Computers & Geosciences 31 (2005) 165–177

www.elsevier.com/locate/cageo

Modelling 3D spatial objects in a geo-DBMS using a 3D primitive

Călin Arens*,[1], Jantien Stoter[2], Peter van Oosterom

Section GIS Technology, Delft University of Technology, Delft, The Netherlands

Received 1 July 2003; received in revised form 31 May 2004; accepted 31 May 2004

Abstract

There is a growing interest in modelling the world in three dimensions, both in applications and in science. At the same time, geographical information systems are changing into integrated architecture in which administrative and spatial data are maintained in one environment. It is for this reason that mainstream Data Base Management Systems (DBMSs) have implemented spatial data types according to the 'Simple Feature Specifications for SQL', described by the OpenGeospatial Consortium. However, these specifications are 2D, as indeed are the implementations in DBMSs. At the Section GIS Technology of TU Delft, research has been carried out in which a 3D primitive was implemented in a DBMS (Oracle Spatial). To explore the possibilities and complications, a fairly simple 3D primitive was chosen to start with: a polyhedron. In the future the study will be extended with more complex primitives, the ultimate aim being to build 3D models with features closer to the real world. Besides the data structure, a validation function was developed to check the geometric accuracy of the data. Rules for validation were established and translated into prototype implementations with the aid of literature. In order to manipulate the data, a list of useful 3D functions was specified. Most of these were translated into algorithms, which were implemented in the DBMS. The algorithms for these functions were obtained from the relevant literature. The research also comprised a comparative performance test on spatial indexing in 2D and 3D, using an R-tree.

Finally, existing software was used to visualize 3D objects structured with the implemented 3D primitive. This research is a first attempt to implement a true 3D primitive in a DBMS. Future research will focus on extending and improving the implementations and on optimizing maintenance and query of 3D objects in DBMSs.

Keywords: Spatial DBMS; 3D data structures; Spatial data types; 3D spatial functions; 3D validation

1. Introduction

Geo-DBMSs make it possible to manage large spatial data sets in data bases that can be accessed by multiple users at the same time. These spatial data sets usually contain 2D data, though more and more applications depend on 3D data. Typical examples include 3D cadastres (Stoter, 2000; Stoter et al., 2002), telecommunications (Kofler, 1998) and urban planning (Cambray, 1993). These applications owe their existence mainly to the growing tendency towards the

*Corresponding author.
E-mail addresses: calin_arens@hotmail.com (C. Arens), stoter@itc.nl (J. Stoter), oosterom@geo.tudelft.nl (P. van Oosterom).
[1]Now at Maersk Data Defence AS, Copenhagen, Denmark.
[2]Now at ITC Enschede, The Netherlands.

0098-3004/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2004.05.013

multifunctional use of space by, for example, buildings above roads and railways and bridges and tunnels (Stoter, 2000). The present Geo-DBMSs do not support 3D primitives, but 3D spatial objects can be modelled with 2D primitives such as polygons. This is attainable thanks to 3D coordinates, which *are* supported by the Geo-DBMSs. In this method, several 2D polygons bound a 3D object. These 2D polygons can be stored in one record (multi-polygon) or multiple records (Stoter and Zlatanova, 2003).

The absence of a real 3D primitive in the Geo-DBMSs, however, creates two major problems:

- Geo-DBMSs do not recognize 3D spatial objects, because they do not have a 3D primitive to model them. This results in DBMS functions that do not work properly (e.g. there is no validation for the 3D object as a whole and functions only work when these objects are projected, because the third dimension is ignored). When 3D objects are stored as one multipolygon or a set of polygons, no relationship exists between the different 2D polygons that define the object. Besides the fact that validation is impossible and that any set of polygons can be inserted, the main disadvantage is that the same coordinates are listed several times (causing inconsistency risks) and there is no information about the outer or inner boundaries (shells) of the polyhedron.
- If 2D polygons that bound a 3D object are stored in multiple records, a 1:*n* relationship exists between the object and the number of records; a clearer and more efficient administration of these large data sets requires a 1:1 relationship between objects in reality and objects in the data base.

Geo-DBMSs were developed for the storage of spatial data according to the OpenGeospatial 'Simple Feature Specification for SQL' (OGC, 1999), because they could guarantee the geometric consistency of the data (in 2D). But now that applications have been built which depend on correct 3D data, new techniques need to be developed to support 3D data as well. The ISO/TC211 spatial schema (ISO, 2003) defines 3D geometry primitives in an abstract (mathematical) manner. However, it is outside the scope of this standard to specify the interface of the 3D geometry primitive(s) within the context of a DBMS (SQL). The actual implementation is even further outside the scope. The work presented in this paper tries to bridge this gap by offering a solution in the form of a design and an implementation of a real 3D primitive in a DBMS context, including validation functions and several geometric and topological functions that e.g. reflect the volume or the distance in 3D between objects. This improves the maintainability of 3D geo-data sets (Stoter and Salzmann, 2003) and paves the way for more realistic applications (Cambray, 1993; Kofler, 1998).

This paper will show how 3D spatial objects can be modelled, i.e. stored, validated and queried, in a Geo-DBMS by using a 3D primitive, and how they can be visualized. Many concepts have been developed in 3D modelling (Molenaar, 1990; Van Oosterom et al., 1994; Pigot, 1995; Pilouk, 1996; Kofler, 1998; Saadi Mesgari, 2000; Zlatanova, 2000). What is innovative about this research is that these concepts have been translated into prototype implementations of a true 3D primitive in a DBMS environment (Oracle Spatial 9i Spatial).[3] As far as we know, this is the first time ever that a Geo-DBMS directly supports a 3D primitive. The implementation was based on a proposal for extending the spatial model of Oracle Spatial 9*i* with support for a 3D primitive (Stoter and Van Oosterom, 2002). In addition to the previous work, which focussed on the storage and format of the 3D primitive, this paper also covers aspects such as 3D (validation) functions, 3D indexing (and benchmarking) and 3D visualization.

The rest of this paper is organized as follows. First, a specific 3D primitive, i.e. the polyhedron, is selected from a number of alternatives and an implementation specification is given (Section 2). The validation of a given polyhedron is discussed next with specific attention to the tolerance value and the validation algorithms (Section 3). Section 4 addresses the need for a 2D or 3D spatial index in order to be able to access the data efficiently. Other 3D geometry functions, such as area, distance, and bounding box, on the 0D to 3D primitives are introduced in Section 5. Two options for the visualization of polyhedron data are given in Section 6: via GIS/CAD programs or via VRML browsers. Finally, conclusions are presented in Section 7.

2. 3D primitive

This section begins by discussing the different alternatives for extending the DBMS with a 3D primitive. After selecting the polyhedron primitive, it describes how this primitive is implemented in Oracle Spatial.

2.1. Choosing a 3D primitive

At the moment, Geo-DBMSs are able to store, validate and query spatial data in 2D coordinate space. 2D spatial objects are stored as 2D primitives (polygons). However, to store 3D spatial objects without encountering the problems

[3]Oracle 9i Spatial User Guide and Reference, 2001. http://www.oracle.com.

mentioned in the introduction, a 3D primitive is necessary. There are a number of 3D primitives that can be used for modelling 3D spatial objects:

- *Tetrahedron* (Stoter and Van Oosterom, 2002): This is the simplest 3D primitive (3-simplex). It consists of 4 triangles that form a closed object in 3D coordinate space (Fig. 1). The object is well defined, because the three points of every triangle always lie in the same plane. It is relatively easy to create functions that work on this primitive. The drawback is that it could take many tetrahedrons to construct one factual object; this does not solve the problem that no 1:1 relationship exists between the factual object and the object's representation in the data base (Section 1).
- *Polyhedron* (Stoter and Van Oosterom, 2002): This is the equivalent of a polygon, but then in 3D (Fig. 2). It is made up of several flat faces that enclose a volume. An advantage is that one polyhedron equals one factual object. Because a polyhedron can have holes in the exterior and interior boundary (shell), it can model many types of objects. A disadvantage is that the buffer operation results in a non-polyhedral object, because this will contain spherical or cylindrical patches, which cannot be represented by the polyhedron primitive. The solution is to approximate the result of the buffer operation (De Vries, 2001).
- *Polyhedron combined with spherical and cylindrical patches* (Stoter and Van Oosterom, 2002): This is the equivalent of the current 2D geometry data model of most Geo-DBMSs (i.e. straight lines and arcs). It makes it possible to model 3D objects even more realistically. However, again, this primitive is not closed under the buffer operation. In addition, modelling with this primitive is a complex task. An example is shown below (Fig. 3).
- *CAD objects*: Here there are many possibilities (Mortenson, 1997), such as Constructive Solid Geometry (CSG, Fig. 4), cell decomposition, octree (Cambray, 1993) and objects with curved faces. These objects either do not fit with the present (OpenGeospatial/ISO) 2D geometry data model or are too complex to model without an advanced graphic user interface.

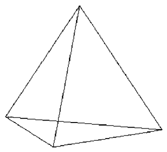
Fig. 1. Tetrahedron.

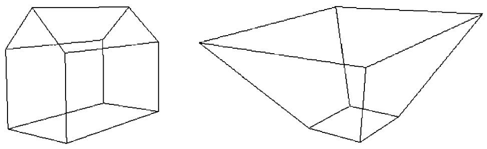
Fig. 2. Collection of polyhedra.

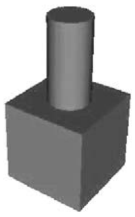
Fig. 3. Polyhedron combined with cylinder.

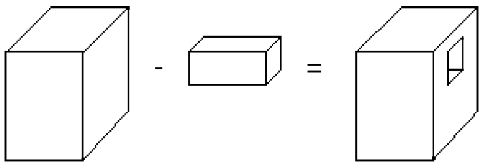

Fig. 4. Example of CSG.

Table 1
Evaluation of possible 3D primitives

	Validation	Realism	Modelling	Algorithms
Tetrahedron	+ +	+	–	+ +
Polyhedron	+	+	+ +	+
Polyhedron combined with spherical and cylindrical patches	–	+ +	–	+/–
CAD objects	–	+ +	+/–	– –

Aguilera (1998) describes some criteria for a 3D primitive. He says that the implementation should lead to valid objects and that it should be easy to create and enable efficient algorithms. Furthermore, the size and redundancy of storage (conciseness) should be taken in consideration. These criteria are evaluated by us in Table 1.

The tetrahedron was not selected, because in general it requires many primitives to model one object and that was one of the disadvantages mentioned earlier on (see Section 1). CAD objects with curved faces can model a spatial object very realistically, but are too complex to model without an advanced graphic user interface and other CAD objects do not fit into the present 2D geometry data model. That leaves the polyhedron option with and without cylindrical/spherical patches. The option with spherical and cylindrical patches would fit better into the present 2D geometry data model, but easy creation and implementation favour the polyhedron without spherical and cylindrical patches at first. Hence, the polyhedron was chosen first as the 3D primitive in this research. If needed, spherical and cylindrical patches could be approximated by several flat faces (Fig. 5). A relatively simple primitive was also more likely to give insight into the problems that could occur when implementing more complex primitives in the future.

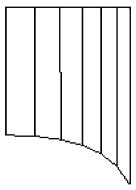
Fig. 5. Approximation of cylindrical patch by several flat faces.

2.2. Implementation

The 3D primitive was implemented in a geometrical model with internal topology. The polyhedron was realized by storing the vertices explicitly (x,y,z) and describing the arrangement of these vertices in the faces of the polyhedron (Fig. 6). This yielded a hierarchical boundary representation (Aguilera, 1998; Zlatanova, 2000). Note that edges were not stored explicitly in this model. This 3D data model can be stored in the existing Oracle Spatial geometry type, which is specified as follows:

```
CREATE TYPE sdo_geometry AS OBJECT (
sdo_gtype NUMBER,                          - - number describing the type of geometry
sdo_srid NUMBER,                           - - number describing the spatial reference
                                           - - this is NULL for local coordinate systems
sdo_point SDO_POINT_TYPE,                  - - one reference point can be stored
sdo_elem_info MDSYS.SDO_ELEM_INFO_ARRAY,- - elements of the geometry
sdo_ordinates MDSYS.SDO_ORDINATE_ARRAY);- - coordinates
```

Managing topological structures between objects (e.g. sharing common faces) is not within the scope of the polyhedron primitive. However, internal topology within one object was maintained since the vertices for one object would be stored only once: faces were defined by internal references to nodes and nodes were shared by faces (Fig. 6).

The interpretation code of the faces (Fig. 6) indicates whether they are part of an outer or inner boundary (of a polyhedron) or part of an outer or inner ring (of a face). Most polyhedra have only an outer boundary (shell), but an inner boundary can, for example, be used to create a hollow object: the inner boundary will then describe this hollow space. Most faces have only an outer ring, but inner rings can be used to create through-holes in polyhedra. These

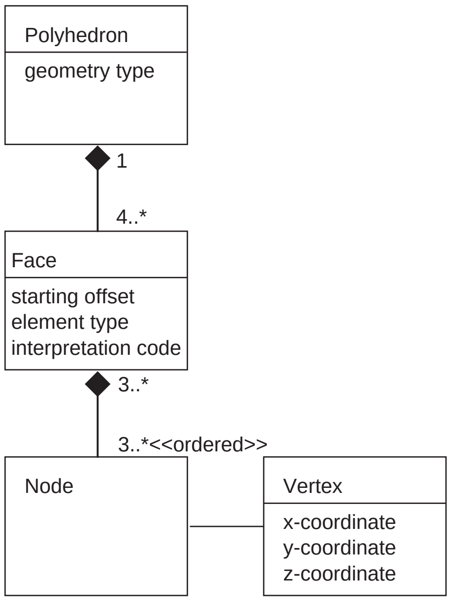

Fig. 6. UML class diagram describing storage of polyhedron primitive.

elements already make it possible to model complex objects, e.g. objects with through-holes or objects that are hollow inside. This set of elements was enough for the functions implemented in the next sections to understand what the 3D spatial objects look like.

It is customary in computer graphics (Van Nieuwenhuizen and Jansen, 2002) to arrange all the vertices of faces in the outer boundaries (outer rings) counter-clockwise, as seen from the outside of an object, and to arrange the vertices of faces in the inner boundaries (outer rings) clockwise (and all inner rings in reverse order). In other words, the normal vector of the face points to the outside of the object. This practice was adopted in the implementation; details and examples in (Arens, 2003).

A table can now be created to hold the polyhedra. This was attained by using the 'CREATE TABLE' command. The table consisted of an id of type NUMBER and a geometry of type MDSYS.SDO_GEOMETRY:

```
CREATE TABLE polyhedron_table (id NUMBER, geometry MDSYS.SDO_GEOMETRY);
```

Then the metadata table is updated. This registers the polyhedron table within the DBMS and makes it possible to create indices on the table:

```
INSERT INTO user_sdo_geom_metadata VALUES (
'POLYHEDRON_TABLE','GEOMETRY',
mdsys.sdo_dim_array(
mdsys.sdo_dim_element('X',−100,100,0.001),
mdsys.sdo_dim_element('Y',−100,100,0.001),
mdsys.sdo_dim_element('Z',−100,100,0.001)),
NULL);
```

The 'INSERT INTO' command is used to insert a polyhedron (cube) in the table (see Fig. 7 with actual polyhedron):

```
INSERT INTO polyhedron_table (id, geometry) VALUES (1, - - id
mdsys.sdo_geometry(3002, - - geometry type: 3D line (for spatial index)
NULL, NULL,
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),
- - starting offset, e_type,int_code
- - first triplet is line for spatial indexing,
- - then the faces of the polyhedron
mdsys.sdo_ordinate_array(
1,1,0, 1,3,0, 3,3,0, 3,1,0,
1,1,2, 1,3,2, 3,3,2, 3,1,2, - - vertices
1,2,3,4, 8,7,6,5, 1,4,8,5, - - bottom, top, front face (nodes)
2,6,7,3, 1,5,6,2, 4,3,7,8 - - back, left, right face (nodes)
)));
```

A 3D R-tree can optionally be created (in order to speed up subsequent querying) by the following SQL 'CREATE INDEX' statement:

```
CREATE INDEX polyhedron_table_index ON polyhedron_table (geometry)
INDEXTYPE IS mdsys.spatial_index parameters('sdo_indx_dims = 3');
```

3. Validation

Large-scale spatial data is very valuable, because of the labour-intensive methods (designing, surveying and processing) that are needed to create it. The DBMS protects the data integrity in a multi-user environment (Kofler, 1998). It is important that the spatial data be checked when it is inserted or changed in the DBMS. Checking the geometry of the spatial objects is called validation. Validation is necessary to ensure that the objects can be correctly manipulated, e.g. it is impossible to compute the volume of a cube when the top face is omitted; this would merely be an open box. Validating may seem fairly easy to humans, but a computer needs a large set of rules to check spatial data. Validation of the values of a data type (such as the polyhedron) is lowest, most fundamental, level of a spatial integrity constraint in the DBMS. Other, more complex, spatial integrity constraints could be imagined, but are not in the scope of this paper. For example, a general constraint in the DBMS/data model ('assertion') could specify that it is not allowed that two polyhedra do overlap.

To allow for checking the spatial data, it is important to give an accurate definition of the 3D primitive. A polyhedron is a bounded subset of 3D coordinate space which is enclosed by a finite set of flat polygons in such a way that each edge of a polygon is shared by exactly one other polygon (Aguilera, 1998). A valid polyhedron bounds a single

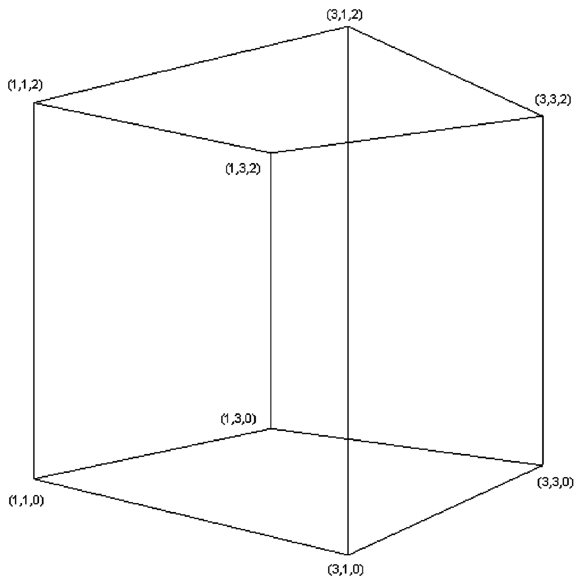

Fig. 7. Polyhedron (cube) defined by its six faces.

volume, which means that from every point (also on the boundary), every other point (also on the boundary) can be reached via the interior. A validation function was implemented on the basis of this definition.

3.1. Tolerance

The validation function and some of the 3D functions (Section 5) have a tolerance value as input parameter. For example, the flat faces of a polyhedron are flat surfaces within a certain tolerance, as the points that make up the polygon can be slightly outside the flat plane, because of the geodetic measuring methods (Stoter and Salzmann, 2003; Stoter and Van Oosterom, 2002; Teunissen and Van Oosterom, 1988) and the finite representation of coordinates in a digital computer. To solve this problem a close-to-zero tolerance value was introduced. It was important that this value was not zero, as this would have introduced errors in the functions if there were any deviations in floating point computations. It should not be too large either; otherwise invalid objects would have been be accepted as valid. A good value for the tolerance is the standard deviation of the geodetic measurements.

3.2. Implementation

The definition of the polyhedron primitive formed the basis for a set of validation rules which were implemented to evaluate the validity of stored objects. All the rules together enforced the correctness of the spatial data.

First of all, a check was needed on the storage of the data. If the validation function was to work properly, the spatial objects needed to be stored as described in Section 2.2. Hence, valid interpretation codes needed to be used and node references in the faces needed to correspond with an existing vertex. Once the spatial object was correctly stored the next test could be carried out.

The next test evaluated the flatness of the faces. These faces should be flat within a given tolerance (Section 3.1). This was tested by estimating a 'least squares' plane through the vertices of the face and then computing the distance from each vertex to this plane. If one of the distances is greater than the tolerance value, the face cannot be flat. At the same time, a test was performed to determine whether the inner boundary of a face was indeed in the same plane as its corresponding outer boundary.

We then ascertained whether the polyhedron bounded a single volume in 3D space (2-manifold polyhedron). This means that the vertices and the edges (2 following vertices) should be 2-manifold, that there should be no intersecting faces and that each polyhedron should be a single object. An example of a non-2-manifold vertex is in Fig. 8, because a single object cannot touch in a vertex. This object bounds two separate volumes in 3D space and should therefore be modelled as two separate polyhedra. A test to determine whether each edge exists exactly twice in opposite order in the polyhedron reveals if the edges are 2-manifold or not (Teunissen and Van Oosterom, 1988).

If the polyhedron is still valid as a whole, the faces have to be checked independently. These faces should be simplicit, which means that they should have an area, should not be self-intersecting and that inner boundaries should not intersect (touch is allowed) their corresponding outer boundaries.

The final test in the validation ascertained whether the vertices in the faces were correctly orientated, i.e. counter-clockwise (looking from the outside) for faces in outer boundaries and clockwise for faces in inner boundaries. Only one face of the polyhedron had to be tested, because if the edges are 2-manifold, the whole object is either orientated correctly or incorrectly. It is important which face is tested. From the bottom face we know that the normal vector should be pointing to negative z-direction. The cross product of two following edges of a convex part of this bottom face gives the normal vector. The z-component of this normal vector should be negative.

If all the criteria in the validation are met, then the spatial object is valid. The following SQL statement tries to validate with our new function the two invalid objects shown in Fig. 9. The result is directly below. Section 5 describes how the validation function was implemented.

```
SELECT validate_polyhedron(geom,0.05) VALID FROM table;
VALID
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Not a 2-manifold object
Not a 2-manifold object
```

Both objects were found to be invalid within a tolerance value of 0.05. Note that the coordinates of these objects are measured in metres. A tolerance value of 0.05 then corresponds to a maximum error of 5 cm. Fig. 10 show how the

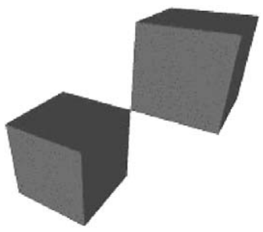
Fig. 8. Invalid object with a non-2-manifold vertex (should be modelled as two objects).

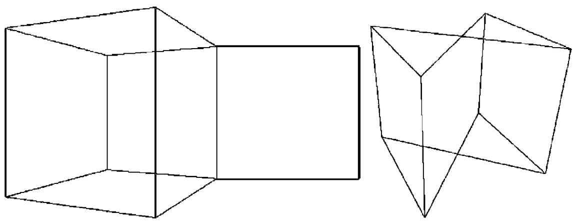
Fig. 9. Invalid objects, left: because of dangling face, right: because of intersecting faces.

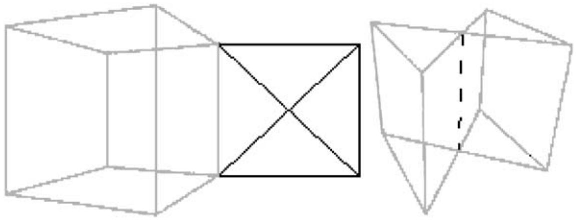
Fig. 10. Corrected objects of Fig. 9.

incorrect objects (of Fig. 9) can be corrected; left object: remove dangling face, right object: split into two neighbour polyhedra (or one multi-polyhedron).

4. Spatial index

4.1. Implementation

No new spatial index was implemented in this research; instead, the present Oracle spatial index could be used. The Oracle spatial index supports R-trees (Guttman, 1984) up to four dimensions and the (2D) quadtree (no support for octree). Use of the Oracle spatial index is made possible by storing the 3D objects in a special way, which could be envisaged as a 3D polyline going through all the coordinates of the defined polyhedron. When creating a 3D R-tree in Oracle, a 3D bounding box is created around this line. This 3D bounding box equals the 3D bounding box around the polyhedron.

4.2. 2D or 3D spatial index?

In many spatial applications the dimensions in the *x*,*y*-plane are greater than in the *z*-direction. For example, a typical city plan covers an area of 5 × 5 km with buildings up to 50 m high. This, plus the fact that queries usually try to find all the objects in a specific (*x*,*y*)-region (possibly with objects that are on top of each other), may make a 3D spatial index less useful in these kinds of applications (Cambray, 1993). In short, the *x*- and the *y*-coordinate are more selective than the *z*-coordinate. This means that a 2D spatial index might work just as well as or better than a 3D spatial index.

A test was performed to determine whether a 2D spatial index might just as well be used as a 3D spatial index (Arens, 2003). The test data set consisted of 1348 objects. In the test (retrieving 3D objects that intersect with a box) the efficiency of the spatial index was measured by determining the number of candidates that were selected by the spatial index compared to the actual number of intersections. SDO_FILTER is the Oracle Spatial function that uses the spatial index to select candidates for spatial queries. It is the only Oracle Spatial function that truly works in 3D (in connection with the 3D R-tree). The following SQL statement shows how to use this filter to retrieve the number of candidates:

```
SELECT COUNT(id) FROM buildings_table WHERE
SDO_FILTER(geometry,(SELECT geometry FROM querywindow WHERE id = 1),
'querytype = WINDOW') = 'TRUE';
```

To retrieve the number of actual intersections, a 3D Boolean intersection function was used which was implemented in this research (Section 5). The function can be used in an SQL statement as follows:

```
SELECT COUNT(id) FROM buildings_table WHERE
intersection(geometry,(SELECT geometry FROM querywindow WHERE id = 1), 0.05) = 1;
```

To use the spatial index in the implemented function, the spatial filter needs to be combined with the intersection function like this:

```
SELECT COUNT(id) FROM buildings_table WHERE
SDO_FILTER(geometry,(SELECT geometry FROM querywindow WHERE id = 1),
'querytype = WINDOW') = 'TRUE'
AND
intersection(geometry,(SELECT geometry FROM querywindow WHERE id = 1), 0.05) = 1;
```

It can be concluded from the test that a 2D index works as well as a 3D spatial index when the query window contains the ground level (Table 2, more details in Arens, 2003). However, if the ground level is not included in a 3D query window, then the 3D R-tree is significantly faster (more efficient), because most objects can be skipped.

With the knowledge that the overhead of a 2D R-tree and a 3D R-tree are both relatively small, there seams to be no reason to build a 2D R-tree on the data set. The 3D R-tree performs just as well as the 2D R-tree when the query window contains the ground level height, but it performs a lot better when it does not. However, the 3D R-tree has one weakness: in case there are many flat objects in the same (e.g. horizontal) plane, then the 3D R-tree may degenerate as the volume contents of many of the 3D bounding boxes is zero (also the boxes belonging to a group of objects). As this content/volume value is used to guide the construction of meaningful spatial clustering, the problem is obvious. Either, the 2D R-tree should be used in such a situation or an improved 3D R-tree algorithm should be implemented (also looking at the area of the boxes when the volume content is zero).

5. 3D Functions

As stated in the introduction, the standard functions in Oracle, as in most Geo-DBMSs, only work when 3D spatial objects are projected onto 2D coordinate space, because the third dimension is ignored, e.g. the area of a face that is standing up (vertical face) is zero, because its 2D projection is a line. Some exceptions are PostGIS[4] and the Spatialware Datablade of MapInfo (based on Informix),[5,6] which do support some geometric calculation such as length and perimeter in 3D of 1D and 2D primitives. The other functions (overlap, area, distance) are also performed only in 2D. To offer realistic functionality, some of the most common functions were implemented in 3D (for 0D up to 3D primitives):

- *Unary function/program to insert data*: Creating data from 3D multi-polygons and VRML.
- *Unary function to validate data*: Validation function (Section 3).

[4]PostGIS, 2003. http://postgis.refractions.net.
[5]Informix, 2003. http://www.informix.com.
[6]MapInfo, 2003. http://www.mapinfo.com.

Table 2
Abstract of query results (Arens, 2003)

Query box	Number of actual intersections	No spatial index		2D R-tree		3D R-tree	
		Number of candidates	Efficiency (%)	Number of candidates	Efficiency (%)	Number of candidates	Efficiency (%)
Including ground level (0–50 m)	509	1348	37.76	510	99.80	510	99.8
Not including ground level (20–50 m)	59	1348	0.04	510	11.57	59	100

- *Binary functions that return a Boolean*: Point-in-polyhedron and intersection[7] tests.
- *Unary functions that return a scalar*: Area, perimeter and volume.
- *Binary functions that return a scalar*: Distance between centroids.
- *Unary functions that return a simple geometry*: Bounding box, centroid, 2D footprint and transformation functions.
- *Binary functions that return a simple geometry*: Line segment representing the distance between centroids.

Functions that return a complex geometry such as tetrahedrization and skeletonization have not been implemented yet, but they are also interesting, because of their analogy with 2D triangulation and generalization (and their possible applications in 3D space).

In order to get high performance and to avoid unnecessary conversions and data communication between DBMS and client, the queries on the data (including the 3D geometric computations) must be performed within the Geo-DBMS itself. This can be realized by storing procedures or functions as part of the data base. In Oracle, these stored procedures and functions can be written in PL/SQL and/or Java, both of them using SQL to access the data. With the help of the spatial index this should lead to a good performance. The functions are implemented in Java, so that they can also be used outside the DBMS environment.

Obviously, functions in 3D require more complex algorithms than 2D functions. This also has a considerable influence on the computational complexity. To maintain a good performance, the algorithms were implemented as efficiently as possible. Spatial data sets can contain many objects, so a slightly more efficient algorithm will already yield a noticeably better performance when querying all these objects.

The following example shows how to compute the area (sum of all faces), volume and perimeter (sum of all edge lengths) of the objects in Fig. 11. The result is directly below.

```
SELECT id, area3d(geom), volume(geom), perimeter(geom) from testobjects;
         ID    AREA3D(GEOM)   VOLUME(GEOM)                PERIMETER(GEOM)
 --------     ---------      ---------     --------------------
          1      22.9530689            5.5                      22.0723224
          2              54             27                              36
          3              58             26                              48
          4             204             98                              96
          5              64             24                              56
```

6. Visualization

To visualize 3D objects it is necessary to use programs that can actually show the third dimension. It is possible to make a viewer, but it is more effective to use existing programs that can access spatial data stored in Geo-DBMSs and to convert the 3D polyhedron data model to a DBMS format which is readable in these programs. We studied two types of options: GIS/CAD programs and VRML viewer.

[7]Of, course, other function based on formal 3D topology relationships can be implemented: touch, in, cross, overlap (same as intersection), disjoint (Van Oosterom et al., 1994).

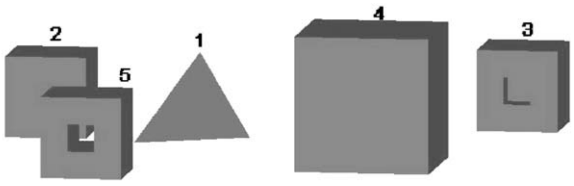

Fig. 11. Set of 5 polyhedra that show storage possibilities.

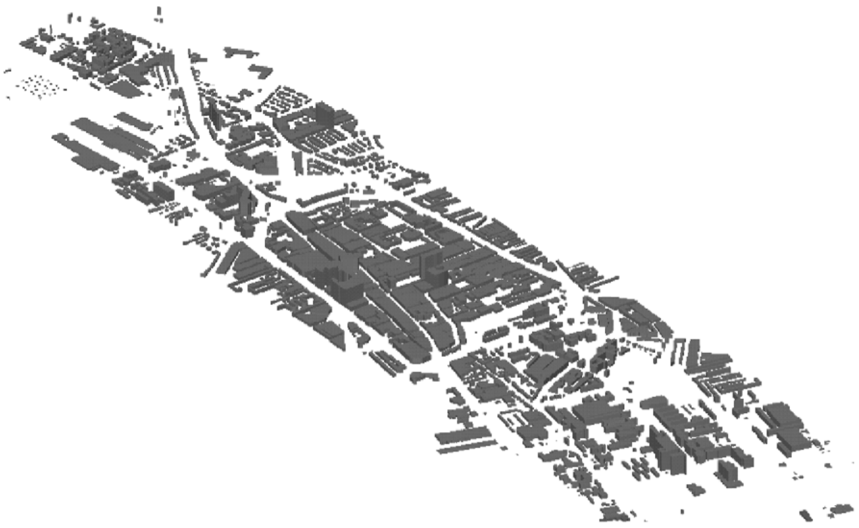

Fig. 12. Visualization of part of Delft (stored in Oracle) in Microstation GeoGraphics.

6.1. *GIS/CAD programs*

GIS/CAD programs that can make a data base connection like MicroStation GeoGraphics[8] and ArcScene/ArcView 3D Analyst[9] can only handle 3D DBMS objects that consist of multiple 2D objects (the present situation, as described in the introduction). The 3D data stored as a 3D data type needs to be converted before it can be visualized (Fig. 12), i.e. the 3D object needs to be split up into multiple 2D polygons in 3D space. This conversion was implemented.

A '2D multi-polygon in 3D space' type can be defined in Oracle Spatial. What distinguishes this from the polyhedron type is that there is no separation between coordinates and face descriptions: faces are described by listing the coordinates. Besides the fact that no validation can be performed, the main disadvantage is that the same coordinates are listed many times and there is no information about the outer or inner boundaries of the polyhedron. The most elegant way to handle this situation is to store the 3D objects in the DBMS using the polyhedron primitive and to define a (DBMS) view to access these polyhedra as multi-polygons for GIS/CAD programs.

6.2. *VRML viewers*

VRML is a language for describing 3D models and making them accessible on the Internet. Interaction and visualization are achieved by plug-ins for web browsers (e.g. Cosmoplayer, Cortona). Since VRML is an open standard and can be used without a licence, it is interesting to look at how 3D data that is stored in the DBMS can be visualized and queried with VRML.

When using VRML,[10] a translation is needed between the 3D type in the data base and the VRML syntax. The geometry in VRML that is useful is the IndexedFaceSet. An IndexedFaceSet is closely related to the storage of the polyhedron type, because it also has a list of coordinates and face descriptions pointing to these coordinates. It has less

[8]Bentley MicroStation GeoGraphics ISpatial edition (J 7.2.x), 2003. http://www.bentley.com/products/geographics/faq.htm.
[9]ESRI ArcView 8 3D Analyst, 2003. http://www.esri.com.
[10]Web3D, 2002. http://www.web3d.org.

information though, because inner boundaries are not explicitly recognizable. Inner boundaries can be specified by creating an edge from and to the outer boundary. It does not matter if one of these edges intersects with another inner boundary; VRML accepts this.

An extra step was needed to convert the VRML file to the polyhedron type: first the VRML file was converted and stored as an SQL loader file. Using the Oracle SQL loader tool it could be loaded into a data base to construct a table from all the geometries that were listed in it. This extra step was taken because it enabled VRML files to be converted without a DBMS connection and it was more efficient to load all geometries into the data base in one run than one-by-one.

Each IndexedFaceSet in the VRML file corresponds to one polyhedron type in a data base. To retrieve the IndexedFaceSets from a VRML file, a Java package called CyberVRML97 for Java[11] was used in our conversion program. Then in the same program, the coordinates and the face descriptions had to be converted to the storage model of the polyhedron type and exported to the SQL loader file.

The reverse function does not use the CyberVRML97 for Java package. Here the data from the data base was written directly to a VRML file, because each geometry in the data base had to be evaluated anyway.

Note that both functions work outside the DBMS; this is because the VRML files are not inside the DBMS.

7. Conclusions

A lot of research has already been carried out into the concepts of 3D data models, including the (abstract) ISO spatial schema (ISO, 2003). The research presented in this paper is a first attempt to provide implementation specifications for a true 3D primitive within the Geo-DBMS, including validation, indexing and spatial functions in 3D. The implementation described in this paper enables users of Geo-DBMSs to consistently maintain their 3D data and perform 3D queries on it. It is also possible to combine 2D and 3D objects in the same DBMS; for example, find all 2D parcels that are intersected by (the projection of) a given polyhedron (representing a tunnel).

The added value, in addition to 3D CAD software, is that Geo-DBMSs can store and manage information on objects and that this information can be queried by numerous other applications, while 3D CAD software focuses more on drawing and visualization. The objective to implement a 3D primitive in a Geo-DBMS in a way that improves the maintainability of 3D spatial data and that paves the way for more realistic applications has therefore been achieved. The internal topology of a polyhedron data type should not be confused with (external) 3D topology structures; see (Van Oosterom et al., 1994) for 3D examples and more discussion on this topic. Basically, the advantages and disadvantages of 3D topological structure management in a DBMS are similar to the situation in 2D (Van Oosterom et al., 2002): redundancy is avoided and consistency is better supported, but more DBMS navigation is needed to reconstruct an object. Topology structures can be useful when many objects are neighbours of each other and share common boundaries (2D or 3D partitions), which have to be maintained during updates.

Acknowledgements

We would like to thank Oracle, ESRI and Bentley for the use of their software and for their support. This publication is the result of the research programme 'Sustainable Urban Areas' (SUA) carried out by Delft University of Technology.

References

Aguilera, A., 1998. Orthogonal polyhedra: study and application. Ph.D. Dissertation, Universitat Politècnica de Catalunya, Barcelona, Spain, 216pp.

Arens, C.A., 2003. Maintaining reality: modelling 3D spatial objects in a GeoDBMS using a 3D primitive. M.Sc. Thesis, Delft University of Technology, The Netherlands, 76pp.

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

De Vries, J., 2001. 3D GIS en grootschalige toepassingen, de opslag en analyse in een geïntegreerde drie-dimensionale GIS M.Sc. Thesis, Delft University of Technology, The Netherlands, 62pp (in Dutch).

[11]Konno, S., CyberVRML97 for Java, 2003. http://www.cybergarage.org/vrml/cv97/cv97java/.

Guttman, A., 1984. R-Trees: a dynamic index structure for spatial searching. In: Proceedings ACM SIGMOD International Conference on Management of Data, Boston, pp. 45–57.

ISO, 2003. ISO/TC 211, ISO International standard 19107:2003, Geographic information—Spatial schema, 8 May 2003.

Kofler, M., 1998. R-trees for the visualizing and organizing large 3D GIS databases. Ph.D. Dissertation, Technical University Graz, Austria, 131pp.

Molenaar, M., 1990. A formal data structure for 3D vector maps. In: Proceedings EGIS, vol. 2, Amsterdam, The Netherlands, pp. 770–781.

Mortenson, M., 1997. Geometric Modelling, second ed. Wiley, New York 523pp.

OGC, 1999. OpenGeospatial Consortium, Inc. OpenGeospatial Simple Features Specification For SQL, Revision 1.1, OpenGIS Project Document 99-049, 5 May 1999.

Pigot, S., 1995. A topological model for a 3-dimensional spatial information system. Ph.D. Dissertation, University of Tasmania, Hobart Australia, 228pp.

Pilouk, M., 1996. Integrated modelling for 3D GIS. Ph.D. Dissertation, ITC, The Netherlands, 200pp.

Saadi Mesgari, M., 2000. Topological cell-tuple structures for three-dimensional spatial data. Ph.D. Dissertation, University of Twente and ITC, ITC Dissertation No. 74, Enschede, The Netherlands, 204pp.

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

Stoter, J.E., Salzmann, M.A., 2003. Where do cadastral needs and technical possibilities meet? In: Cadastral Systems III, 3D Cadastres. In: van Oosterom, P.J.M., Lemmen, C.H.J. (Eds.), Computers, Environment and Urban Systems (CEUS), July 2003, vol. 27(4) pp. 395–410.

Stoter, J.E., Van Oosterom, P.J.M., 2002. Incorporating 3D geo-objects into a 2D geo-DBMS. In: Proceedings FIG ACSM/ASPRS, Washington DC, USA, 12pp, CD-ROM.

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

Stoter, J.E., Salzmann, M.A., Van Oosterom, P.J.M., Van der Molen, P., 2002. Towards a 3D cadastre. In: Proceedings FIG ACSM/ASPRS, Washington DC, USA, 12pp, CD-ROM.

Teunissen, W.J.M., Van Oosterom, P.J.M., 1988. The creation and display of arbitrary polyhedra in HIRASP. Technical Report No. 88-20, University of Leiden, The Netherlands.

Van Nieuwenhuizen, P.R., Jansen, F., 2002. Computer Graphics Lecture Notes, Delft University of Technology, The Netherlands, 95pp.

Van Oosterom, P., Vertegaal, W., Van Hekken, M., Vijlbrief, T., 1994. Integrated 3D Modelling within a GIS. In: Proceedings of the International GIS Workshop AGDM'94 (Advanced Geographic Data Modelling), Delft, The Netherlands, 12–14 September 1994, pp. 80–95.

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

Zlatanova, S., 2000. 3D GIS for urban development. Ph.D. Dissertation, ITC, Publication 69, Enschede, The Netherlands, 222pp.