Proceedings of the 6th AGILE
Dates – Lieu

MODELLING 3D SPATIAL OBJECTS IN A GEO-DBMS USING A 3D PRIMITIVE

Călin Arens, Jantien Stoter and Peter van Oosterom

Section GIS-Technology, Department of Geodesy,
Faculty of Civil Engineering and Geosciences,
Delft University of Technology,
Thijsseweg 11, 2629 JA Delft, The Netherlands
E-mail: calin_arens@hotmail.com, J.E.Stoter@geo.tudelft.nl, oosterom@geo.tudelft.nl

1. INTRODUCTION

Geo-DBMSs make it possible to manage large 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 [1], telecommunications [2] and town planning [3]. 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 [1]. The present Geo-DBMSs do not support 3D primitives, but 3D spatial objects can be modelled by using 2D primitives such as polygons. This is possible by using 3D coordinates, which are supported by the Geo-DBMSs. In this way, several 2D polygons bound a 3D object. These 2D polygons can be stored in one record (multi-polygon) or multiple records.

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

- The Geo-DBMSs do not recognize 3D spatial objects, because they do not have a 3D primitive to model the 3D object. 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 [4]).
- In the case 2D objects, that bound a 3D object, are stored in multiple records, a 1:n relationship exists between the object and the number of records; a better administration of these large datasets requires a 1:1 relationship between objects in reality and objects in the database.

Geo-DBMSs were developed 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, new techniques need to be developed to support 3D data as well. 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 in 3D between objects. This improves the maintainability of 3D geo-datasets [5] and opens the door to more realistic applications [2], [3].

This paper will show how 3D spatial objects can be modelled, i.e. stored, validated and queried, in a Geo-DBMS using a 3D primitive and how these objects can be visualised. Many concepts have been developed in the area of 3D modelling [2], [6], [7], [8], [9], [10]. The innovation of this research is that the developed concepts have been translated into prototype implementations of a true 3D primitive in a DBMS environment (Oracle Spatial 9i Spatial [11]). As far as we know, this is the first time ever that a Geo-DBMS directly supports a 3D primitive. The implementation is based on a proposal for extending the spatial model of Oracle Spatial 9i with support for a 3D primitive [12].

2. 3D PRIMITIVE

This section first discusses the different alternatives for the extension of the DBMS with a 3D primitive. After selecting the polyhedron primitive, it is described how this primitive is implemented within 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). To store 3D spatial objects without the problems mentioned in the introduction, a 3D primitive is necessary. There are a number of 3D primitives possible to model 3D spatial objects:

- Tetrahedron [12]: This is the simplest 3D primitive (3-simplex) and 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 the 4 triangles always lie in the same plane. It is relatively easy to create functions that work on this primitive. The disadvantage is that it could take many tetrahedrons to construct one factual object; this does not solve the problem of not having a 1:1 relationship between the factual object and the object's representation in the database (sect. 1).

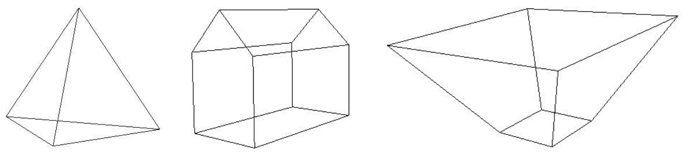


Fig. 1 Tetrahedron (left) and collection of polyhedra (centre and right).

- Polyhedron [12]: This is the equivalent of a polygon, but then in 3D (Fig. 1). It is made up by 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, it can model many types of objects. A disadvantage is that the buffer operation results into a non-polyhedral object, because it 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 [13].

- Polyhedron combined with spherical and cylindrical patches [12]: This is the equivalent of the current 2D geometry data model of most Geo-DBMSs (i.e. straight lines and circular arcs). This possibility makes it possible to model 3D objects even more realistic and is closed under the buffer operation. However, modelling with this primitive is a complex operation. (Fig. 2) shows an example.

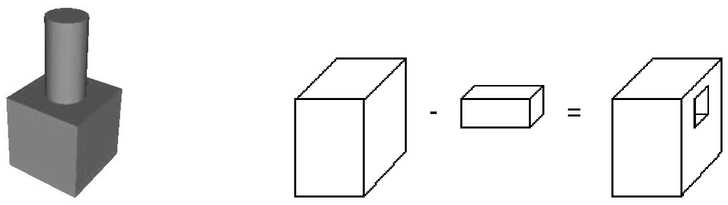


Fig. 2 Polyhedron combined with cylinder (left) and example of CSG (right).

- CAD objects: There are many possibilities [14], such as Constructive Solid Geometry (CSG, Fig. 2), cell decomposition, octree [3] and objects with curved faces. These objects either do not fit with the present (OpenGIS/ISO) 2D geometry data model or are complex to model without an advanced graphic user interface.

To choose a suitable 3D primitive some criteria have to be evaluated [15]. The implementation should lead to valid objects. It should be easy to create and enable efficient algorithms. Further more, the size and redundancy of storage (conciseness) should be taken in consideration. These criteria are evaluated in (Table 1).

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

Table 1 Evaluation of the possible 3D primitives.

The tetrahedron is not suitable, because there are several primitives necessary to model one object and that was one of the problems. CAD objects with curved faces can model a spatial object very realistic, but are complex to model without an advanced graphic user interface and other CAD objects do not fit within the present 2D geometry data model. That leaves the polyhedron option with and without the cylindrical/spherical patches. The one with spherical and cylindrical patches would fit better to the present 2D geometry data model, but ease of creation and implementation favour the polyhedron without spherical and cylindrical patches at first. Therefore, the polyhedron is chosen as the 3D primitive in this research to start with. If needed, spherical and cylindrical patches are approximated by several flat faces (Fig. 3). It is also expected that choosing a relatively simple primitive will give more insight in the problems that occur when implementing more complex primitives in the future.

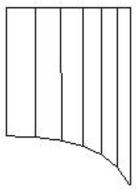


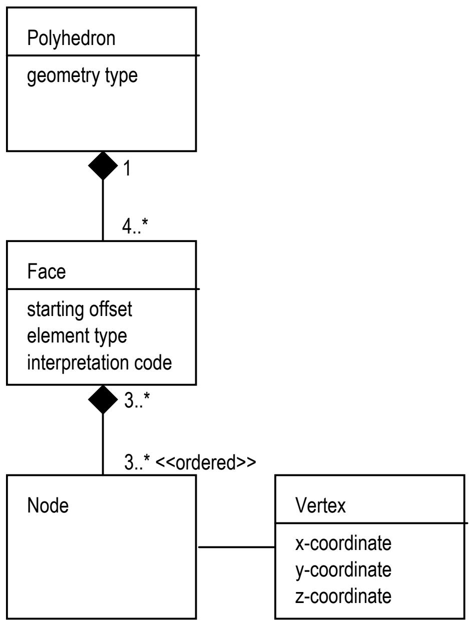
Fig. 3 Approximation of cylindrical patch by several flat faces.

2.2 Implementation

The 3D primitive is implemented in a geometrical model with internal topology. 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. 4). This yields a hierarchical boundary representation [10], [15]. Note that edges are not stored explicitly in this model.

Topology between objects is not maintained. Internal topology is maintained since the vertices for one object will be stored only once: faces are defined by internal references to nodes and nodes are shared between faces (Fig. 4).

The interpretation code of the faces (Fig. 4) describe if these faces are 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 implemented functions in the next sections to understand what the 3D spatial objects look like.



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

INSERT INTO polyhedron_table (id,
geometry) VALUES (1, --id
mdsys.sdo_geometry(3002,
-- geometry type: 3D line
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,
-- then the faces
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)
)));
```

Fig. 4 UML diagram describing the storage of the polyhedron primitive and example.

In the field of computer graphics [16] it is a custom to order all the vertices of outer boundary (ring) counter-clockwise, seen from the outside of an object, and the vertices of inner boundaries (rings) clockwise. This practice is taken over in the implementation (details and examples in [19]).

3. VALIDATION

Large-scale spatial data is very valuable, because of the labour intensive methods (designing, surveying and processing) that create these data. The DBMS protects the data integrity in a multi-user environment [2]. It is important that the spatial data is checked when it is inserted in the DBMS or when it is changed 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, e.g. it is impossible to compute the volume of a cube when the top face is omitted; this would be an open box without a volume. Validating is quite easy for the human eye, but a computer needs a large set of rules to check the spatial data.

To allow for checking the spatial data, it is important to give an accurate definition of the 3D primitive. A polyhedron is defined as a bounded subset of 3D coordinate space enclosed by a finite set of flat polygons such that every edge of a polygon is shared by exactly one other polygon [15]. Note that the polyhedron should bound a single volume, which means that from every point (also on the boundary), every other point (also on the boundary) can be reached via the interior.

Based on this definition, a validation function has been implemented.

3.1 Tolerance

The validation function and some of the 3D functions (sect. 5) have a tolerance value as input. For example, the flat faces of a polyhedron are flat surfaces within a certain tolerance, because the points that make up the polygon can be slightly out of the flat plane, because of the geodetic measuring methods [5], [12], [17] and the finite representation of coordinates in a digital computer. To solve this problem a close to zero tolerance value is introduced. It is important for these functions that these value is not equal to zero, because this will introduce errors in the functions if there are any deviations in floating point computations. This tolerance value should also not be too large, otherwise invalid objects will 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 is the basis for a set of validation rules that have been implemented to evaluate the validity of stored objects. All the rules together enforce the correctness of the spatial data.

First of all, a check is needed on the storage of the data. It is important for the validation function to work properly, that the spatial objects are stored as described in (§2.2). This means that valid interpretation codes need to be used and that node references in the faces should correspond with an existing vertex. If the spatial object is correctly stored the next test can be carried out.

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

Then it is tested if the polyhedron bounds a single volume in 3D space (2-manifold polyhedron). This means that the vertices and the edges (2 following vertices) should be 2-manifold, there are no intersecting faces and that each polyhedron should be a single object. An example of a non-2-manifold vertex is in (Fig. 5), 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. To test if each edge exists exactly 2 times in opposite order in the polyhedron, reveals if the edges are 2-manifold or not [17].

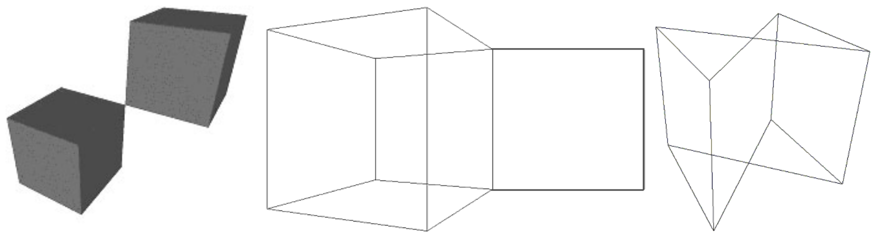


Fig. 5 Invalid objects with a non-2-manifold vertex (left, should be modelled as two objects), dangling face (centre) and intersecting faces (right).

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) with its belonging outer boundary.

The final test of the validation is to check if the vertices in the faces are orientated correctly, i.e. counter-clockwise for outer boundaries and clockwise for inner boundaries. Only one face of the polyhedron has to be tested, because if the edges are 2-manifold, the whole object is either orientated correctly or incorrectly. It is important which face to test. 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 validates the two objects shown in (Fig. 5). The result is right below it. How the validation function has been implemented is described in (sect. 5).

```
SELECT validate_polyhedron(geom,0.05) VALID FROM table;

VALID
--------------------------------------------------------------------
Not a 2-manifold object
Not a 2-manifold object
```

Both objects are detected to be invalid within a tolerance value of 0.05. Note that the coordinates of these objects are measured in meters. A tolerance value of 0.05 then corresponds to a maximum error of 5 centimetres.

4. SPATIAL INDEX

4.1 Implementation

No new spatial index is implemented in this research; instead the present Oracle spatial index can be used. The Oracle spatial index supports R-trees [18] up to 4 dimensions and the (2D) quadtree (no support for octree). Using the Oracle spatial index is made possible by storing the 3D objects in a special way. A 3-dimensional polyline going through all the coordinates of the defined polyhedron is imagined. When creating a 3D R-tree in Oracle, a bounding volume is created around this line. This bounding volume around the line equals the bounding volume around the polyhedron.

4.2 2D or 3D spatial index?

In many spatial applications the dimensions in the x,y-plane are larger than in the z-direction. For example, a city plan typically covers an area of 5x5 kilometres with buildings up to 50 meters tall. This, plus the fact that queries usually try to find all the objects in a specific (x,y)-region (with possibly objects that are on top of each other), may make a 3D spatial index less useful in these kinds of applications [3]. In short, the x- and y-coordinate are more selective than the z-coordinate. This means a 2D spatial index might work just as good or better than a 3D spatial index

A test was executed to see if one might just as well use a spatial index and not a 3D spatial index [19]. The test dataset 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 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 is implemented (sect. 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, you have to combine the spatial filter 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;
```

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 (0-50m)	509	1348	37,76%	510	99,80%	510	99,80%
Not including ground level (20-50m)	59	1348	0,04%	510	11,57%	59	100%

Table 2 Abstract of query results from [19].

From the test it can be concluded that a 2D index works as good as a 3D spatial index when the query window contains the ground level (Table 2, more details in [19]). 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 is no reason to build a 2D R-tree on the dataset. The 3D R-tree performs equally well as the 2D R-tree in case the query window contains the ground level height, but it performs a lot better when this query window does not contain the ground level height.

5. 3D FUNCTIONS

As stated in the introduction, the standard functions in Oracle, just as in most Geo-DBMSs, only work with the projection of 3D spatial objects on 2D coordinate space, because the third dimension is ignored, e.g. the area of a face that is standing up is zero, because its 2D projection is a line. Some exceptions are PostGIS [20] and the Spatialware Datablade of MapInfo (based on Informix [21], [22]) that do support geometry calculation such as length and perimeter in 3D. The other functions (overlap, area, distance) are also performed only in 2D. To offer realistic functionality, some of the most common functions have been implemented in 3D:

- Function to insert data: Creating data from 3D multi-polygons and VRML.
- Function to validate data: Validation function (sect. 3).
- Functions that return a Boolean: Point-in-polyhedron and intersection test.
- 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 tetrahedrisation and skeletonisation are not implemented yet, but are also interesting, because of their analogy with 2D triangulation and generalisation.

In order to get high performance and to avoid unnecessary conversions and data communication between DBMS and client, the data should be queried in the Geo-DBMS itself. This can be done by storing procedures or functions as part of the database. 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 good performance. The functions are implemented in Java, so that these can also be used outside the DBMS environment.

It is clear that functions in 3D require more complex algorithms than 2D functions. This also has a big influence on the computational complexity. To maintain good performance, there has to be a lot of emphasis on keeping the algorithms as efficient as possible. Spatial datasets can contain many objects, so a slightly algorithm already will yield noticeable better performance when querying all these objects.

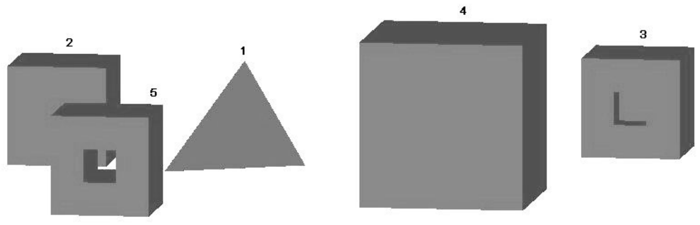


Fig. 6 A set of 5 polyhedra that show the storage possibilities.

The following example shows how to compute the area, volume and perimeter of the objects in (Fig. 6). The result is right below it.

```
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. VISUALISATION

To visualise 3D objects it is necessary to use programs that actually can show the third dimension. It is possible to make a viewer, but it is easier and better 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 readable to these programs. We studied two options: GIS/CAD programs and VRML.

6.1 GIS/CAD programs

GIS/CAD programs that can make a database connection like Microstation GeoGraphics [23] and ArcScene / ArcView 3D Analyst [24] can only handle 3D DBMS objects that consist of multiple 2D objects (the present situation described in the introduction). The 3D data stored as a 3D data type needs a conversion before it can be visualised (Fig. 7), i.e. splitting up the 3D object into multiple 3D polygons.

A 3D multi-polygon type can be defined in Oracle Spatial. The difference with the polyhedron type is that there is no separation between coordinates and face descriptions: faces are described by listing the coordinates. Beside the fact that no validation can be performed, the main disadvantage is that the same coordinates are listed multiple times and there is no information about outer or inner boundaries of the polyhedron.

6.2 VRML

VRML is VRML is a language to describe 3D models and to make them accessible on the Internet. Interaction and visualisation is done by plug-ins for web browsers (e.g. Cosmoplayer, Cortona). Since VRML is an open standard and can be used without licenses, it is interesting to look how 3D data that is stored in the DBMS can be visualised and queried with VRML.

When using VRML [25], there needs to be a translation between the 3D type in the database and the VRML syntax. The type of 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 information though, because inner boundaries are not explicitly recognisable. 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.

There is an extra step to convert the VRML-file to the polyhedron type: first the VRML-file is stored as an SQL-loader file. With the Oracle tool SQL-loader this file can be loaded into a database to construct a table from all the geometries listed in this file. This extra step is taken, because it gives the possibility to convert VRML-files without a DBMS connection and it is more efficient to load all geometries in one run into the database than one by one.

Each IndexedFaceSet in the VRML-file corresponds to one polyhedron type in a database. To retrieve the IndexedFaceSets from a VRML-file, a Java package called CyberVRML97 for Java [26] is used. From here, the coordinates and the face descriptions have to be converted to the storage model of the polyhedron type and exported to the SQL-loader file.

The vice-versa function does not use the CyberVRML97 for Java package. Here the data from the database is written to a VRML file directly, because each geometry in the database has to be evaluated anyway.

Note that both functions work outside the DBMS, because the VRML-files are not inside the DBMS.

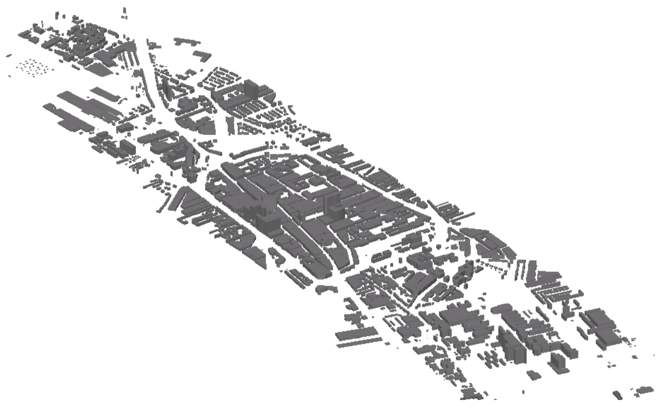


Fig. 7 Visualisation of part of Delft in Microstation GeoGraphics.

7. CONCLUSIONS

There has already been a lot of research on the concepts of 3D data models. This research is a first attempt to implement a true 3D primitive in the Geo-DBMS including validation functions, indexing and spatial functions in 3D. The implementation described in this paper enables users of Geo-DBMSs to add their 3D data and perform 3D queries on them. The added value above 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 visualisation. The objective to implement a 3D primitive in a Geo-DBMS in a way that the maintainability of 3D spatial data improves and that the door is opened to more realistic applications is hereby satisfied.

8. BIBLIOGRAPHICAL REFERENCES

[1] Stoter, J.E., «Needs, possibilities and constraints to develop a 3D cadastral registration system», *22nd Urban Data Management Symposium 'Urban and Rural Data Management Common Problems - Common Solutions', Delft, The Netherlands*, vol. III, 43-58, 2000.

[2] Kofler, *R-trees for the Visualisation of Large 3D GIS Database*, PhD Thesis, Technical University Graz, Austria, 1998.

[3] Cambray, «Three-dimensional modelling in a geographical database», *11th International Conference on Computer Assisted Cartography*, 338-347, 1993.

[4] Stoter, J.E. et al., «Towards a 3D cadastre», *FIG, ACSM/ASPRS Washington DC, USA*, 2002.

[5] Stoter, J.E. and M. Salzmann, «Towards a 3D cadastre: where do cadastral needs and technical possibilities meet?», *International Workshop on 3D Cadastres, Registration of properties in strata, Delft, The Netherlands*, 2001.

[6] Molenaar, M., «A Formal Data Structure for 3D Vector Maps», *EGIS'90 Amsterdam, The Netherlands*, vol. 2, 770-781, 1990.

[7] Pigot, S., *A Topological Model for a 3-Dimensional Spatial Information System*, PhD Thesis, University of Tasmania, Australia, 1990.

[8] Pilouk, M., *Integrated Modelling for 3D GIS*, PhD thesis, ITC, The Netherlands, 1996.

[9] Saadi Mesgari, M., *Topological Cell-Tuple Structures for Three-Dimensional Spatial Data*, PhD Thesis, University of Twente and ITC, ITC Dissertation Number 74, Enschede, The Netherlands, 2000.

[10] Zlatanova, S., *3D GIS for urban development*, PhD Thesis, ITC publication 69, Enschede, The Netherlands, 2000.

[11] Oracle, *Oracle 9i Spatial User Guide and Reference, Release 9.0.1, Part Number A88805-01*, 2001.

[12] Stoter, J.E. and P.J.M. van Oosterom, «Incorporating 3D geo-objects into a 2D geo-DBMS», *FIG, ACSM/ASPRS, Washington DC, USA*, 2002.

[13] De Vries, J., *3D GIS en grootschalige toepassingen, De opslag en analyse in een geïntegreerde drie-dimensionale GIS*, MSc thesis (in Dutch), 2001.

[14] Mortenson, M., *Geometric Modelling 2nd ed*, 1997.

[15] Aguilera, A., *Orthogonal polyhedra: study and application*, PhD thesis, Barcelona, Spain, 2001.

[16] Nieuwenhuizen, van and Jansen, *Computer graphics lecture notes*, 2000.

[17] Teunissen and Van Oosterom, *The creation and display of arbitrary polyhedra in HIRASP*, 1988.

[18] Guttman, A., «R-Trees: A dynamic index structure for spatial searching», *ACM SIGMOD international conference on management of data*, 45-57, 1984.

[19] Arens, C.A., *Maintaining Reality: Modelling 3D spatial objects in a GeoDBMS using a 3D primitive*, MSc thesis, March 2003

[20] PostGIS, *postgis.refractions.net*, 2003.

[21] MapInfo, *www.mapinfo.com*, 2003.

[22] Informix, *www.informix.com*, 2003.

[23] Bentley MicroStation GeoGraphics ISpatial edition (J 7.2.x), *http://www.bentley.com/products/geographics/faq.htm*, 2003.

[24] ESRI, ArcView 3.2a, 3D Analyst, *www.esri.com*, 2003.

[25] Web3D, *www.web3d.org*, 2002.

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