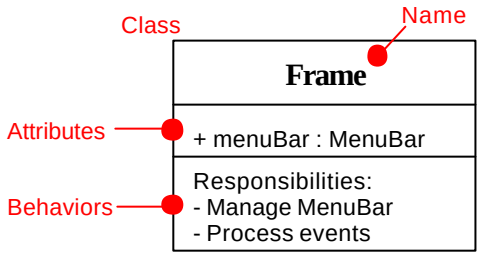



Figure 3.1: Sample of UML diagram

An attribute can be referred to by one or more of the following elements:

- Visibility - In terms of the object-oriented code which may implement the class, is this attribute visible to all (+), to only those dasses which are sub-types of this class (#), or to this class only (-).

- Name - This is the only required element.

- Multiplicity - Object-orientation is not constrained by the relational notion that an object may have only one value for an attribute. This parameter lets you

define that it may have more than one, up to five, etc. If the lower limit is zero, then occurrences of the related entity are optional.

- Type - This is the data type of the attribute (number, character, etc.). The values for this depend on the model's environment.

- = Initial value - Here can be specified a default value.

*Relationships* (Associations): A relationship is called an association in the object-oriented world. Rather than using graphic symbols, all the information on a UML association is conveyed by characters. Several elements involve in providing the association for UML are:

- Cardinality / Optionality: Both cardinality and optionality are conveyed by characters in the form:

  <lower limit>

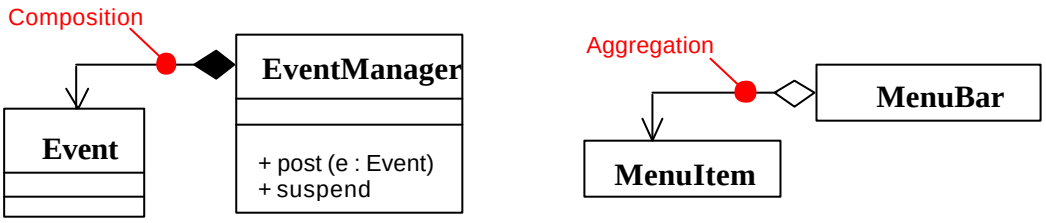
  ..

  <upper limit>

  ...

where the <lower limit> denotes the optionality (nearly always 0 or 1, although conceivably it could be something else), and the <upper limit> denotes the cardinality. The <upper limit> may be either an asterisk (*) for the generic "more than one", or it may be an explicit number, a set of numbers, or a range. For example, "0..*" means "may be one or more" (zero, one, or more), and "1..1" means "must be exactly one". Since they are most common, "0..*" may be abbreviated "*", and "1..1" may be abbreviated "1".

- "Part of/composed of": Extra symbols represent the particular association where each object in one class is composed of one or more objects in the other class. (Each object in the second class must be part of one and only one object in the first class) The association acquires a diamond symbol next to the parent ("composed of") class. If the association is mandatory and the referential integrity rule is "cascade delete" - that is, deletion of the parent deletes all the children - this is called "composition" and the diamond is solid. If the association is optional to the parent (and therefore has the referential integrity rule "nullify delete") - that is, a parent can be deleted without affecting the children - then the diamond is open and is called "aggregation". The notation does not address the "restricted" rule, in which deletion of a parent is not permitted if children exist. Nor does it address referential integrity rules for any other kind of association.




(a) (b)

Figure 3.2: (a). Composition, and (b). Aggregation

### 3.2.2 Data Types and Collection Types

There are many different spatial standards on how to model geographic features to spatial objects in databases. It depends on which kind of DBMS solution is used, which spatial data types are available to use. However, the OpenGeospatial Consortium

(OGC, 1999) produced a standard specification on simple features that established some consensus on this matter by creating a paradigm for database users and vendors to follow in handling two-dimensional spatial data. The simple feature model has since been adapted and developed further by the International Standards Organization (ISO), e.g. in ISO/TC211-19107: Geographic Information Spatial Schema (ISO 2003). The discussion on spatial standard started by introducing the OGC simple feature specification that is currently applicable in most spatial enabled databases.

The OGC defines spatial model as collections of spatial objects that referred as Geometry. It is associated to spatial extent in space, which defined in spatial reference system. The Geometry class is divided into four subclasses as geometric primitives: Point, Curve, Surface and GeometryCollection. All these classes provide methods to their subclasses to be extended the model in the future (Ingvarsson, 2005). For example, the Surface was used for this study in extending the two-dimensional spatial data type to three-dimension, i.e. polyhedron. Point represents a 0-dimensional object located in space. Linear feature connects at least two Points to create a 1-dimensional LineString, which denotes a class Curve. Arc or Spline could also be considered as class of Curve. The simplest LineString is a straight Line segment that only connects two Points. If the LineString is simple (i.e. does not cross itself) and closed with it's begin and end points connected, it is considered to be LinearRing. At least one LinearRing is necessary to define the boundary of a 2-dimensional Polygon, a specialization of the geometric primitive Surface. Extra LinearRings define interior boundaries of the Polygon. Other classes than Polygon can be considered as specialization of Surface, e.g. TriangulatedSurface.

A collection of one ore more Points, LineStrings or Polygons can be stored in as collection objects, e.g. MultiPoint, MultiLinestring or MultiPolygon. The MultiLineString and MultiPolygon are specializations of MultiCurve and MultiSurface

respectively, with everything a specialization of the geometric primitive GeometricCollection. The OGC specification on simple features is given in Figure 3.3.

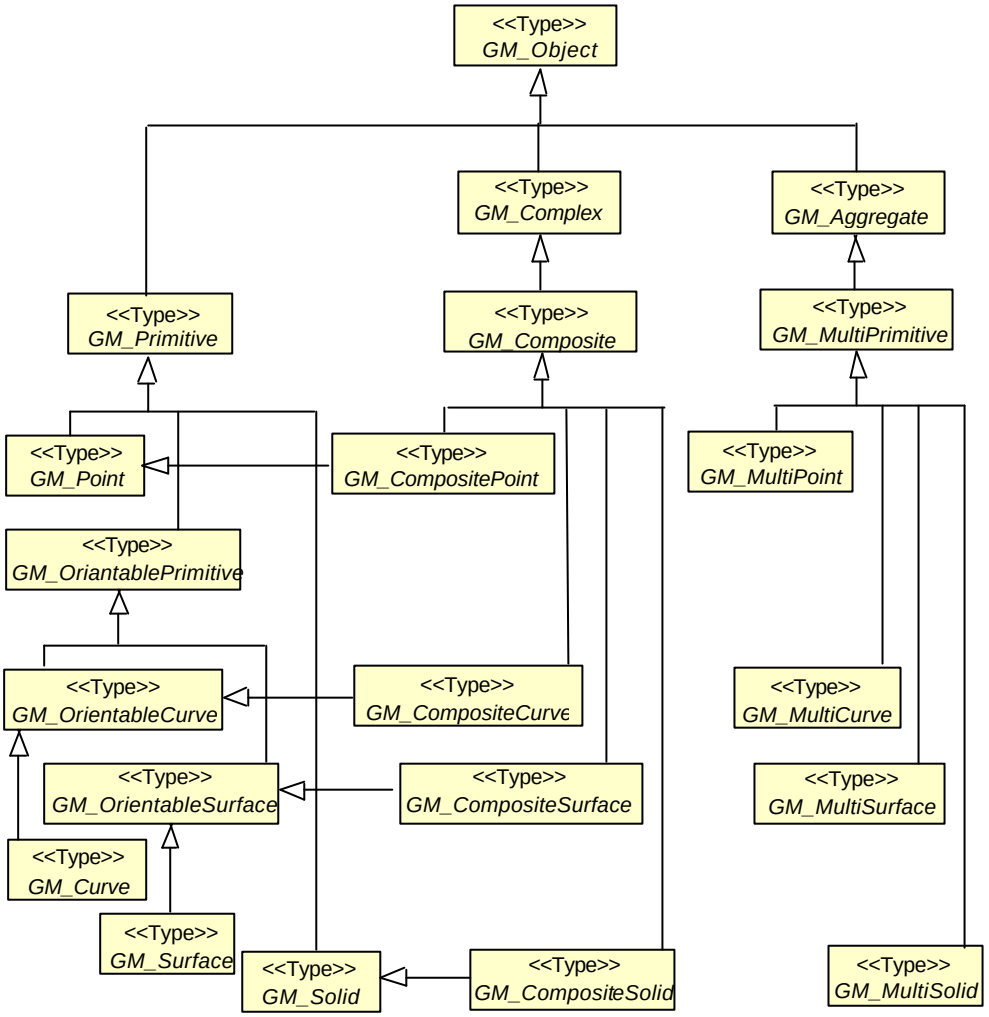



Figure 3.3: Geometry basic classes given in OGC

The Geometry class given by OGC specification is GM_Object. It has three subclasses: GM_Primitives, GM_Complex and GM_Aggregate. The specialization of GM_Primitives are GM_Point, GM_Curve and GM_Surface similar to Point, Curve and Surface. To stores sequence of GM_Primitives, GM_Complex is used, as if implemented in GM_CompositeCurve. GM_Aggregate is a function similarly as

GeometricCollection in the OGC specification. The spatial extent is not associated with root class, i.e. GM_Object, but to GM_Primitives. Consequently, the GM_Complex and GM_Aggregate are extending from primitives, with spatial reference dependence on the class GM_Primitive.

### 3.2.3 OGC Well-Known-Text

There are two ways of expressing spatial object, especially within DBMS environment like data insertion. They are defined by OGC, i.e. the Well-Known Text (WKT) form and the Well-Known Binary (WKB) form. Both WKT and WKB record the information about data type and the coordinates in order to form an object. The purpose of identifying this specification is to implement the method of expressing spatial object into three-dimension for the study. Although OGC only defines these methods in two-dimension, the method could be extended into three-dimension, with minor modification from the coordinate structure. For the sake of this study, only OGC WKT was given attention, and some examples of the text representations (WKT) are as follows:

- POINT($x_1$ $y_1$)
- LINESTRING($x_1$ $y_1$, $x_2$ $y_2$, $x_3$ $y_3$)
- POLYGON($x_1$ $y_1$, $x_2$ $y_2$, $x_3$ $y_3$, $x_4$ $y_4$, $x_5$ $y_5$, $x_1$ $y_1$)
- MULTIPOINT($x_1$ $y_1$, $x_2$ $y_2$)
- MULTILINESTRING(($x_1$ $y_1$, $x_2$ $y_2$, $x_3$ $y_3$),($x_4$ $y_4$, $x_5$ $y_5$, $x_6$ $y_6$))
- MULTIPOLYGON(($x_1$ $y_1$, $x_2$ $y_2$, $x_3$ $y_3$, $x_4$ $y_4$, $x_5$ $y_5$, $x_1$ $y_1$),( $x_6$ $y_6$, $x_7$ $y_7$, $x_8$ $y_8$, $x_9$ $y_9$, $x_{10}$ $y_{10}$, $x_6$ $y_6$))
- GEOMETRYCOLLECTION(POINT($x_1$ $y_1$),LINESTRING($x_1$ $y_1$, $x_2$ $y_2$))

The OGC specification also requires other storage format such as include the spatial referencing system identifier (SRID). The SRID is required if spatial objects are inserted into the database. Input/Output of these formats are available using the following interfaces:

- bytea WKB = asBinary(geometry);
- text WKT = asText(geometry);
- geometry = GeomFromWKB(bytea WKB, SRID);
- geometry = GeometryFromText(text WKT, SRID);

A valid insert statement to insert a spatial object is given as follows:

INSERT INTO table (*geometry*, *name*) VALUES (GeomFromText('POINT 100 200)', 312), 'City A');

### 3.2.4 OGC Abstract Specifications for 3D Solids

According to the Spatial Schema, spatial characteristics are described by one or more spatial attributes whose value is given by a geometric object (GM_Object) or a topological object (TP_Object).

Geometry provides the means for the quantitative description, by means of coordinates and mathematical functions, of the spatial characteristics of features, including dimension, position, size, shape, and orientation. The mathematical functions

used for describing the geometry of an object depend on the type of coordinate reference system used to define the spatial position. Geometry is the only aspect of geographic information that changes when the information is transformed from one geodetic reference system or coordinate system to another.

Topology deals with the characteristics of geometric figures that remain invariant if the space is deformed elastically and continuously – for example, when geographic data is transformed from one coordinate system to another. Within the context of geographic information, topology is commonly used to describe the connectivity of an n-dimensional graph, a property that is invariant under continuous transformation of the graph. Computational topology provides information about the connectivity of geometric primitives that can be derived from the underlying geometry.

#### 3.2.4.1 GM_Solid

OGC defines 3D object as GM_Solid (OGC 2001) and it is a subclass of GM_Primitive and is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces. The boundary defines a sequence sets of GM_Surfaces that limit the extent of this GM_Solid (see Figure 3.4). These surfaces shall be organized into one set of surfaces for each boundary component of the GM_Solid. Each of these shells shall be a cycle (closed composite surface without boundary). In general, a solid in a bounded 3-dimensional manifold has no distinguished exterior boundary. In cases where "exterior" boundary is not well defined, all the shells of the GM_SolidBoundary shall be listed as "interior". The GM_OrientableSurfaces that bound a solid shall be oriented outward – that is, the "top" of each GM_Surface as defined by its orientation shall face away from the interior of the solid. To represent a

3D solid as a volumetric object, GM_Solid is the best abstract spesification defined by OGC. Other than the GM_Solid, some feature geometry such as GM_Composite also involves 3D solid object with other primitives, e.g. point, line, and polygon.

There are some functions or operations that could be implemented by using GM_Solid. The function/operations are:

- Area: the operation shall return the sum of the surface areas of all of the boundary components of a solid. For example:

```
GM_Solid::area() : Area
```

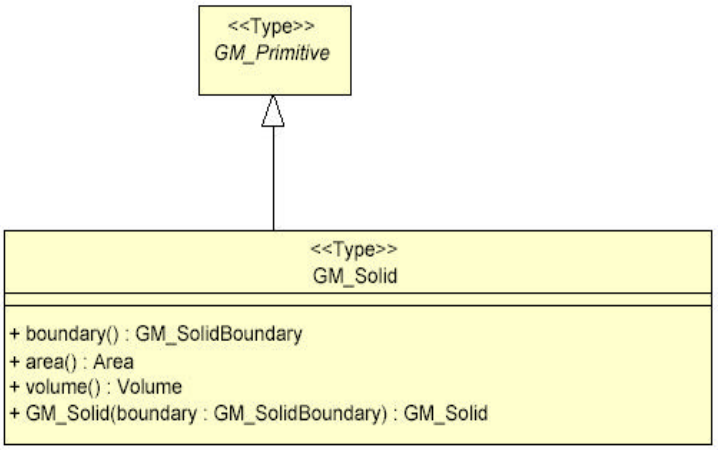



Figure 3.4: GM_Solid data type defined by OGC

- Volume: the operation shall return the volume of this GM_Solid. This is the volume interior to the exterior boundary shell minus the sum of the volumes interior to any interior boundary shell. For example:

```
GM_Solid::volume() : Volume
```

- GM_Solid (constructor): since this standard is limited to 3-dimensional coordinate reference systems, any solid is definable by its boundary. The default constructor for a GM_Solid is from a properly structured set of GM_Shells organized as a GM_SolidBoundary. For example:

```
GM_Solid::GM_Solid(boundary : GM_SolidBoundary) :
GM_Solid
```

Although the OGC does not discuss some operations that refer to 3D solid e.g. 3D intesection between 2 solids, in order to extend to third dimension, similar specifications could be given to the 3D operations, if the z coordinate is considered. The notion for operations provided by OGC as given below:

```
return-type type-1::operation(type-2, type-3 … )
```

Example:

```
Double Precision Geometry 1::Distance(Geometry 2,
Geometry 3)
operation(name-1 : type-1, name-2 : type-2, name-3 :
type-3 …) : return-type, …
```

Example:

```
3D Intersects(A1:Geometry 1, A2:Geometry 2) : Geometry
3
```

There are other 3D objects being considered in OGC specification, i.e. cone, sphere and, etc. Some 3D object are not considered as volumetric solids but still apears in 3D space, i.e. free-form curve and surface. Figure 3.5 denotes the complete list of 3D objects (with highlighted part) considered in OGC specification.

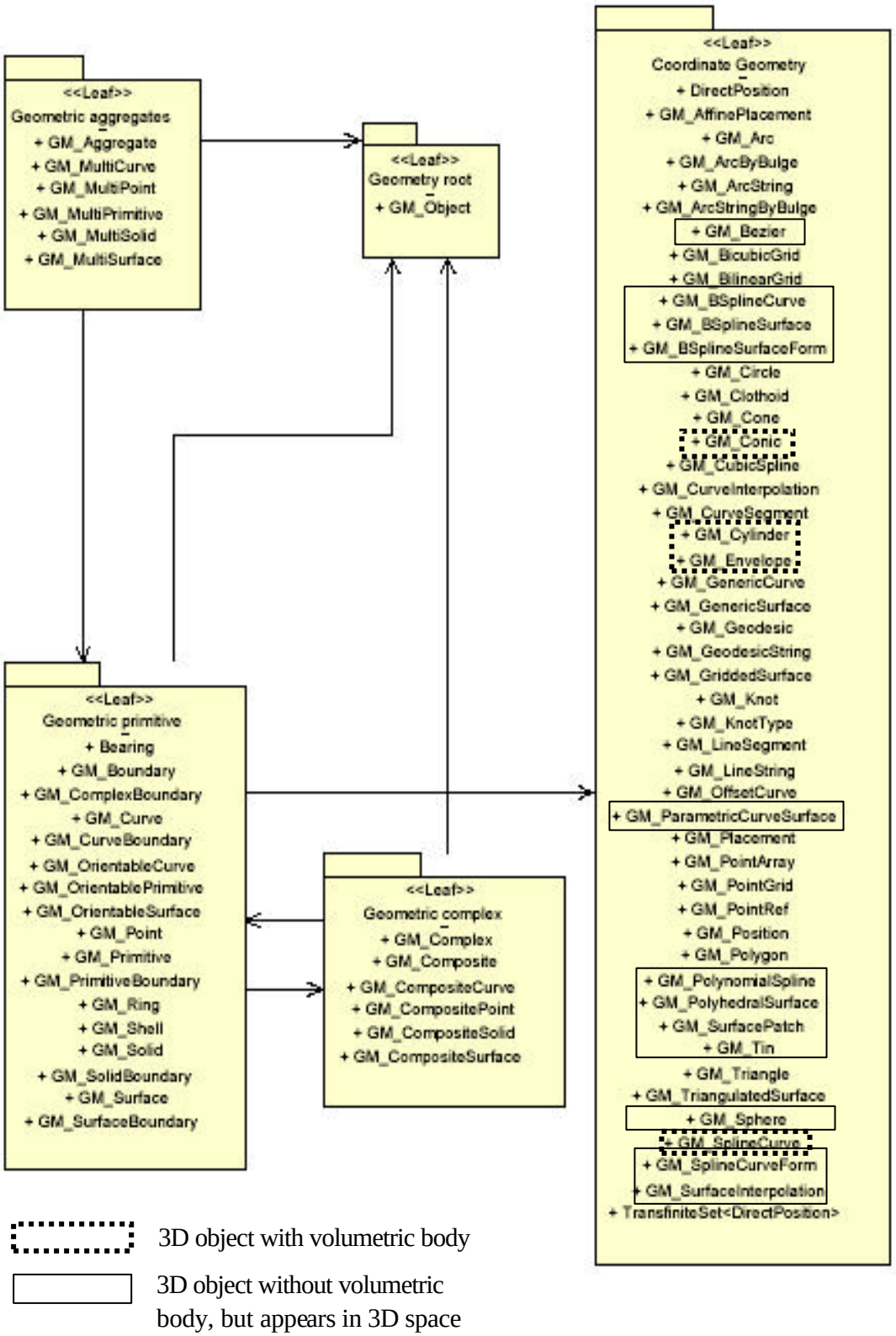
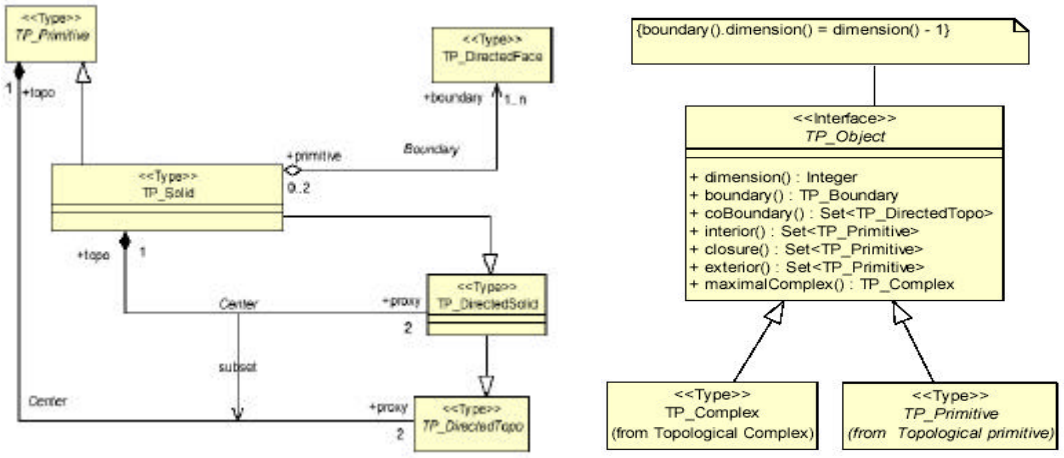



Figure 3.5: Geometry package in OGC abstract specification

### 3.2.4.2 TP_Solid

The OGC also defines solid object that implement the use of topology, i.e. TP_Solid. The main purpose of using topology is to accelerate computational geometry. Geometric calculations such as containment (point-in-polygon), adjacency, boundary, and network tracking are computationally intensive. For this reason, combinatorial structures known as topological complexes are constructed to convert computational geometry algorithms into combinatorial algorithms. Another purpose is, within the geographic information domain, to relate feature instances independently of their geometry. The class "TP_Solid" (see Figure 3.6a) provides topological primitives for GM_Solid. For TP_Solid, the operation “boundary” defined at TP_Object shall return a collection of faces or their negative proxies. This operation is overridden from TP_Object (see Figure 3.6b).




(a) (b)

Figure 3.6: (a). TP_Solid defined, and (b). TP_Object defined by OGC

## 3.3 OGC Implementation Specifications for DBMS

The GM_Solid defined by OGC as a general 3D primitive in abstract specification (OGC 1999a). However, the existing implementation (for SQL) of 3D solid (e.g. polyhedron, tetrahedron) is not available due to the absent of 3D data type (as 3D primitive) within existing DBMS. A volumetric object could be modelled by using multi collection of same or different geometries. OpenGIS implementation specification for 3D solid object can be referred to as *PolyhedralSurface*, and *MultiPolygon*. A *PolyhedralSurface* is a contiguous collection of polygons, that share common boundary segments. It is a subtype of *Surface*. The primitive of *PolyhedralSurface* and *MultiPolygon* are referred to *Polygon* (see Figure 3.8). The different between these two geometries is that the polygons that construct *PolyhedralSurface* must share boundary to the neighboring polygons. The *MultiPolygon* is flexible, i.e. share boundary may not exist for certain polygon(s). For each pair of polygons that "touch", the common boundary shall be expressible as a finite collection of LineStrings. Each such LineString shall be part of the boundary of at most 2 polygon patches. A TIN (triangulated irregular network) is a PolyhedralSurface consisting only of Triangle patches. For any two polygons that share a common boundary, the "top" of the polygon shall be consistent. This means that when two LinearRings from these two Polygons traverse the common boundary segment, they do so in opposite directions. Since the Polyhedral surface is contiguous, all polygons will be thus consistently oriented. This means that a non-oriented surface shall not have single surface representations. Figure 3.7 shows an example of such a consistently oriented surface (from the top). The arrows indicate the ordering of the linear rings that from the boundary of the polygon in which they are located. The methods of implementing the polyhedral surface in DBMS is given as below (see Figure 3.9):

```
NumPatches (): Integer – Returns the number of
including polygons.
```

PatchN (N: Integer): Polygon – Returns a polygon in this surface, the order is arbitrary.

BoundingPolygons (p: Polygon): MultiPolygon – Returns the collection of polygons in this surface that bounds the given polygon “p” for any polygon “p” in the surface.

IsClosed (): Integer – Returns 1 (True) if the polygon closes on itself, and thus has no boundary and encloses.

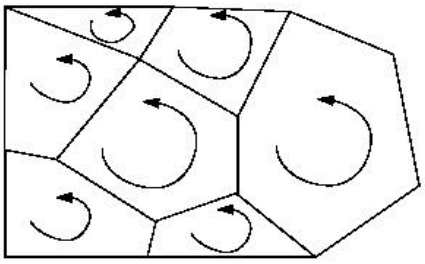


Figure 3.7: *PolyhedralSurface* with consistent orientation

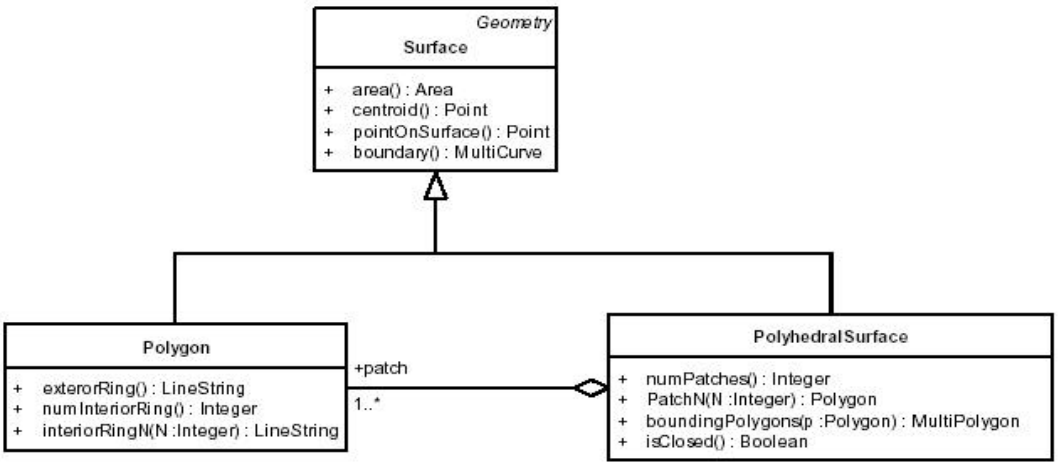



Figure 3.8: Implementation specification for *PolyhedralSurface*

In the implementation specification, OGC provides the geometry function that does not limit to any dimensions. Only DBMS itself decides the implementation of the standard functions (specified by OGC) that considers the third dimension or not. Some of the standard functions given by OGC (Simple Feature Specification for SQL, Revision 1.1) are:

```
Intersection (g1 Geometry, g2 Geometry): Geometry
   Return a Geometry that is the set intersection of
geometries g1 and g2.

Difference (g1 Geometry, g2 Geometry): Geometry
   Return a Geometry that is the closure of the set
difference of g1 and g2.

Union (g1 Geometry, g2 Geometry): Geometry
Return a Geometry that is the set union of g1 and g2.
SymDifference(g1 Geometry, g2 Geometry): Geometry
Return a Geometry that is the closure of the set
symmetric difference of g1 and g2 (logical XOR of
space).

Buffer (g1 Geometry, d Double Precision) : Geometry
Return as Geometry defined by buffering a distance d
around g1)

ConvexHull(g1 Geometry) : Geometry
Return a Geometry that is the convex hull of g1.
```

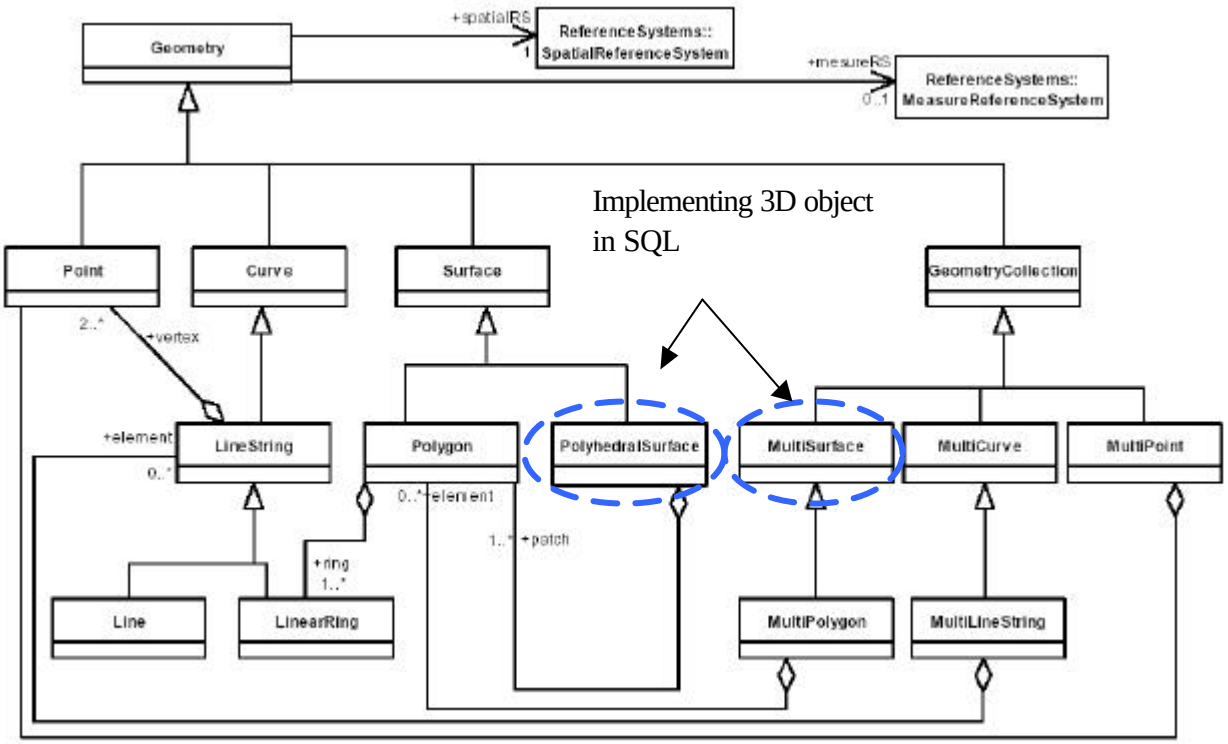



Figure 3.9: SQL Geometry type hierarchy

## 3.4 Summary and Conclusion

The OGC abstract and implementation specifications provide the standard method to define spatial objects and their functionalities. The research was attempted to extend the existing two-dimensional standard specifications to the third dimension. UML diagram was used to present the design of spatial object, in order to define the properties and functions for 3D spatial object. The study also discussed the Well-Known-Text, as what PostGIS expresses 3D object in DBMS, e.g. point, line, and polygon in 3D. The solid object will follow the WKT method in order to express Polyhedron as a solid object within DBMS. The existing implementation (for SQL) of 3D solid (e.g. polyhedron, tetrahedron) is not available due to the absent of 3D data type

(as 3D primitive). An alternative to model volumetric object is to use multi collection of same or different geometries. The study will implement multi collection of *Surface*, similar to *MultiPolygon*, with minor modification, in order to create a solid object, i.e. Polyhedron. This user-defined solid object will be used as input for 3D spatial operation, which extend the standard implementation specification for SQL. Next chapter explains the related works and study for 3D spatial operations for geometry and topology data types.

# CHAPTER IV

# FUNDAMENTAL SET AND EULER THEORIES

## 4.1 Introduction

The second stage of the research started with a review fundamental set and Euler theories. The strength of set theory is focused on the topological data modeling for 3D spatial data object. On the other hand, the strength of Euler theory is defined in the validation function for 3D spatial object, for this research, polyhedron. This chapter discusses the fundamental issue for set and Euler theories. The discussion continues by extending these theories for 3D spatial data modeling for 3D GIS. The discussion will be implemented in the next chapter to verify the developed topological data structure and the validation function is applicable within DBMS environment.

## 4.2 Set Theory

Set theory is the mathematical theory of sets, which represent collections of abstract objects. It often encompasses as Venn diagrams about collections of objects. Set theory provides the language in which mathematical objects are described. Along with logic and the predicate calculus, it is one of the axiomatic foundations for mathematics, allowing mathematical objects to be constructed formally from the undefined terms of "set" and "set membership".

In naive set theory, sets are introduced and understood using what is taken to be the self-evident concept of sets as collections of objects considered as a whole. In axiomatic set theory, the concepts of sets and set membership are defined indirectly by first assuming certain axioms which specify their properties. In this conception, sets and set membership are fundamental concepts like point and line in Euclidean geometry, and are not themselves directly defined.

### 4.2.1 Naive Set Theory

In naive set theory, a set is described as a well-defined collection of objects. These objects are called the elements or members of the set. Objects can be anything: numbers, people, other sets, etc. For instance, 4 is a member of the set of all even integers. Clearly, the set of even numbers is infinitely large; there is no requirement that a set be finite.

If x is a member of A, then it is also said that x belongs to A, or that x is in A. In this case, it can be written as $x \in A$. The symbol $\notin$ is sometimes used to write $x \notin A$, meaning "x is not in A".

Two sets A and B are defined to be equal when they have precisely the same elements, that is, if every element of A is an element of B and every element of B is an element of A. Thus, a set is completely determined by its elements; the description is immaterial. For example, the set with elements 2, 3, and 5 is equal to the set of all prime numbers less than 6. If the sets A and B are equal, this is denoted symbolically as A = B (as usual).

It is also allowed for an empty set, often denoted Ø and sometimes {}: a set without any members at all. Since a set is determined completely by its elements, there can only be one empty set.

### 4.2.2 Axiomatic Set Theory

Axiomatic set theory is a version of set theory, in which axioms are taken as uninterpreted rather than as formalizations of pre-existing truths. Axiomatic set theory is a first order logical structure. First order logic works with propositions, i.e., logical statements constructed according to the rules of logic and that can take two values. For convenience we call these two values "True" and "False". Set theory, and thus the entire body of mathematics reduces to logical propositions that use the following elements:

1. Variables (e.g., $a, b, \cdots x, y, z$), which stand for sets.

2. The predicate $\in$, which stands for element inclusion. For example, if the proposition $(x \in y)$ takes the value true, we know that both x and y are sets and that x is an element of y. For example, the proposition

   $\{1, 2, 3\} \in \{\{1, 2\}, \{4, 5\}, \{1, 2, 3\}\}$ (3) takes the value "True".

3. Logical operators
   (a) $\neg P$, where $\neg$ is the logical "negation" operator.
   (b) $P \wedge Q$, where $\wedge$ is the logical "and" operator.
   (c) $P \vee Q$, where $\vee$ is the logical "or" operator.
   (d) $P \rightarrow Q$, where $\rightarrow$ is the logical "implication" operator.
   (e) $P \leftrightarrow Q$, where $\leftrightarrow$ is the logical "bijection" operator.
   (f) $\forall x P$ is the logical "for-all" quantifier.
   (g) $\exists x P$ is the logical "exists" quantifier.

All propositions in set theory are built out of atomic propositions of the form $(x \in y)$ connected using the logical operators. If P and Q are propositions, e.g., P could be $(x \in y)$ and Q could be $(y \in z)$ then $\neg P$, $P \wedge P$, $P \vee Q$, $P \rightarrow Q$, $P \leftrightarrow Q$, $\forall x P$ and $\exists x P$ are also propositions. The effect of the connectives on the truth value of propositions is expressed in Table 4.1:

| P | Q | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \rightarrow Q$ | $P \leftrightarrow Q$ |
|---|---|---|---|---|---|---|
| T | T | F | T | F | T | T |
| T | F | F | F | T | F | F |
| F | T | T | F | T | T | F |
| F | F | T | F | T | T | T |

Table 4.1: Logical operators, T = TRUE; F = FALSE

## 4.3 3D Topological Data Modeling for DBMS

A lot of work had been done in the management of 3D topological data in geo-DBMS environment. Existing 3D spatial model (3D FDS by Molenaar (1990), TEN by Pilouk (1996), and SSS by Zlatanova (2000)) are mainly aimed to perform visualization analysis. Different spatial models involve different primitives and constructive object, and thus, performances are differed to each other. All the faces that construct a body, in this case, a polyhedron, should be mentioned in the relationship model. Following this strategy, three models will be discussed extensively. Topological structures could be used to represent planar partitions or linear network without redundancy. Relational DBMSs can store very well the topological references: area left and right of a boundary, boundary to boundary references etc. They also support certain topology structure management, as they are able to do consistency checks, providing spatial queries and functions. Currently, the DBMSs do seem to offer the most flexible platform for implementing the complex features. DBMS offers the possibility to navigate through the database, controlled by programming the implementation of methods.

### 4.3.1 3D Formal Data Structure (3D FDS)

The design of 3D FDS (Molenaar, 1990) design is based on the decomposition of a feature into identifier, geometry, and theme. The model consists of three fundamental levels (see Figure 4.1): feature (related to a thematic class), four elementary objects (point, line, body and surface) and four primitives (node, arc, face and edge). A node and an arc will be created in the first place. Edge is extended by arc in the sense that the series of arcs could be backward or forward. Edges are additional geometric primitives providing the link between arcs and faces and so permitting the unique reference to left

and right bodies that are 3D features. Faces are 2D geometric primitives in addition to nodes and arcs, respectively of 0 and 1 dimension. The data model distinguishes four types of features, i.e. point, line, surface and body. A point feature consists of a node; a line feature consists of one or more arcs; a surface consists feature of one or more faces; and a body feature is bounded by faces. The role of the edge is dual, i.e. to define the border of a face (relationship face-arc) and establish an orientation for a face, which is needed to specify left and right body. The number of arcs constituting an edge is not restricted. Arcs are straight lines and faces are planar. The surface has one outer boundary and may have several non-nested boundaries, i.e. may have holes or islands. The body has one outer surface without a boundary and can have several non-nested bodies or holes.

The fundament concept of 3D FDS is referred to a single-valued map that able to partition the space into non-overlapping objects, and thus ensuring 1:1 relationships between elementary object and primitives of same dimensions, e.g. surfaces and face. Primitives of different dimensions can overlap, e.g. relationships node-on-face, arc-on-face, node-in-body and arc-in-body are explicitly stored. Thus, singularities are permitted in this model. In general, 3D FDS contains all the necessary data to visualize the geometry of objects. The model also provides the orientation of faces, which is crucial for the correct rendering. Therefore, The 3D FDS is suitable for direct representation and query-based spatial analysis. However, with the absent of 3D primitive data type, e.g. tetrahedron or polyhedron, the model is not suited to solving problems for applications that involve complex computation.

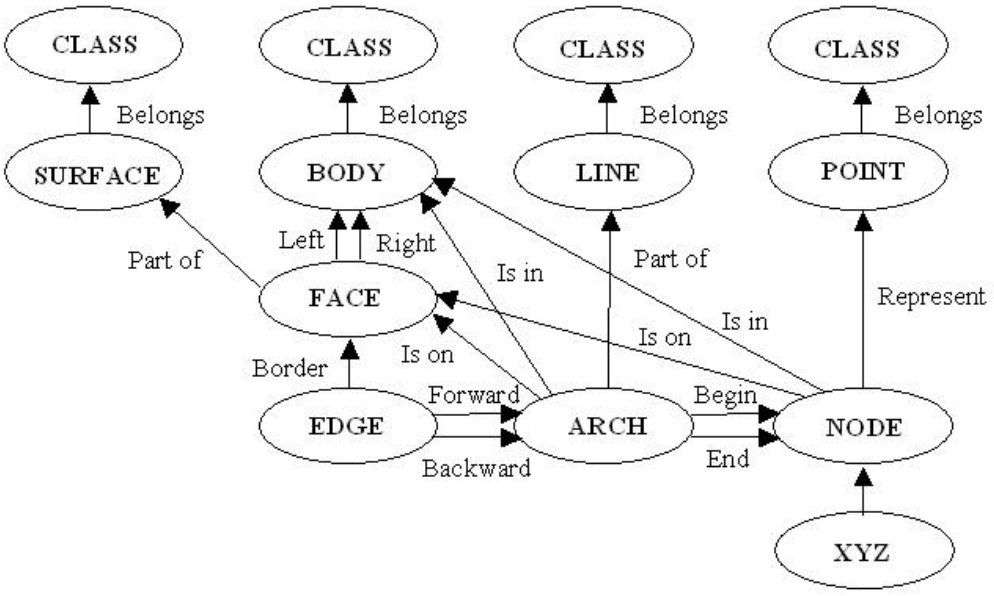



Figure 4.1: 3D FDS

### 4.3.2 TEtrahedron Network (TEN)

TEN was introduced by Pilouk (see Pilouk 1996) to overcome some difficulties of 3D FDS in modelling objects with indiscernible boundaries. TEN (Figure 4.2) had extended the TIN-based data model to facilitate handling of 3D objects in order to stretch the capability of the integrated data model in both dimensionality and computability. The triangular network had been generalized into a tetrahedral network. The general properties of a tetrahedron are the same as a triangle's, where each is a simplex of its dimension and convex. Some important properties of their networks, for example, locality, fidelity and capability of embedding features are also the same. The latter indicates that the geometry of the features can also be maintained within the tetrahedral network (TEN), which means that TEN also has the capability of maintaining human knowledge about the real world. However, TEN created a body-feature that

involves a contiguous set of tetrahedrons as a whole, so as to surface-feature of triangles, and line-feature of arcs. The model constraints self-overlapping or self-intersecting of a feature are not allowed, and thus, avoiding singularity in the object. According to the definitions, TEN has four primitives, i.e. node, arc, triangle, and tetrahedron. The ARC table gives the relationship of arc-node; the TRIANGLE table contains the tetrahedron-triangle-edge link. A body object is composed of tetrahedrons, a surface object of triangles, a line object of arcs and a point object of nodes. The general rule for creating the model is based on the fact that each node is part of an arc, each arc is part of a triangle and each triangle is part of a tetrahedron. The model is appropriate for representing irregularities in the real world.

The complete tetrahedronization constrained the triangles for both 2.5D and 3D objects. For the logical model, the list of the triangles will be increased if certain triangles are placed in the same plane. Moreover, the space is completely subdivided into tetrahedrons, which the interiors of objects, as well as the open space, are also decomposed into tetrahedrons. These tetrahedrons will lead to database size expansion.

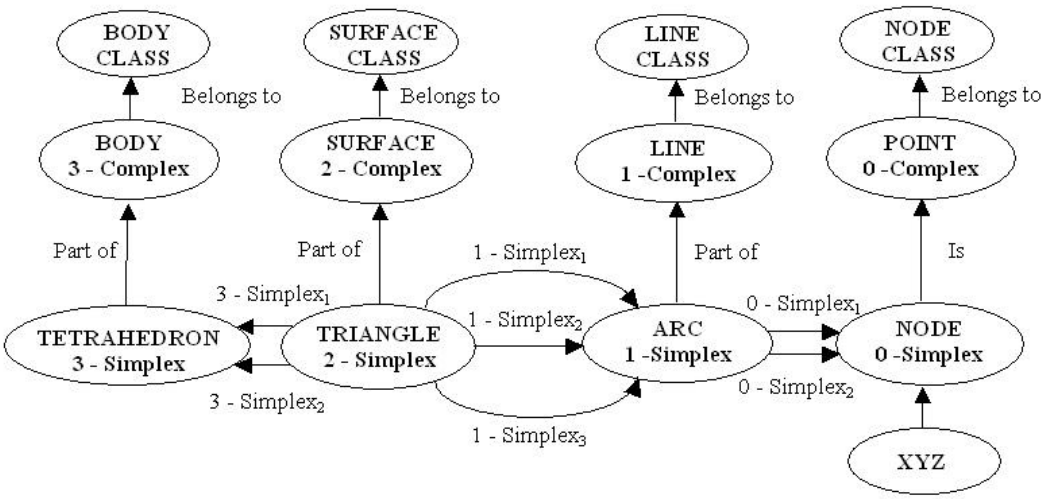



Figure 4.2: TEN

### 4.3.3 Simplified Spatial Model (SSM)

Simplified Spatial Model (Figure 4.3) was developed by Zlatanova (2000). It was named as "simplified" because arcs are not used to construct objects. 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. The Simplified Spatial Model differs from other 3D spatial topological models in the number of constructive objects used, i.e. nodes and faces. The geometric objects (known as complexes, cell complexes, feature objects) are the same. It is similar to 3D FDS and some modifications of the cell model in permitting singularities. The model allows arbitrary shapes (but convex faces) as the cell models and 3D FDS do. Similarly to TEN, triangulation of real surfaces is a basic operation to resolve interactions (point on, line on surface or face) with other objects and complex shapes (holes, concave faces). The complete triangulation of all the surfaces may be considered a modification of TEN. The TRIANGLE and ARCLINE tables will contain only the identifiers of the nodes instead of arcs.

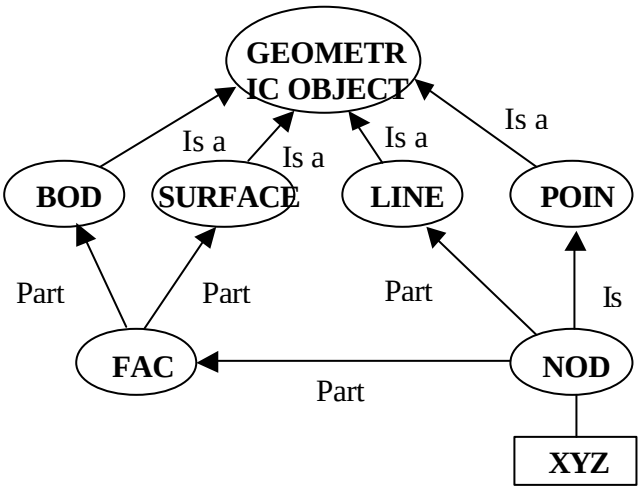



Figure 4.3: Simplified Spatial Model

### 4.3.4 Modified Topological Structure Model

In this section, the definition of a modified topological model will be given. In order to produce a model that implements only the most essential primitives, arc and surface are not used to construct objects. Due to the arcs store multiple nodes information within Geo-DBMS database, lines are used to reduce datasets storage and improve the abilities of data retrieval and spatial query. Besides, a surface created from multiple faces also omitted from this model. This is because a single but planar face may give a common boundary either for two solid objects, or topological space and an object. On the other hand, a surface that created by faces may or may not represent a common boundary between two solid objects. Furthermore, topological relationship that implements surface will become more complex, i.e. more relationship will be involved within node-surface, line-surface, surface-surface, or even surface-solid (Chen *et. al*, 2005). Therefore, three primitives are given, i.e. node, line, and face.

The geometry of each spatial object can be associated with four abstractions of geometric objects, i.e. *point*, *line*, *face* and *solid*. A *point* is a type of spatial object that does not have shape or size but position is the topological space. A *line* is a type of a spatial object that has length and position. A *face* is a type of spatial object that has position and area. A *solid* is a type of spatial object that has a position and a volume. Figure 4.4 denotes the conceptual design of modified topological data model. The three primitive objects are used to compose the four feature objects, i.e. *point*, *line*, *face* and *solid object*. However, the feature objects are constructed using these rules as follows: nodes construct points and lines; lines construct faces; faces construct solid objects.

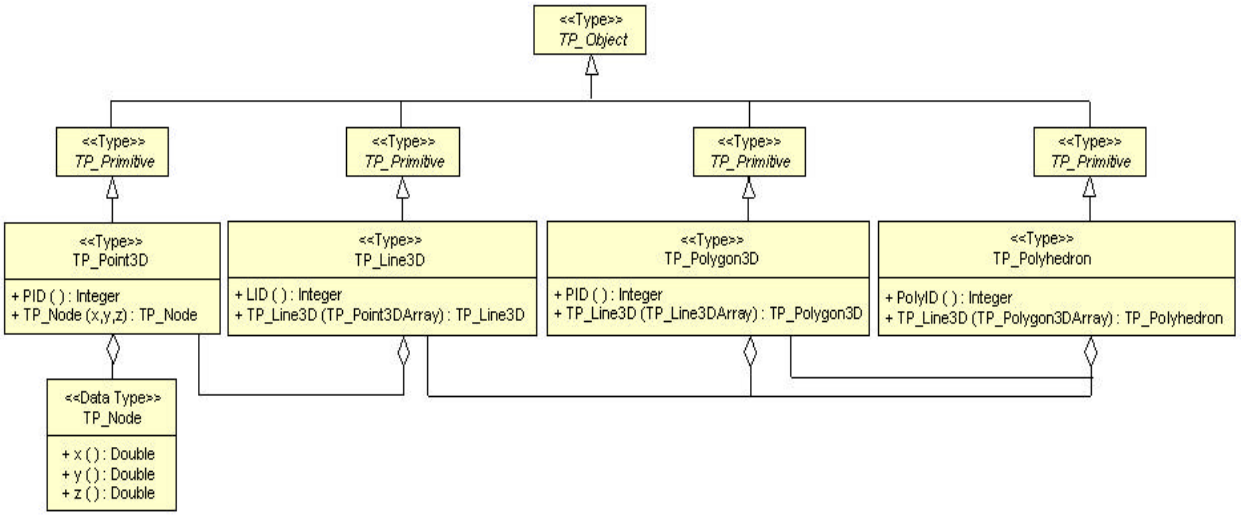
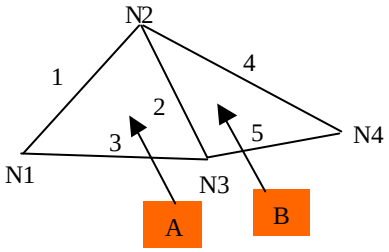



Figure 4.4: Modified topological data structure

## 4.4 Implementation of Set Theories for Spatial Data Modeling

In GIS, the vector data model that used for geographic phenomena may be represented by geometric entities (primitives) like points, lines, polygons, and volumetric solid object. With the integrated Geo-DBMS module (such as Oracle Spatial, 2002), geometric objects can be stored together with topological information. These vector data models that include the description of topology, as well as the location of the spatial entities will be stored. In general, topology in GIS requires a data structure, where common boundary between two adjacent areas is stored as a single line, simplifying the map maintenance. On the other hand, geometric entities require full data insertion of any object. For instance, two triangles that share a common boundary stores topological information of six lines. However, in geometric condition, they will be stored twice of four nodes. Figure 4.5 gives the difference between topological and geometric data storage in Geo-DBMS.




**TOPOLOGICAL DATA**

| FaceID | LineID |
|---|---|
| A | 1-**2**-3 |
| B | 4-5- |

Note: 1). **2** represents the common boundary
2). Line **2** represents node N2 to N3

Figure 4.5: Topological data structure

Beside reducing data storage, Geo-DBMS allows multi-user control on shared data and crash recovery, automatic locks of single objects while using database transactions, advanced database protocol mechanisms to prevent the loss of data, data security, data integrity and operations that comfortably retrieve, insert and update data (Bruenig and Zlatanova (2004), Patel *et. al* (1997)). In this research, capabilities of topology in handling spatial datasets in GIS will be discussed. It involves spatial primitives that deal with spatial data recognitions (semantics) and their relationships among objects. A new framework for representing spatial model is introduced. The model implements topological mechanism for object semantic and relationship that able to represent the real world, i.e. node, line, face, solid3D.

### 4.4.1 Geometric Properties

*Node* is defined as single coordinate triplet represented by (x, y, z) in three-dimensional space ($\mathbf{R}^3$). It appears as 0D object in 3D Euclidean space. It is used to represent objects that are best described as shape- and sizeless, single-locality features. The location of each different node must be unique. The interior of a point is the empty set, denoted by $P^\circ$. For any cases, interior of point will not be related to any kind of

primitives due to the intersection results is an empty set. The boundary of a point, denoted by $\partial P$, is the point by itself. Finally, the exterior of a point is denoted by $P^-$.

*Line* is defined as series of nodes connecting together with an appropriate sequence. It is a one-dimensional spatial entity that defines a path through 2D or 3D space. The geometry of line is defined by an ordered collection of two or more distinct coordinate tuples. The orientation of a line is defined by the ordering of its coordinate tuples.

A line is used to represent one-dimensional objects such as road, railways, canals, rivers, or even power lines in 3D Euclidean space. A line connecting two nodes forms an arc. However, two end nodes connecting a vertex or more vertices define a line. The straight parts of a line (or an arc) between two consecutive vertices are called line segments. Collections of (connected) lines may represent as network in real applications. These end points are defined as the border of a line, denoted by $\partial l$. The interior of line is defined as the line segment itself, denoted by $l^o$. The exterior of line is denoted by $l^-$.

*Face* is defined as a planar areal object. This areal feature is determined by series of lines with an appropriate connected sequence, forming a closed boundary. It appears as 2D object in 3D Euclidean space. A face connects three points forms a triangle, otherwise, with at least four points forms a polygon. In most cases, the coordinate tuples of the bounding edges of a face are not necessarily coplanar. Any entity nodes that are contained within the face also help to define its three-dimensional shape. The starting and ending points are the same point that defined as the border of polygon, denoted by $\partial Po$.

In three-dimensional space, a face shares only between two solid objects. This implies a face has two sides. The orientation of a face is defined by the order of the edges that make up its outer boundary. The "top" side of a face is the side for which the outer boundary is defined in counter-clockwise order; otherwise the inner boundary defines in clockwise order. In order to support vertical and overhanging surfaces, the orientation of the "top" side of a face must be capable of being defined to be an arbitrary direction, not necessarily parallel to the positive Z-axis. Therefore, a face must have an explicit "up" vector. Either a simple face or face with holes, the properties of face will not affect any topological relationship between other objects. The different between these two kinds of face is the face with hole consists of two or more borders, whereas the simple face only remains one border. The interior of a face is defined as the area within its boundary, denoted by $Po^{o}$. The exterior of face is denoted by $P^{-}$.

A *solid* is defined as indexed set of faces (polygons) joining together that forms a volumetric object. It appears as 3D object in 3D Euclidean space. A solid may be topologically linked to nodes, edges, and faces. These nodes, lines, and faces that form a solid are defined as the border of solid, denoted by $\partial So$. In the topological model, either a simple 3D object or solid object with holes, the properties of polygon will not affect the topological relationship between other objects. The different between these two kinds of solid is the one with hole consists of two or more borders, whereas the simple solid only remains one border. The interior of solid is defined as the closure of all borders, denoted by $So^{o}$. The exterior of solid is denoted by $So^{-}$.

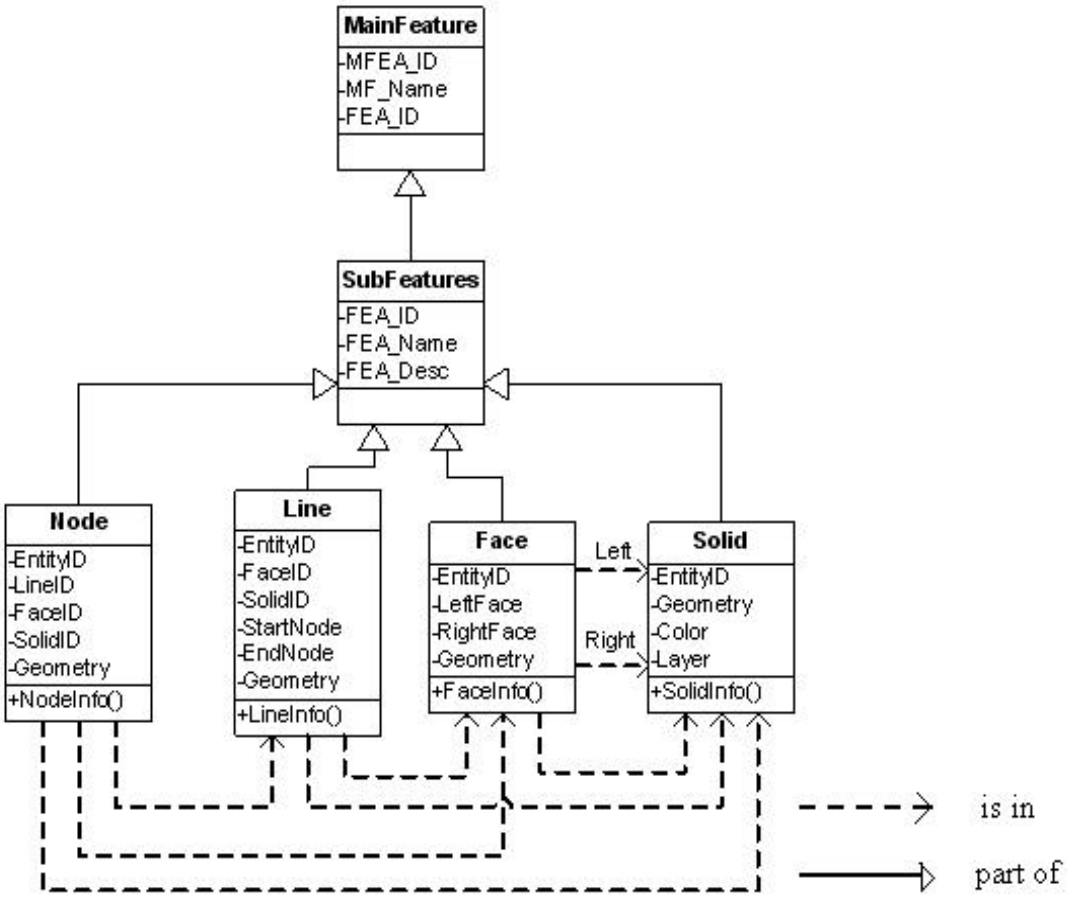
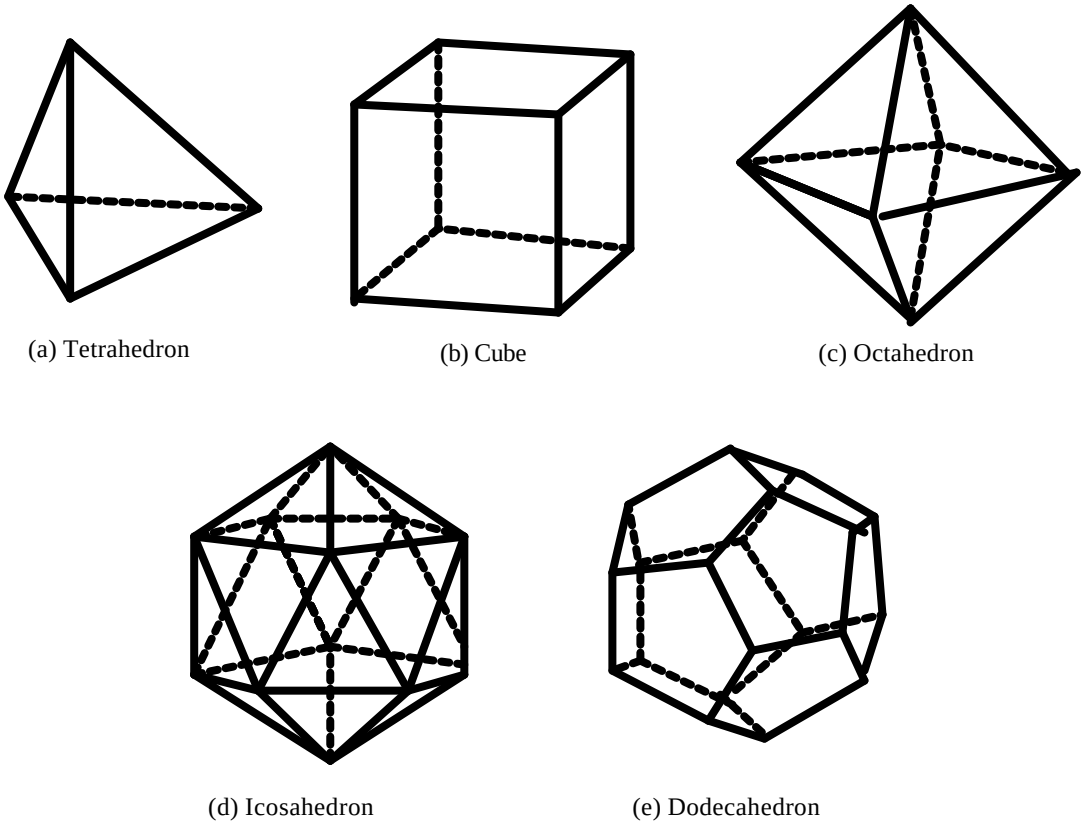



Figure 4.6: The conceptual design of topological data structure

## 4.5 Euler Theory

In algebraic topology, the Euler characteristic is a topological invariant, a number that describes one aspect of a topological space's shape or structure. It is commonly denoted by **?** (Greek letter chi). The Euler characteristic was originally

defined for polyhedra and used to prove various theorems about them, including the classification of the Platonic solids (see Figure 4.7).



(a) Tetrahedron (b) Cube (c) Octahedron

(d) Icosahedron (e) Dodecahedron

Figure 4.7: Sample of Platonic solids

The Euler characteristic, ? was classically defined for polyhedra, according to the formula

$$? = V - E + F$$

where $V$, $E$, and $F$ are respectively the numbers of vertices, edges and faces in the given polyhedron. A convex polyhedron is homeomorphic to a sphere, so its Euler characteristic is

$$? = V - E + F = 2$$

This result is known as Euler's formula, and can be applied not only to polyhedra but also to embedded planar graphs. A proof is given below.

| **Platonic solids** | Vertices | Lines | Faces | Euler characteristic (V-E+F) |
|---|---|---|---|---|
| Tetrahedron | 4 | 6 | 4 | 2 |
| Cube | 8 | 12 | 6 | 2 |
| Octahedron | 6 | 12 | 8 | 2 |
| Dodecahedron | 20 | 30 | 12 | 2 |
| Icosahedron | 12 | 30 | 20 | 2 |

Table 4.2: Platonic solids (without hole) that apply the Euler characteristic

### 4.5.1 The Generalization of Euler's Formula

A slight generalization of Euler's formula to take into account polyhedron that having holes.

$$V - E + F = 2 - 2G$$

where G is the number of holes in the polyhedron. Thus the Euler characteristic is 2 for a regular polyhedron but 0 for a torus-like polyhedron (see Figure 4.8). It was the French mathematician Henri Poincaré who fully generalized Euler's formula.

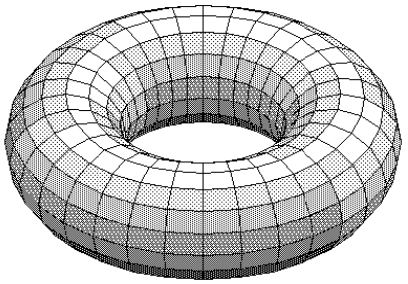


Figure 4.8: Torus-like polyhedron

### 4.5.2 The Euler-Poincaré Formula

The Euler-Poincaré formula describes the relationship of the number of vertices, the number of edges and the number of faces of a manifold. It has been generalized to include potholes and holes that penetrate the solid. To state the Euler-Poincaré formula, we need the following definitions:

V: the number of vertices

E: the number of edges

F: the number of faces

G: the number of holes that penetrate the solid, usually referred to as genus in topology

S: the number of shells. A shell is an internal void of a solid. A shell is bounded by a 2-manifold surface, which can have its own genus value. Note that the solid itself is counted as a shell. Therefore, the value for S is at least 1.

L: the number of loops, all outer and inner loops of faces are counted.

Then, the Euler-Poincaré formula is the following:

$$V - E + F - (L - F) - 2(S - G) = 0$$

Examples

1). A cube has eight vertices (V = 8), 12 edges (E = 12) and six faces (F = 6), no holes and one shell (S=1); but L = F since each face has only one outer loop. Therefore, we have

$$V-E+F-(L-F)-2(S-G) = 8-12+6-(6-6)-2(1-0) = 0$$

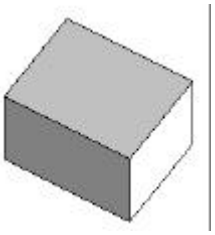


Figure 4.9: A cube

2). The following solid has 16 vertices, 24 edges, 11 faces, no holes, 1 shell and 12 loops (11 faces + one inner loop on the top face). Therefore,

$$V-E+F-(L-F)-2(S-G) = 16-24+11-(12-11)-2(1-0)=0$$

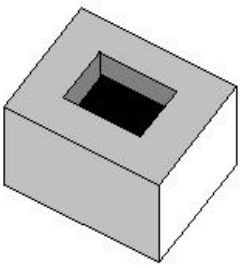


Figure 4.10: A cube with one inner loop on the top face

3). The following solid has 16 vertices, 24 edges, 10 faces, 1 hole (i.e., genus is 1), 1 shell and 12 loops (10 faces + 2 inner loops on top and bottom faces). Therefore,

V-E+F-(L-F)-2(S-G) = 16-24+10-(12-10)-2(1-1)=0

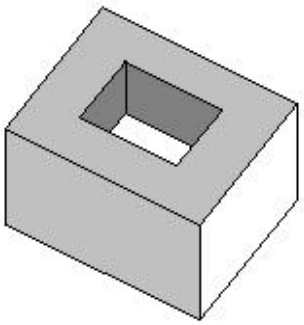


Figure 4.11: A cube with hole

### 4.5.3 Validation Using TEtrahedral irregular network (TEN)

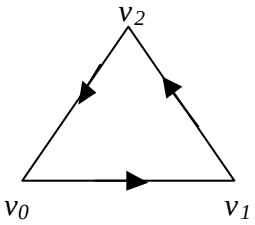
The basic preference for these simplex-based data structures is based on certain qualities of simplexes:

- Well defined: a n-simplex is bounded by n + 1 (n - 1)-simplexes. E.g.
  - 3-simplex (TEN) is bounded by 4 2-simplexes (4 triangles)
  - 2-simplex (triangle) is bounded by 3 1-simplexes (3 edges)
  - 1-simplexes (edge) is bounded by 2 0-simplexes (2 nodes)

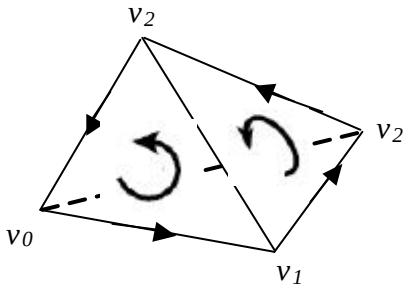
- Flatness of faces are defined by three points (triangle)

The TEN structure consist of nodes, edges and triangles. A formal definition of a n-simplex $S_n$ can be given: a n-simplex $S_n$ is the smallest convex set in Euclidian space $IR_m$ containing n + 1 points $v_0, \ldots, v_n$ that do not lie in a hyperplane of dimension less than n. As the n-dimensional simplex is defined by n + 1 nodes, it has the following notation: $S_n = <v_0, \ldots, v_n>$. The boundary of a n-simplex is defined by the following sum of n - 1 dimensional simplexes (Figure 4.12):


$S_1 = <v_0, v_1>$ $\quad \partial S_1 = <v_1> - <v_0>$




$S_2 = <v_0, v_1, v_2>$ $\quad \partial S_2 = <v_1, v_2> - <v_0, v_2> + <v_0, v_1>$




$S_3 = <v_0, v_1, v_2, v_3>$ $\quad \partial S_3 = <v_1, v_2, v_3> - <v_0, v_2, v_3> + <v_0, v_1, v_3> - <v_0, v_1, v_2>$

Figure 4.12: TEN model

### 4.5.4 Validating the data structure

A TEN is valid when it is a full decomposition of space, i.e. there are no overlaps or gaps in the structure. Additional requirements are that each tetrahedron has positive orientation, meaning that the normal vectors of the bounding triangles all point outwards. The data structure can be validated by applying the Euler- Poincaré formula:

N- E+F - V = 0 with N the number of nodes,
E the number of edges,
F the number of faces, and
V the number of volumes (including the exterior).

As can be seen in Figure 4.10, the Euler- Poincaré formula holds for all simplicial complexes, including simplicial complexes that consist of simplexes of different dimensions. Due to this characteristic dangling edges and faces cannot be detected, but for instance holes (i.e. missing faces) can be detected. Within the simplicial complex-based approach the validation strategy is to start with a valid tetrahedronization and to check every update for correctness before committing it to the database. As a result one will migrate from one valid state into another valid state. This strategy will also include the application of for instance flipping algorithms for the deletion of vertices; as such algorithms are designed to maintain a valid TEN during each step of the process. Other correctness checks can be implemented, like for instance a check on the triangle view to ensure that every triangle appears two times (with opposite sign, ignoring the inherited object id's). Also validation on feature level can be considered, for instance one can check whether all constrained triangles form a valid polyhedron.

## 4.6 Summary and Conclusion

The implementations of set and Euler theories for topological data structure and object validation were given in this chapter. The review of existing topological model, e.g. 3D FDS, TEN, and SSM were discussed in order to provide background theories related to topological model, and later with the implementation of set theories, an modified topological data structure for 3D solid object will be developed. All related geometric properties were mentioned in the design of the modified topological data structure. The developed 3D spatial data modelling will be validated using Euler Poincaré Theorem. The implementation of developed idea will be verified in the next chapter, where the experiment is carried out within geo-DBMS environment.

# CHAPTER V

# IMPLIMENTATION AND EXPERIMENT

## 5.1 Introduction

The review of concept and background of set and Euler theories for 3D spatial data modeling had been discussed in chapter 4. The approach mentioned from the previous chapters will be tested. This chapter explains the methodology of constructing new 3D topological data structure (implements the set theory) and validation function (implements the Euler theory) in DBMS environment. The intention of developing the topological data structure is to create a new data type in 3D, i.e. polyhedron that implements the set theory for structuring 2D primitives in order to construct a 3D solid object. The characteristics of user-defined data type in DBMS environment will be discussed at the beginning of the research. It is followed by the validation function that implements the Euler theory to validate the input data for DBMS. The implementation is divided into 2 phases, where the first part deals with the topological database of spatial data modeling, and the second part deals with the spatial function, i.e. the validation function.

## 5.2 User-defined function for DBMS

Existing DBMS only manage to provide two-dimentional objects with 3D coordinates. The absent of 3D primitive, e.g. tetrahedron or polyhedron, causes the inavailability of 3D operations. However, some of the commercial DBMS, e.g. PostgreSQL, enable user to create new user-defined datatype and functions. This user-defined datatype and functions can be written in C (or a language that can be made compatible with C, such as C++). For the purpose of research, only user-defined function will be discussed, as it can be implemented for validation function. However, for the 3D data type, i.e. polyhedron, the complete topological data structure will be developed within geo-DBMS.

### 5.2.1 Calling Conventions Version 0 for C-Language Functions

Although this approach is now deprecated, it's easier to get a handle on initially. In the version-0 method, the arguments and result of the C function are just declared in normal C style, but being careful to use the C representation of each SQL data type as shown above.

Here are some examples:

```
#include "postgres.h"
#include <string.h>

int add_one(int arg)
{
    return arg + 1;
}

float8 *add_one_float8(float8 *arg)
{
    float8    *result = (float8 *)
```

```
    palloc(sizeof(float8));
    *result = *arg + 1.0;
    return result;
}
```

Supposing that the above code has been prepared in file *sample.c* and compiled into a shared object, we could define the functions to PostgreSQL with commands like this:

```
CREATE FUNCTION add_one(integer) RETURNS integer
     AS 'DIRECTORY/funcs', 'add_one'
     LANGUAGE C STRICT;

CREATE FUNCTION add_one(double precision) RETURNS double
precision
     AS 'DIRECTORY/funcs', 'add_one_float8'
     LANGUAGE C STRICT;
```

The DIRECTORY stands for the directory of the shared library file (for instance the PostgreSQL tutorial directory, which contains the code for the examples used in this section). (Better style would be to use just 'funcs' in the AS clause, after having added DIRECTORY to the search path. In any case, user may omit the system-specific extension for a shared library, commonly *.so)

Notice that user has specified the functions as "strict", meaning that the system should automatically assume a null result if any input value is null. By doing this, user avoid having to check for null inputs in the function code. Without this, user would have to check for null values explicitly, by checking for a null pointer for each pass-by-reference argument.

Although this calling convention is simple to use, it is not very portable. On some architecture, there are problems with passing data types that are smaller than *integer*. Also, neither there is no simple way to return a null result, nor to cope with

null arguments in any way other than making the function strict. The version-1 convention, presented next, overcomes these objections.

### 5.2.2 Calling Conventions Version 1 for C-Language Functions

The version-1 calling convention relies on macros to suppress most of the complexity of passing arguments and results. The C declaration of a version-1 function is:

```
Datum funcname(PG_FUNCTION_ARGS)
PG_FUNCTION_INFO_V1(funcname);
```

In addition, the macro call must appear in the same source file. (Conventionally. it's written just before the function itself.) This macro call is not needed for internal-language functions, since PostgreSQL assumes that all internal functions use the version-1 convention. It is, however, required for dynamically-loaded functions.

In a version-1 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. Here denotes the same functions as above, coded in version-1 style:

```
#include "postgres.h"
#include <string.h>
```

```
#include "fmgr.h"

PG_FUNCTION_INFO_V1(add_one);

Datum add_one(PG_FUNCTION_ARGS)
{
    int32   arg = PG_GETARG_INT32(0);
    PG_RETURN_INT32(arg + 1);
}

PG_FUNCTION_INFO_V1(add_one_float8);

Datum add_one_float8(PG_FUNCTION_ARGS)
{
    float8   arg = PG_GETARG_FLOAT8(0);
    PG_RETURN_FLOAT8(arg + 1.0);
}
```

The CREATE FUNCTION commands are the same as for the version-0 equivalents.

At first glance, the version-1 coding conventions may appear to be just pointless obscurantism. They do, however, offer a number of improvements, because the macros can hide unnecessary detail. An example is that in coding add_one_float8, user no longer needs to be aware that `float8` is a pass-by-reference type. Another example is that the `GETARG` macros for variable-length types allow for more efficient fetching of values.

One big improvement in version-1 functions is better handling of null inputs and results. The macro `PG_ARGISNULL(n)` allows a function to test whether each input is null. (Of course, doing this is only necessary in functions not declared "strict".) As with the `PG_GETARG_xxx()` macros, the input arguments are counted beginning at zero. Note that one should refrain from executing `PG_GETARG_xxx()` until one has verified that the argument isn't null. To return a null result, execute `PG_RETURN_NULL()`; this works in both strict and non-strict functions.

### 5.2.3 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 precise, a *shared library* needs to be created.

Creating shared libraries is generally analogous to linking executables. 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. (Object files intended for executables are usually not compiled that way.) The command to link a shared library contains special flags to distinguish it from linking an executable. In the following examples, it is assumed that the source code is *sample.c* and a shared library *sample.so* will be created. The intermediate object file will be called *sample.o* unless otherwise noted. A shared library can contain more than one object file, but only one will be used

The dynamic loading feature is what distinguishes "C language" functions from "internal" functions, and the actual coding conventions are essentially the same for both. Hence, the standard internal function library is a rich source of coding examples for user-defined C functions. The dynamic loading feature involves 2 processes:

- *Dynamic loading* is what PostgreSQL does to an object file. The object file is copied into the running PostgreSQL server and the functions and variables within the file are made available to the functions within the PostgreSQL process. PostgreSQL does this using the dynamic loading mechanism provided by the operating system. The syntax that runs in Linux platform is to produce an object file called *sample.o* that can then be dynamically loaded into PostgreSQL.

```
gcc -fpic -c sample.c
```

- *Loading and link editing* is what user does to an object file in order to produce another kind of object file (e.g., an executable program or a shared library). User performs this using the link-editing program. This share library will be registered within PostgreSQL environment.

```
gcc -shared -o sample.so sample.o
```

The methodology of creating user-defined function could be represented as Figure 5.1.

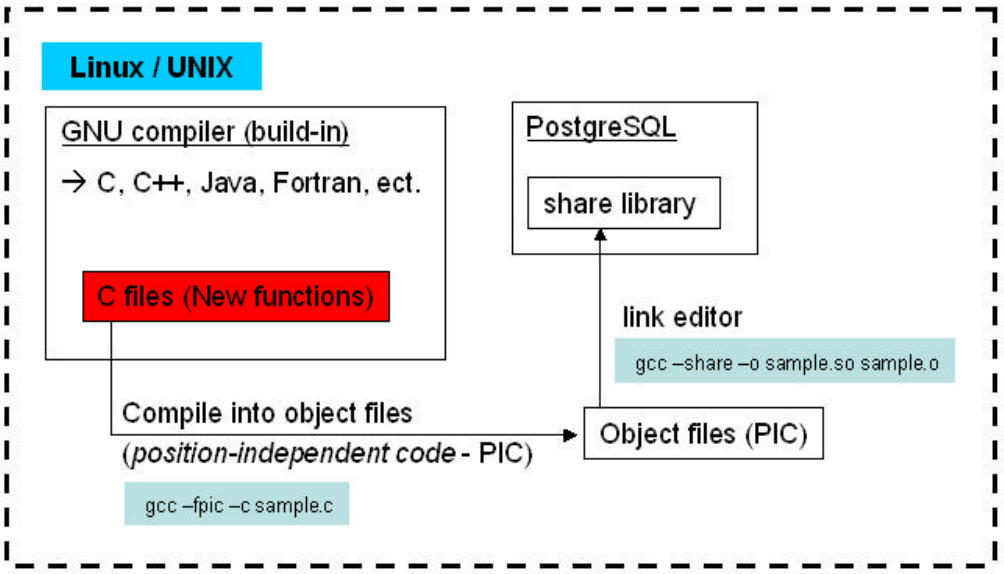



Figure 5.1: The methodology of creating new function in DBMS environment

## 5.3 Querying 3D Topological Data Structure

The spatial query for the 3D topological data structure could be done using the SQL within geo-DBMS environment. The following query shows the results of "all faces in a building" (see Figure 5.2).

```
psql>  SELECT  face_ID  FROM  Polyhedron_Table,  where
psql>  Polyhedron_ID = 121;


Polyhedron_ID    face_ID
-------------  ---------
          121         12
          121         16
          121         24
          121         56
          121         57
          121         58
          121         67
          121         78
          121         79
          121         82
          121         83
          121         85
          121         86
```

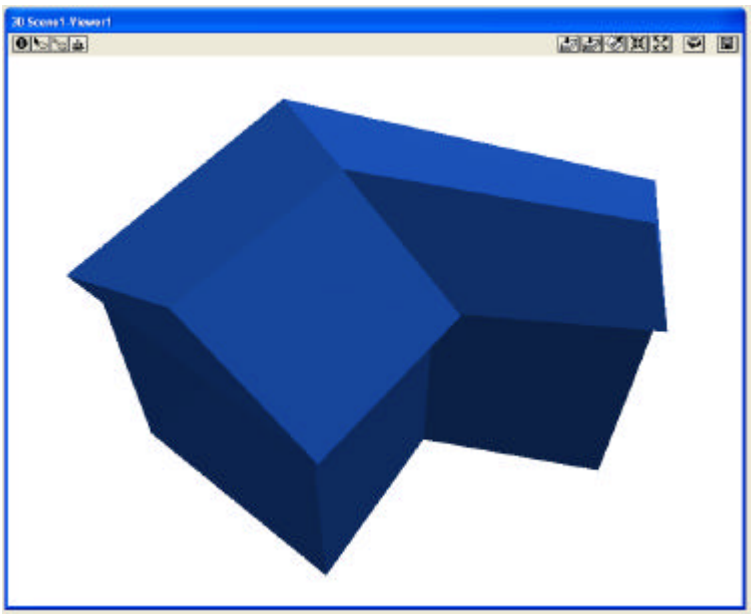


Figure 5.2: Sample of polyhedron

## 5.4 Validation of The 3D Spatial Object

It is important that the spatial data is checked when it is inserted in the DBMS. This check on the geometry of the spatial objects is called validation. Valid objects are necessary to make sure the objects can be manipulated in a correct way. For this research, the validation function is based on the Euler Poincaré theorem. The Euler-Poincaré formula describes the relationship of the number of vertices, the number of edges and the number of faces of a manifold. It has been generalized to include potholes and holes that penetrate the solid. To state the Euler-Poincaré formula, we need the following definitions:

V: the number of vertices
E: the number of edges
F: the number of faces
G: the number of holes that penetrate the solid, usually referred to as genus in topology
S: the number of shells. A shell is an internal void of a solid. A shell is bounded by a 2-manifold surface, which can have its own genus value. Note that the solid itself is counted as a shell. Therefore, the value for S is at least 1.
L: the number of loops, all outer and inner loops of faces are counted.

Then, A polyhedron is valid when it follows the Euler-Poincaré formula given as below:

$$V - E + F - (L - F) - 2(S - G) = 0$$

The following SQL statement runs the validation function on column geom.

```
psql> SELECT validate_polyhedron(geom,0.05) VALID FROM
psql> Polyhedron_Table, where Polyhedron_ID = 121;
```

The result:

```
VALID
-------------------------------------------------
Storage is valid
```

## 5.5 Summary and Conclusion

In this chapter, the implementation of topological data structure and validation function based on Euler Poincaré theorem was presented. However, there are many issues that need to be given attention in order to improve the current situation of 3D spatial modelling. These issues are in-line to the requirements of 3D spatial modelling. Many kinds of 3D modeling are basically based on their applications. The use of 3D modeling, e.g. simulation of 3D city, has become a common application for both the military and private sector. Detailed 3D city models are one of the popular mapping products that have seen widespread use in recent years. Besides, the spatial operations for the 3D city model also become important for urban planning, telecommunication, environment analysis, transportation, risk management and visualization of proposed development. The future challenges for 3D GIS modeling also involves the interoperability between different applications, data model, the integration between DBMS and visualization, and the linkage between data model and data acquisition.

# CHAPTER VI

# CONCLUSIONS AND RECOMMENDATIONS

## 6.1 Conclusions

Research and development within the scope of 3D-GIS is now extensive. This research can only deal with some parts of it. The emphasis here is on the development of 3D spatial data modelling for topological data structure. Some methods utilizing the 3D spatial modeling with respect to the development introduced in this report are also given. Several problems associated with 3D GIS were identified in chapter 1. The scope of this thesis, however, restricts the emphasis to various stages of the development of new 3D data type, i.e. polyhedron.

The review of the current situation of 3D modeling indicates that existing researches do not provide adequate, complete and optimized 3D modeling tools and operations for earth science applications needing to model the spatial objects. Moreover, different spatial modelings deal with different applications. The review of the existing DBMS was given in chapter 2, whereas the chapter 3 deals with the OGC standards for geospatial modelling. These reviews reflect the incompleteness, probably weaknesses for 3D GIS in certain 3D applications. Thus, the study commences with a review of all the existing of geo-spatial modeling that deals with

3D object and spatial operations within DBMS. Although different theories and approaches abound, only those supporting the design of a 3D modeling are reviewed. By relating and bringing some order into those theories and approaches, the study also contributes to the further development of spatial data modelling that implement the set and Euler theories.

With respect to the spatial and topological model, some reviews of fundamental issue related to the set and Euler theories were discussed in chapter 4. These topological models deal with spatial object construction and relationship among primitives. In order to implement the topological model, the research attempted to extend to the third dimension. A new 3D data type, also polyhedron, was created for this topological model. The topological data structure is applicable within DBMS environment.

## 6.2 Recommendations

The 3D spatial data modeling implements the set and euler theories 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.

New data types can also be implemented in other DBMS, e.g. Oracle Spatial, similar to the work by Pu (2005) that had been done using free-from objects. The reason of using PostgreSQL in this research is that PostgreSQL follows the specifications of Open GIS Consortium (OGC, 2006). The most important for user is the commercial issues, which it is an open source technology and suitable for

educational purposes. Some of the comparisons among DBMSs could be found at Konrad et al. (2006).

The test data set consists currently of very simple objects as it can be seen from the example, but some more experiments with real data sets are planned and can be implemented in future. It is also interesting to compare the implementation with the shortly coming implementation of Oracle Spatial 11.

Future research will concentrate a very important issue. That 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. 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, e.g. using Web Feature Service (WFS).

## LIST OF REFERENCES

Abel, D. J., Ooi, B. C., Tan, K. L., Power, R., and Yu, J.X. (1995). Spatial Join Strategies in Distributed Spatial DBMS. Proc. 4th Intl. Symposium on Large Spatial Databases. 348-367.

Abdul-Rahman, A., Drummond, J. and Shearer, J. (1998). Representation of 3D Spatial Objects. *Proceedings of American Congress on Surveying and Mapping (ACSM) Conference and Exhibition*. Baltimore, USA.

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

Aronoff, S. (1989) Geographic Information System: A Management Perspective. WDL Publications, Ottawa, Canada.

Basu, S. (1999). On Bounding the Betti Numbers and Computing the Euler Characteristics of Semi-algebraic Sets, Discrete and Computational Geometry. Vol. 22 pp. 1-18.

Basu, S., Pollack, R., Roy, M.F. (2003). Algorithms in Real Algebraic Geometry, Springer-Verlag.

Berry, J. K. (1993). *Beyond Mapping: Concepts, Algorithms, and Issues in GIS*. Fort Collins, Colorado: GIS World.

Bruenig, M. and Zlatanova, (2004). 3D Geo-DBMS. S. http://www.directionsmag.com/article.php?article_id=694.

Clarke, K. C. (1986). Advances in geographic information systems. Computers, Environment and Urban Systems, 10(3/4):175-186.

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

Corbett, J. P. (1979). Topological Principles in Cartography. Technical Report 48, US Bureau of the Census, Washington DC.

Egenhofer, M. (1992). Why not SQL!. *International Journal of Geographical Information Systems*, 6(2):71-85.

Egenhofer, M. (1996). Spatial-Query-by-Sketch. In: Burnett, M. and Citrin, W. ed. *VL'96: IEEE Symposium on Visual Languages*. Boulder, CO, IEEE Press. 60-67.

Egenhofer, M., Frank, A., Jackson, J. (1989). A Topological Data Model for Spatial Databases. In: Proceedings of First Symposium SSD'89. pp 271–286.

Goodchild, M., and Gopal, S. (1989). *Accuracy of Spatial Databases*. Taylor & Francis.

Guttman, A. (1984). R-trees: A Dynamic Index Structure for Spatial Searching. In: Proceedings of ACM SIGMOD International Conference on Management of Data. 47-57.

Güting, R. H. (1994a). An Introduction to Spatial Database Systems. VLDB Journal 3. 357-399.

Güting, R. H. (1994b). GraphDB: A Data Model and Query Language for Graphs in Databases. Technical Report Informatik Berichte 155-2/1994, Fern Universitat, Hagen.

Güting, R. H. (1989). Gral: An Extensible Relational Database System for Geometric Applications. 15th Int. Conf. on Very Large Data Bases. 33-44.

Guting, R. H., and Schneider, M. (1993). Realms: A Foundation for Spatial Data Types in Database Systems. Symposium on Large Spatial Databases. 14-35.

Hadzilacos, T. and Tryfona, N. (1996). Logical Data Modelling for Geographic Applications. International Journal of Geographical Information Systems. 10:179-200.

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

ITTIA. (2005). db.* Documentation, ITTIA Embedded Database Development, URL: http://www.ittia.com/dbstar/manual/UG_Gloss.htm.

Lipeck, U. and Neumann, K. (1987). Modelling and Manipulating Objects in Geoscientific Databases. 5th Int. Conf. on the Entity-Relationship Approach. 67-86.

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

Manola, F. and Orenstein, J. A. (1986). Toward a General Spatial Data Model for an Object-Oriented DBMS. 12th Int. Conf. on Very Large Data Bases. 328-335.

Molenaar, M. (1990). A Formal Data Structure for Three-dimensional Vector Maps. In: Proceedings of the Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland.

Mounsey, H., and Tomlinson, R. F. (1988). *Building Databases for Global Science*. Taylor and Francis, New York.

MySQL 6.0 Reference Manual. (2007).
URL: http://dev.mysql.com/doc/refman/6.0/en/index.html

Oracle Spatial Documentation (2007).
URL: http://www.oracle.com/technology/documentation/spatial.html

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

Paredaens, J., and Kuijpers, B. (1998). Data Models and Query Languages for Spatial Databases. Data & Knowledge Engineering. 25(1-2):29-53.

Patel, J., Yu, J.B., Kabra, N., Tufte, K., Nag, B., Burger, J., Hall, N., Ramasamy, K., Lueder, R., Ellmann, C., Kupsch, J., Guo, S., Larson, J., DeWitt, D., Naughton, J. (1997). Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. Proceedings of ACM SIGMOD Conference. pp. 336—347.

Peuquet, D. J. (1984). A Conceptual Framework and Comparison of Spatial Data Models. Cartographica. 21:66- 113.

Pigot, S. (1992). A Topological Model for a 3D Spatial Information System. In: Proceedings of the 5th International Symposium on Spatial Data Handling. Pp. 344–360.

Pigot, S. (1995). A topological model for a 3-dimensional Spatial Information System. PhD thesis, University of Tasmania, Australia.

PostGIS. (2007). PostGIS Manual, Refractions Research.
URL: http://postgis.refractions.net/docs/

PostgreSQL Documentation. (2006). URL: http://www.postgresql.org/docs/

Pilouk, M. (1996). *Integrated Modelling For 3D GIS*. ITC Enschede, Netherlands: PhD thesis.

Pu, S. (2005). *Managing Freeform Curves and Surfaces In a Spatial DBMS*. TU Delft: Msc thesis.

Raper, J. F., and Maguire, D. J. (1992) Design models and functionality in GIS. Computers & Geosciences. 18:387-394.

Rigaux, P., Scholl, M., and Voisard A. (2002). Spatial Databases – With Application To GIS. Morgan Kaufmann publishers, San Francisco, 410 p.

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.

Shunji M., (1999) GIS Work Book. National Space Development Agency of Japan (NASDA), Asian Institute of Technology (AIT).

Smith, D.R., and Paradis, A.R. (1989). Three-Dimensional Display of Geologic Data. In: Raper, J.F. ed. *Three-Dimensional Applications In Geographical Information System*. Taylor and Francis: London. 150 p.

Smith, T. R., Menon, S., Star, J. L., and Estes, J. E. (1987). Requirements and Principles for the Implementation and Construction of Large-Scale Geographic Information Systems. International Journal of Geographical Information Systems. 1(1):13-31.

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.

Taylor, G. (2000). *The Third Dimension (And Beyond?)*. http://www.comp.glam.ac.uk/pages/staff/getaylor/papers/sw3d.PDF

Thompson, D., and Laurini, R. (1992). Fundamentals of Spatial Information Systems. Number 37 in APIC Series. Academic Press.

Tsichritzis, D. C., and Lochovsky, F. H. (1977). Data Base Management Systems. Academic Press, New York.

Van Roessel, J. W. (1987). Design of a Spatial Data Structure Using the Relational Normal Form. International Journal of Geographical Information Systems. 1:33-55.

Zlatanova, S. (2000). *3D GIS for Urban Development*. ITC, The Netherlands: PhD Thesis.

Zlatanova, S., Abdul-Rahman, A., and Shi, W. Z. (2004). Topological Models and Frameworks for 3D Spatial Objects. *Computers & Geosciences*. 30(4): 419-428.