DESIGN AND IMPLEMENTATION OF POLYHEDRON AS A PRIMITIVE TO REPRESENT 3D SPATIAL OBJECT

[1]Mr.N.Subhash Chandra, [2]Dr. A.Govardhan
[1]Research Schalor,JNT University, Hydearbad,A-P,INDIA
[2]Professor, JNT University, Hydearbad,A-P,INDIA
[1]subhashchandra_n@yahoo.co.in, [2]govardhan_cse@yahoo.co.in

ABSTRACT

Spatial objects of our world have an intrinsic three-dimensional (3D) nature; 3D data modeling and 3D data management have been neglected in spatial database systems and Geographical Information Systems, which map geometric data mainly to two-dimensional abstractions. But increasingly the third dimension becomes more and more relevant for application domain like pollution control, water supply, soil engineering, urban planning and aviation.
Spatial-DBMSs were chosen to store spatial data, because they could guarantee the safety of the data (in 2D). But with the arrival of applications depending upon correct 3D data, the present techniques do not suffice. The solution for this problem is to implement a real 3D primitive, including functions that e.g. return the volume or the distance between objects in 3D and validation functions. This improves the maintainability of 3D Spatial-datasets .3D Spatial objects are stored with the polyhedron as (3D) primitive. The proposed primitive is easy for users to model objects, can easily validated, because the algorithms are not too difficult to implement and still result in realistic objects.

Keywords *3D spatial data, GIS*

1 INTRODUCTION

Spatial-DBMSs make it possible to manage spatial datasets in databases that can be accessed by multiple users at the same time. These spatial datasets usually contain 2D data, while more and more applications depend on 3D data. Some examples are 3D cadastres, telecommunications and town planning. These applications mainly come from the ever-growing tendency of using living space multifunctional by building in the vertical direction, e.g. apartments, buildings over spanning a road, tunnels and bridges.

2D Spatial data can be modeled in the Spatial-DBMS with 2D primitives. However, the present Spatial-DBMSs do not support 3D primitives, but 3D spatial objects can be modeled by using 2D primitives such as polygons in 3D space[2]. This is possible by using 3D coordinates, which are supported by the Spatial-DBMSs. In this way, several 2D polygons bound a 3D object. These 2D polygons can be stored in one (multi-polygon) or multiple records. The absence of a real 3D primitive in the Spatial-DBMSs however, results into two problems:

1) The Spatial-DBMSs do not recognize 3D objects, because they do not have a 3D primitive to model these objects.

2) This results into DBMS functions not working properly (e.g. there is no validation for the 3D object as a whole and functions only work with the projection of these objects, because the third dimension is ignored).
3) In some cases the 2D objects, that bound a 3D object, are stored in multiple records; a better administration of these large datasets requires a 1:1 relationship between objects in reality and objects in the database, because then there is a clear connection between the object in the database and the object in reality.

Spatial-DBMSs were chosen to store spatial data, because they could guarantee the safety of the data (in 2D). But with the arrival of applications depending upon correct 3D data, the present techniques do not suffice. The solution for this problem is to implement a real 3D primitive, including validation functions and functions that e.g. return the volume or the distance between objects in 3D.Many concepts have been developed in the area of 3D

modeling. The innovation of this project is that the developed concepts have been translated into prototype implementations of a true 3D primitive in a DBMS environment.

Furthermore, it is important that the 3D data in the database can be visualized. Project emphasis on choice of a 3D primitive to model the 3D objects and describes its Implementation, the implementation of the most commonly used functions (e.g. area, volume, point-in-polyhedron and bounding box) and 3D also methods to visualize 3D spatial objects that are stored as 3D primitives.

1.1 3D Primitive

Now, Spatial-DBMSs are able to store, validate and query spatial data in 2D coordinate space. 2D spatial objects are stored as 2D primitives (polygons)[3][8]. To store 3D spatial objects, without the problems mentioned in the introduction, a 3D primitive is necessary.

1.2 Definition of a polyhedron

A polyhedron is the 3D equivalent of a polygon (in 2D space) and can be defined as a bounded subset of 3D coordinate space enclosed by a finite set of flat polygons (called faces) such that every edge of a polygon is shared by exactly one other polygon . Note that the polyhedron should bound a single volume, i.e. from every point (can be on boundary), every other point (can be on boundary) can be reached via the interior. The characteristics of the polyhedron primitive are[4][5] :

Flatness: The polygons that make up the polyhedron have to be flat. This means that all points that make up the polygon must be in the same plane. For three points this is always true, but for more than three points this is not always true , because of the geodetic measuring and processing methods and the finite representation of coordinates in a digital computer. Furthermore, inner rings (hole in polygon) of a face have to be in the same plane as the outer ring that it belongs to

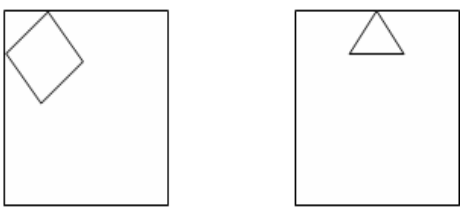

Fig 1 Left: Invalid polyhedron, because the hole divides the polyhedron in two volumes. (Note that this object is stored as one volume minus a hole (2 connected inner rings) and not as two separate volumes)Right: Valid polygon with inner ring that touches boundary.

2-Manifold: This characteristic looks at a polyhedron as a whole; it should bound only one volume. This means that from every point on the boundary, you should be able to reach every other point on the boundary via the interior (Fig 6). For the object to be valid, the faces where the hole starts and ends have to be modeled as a face with one or more inner rings. The edges and vertices should be 2-manifold. This means an edge is adjacent to exactly two faces and a vertex is the apex of only one cone of faces (i.e. two or more shells do not touch in one vertex)

Simplicity: This characteristic looks at the faces of a polyhedron. The polyhedron has to be composed of simple features. These are closed polygons that are not self-intersecting and have no inner rings. The faces of a polyhedron however, are allowed to have inner rings, as long as the faces together form a closed polyhedron. That is the reason this characteristic is called simplicity and not just simple. The inner rings of faces are not allowed to interact with the outer ring, except for touching boundaries. Furthermore, the vertices that span a face are not allowed to lie all on a straight line, i.e. the face has to have an area. A face has exactly one outer ring and zero or more inner rings. Finally, each edge has exactly 2 vertices. Only straight line segments are allowed, so there is no necessity for an edge to have more than 2 vertices. Note that two or more (but not all) edges are allowed to lie on a straight line, if this is more convenient for modeling an object

Orientable: There has to be a clear outside and inside of the polyhedron. In the field of computer graphics the normal vectors of faces point from inside to outside. This means that the vertices in a face must be specified in counter-clockwise order seen from the outside of the object. Note that the vertices in inner rings of faces need to be ordered in opposite direction (clockwise). All polyhedron need to fulfill these characteristics. The validation function is able to check if these characteristics are met.

2 POLYHEDRON MODEL

The polyhedron can be stored by storing the vertices explicitly (x,y,z) and describing the arrangement of these vertices in the faces of the polyhedron (Fig 8 shows an UML diagram). This yields a hierarchical boundary representation. Note that edges are not stored explicitly in this model. There are tags that describe if the face description is an outer or inner boundary (of a polyhedron) or an outer or inner ring (of a face). With these elements it is already possible to model complex objects, e.g. objects with through-holes or objects that are hollow inside. This set of elements is enough for the functions to understand what the 3D spatial objects look like.

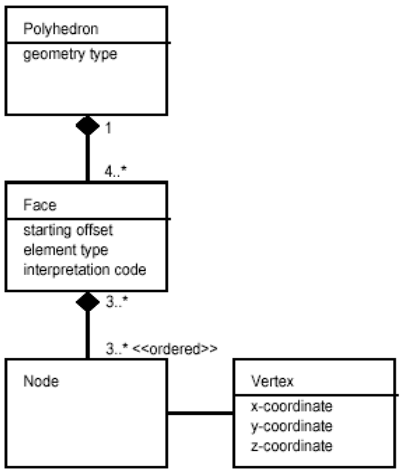

Fig 2 UML diagram of polyhedron storage.

The 3D primitive is implemented in a geometrical model with internal topology. This means that topology between objects is not maintained. Internal topology (topology within 3D objects) is maintained since the vertices for one object will be stored only once: faces are defined by internal references to the nodes and nodes are shared between faces.

There is a special geometry type in the object-relational model in Oracle Spatial 9i. This type is called sdo_geometry and is defined 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_ARRAY);

This type is stored in the MDSYS scheme. The meaning of the elements of sdo_geometry is :

- sdo_gtype: This indicates the type of geometry (point, linestring, polygon, multipoint, multiline string, multi polygon) and the dimension (0D, 1D, 2D, 3D) of its embedding space. Each geometry type has its own code, e.g. a 2D polygon has sdo_gtype = 2003. The first digit is the dimension and the last digit is the geometry type.

- sdo_srid: This is a reference to the spatial reference system used by the coordinates. In this research local (Cartesian-) coordinates are used, so no sdo_srid is specified (NULL). Non projected reference systems have to be converted to Cartesian coordinates first.

- sdo_point: This element is used when only points are stored as single object or when a point is stored in addition to the other geometry. The SDO_POINT_TYPE has an x-, y- and zelement.

- sdo_elem_info: This specifies the elements of the geometry with references to the coordinates (starting_offset), information about the element itself (e_type) and an interpretation code (e.g. straight line, rectangle, circle) on how to interpret the coordinates. This is stored in a variable array of numbers. A rectangular polygon specified by two coordinates is e.g. stored as sdo_elem_info_array = (1,1003,3).

- sdo_ordinates: This is a variable array of numbers and contains the coordinates.

To extend Oracle Spatial 9i with polyhedron geometry, a new set of codes is necessary. The proposal for these codes as described in [4] is the starting point for this research. The data model is a geometric model, defined with internal topology; the vertices are stored only once per polyhedron. each face of the polyhedron is described with a reference to the point number of the vertices.

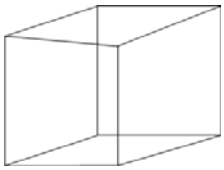

Fig 3 Cube with its Coordinates

The SQL to insert this geometry in Oracle Spatial 9i, as a polyhedron primitive by means of the proposed codes, is:

INSERT INTO table (id, geometry) VALUES (2, mdsys.sdo_geometry(3008, NULL, NULL,

mdsys.sdo_elem_info_array(

25,1006,1, 29,1006,1, 33,1006,1, 37,1006,1, 41,1006,1, 45,1006,1),-- 25 is the first face, the first 24 are used by the coordinates

mdsys.sdo_ordinate_array(

1,1,0, 1,3,0, 3,3,0, 3,1,0, 1,1,2, 1,3,2, 3,1,2, 3,3,2, –- the coordinates

1,2,3,4, -- bottom face starts at index 25

8,7,6,5, -- top face starts at index 29

1,4,8,5, -- front face starts at index 33

2,6,7,3, -- back face starts at index 37

1,5,6,2, -- left face starts at index 41

4,3,7,8 -- right face starts at index 45

)));

This means that elements of sdo_geometry are:

- sdo_gtype = 3008 (3D polyhedron)

- sdo_srid = NULL (no spatial reference system)

- sdo_point = NULL (no point type)

- sdo_elem_info = 6 times x,1006,1 (exterior polyhedron boundary, x is where the face starts)

- sdo_ordinates = 8 coordinate triplets and 6 face descriptions

3 IMPLEMENTATION

Oracle Spatial ignores all elements with sdo_gtype or e_type = 0. If the sdo_gtype = 0, the object is ignored by the spatial index. On the other hand, sdo_gtype = 3008 is not recognized and therefore it is also not possible to create a spatial index on that sdo_gtype. Therefore, an existing sdo_gtype = 3002 is chosen. This is a 3- dimensional polyline going through all the coordinates of the defined polyhedron. When creating a 3D spatial index (which is possible in Oracle), a bounding box is created around this line. This bounding box is equal to the bounding volume around the polyhedron. The drawback of using an existing sdo_gtype is that application will be confused, because there is no difference between a 3D polyline and a polyhedron.

In order to store the line, an entry in the sdo_elem_info is necessary. If the cube from §2.3 is taken (Fig 10), it will look like this:

INSERT INTO table (id, geometry) VALUES (2,

mdsys.sdo_geometry(3002, NULL, NULL, -- 3002 = 3D line

mdsys.sdo_elem_info_array(1,2,1, 25,0,1006, 29,0,1006, 33,0,1006, 37,0,1006,41,0,1006, 45,0,1006), -- first triplet is line, then the faces

mdsys.sdo_ordinate_array(1,1,0, etc., 1,2,3,4, etc.)));

This means that elements of sdo_geometry are:

- sdo_gtype = 3002 (3D line)

- sdo_srid = NULL (no spatial reference system)

- sdo_point = NULL (no point data)

- sdo_elem_info = 1,2,1 (straight line) – x,0,1006 (6 times a exterior polyhedron boundary, x is where the face starts)

- sdo_ordinates = (8 coordinate triplets and 6 face descriptions)

E_type = 0 is necessary in this implementation for Oracle Spatial to ignore this element. The interpretation code is free to choose and thus takes the role that the e_type had. This is why

the information about the element is moved to this position. The rest of the implementation is the same as described in the Table 1 shows an overview of the storage options.

Sdo_gtype		3002: 3D line tocreate index on
Sdo_elem_info	Starting offset	Points to the starting offset of a face in sdo_ cordinates
	E_type	Is always 0,to ensure proper working
	Interpretation code	1006 Outer ring of exterior polyhedron boundary(face) 1106 Inner ring of exterior polyhedron boundary(face) 2006 Outer ring of interior polyhedron boundary(face) 2106 Inner ring of interior polyhedron boundary(face)
Sdo_ordinates		The ordinate triplets that store vertices The face description that point to the ordinate triplets

Table 1 Overview of storage options in the implementation of 3D primitive.

4 3D FUNCTIONS

4.1 Functions used in the conversion

These functions are used when converting spatial data to the polyhedron type and back. There are a number of conversion functions implemented. These are the present possibilities to convert spatial data in other formats to the polyhedron type and vice versa: - It is possible to manually insert a polyhedron into a record of a spatial table in the database.

- The function used to convert multi-polygons (standard type in Geo-DBMS) that together form a polyhedron to the polyhedron type itself. This means that if spatial data is available in this format or if spatial data can be stored in this format; these data can be converted to a real 3D primitive. The vice versa function works exactly opposite and is especially useful to visualise the polyhedron. An advantage is that data can be inserted by GIS/CAD front-ends. Then there is a function that converts a body, face and node table (topology) to the polyhedron type.

Once the conversion to the polyhedron type has taken place, the user can decide to validate the polyhedron to see if they are correctly modeled. This is recommended, because all other DBMS 5 3D functions expect the polyhedron to be valid. The validation function related to the validation function is the function to correct the orientation of the faces of a polyhedron (fix orientation).

4.2 Functions that return a Boolean

These functions return a Boolean, i.e. true or false. A well-known Boolean function is the point in polyhedron function. This function determines whether a point is inside a polyhedron or not. - Generate a random unit vector. The point to test plus the direction of this vector form a random ray away from the point. The choice for a random vector is made, because if a fixed vector is chosen, there is a chance that the vector will intersect with the boundary of the polyhedron. This results in undesirable results in the function.

- Test for each plane if the ray intersects with it.

- If the number of intersections is even, then the point is outside the polyhedron, if this number is odd, then the point is inside the polyhedron.

A problem arises with this algorithm. If the ray hits the boundary of one or more of the faces, it is undetermined if the point is inside or outside the polyhedron. The solution is that if the ray hits a boundary of a face, then the algorithm is started over with a different random ray. If the ray intersects with the boundary again, then

another random ray is generated and so on until a good ray is found.

4.3 Unary functions that return a scalar

These are functions that work on a single polyhedron (unary) that return a number (scalar). Three of these kinds of functions are implemented:

4.3.1 Area

This function returns the true 3D surface area for a 3D polygon or the summation of the area of all faces that span a polyhedron. The area of a face is computed by projecting the face on 2D coordinate space. This projection takes place on the largest component of the normal vector n of the face. This evades numerical problems.

4.3.2 Volume

This function returns the volume of a polyhedron. The general idea of the algorithm that is used here to compute the volume is to multiply the area of each face by a depth, just like one would compute the volume of a box. With a polyhedron this results in computing overlapping boxes for each face, but by using the right orientation of vertices, these volumes are either positive or negative. By summing these volumes, the overlapping volumes disappear and the result is the right volume of the polyhedron as a whole.

4.4 Functions that return simple geometry

There are many unary functions possible that return simple geometry. With simple geometry is meant, geometry that represents single simple objects (like single points or a cube). There are many functions like this possible; these are implemented:

Bounding box: This is the smallest possible orthogonal box around a polyhedron. This function is implemented by searching for the smallest x-, y- and z-coordinate and the largest x-, y- and z-coordinate. The first triplet forms the lower left front vertex and the second triplet forms the upper right back vertex of the box, looking to positive y and the x,z-plane. The other 6 vertices can simply be constructed from these two. The bounding box can be used as a simplified model of the polyhedron.

Transformation: This function should return geometry as a result of scaling, translating or rotating. If the object is valid, just the vertices need to be changed; the face descriptions stay the same. The transformation is implemented as three functions:

Scaling: This function multiplies each vertex with a scalar. It is possible to scale x, y and z separately. The polyhedron (average coordinate) is first translated to the origin (0,0,0), so that the scaling takes place in all directions. After the scaling it is translated back to its original position.

Translation: This function translates each vertex with a translation vector.

Rotation: This function has two parameters, the rotation angle (è) and the rotation axis. First the object (average coordinate) is moved to the origin (0,0,0), then it is rotated and then it is moved back to its original position. The rotation multiplies every vertex with a rotation matrix. The rotation axis defines the elements of the rotation matrix.

Circumscribed sphere: The sphere around the polyhedron through its vertices.

Convex hull: This function removes concavities in the polyhedron, e.g. through holes. This will result in a convex polyhedron.

Inscribed sphere: The largest possible sphere inside the polyhedron.

- Point that is inside the polyhedron: Returns a point that is certainly inside the polyhedron. This can be used as label point in applications.

4.5 Binary functions that return simple geometry

A relatively simple function is the one that returns the line segment representing the distance between two objects. This function has been implemented. It is simply the line between the average coordinates of two polyhedrons.

5 CONCLUSIONS

This paper concentrates on polyhedron as 3D primitive to represent 3D spatial object. The polyhedron is stored as a hierarchical boundary representation, which means that the edges are not stored explicitly and vertices only

need to be stored once. For each polyhedron is stored the set of faces, which consist of a set of ordered nodes. Using functions that are for 3D objects, because these functions work with the 2D projection of the 3D objects. Instead, some of the most commonly used functions are implemented in 3D. Most of these work on the polyhedron primitive.

This paper recommends instead of the polyhedron primitive, an even more realistic primitive can be implemented, e.g. the polyhedron with spherical and cylindrical patches.

6 REFERENCES

[1] B.de Cambray. "Three –Dimensional (3D) Modeling in a Geographical database", *11th int .Symp. on Computer-Assisted Cartography*, pp.338- 347, 1993.

[2] Markus Schneider , B.E. Wenrich, "Use of rational numbers in the design of robust geometric primitives of 3D Spatial database systems", GIS'05,Nov 4-5,2005.

[3] P. van Oosterom, W. Vertegaak, M.van Hekken, and T. Vijlbrief. " Integrated 3D Modeling within a GIS." *Int . Workshop on Advanced Geographic Data Modeling*, pp. 80-95,1994.

[4] S. Zlatanova and K. Tempfli. "Data Structuring and Visualization of 3D Urban Data". *Int. Conf.of the Association of Geographic Laboratories in Europe*, 1998.

[5] M. Schneider." Spatial Data Types for Database Systems-Finite Resolution" ,Springer,1288,1997

[6] Philip J. Schneider David H.Eberly "Geometric Tools for Computer graphics", Morgan Kaufmann Publishers.

[7] Philippe Riqaux,Michel Scholl,Agnes Voisard ,"Spatial Databases With Applications To GIS" : Morgan Kaufmann Publishers.

[8] N.Subhash Chandra, A.Govardhan," Three dimensional data types for spatial abstract data model", Journal of Computer science, sep-oct,2006, volume 2,number 2, pages 171-175.