Data Structuring for 3D Spatial Objects

Harith Fadzilah Abd Khalid and **Alias Abdul Rahman**

Departments of Geoinformatics,
Faculty of Geoinformation Science and Engineering,
Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia.
E-mail: {harith, alias}@fksg.utm.my.

Abstract

This paper describes one aspect of 3D GIS research, that is the data structuring for 3D objects and incorporate with geo-DBMS. Geo-database can manage large spatial data sets and able to provide access to multi users. Lately there are growing tendency towards the multifunctional use of space i.e. above and below the ground surface. Applications such as urban planning, telecommunication and 3D cadastre depend on 3D data. Thus, the need for managing the 3D spatial datasets especially for large area is inevitable. A discussion on how to establish these data structures for managing 3D spatial objects of large areas form major discussion of this paper. 3D primitives like points, lines, surfaces (or polygons) and solids are part of the primitives. The mapping of this structure in Oracle Spatial DBMS will be highlighted as well. Our presentation also will highlight the future outlook of the data structuring approach for managing large volume of 3D spatial objects and relates with current research development in 3D GIS.

Keywords: data structures, 3D spatial objects, geo-DBMS, 3D GIS

1.0 Introduction

The surrounding that we live in actually consists of abundance of 3D spatial objects. Lately there is a demand in applications such as urban planning (Cambry, 1993), telecommunications and cadastres (Stoter, 2000) which depend on 3D data. These applications made complicated because of the tendency towards the multifunctional use of space such as buildings above roads and railways and bridges (Stoter, 2000). Presently the way we manage and administer these 3D spatial objects is through Data Base Management Systems (DBMSs) or specifically Geo-DBMS where these 3D spatial objects are treated as 2D spatial objects. Although the present Geo-DBMS do not support 3D primitive, 3D spatial objects can be modeled with 2D primitive polygons. With several 2D polygons bound a 3D object, it can be stored in one record (multi-polygons) or multiple records (Stoter and Zlatanova, 2003). This is achievable because Geo-DBMSs support 3D coordinates.

This paper will discuss part of 3D GIS where a practical approach to model a 3D object using the available functions provided by the Geo-DBMS. The first section described the reality that our surrounding objects should be better administered in a realistic approach of data management system. Then in second section, we describe what is geo-database. The third section illustrates the complex data type, SDO_GEOMETRY for storing spatial object in Oracle Spatial. In fourth section, we describe our experience with a simple 3D objects structured as polyhedron implemented in Oracle Spatial based on an

approach done by Stoter et al (2002) and Arens et al (2005). Finally we end with concluding remark and future work.

2.0 Geo-database

Basically DBMS was design for non-spatial data. Until the last several years, developments of DBMS were significant to cater the spatial data. According to Bruenig et.al (2004), there are two approaches of development in DBMS linking to spatial data. The first is the 'top-down' approach. In this approach, the DBMS functionality has been constructed 'under' the GIS application and the GIS application accesses 'top-down' to the geodata stored in the DBMS. ESRI is one of the GIS vendors that initially implementing this approach to store both non-spatial and spatial data.

The other development of DBMS is the 'bottom-up' approach. In this approach, the DBMS supports the spatial data type; meaning that it extends 'low level' DBMS data type and indexes to use them in the upper level of GIS applications. Data analysis on the spatial and non-spatial parts of objects can be executed. Lately more and more commercial DBMSs provide spatial extensions to support spatial objects.

Mainstream DBMS such as Oracle (Oracle, 2001), IBM DB2 (IBM, 2000), Informix (Informix, 2000) and Ingres (Ingres, 1994) have implemented spatial data type and spatial functions or less similar to the OpenGIS Consortium (OGC, 1998) Simple Features Specification for SQL (OGC, 1999). According to the OpenGIS specifications, the spatial object is represented by two structures, i.e. *geometrical (i.e. simple feature specifications)* and *topological (i.e. complex feature specifications).* The geometric structure provides direct access to the coordinates of individual objects, while the topological structure encapsulates information about their spatial relationships. Presently, geometrical model has been implemented in mainstream DBMSs.

In other word, a full-fledged DBMS which has capabilities for handling spatial data is also called Geo-database or Geo-DBMS (Gunting, 1994). A Geo-DBMS knows primitive and composed geometric data types i.e. point, line and polygon, in the same way as its primitive standard data types such as character, string, integer, real etc. Presently the implementations of spatial data type in mainstream DBMSs are basically 2D and the spatial features are stored in a geometrical model without the internal topology. Topological relationships between geometries can be retrieved by the use of spatial operators.

3.0 Geometrical Model In Oracle Spatial

Oracle, IBM DB2, Informix and PostGIS support geometric functions defined by OGC (Oosterom et al., 2002). OGC specifications (OGC, 2001) are until now 2D and the implementations of spatial data types in mainstream DBMSs stated above are actually in 2D, but some of these DBMSs capable of supporting 3D coordinates in their spatial data type. The standard SQL statement "*select attribute_a from table_b where a<100*" is the same in every DBMS, however, if geometries are concerned, different types of queries have to be executed in different DBMSs. For example, Oracle Spatial does not have explicitly implemented data types such as point, line and polygon. There is only one complex data type, SDO_GEOMETRY, composed of several parameters including type geometry, dimension, and an array with the x,y,z coordinates.

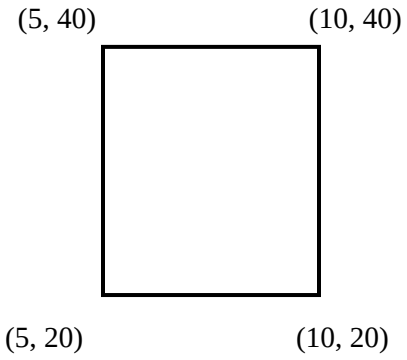
In this investigation, we have tested the Oracle Spatial to store 3D spatial object. Before we move to 3D object, lets have a look on how the Oracle Spatial provides the geometrical model of handling spatial features. The basic building block for representing an object in Oracle Spatial is an element. Points, lines

and polygons are the primitive elements which are stored by their x and y coordinates. To model a spatial feature, a single element or a collection of homogeneous or heterogeneous elements to is used to form a geometry object. Oracle used a variety of geometric types (point, linestring, polygon, multipoint, multilinestring, multipolygon). A geometry is stored as an object in a single row, in a column of type SDO_GEOMETRY. Oracle Spatial defines the object type SDO_GEOMETRY as:

```
CREATE TYPE sdo_geometry AS OBJECT  (
  SDO_GTYPE NUMBER,
  SDO_SRID NUMBER,
  SDO_POINT SDO_POINT_TYPE,
  SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
  SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRY);
```

SDO_GTYPE indicates the type of the geometry (point, linestring, polygon, multipoint, multilinestring, multipolygon). SDO_SRID is a reference to the spatial reference system used for the ordinates. SDO_ELEM_INFO is defined using a varying length array of numbers. These attributes show how to interpret the ordinates stored in the SDO_ORDINATES attribute. They include for every element the offset where the element starts in the array, type of element (point, linestring consisting of straight lines, linestring consisting of circular arcs, polygon) and an interpretation code. One geometry object composed of one or more elements.

Below is how a polygon is represented using the Oracle Spatial:




```
SDO_GEOMETRY Column = (
SDO_GTYPE = 2003
SDO_SRID = NULL
SDO_POINT = NULL
SDO_ELEM_INFO = (1,1003,1)
SDO_ORDINATES = (5,20, 10,20, 10,40, 5,40, 5,20))
```

In SDO_GTYPE =2003, the *2* indicates two-dimensional and the *3* indicates a polygon. In SDO_ELEM_INFO, the final *1* in 1,1003,1 indicates that this is a polygon containing straight lines. Also

the first position (‘1’) indicates that the first element (in this case only one polygon) starts at offset 1 in the coordinate list.

To elaborate further, let see how a box is stored as a geometry in Oracle (SDO_GEOMETRY type) in the ‘geom2d’ table and progress further the same box with height 50 stored in the ‘geom3d’ through SQL statements. These tables are representing the geometries of the object.

```
/* creation of the tables */
create table geom2d (
 shape mdsys.sdo_geometry not null,
 ID number (11) not null);
create table geom3d (
 shape mdsys.sdo_geometry not null,
 ID number (11) not null);
```

```
/* inserting the data */
/* a 2D box */
insert into geom2d (shape, ID) values (
  mdsys.SDO_GEOMETRY (2003, NULL, NULL, mdsys.SDO_ELEM_INFO_ARRAY(1, 1003, 1),
  mdsys.SDO_ORDINATE_ARRAY(0,0, 100,0, 100,100, 0,100, 0,0)), 8);

/* a 3D box */
insert into geom3d (shape, ID) values (
  mdsys.SDO_GEOMETRY (3003, NULL, NULL, mdsys.SDO_ELEM_INFO_ARRAY(1, 1003, 1),
  mdsys.SDO_ORDINATE_ARRAY(0,0,50, 100,0,50, 100,100,50, 0,100,50, 0,0,50)), 9);
```

The metadata table for the geometries is maintained in Oracle which describes the dimension, lower and upper bounds and tolerance in each dimension. The following shows how the information on the tables, geom2d and geom3d inserted in the metadata table. And finally a spatial index (r-tree) is created on the tables (to speed up spatial queries)

```
/* inserting metadata */
/* 2D table */
insert into user_sdo_geom_metadata values
 (‘GEOM2D’, ‘SHAPE’, mdsys.SDO_DIM_ARRAY (
  mdsys.SDO_DIM_ELEMENT (‘X’, 0, 500, 0.5),
  mdsys.SDO_DIM_ELEMENT (‘Y’, 0, 500, 0.5)), NULL);

/* 3D table */
insert into user_sdo_geom_metadata values
 (‘GEOM3D’, ‘SHAPE’, mdsys.SDO_DIM_ARRAY (
  mdsys.SDO_DIM_ELEMENT (‘X’, 0, 500, 0.5),
  mdsys.SDO_DIM_ELEMENT (‘Y’, 0, 500, 0.5),
  mdsys.SDO_DIM_ELEMENT (‘Z’, 0, 300, 0.5)), NULL);
```

```
/* creating index */
/* 2D table */
```

```
create index GEOM2D_I on GEOM2D(SHAPE)
 indextype is mdsys.spatial_index;
analyze table GEOM2D compute statistics;

 /* 3D table */
create index GEOM3D_I on GEOM3D(SHAPE)
 indextype is mdsys.spatial_index parameters('sdo_indx_dims=3');
analyze table GEOM3D compute statistics;
```

4.0 3D Objects in Oracle Spatial

The example shown in section 3 shows that 3D objects can be stored in mainstream Geo-DBMS even tough the geometry types are 2D. There are two ways of storing 3D objects in Geo-DBMS: using the existing data types or creating user-define spatial types. In this paper we will discuss on the first approach. In this approach the 3D objects are represented as a set polygons (with 3D coordinates) composing the object. This approach also has the advantage of being 'understandable' for all the front-ends (GIS/CAD/AEC) since it is supported by the DBMS. In Oracle Spatial, 3D object can be represented as **polyhedron** (body with flat faces) which is 3D polygon stored as a list of polygons or as multipolygon (one polygons consisting of several polygons). Figure 1.0 is the UML class diagram that shows the data structure of polyhedron data model.

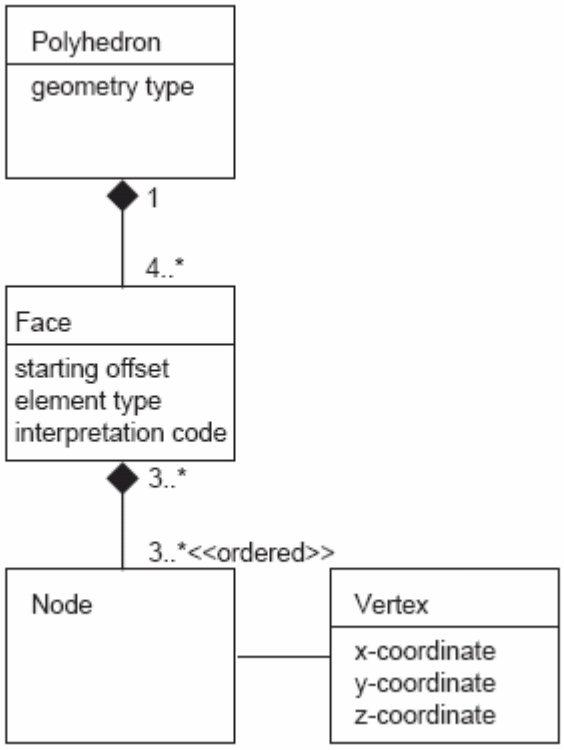


Figure 1.0 Polyhedron Data Structure (Arens,C et.al. 2005)

To show the above data structure be implemented in Oracle Spatial, lets take an example of a cube (Figure 2.0).

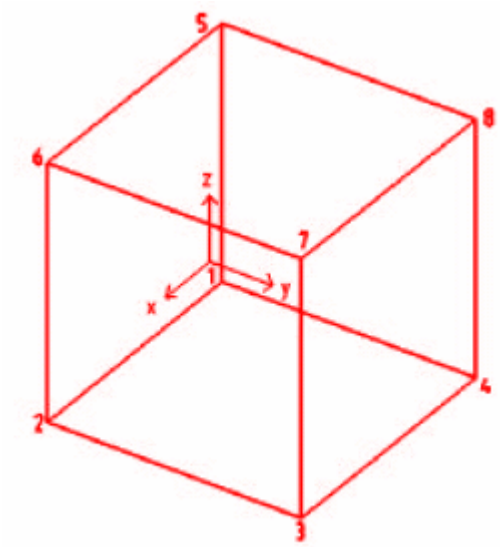


Figure 2.0 Cube to be stored in the Oracle Spatial (Stoter, J. et. al. 2002)

In the first option i.e. defining a 3D objects as a list of polygons, two tables are used: a table 'BODY' and a table 'FACE'. In the table 'BODY' the 3D spatial object is defined by set of records representing a polyhedron with references to the (flat) faces it consists of. In the table 'FACE' the actual geometries of faces are stored as 3D polygons (SDO_GTYPE: 3003, SDO_ELEM_INFO: (1,1003,1)). This model is partly a topological model, since the body is defined by references to the faces and the faces can be by neighbor-bodies. The generated tables for the cube look as follows:

Table 1.0 The BODY and FACE table for a cube is stored as a set of polygons (Stoter, J. et.al.2003)

BODY table	
BID	*FID*
1	1
1	2
1	3
1	4
1	5
1	6

FACE table	
FID	*sdo_ordinate_array*
1 (lower face)	(x4,y4,z4, ,x3,y3,z3, x2,y2,z2, x1,y1,z1, x4,y4,z4)
2 (side 1)	(x3,y3,z3, ,x4,y4,z4, x8,y8,z8, x7,y7,z7, x3,y3,z3)
3 (side 2)	(x4,y4,z4, ,x1,y1,z1, x5,y5,z5, x8,y8,z8, x4,y4,z4)
4 (side 3)	(x1,y1,z1, ,x2,y2,z2, x6,y6,z6, z5,y5,z5, x1,y1,z1)
5 (side 4)	(x3,y3,z3, ,x2,y2,z2, x6,y6,z6, z7,y7,z7, x3,y3,z3)
6 (upper face)	(x5,y5,z5, ,x6,y6,z6, x7,y7,z7, z8,y8,z8, x5,y5,z5)

In the second representation a body is stored as one record instead of a set of records. The multipolygon, which is also supported in Oracle Spatial, is used for this representation. The resulting table 'BODY' in which the cube of the example is stored looks as follows:

Table 2.0 BODY table for a cube is stored as one multipolygons (Stoter, J. et.al.2003)

BODY table	
Body id	Geometry
1	SDO_GEOMETRY(3007, -- 3007 indicates a 3D multipolygon NULL, NULL, SDO_ELEM_INFO_ARRAY(-- the offset of the polygons is specified 1, 1003, 1, 16, 1003, 1, 31, 1003, 1, 46, 1003, 1, 61, 1003, 1, 76, 1003, 1), SDO_ORDINATE_ARRAY(x4,y4,z4, ,x3,y3,z3, x2,y2,z2, x1,y1,z1, x4,y4,z4, --end of 1st (lower) polygon x3,y3,z3, ,x4,y4,z4, x8,y8,z8, x7,y7,z7, x3,y3,z3, x4,y4,z4, ,x1,y1,z1, x5,y5,z5, x8,y8,z8, x4,y4,z4, x1,y1,z1, ,x2,y2,z2, x6,y6,z6, z5,y5,z5, x1,y1,z1, x3,y3,z3, ,x2,y2,z2, x6,y6,z6, z7,y7,z7, x3,y3,z3, x5,y5,z5, ,x6,y6,z6, x7,y7,z7, z8,y8,z8, x5,y5,z5 --end of last (upper) polygon))

As we can see, the advantage of 3D multipolygons is that it is recognized as one object by front-end applications (GIS/CAD) that can access, visualize, and edit these data and post the changes back to the

database. Another advantage of this option is the one-to-one correspondence between a record and an object.

5.0 Concluding Remarks

Nowadays, GIS is changing and becoming an integration of database management, powerful editing and realism visualization inherited from the advanced computing development. At present, 3D GIS mostly focus on the geometrical rather that the topological aspect. The above geometric approach of using set of polygons as polyhedron for the data structure is encouraging us to model a large volume of 3D spatial objects and managing them in a proper manner in a Geo-DBMS. At present, this could be done by the help of photogrammetry method which capable of capturing 3D data set in large area. Even though 3D spatial object modeled by the method stated above and manageable in Geo-DBMS, it still lack of real true 3D data type (3D volumetric) which can function in a proper 3D environment function such as validation, volume, length, intersection, overlap, etc (Alias et. al, 2002).

Our future works will be on how to use these data structure and manage the 3D spatial objects in a geo-DBMS. This will definitely relates to designing the geo-DBMS, spatial indexing and also the visualization with front-end application software.

Reference

Rahman, A.A. , Zlatanova, S., Pilouk M., (2002). Trends in 3D GIS Development. In Journal of Geospatial Engineering, Vol. 2, No.2 pp. 1-10

Arens, C., J. Stoter, P.v. Oosterom (2005). Modelling 3D spatial objects in a geo-DBMS using a 3D primitive. In Computer & Geosciences 31 (2005) pp. 165-177.

Bruenig, M and Zlatanova, S. (2004). 3D-GeoDBMS, Directions Magazine.

Cambray, B., (1993). Three-dimensional modeling in a geographical database. In: Proceedings 11th International Conference on Computer Assisted Cartography, MN, USA, pp. 338-347.

Güting, R.H, (1994). An Introduction to Spatial Database Systems. VLDB Journal Vol. 3 no. 4, pp357-399

IBM (2000). IBM DB2 Spatial Extender User's Guide and Reference. Special web relese edition.

Informix (2000). Informix Spatial DataBlade Module User's Guide, Part no. 000-6868.

Ingres (1994). INGRES/Object Management Extension User's Guide, Release 6.5 (1994). CA-OpenIngres.

OGC (1998). The OpenGIS Guide, Third edition. An introduction to Interoperable Geo-processing. The OGC Project Technical Committee of OpenGIS Consortium, edited by Buhler and K. McKee, L., Wayland, Mass., USA

OGC (1999). OpenGIS Simple Features Specification for SQL. Revision 1.1, OpenGIS Project Document 99-049

OGC (2001). OpenGIS Specification, 2001, available on http://www.opengis.org/techno/spec.htm.

Oosterom, P. v, J. Stoter, W. Quak and S. Zlatanova, (2002). The balance between geometry and topology, in: Advances in Spatial data Handling, 10th International Symposium on Spatial Data Handling, D. Richardson and P. van Oosterum (Eds.), Springer-Verlag, Berlin, pp. 209-224

Oracle (2001). Oracle Spatial User's Guide and Reference Release 9.0.1 Part Number A88805-01, June 2001

Stoter, J.E., (2000). Needs, possibilities and constraints to develop a 3D cadastral registration system. In: Proceedings 22nd Urban Data Management Symposium 'Orban and Rural data Management Common Problems-Common Solutions', vol. III, Delft, The Netherlands, pp. 43-58.

Stoter, J.E and Zlatanova (2002). 3D Large Scale Modelling, Workshop on 3D cadastre and Large Scale Modelling, organized during UDMS 2002, October 2002, Prague, Tsjech Republic

Stoter, J. E, Zlatanova, S., (2003). Visualising and editing of 3D objects organized in a DBMS. In: Proceedings EUROSDR Workshop: Rendering and Visualisation, January 2003, Enschede, the Netherlands, 14pp, CD-ROM.