Visualisation and editing of 3D objects organised in a DBMS

Jantien Stoter and Siyka Zlatanova

Delft University of Technology, The Netherlands

Abstract

The developments in DBMSs are quite significant in the last several years. Mainstream DBMSs (Oracle, IBM DB2, Informix, Ingres) have already implemented spatial data types and spatial operators similar to the OpenGIS Consortium Simple Features Specification for SQL. The implementation consists of an SQL extension that supports storage, retrieval, query and update of simple spatial features (points, lines and polygons). Currently the implementations are only 2D, i.e. point, line and polygon are supported. However, in most DBMSs 3D coordinates can be used to represent 2D features in 3D (3D points, 3D lines and 3D polygons). Specifications for complex features (topology) as well as for 3D geometry objects are currently being developed by the OpenGIS Consortium in co-operation with ISO. Furthermore many frond-end applications (CAD, GIS) offer tools to access, to edit and to post data to the DBMS. The greatest benefits of such DBMS-GIS/CAD integration are in the area of visualisation and data editing. Having 3D data stored in a DBMS, the user has the possibility to extract only a limited set of data and thus critically reduce the time for loading. Locating, editing and examining a particular object also becomes quick, simple and convenient.
This paper summarises our experiments on storage and visualisation of 3D data, using the current tools provided by Oracle Spatial 9i, MicroStation Geographics (Bentley), ArcGIS (ESRI) and VRML.

1. Introduction

Spatial data, which is used in GISs, are mostly part of a complete work- and information-process. Therefore in many organisations there is a growing need for a central DBMS: a system in which spatial data and attribute data are maintained in one integrated environment. Many DBMSs are capable of maintaining spatial data in 2D, since spatial data types in 2D have been implemented in DBMSs. Relevant questions now are: how can 3D data be stored in DBMSs and are GIS front-ends capable of visualising and editing spatial data that is stored in a DBMS? One of the statements of the EuroSDR workshop on visualisation and rendering is ‘technically we can do many things‘. This paper focuses on the question if this really is the case when storing 3D objects in a DBMS and also when visualising these data in GIS and CAD front-ends. The first section describes how spatial data can be represented both in 2D and 3D in the DBMS in which we use Oracle Spatial 9i as example. The second section describes our experiences on visualising (end editing) these data in two commercial mainstream GIS/CAD front-ends: Bentley and ESRI. In this section we also describe our experiences on visualising these data with an open standard: VRML. We end with a discussion in which we will conclude what technically is possible concerning visualising 3D objects organised in a DBMS and what kind of further research is needed.

2. Spatial objects in DBMSs

According to the OpenGIS specifications, the spatial object is represented by two structures, i.e. *geometrical (i.e. simple feature specifications)* and *topological (i.e. complex feature specifications).* While the geometric structure provides direct access to the coordinates of individual objects, the topological structure encapsulates information about their spatial relationships. The geometrical model has been implemented in mainstream DBMSs and will be discussed in this paper. Recently topology has become available in a commercial package (Laser-scan Radius based on Oracle Spatial, Laser-scan Radius (2002)). For a discussion on topology in DBMSs see (Stoter and Zlatanova, 2002).

2.1 Geometrical model in the DBMS

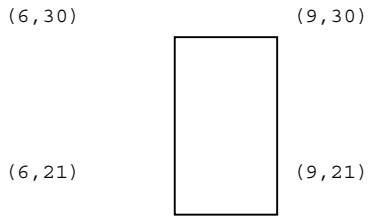
Mainstream DBMSs (Oracle (Oracle, 2001), IBM DB2 (IBM, 2000), Informix (Informix, 2000), Ingres (Ingres, 1994)) have implemented spatial data types and spatial operators more or less similar to the OpenGIS Consortium (OGC, 1998) Simple Features Specification for SQL (OGC, 1999). The implementation consists of an SQL extension that supports storage, retrieval, query and update of simple spatial features (points, lines and polygons).

The spatial features are stored in a geometrical model, without internal topology. Topological relationships between geometries can be retrieved by the use of spatial operators. In Ingres the support for topological relationships is minimal. Oracle, IBM DB2, Informix, PostGIS support geometric functions defined by OGC and often more functions than these (Oosterom et al., 2002). OGC specifications (OGC, 2001 (1)) are until now 2D. Also the implementations of spatial data types in mainstream DBMSs are basically 2D, although specifications for complex features (topology) as well as for 3D geometry objects are currently being developed by the OpenGIS Consortium in co-operation with ISO (OGC, 2001 (2)).

To illustrate the spatial possibilities within DBMSs Oracle Spatial 9i (Oracle, 2001) is used. Although Oracle is not completely OpenGIS compliant, we use Oracle Spatial 9i for our experiments because Oracle is one of the most used DBMSs. The supported spatial features in Oracle are point, line and polygons (including arcs, boxes and mixed geometry sets). The object-relational model in Oracle defines the object type sdo_geometry as:

```
CREATE TYPE sdo_geometry AS OBJECT (
SDO_GTYPE NUMBER,                    -- type of the geometry (e.g. point, linestring, polygon)
SDO_SRID NUMBER,                     -- reference to the spatial reference system
SDO_POINT SDO_POINT_TYPE,
SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,          -- indicates how the ordinate array should
                                                  be interpreted
SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);          -- list of coordinates
```

An example of using the Oracle object-relational model to represent a polygon:




```
SDO_GEOMETRY Column = (
SDO_GTYPE = 2003
SDO_SRID = NULL
SDO_POINT = NULL
SDO_ELEM_INFO= (1,1003,1)
SDO_ORDINATES =(6,21, 9,21, 9,30, 6,30, 6,21))
```

In sdo_gtype=2003, the first position indicates the dimension (2D in this case), the last position indicates the element type (3 indicates a polygon). In sdo_elem_info, the combination 1003,1 in 1,1003, 1 indicates that this is a polygon containing straight lines. The first position ('1') indicates that the first (and only) element starts at offset 1 in the coordinate list.

The next SQL statements illustrate how a box (with 0,0 as lower left and 100,100 as upper right coordinates) is stored as a geometry type in Oracle (sdo_geometry type) in the 'geom2d' table. The same box with height 50 is stored in the 'geom3d' table.

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

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

```
  mdsys.SDO_ORDINATE_ARRAY(0,0, 100,0, 100,100, 0,100, 0,0)), 8);
/* a 3D box */
insert into geom3d (shape, ID) values (
  mdsys.SDO_GEOMETRY(3003, NULL, NULL, mdsys.SDO_ELEM_INFO_ARRAY(1, 1003, 1),
  mdsys.SDO_ORDINATE_ARRAY(0,0,50, 100,0,50, 100,100,50, 0,100,50, 0,0,50)), 9);
```

Besides the tables representing the geometries of the objects, metadata is maintained in Oracle describing the dimension, lower and upper bounds and tolerance in each dimension. In the following statements the information on the tables geom2d and geom3d is inserted in the metadata table. Finally, a spatial index (r-tree in 2D and in 3D) is created on the tables (to speed up spatial queries).

```
/* inserting metadata */
/* 2D table*/
insert into user_sdo_geom_metadata values
('GEOM2D', 'SHAPE', mdsys.sdo_dim_array(
mdsys.sdo_dim_element('X', 0, 500, 0.5),
mdsys.sdo_dim_element('Y', 0, 500, 0.5) ), NULL);
/*3D table*/
insert into user_sdo_geom_metadata values
('GEOM3D', 'SHAPE', mdsys.sdo_dim_array(
mdsys.sdo_dim_element('X', 0, 500, 0.5),
mdsys.sdo_dim_element('Y', 0, 500, 0.5),
mdsys.sdo_dim_element('Z', 0, 300, 0.5)
),  NULL);

/* creating index */
/* geom2d*/
create index GEOM2D_I on GEOM2D(SHAPE)
 indextype is mdsys.spatial_index;
analyze table GEOM2D compute statistics;
/* geom3d*/
create index GEOM3D_I on GEOM3D(SHAPE)
 indextype is mdsys.spatial_index parameters('sdo_indx_dims=3');
analyze table GEOM3D compute statistics;
```

2.2 3D objects in DBMSs

Most DBMSs (Oracle, Postgres, IBM, Ingres, Informix) support the storage of 3D points, lines and polygons, as was illustrated by example in the previous section. Using 3D polygons, 3D objects can be represented as polyhedrons (body with flat faces) in two ways: as a list of polygons or as multipolygon (one polygons consisting of several polygons). To illustrate this, the cube in figure 1 will be used.

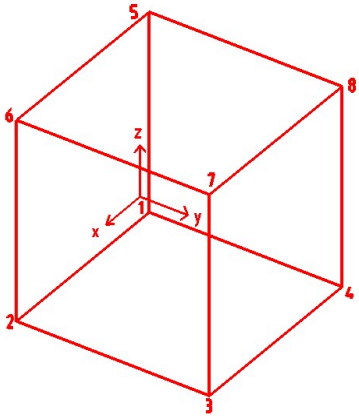


Figure 1: Cube to be stored in the DBMS

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

Table 1: The body and face table in case the cube is stored as a set of polygons.

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

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

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

Table 2: BODY table in case the cube is stored as one multipolygon

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

An advantage of 3D multipolygons (compare to list of polygons) is that it is recognised as one object by front-end applications (GIS/CAD) that can access, visualise, and edit these data and post the changes back to the database. Another advantage of the 3D multipolygon approach is the one-to-one correspondence between a record and an object. A disadvantage of both representations is that the topology structure between objects cannot be used, which implies that there is redundant storage of coordinates (and in the 3D multipolygon solution also of faces).

These disadvantages can be overcome with the implementation of real 3D (volumetric) data types. Stoter and Oosterom, 2002 proposed an extension of Oracle Spatial 9i with support of a true 3D data type: the polyhedron primitive. This primitive is currently being implemented including the data model, validation functions and spatial functions in 3D (Arens et al. 2003).

2.3 3D analyses in DBMSs

Our experiments have showed that it is possible to maintain objects with 3D coordinates in Oracle Spatial 9i. However, the current implementations of geometry operators (e.g. compute area of 3D polygon) in Oracle Spatial 9i omit the z-value. Many other DBMSs support similar set of geometry operators as most of them also skip the z coordinate. Some exceptions are PostGIS (PostgreSQL) (PostGIS, 2002) and the Mapinfo Spatialware Datablade (Mapinfo, 2002) (based on Informix) that do support geometry calculation such as length and perimeter in 3D. This is illustrated in the next PostGIS example. First, four tables are created: line2D, line3D, polygon2D and polygon3D in which respectively a 2D line, a 3D line, a 2D polygon and a 3D polygon are inserted:

```
/* a table with a 2D line */
create table line2d (id int4)\g
select addgeometrycolumn('test', 'line2d', 'shape', 0, 'LINESTRING', 2)\g
insert into line2d (id, shape) values(1, geometryfromtext('LINESTRING(1 1, 2 2)', 0))\g

/* a table with a 3D line */
create table line3d (id int4)\g
select addgeometrycolumn('test', 'line3d', 'shape', 0, 'LINESTRING', 2)\g
insert into line3d (id, shape) values(1, geometryfromtext('LINESTRING(1 1 0, 2 2 50)', 0))\g

/* a table with a 2D polygon */
create table polygon2d (id int4)\g
select addgeometrycolumn('test', 'polygon2d', 'shape', 0, 'POLYGON', 2)\g
insert into polygon2d (id, shape) values(1, geometryfromtext('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))',
0))\g

/* a table with a 3D polygon */
drop table polygon3d\g
create table polygon3d (id int4)\g
select addgeometrycolumn('test', 'polygon3d', 'shape', 0, 'POLYGON', 3)\g
insert into polygon3d (id, shape) values(1, geometryfromtext('POLYGON((0 0 0, 1 0 0, 1 1 100, 0 1 100,
0 0 0))', 0))\g
```

In the next step the following queries are executed:

```
select length(shape) from line2d\g
select length3d(shape) from line3d\g
select perimeter(shape) from polygon2d\g
select perimeter3d(shape) from polygon3d\g
```

As the results show, length and perimeter do work in 3D:

```
   length
------------------
 1.4142135623731
(1 row)

   length3d
-------------------
 50.0199960015992
(1 row)

   perimeter
-----------
         4
(1 row)

   perimeter3d
------------------
 202.009999750012
(1 row)
```

The other functions (overlap, area, distance) in PostGIS (and also in the SpatialWare Datablade of MapInfo) are performed in 2D. PostGIS also has a box3D function that gives the maximum extents in 3D as result.

3. Visualising spatial objects stored in DBMSs

Once 3D data has been stored in a DBMS, it can be optimally maintained together with the attribute information. The next issue is to get access to these data by means of GIS/CAD front-ends. In this section three front-ends to visualise 3D data that is organised in Oracle Spatial 9i will be examined: Bentley (Bentley, 2002), ESRI (ESRI, 2002) and VRML (Ames et al., 1997).

3.1 GeoGraphics iSpatial, Bentley

Geographics iSpatial is an extension of MicroStations that access the geometric types of Oracle Spatial 9i. The organisation of data within Geographics iSpatial is defined in a project hierarchical structure. *Project* represents the data for the entire study area. The second level is the *category*, which groups features with a similar theme (e.g. buildings, rivers). One project can have many categories but a category may belong to only one project. *Feature* is at the third level and represents one or more objects (e.g. the bank building, the school building) with the same semantic attributes. A category may have many features but a feature may belong to only one category. Feature is the basic structural unit in GeoGraphics iSpatial. The geometry of the objects is described in *spatial layer,* which (among all) contains the name of the relational table and column that contains the geometric types (as they are defined in Oracle Spatial).

Visualisation of spatial data from Oracle Spatial is relatively simple and straightforward. The user has to create a Project and connect to Oracle Spatial. GeoGraphics iSpatial check the Oracle metadata table for the name of the table(s) and corresponding columns that contain spatial data. These are supplied to the user for display. Note, in this case, the geometries will be visualised, but will not be available for editing.

Much more steps have to be taken in order to colour objects separately or to perform an 'identify' on objects and also in order to edit and post spatial objects in Oracle Spatial 9i. In general, each spatial object (in the Oracle database) has to be assigned to a feature (in GeoGraphics), but depending on the original location of data (Oracle of MicroStation), different steps have to be followed.

We completed a number of case studies following the two different approaches of representing 3D objects, i.e. 3D polygons and 3D multipolygons, having the data initially organised either in Oracle (user-defined tables) or in MicroStation (graphics in a DGN file).

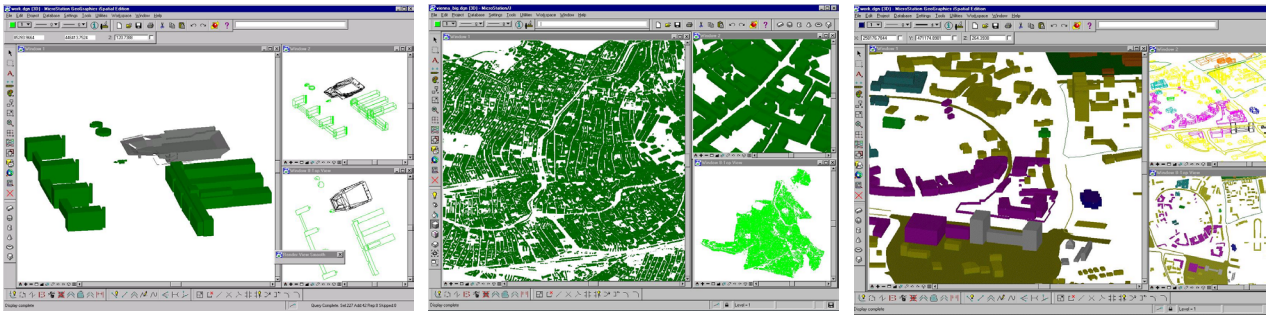


Figure 2: Experimental data sets: TUDelft (left), Vienna (middle) and Enschede (right)

We have used several data sets. Figure 2 shows some of them: several buildings of the campus areas in TUDelft (figure 2, left), 21 000 buildings of the city of Vienna (figure 2, middle) and central part of Enschede (right). In the first two cases, the 3D objects where primarily organised in the Oracle Spatial 9i and accessed from GeoGraphics iSpatial. The required steps to assess and query the objects are described in details in Zlatanova et al 2002. Here, we summarise the most important ones. The user has to:

1. create semantics, i.e. project, features and categories
2. create a spatial layer (in GeoGraphics) and reference it to the corresponding table and column that contains the geometries (in our case the tables 'BODY' and 'FACE'). The result of this

operation is significant modification of the use-defined tables, i.e. 10 new columns are introduced.

3. link features (the code of the feature) to the corresponding spatial objects (id of spatial object). Running an appropriate script within Oracle is one of the easiest ways to complete this operation in case of many objects.

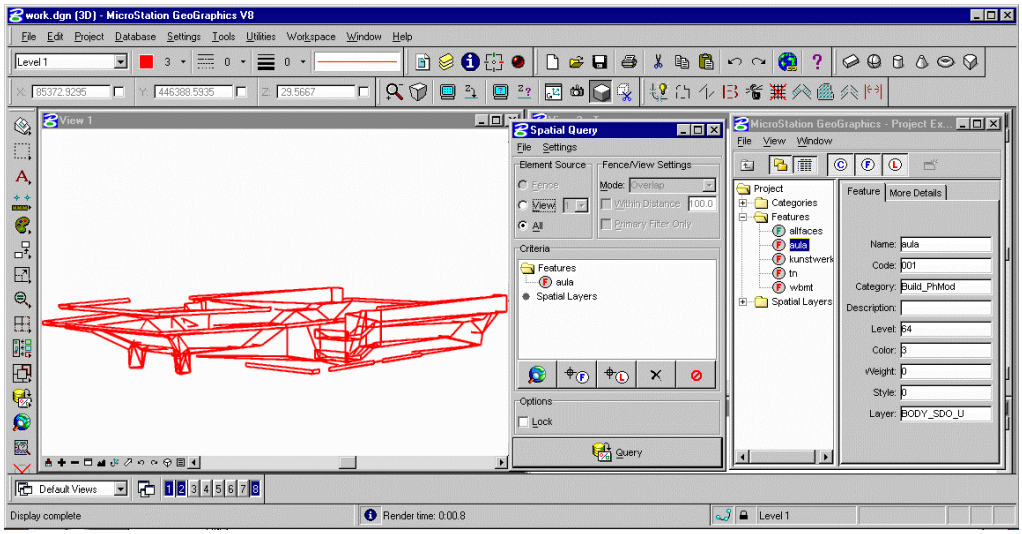


Figure 3: Query of the Aula building, TUDelft

The third data set was obtained by a semi-automatic procedure for 3D reconstruction developed in ITC, Enschede. The current procedure is an extension to the one presented in (Tempfli, 1998). The manual digitising points (in *SocketSet*) characterising roofs of buildings in a photogrammetric stereo model (from aerial photographs) creates a 'skeletal point cloud'. The 3D reconstruction then consists of automatically computing and assembling all the faces (roof faces and walls) of the building from this point cloud. The model obtained in this way contains planar closed polygons, which normal vector points towards the outside of the building (to ensure correct 3D visualisation). All reconstructed objects are organised in a topological data structure 3D FDS (Molenaar, 1990). The software creates also a new DGN file with the 3D reconstructed object. These 3D objects were imported into Oracle Spatial directly from the DGN file by the following major steps:

1. Creating project, features, categories and spatial layers.
2. Selecting the entire geometry (polygons or groups of polygons) per spatial objects in GeoGraphics and attaching a feature to it.
3. Posting the spatial objects to the database.

In both approaches, after completing all the required steps, it was possible to query, visualise and edit the objects. The query can be performed on the basis of the semantic characteristics of the objects as they are defined in GeoGraphics. For example, query on feature 'buildings' will result in visualising all the buildings. Figure 3 shows the result of querying the 'aula' building. The feature 'aula' is attached to the geometry of only one spatial object. Apparently, such possibility brings advantages for editing and updating large 3D models. Instead of working with the entire model (e.g. Vienna model consists of 21 000 buildings), the user can query and work with only one object. Thus rendering of thousands of polygons can be easily avoided.

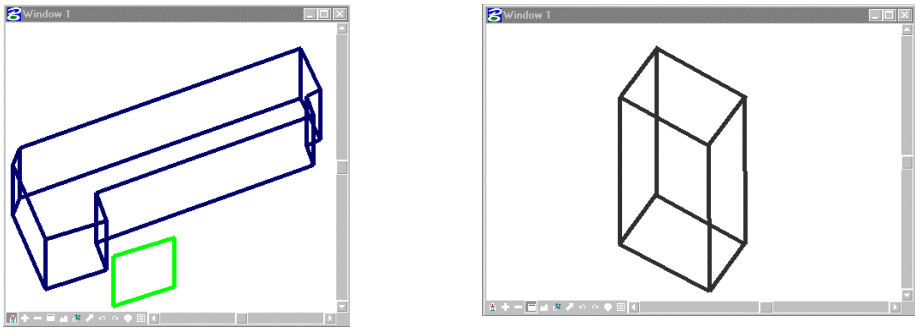


Figure 4: 3D object represented as a set of polygons: in GeoGraphics one of the polygons is shifted (left) and part of the relational table BODY_SDO in Oracle (right)

It should be noted that MicroStation iGeoGraphics interprets the two representations (i.e. 3D polygons and 3D multipolygon) in a different manner. In the first case the building is visually one object, but practically, it is a set of individual polygons (Figure 4, left). The entire building can be selected only by placing a fence around all the polygons. In the second case, the building is one 'group', i.e. a single click of the mouse will highlight the entire building (Figure 4, right). In order to edit the object, however, the group has to be 'dropped' into constructing individual polygons. To send the changes back to the database, grouping of the objects will be again required. Otherwise, the object will be considered a set of several new polygons.

3.2 ArcMap and ArcScene, ESRI

(ESRI, 2002) software (ArcMap (for 2D) and ArcScene (for 3D), both part of the complete package ArcGIS) is able to visualise data that is stored as a sdo_geometry type in Oracle with ArcSDE. ArcSDE is a gateway that facilitates managing spatial data in a DBMS (IBM DB2, IBM Informix, Microsoft SQL Server, and Oracle). Originally ArcSDE was developed for the SDE binary format, which is a format for spatial data types in the DBMS developed by ESRI. Since spatial data types have become available in DBMSs, ArcSDE now also supports the spatial data types of Oracle.

There are two methods to access data stored in a DBMS via ArcSDE:

1. via SDE client/server software
2. via 'direct connect', which does not use the SDE server, but only the SDE client software which is part of ArcGIS.

For the user who visualises the data both connections work similar. The difference is the way the connection is defined and how the connection works behind the GUI. The 'direct connect' makes direct connection to the DBMS without using the ArcSDE application server. On the other hand, for the 'direct connect' one needs Oracle client software (Net8), which is not needed for the ArcSDE connection. According to the manual, the 'direct connect' is easier to install and maintain. However, for this connection one still needs the tables of the user 'sde' in Oracle (the ArcSDE system tables) in combination with the Oracle Spatial metadata tables. An advantage of the 'direct connect' is that you do not need an ArcSDE licence to visualise data, in contrast with the SDE client/server connection.

The steps to visualise data organised in the DBMS in ArcGIS are:

1. insert metadata in the Oracle Spatial metadata tables
2. register the table that contains the geometry in the SDE system tables
3. define the DBMS connection in ArcCatalog (the ArcGIS program for management of GIS-layers)
4. obtain the data in ArcMap or ArcScene

Step 2: Registering the table containing geometries

For both connections the table containing geometries needs to be registered as a sde-layer. The registration of a table 'test2d' containing line features in a geometry column 'shape' and a primary key in the column 'ID' is registered as follows:

```
sdelayer -o register -l test2d,shape -k SDO_GEOMETRY -e l -u stoter -p password@database_name -i
sde:oracle9i -c id -C USER
```

For 3D information, the element type should be polygons in 3D, and also multipolygons need to be supported. This is handled by the –e a3+ option (instead of –e l). Furthermore a keyword is needed that is available in the sde.dbtune table to describe the right dimensions and tolerances, for example –k TEST3D (instead of –k SDO_GEOMETRY, which is the default).

The registration of a sde-layer also works without an ArcSDE license and can be performed both on the client and the server. The registration only edits the sde tables in Oracle. In these tables ArcSDE maintains the information of the geometry tables (dimension, spatial data types, geometry column). When the layer is 'unregistered' (with the sdelayer and the sdetable command) sometimes the Oracle metadata table are updated (i.e. entity for the table with geometry is deleted): this should not be the case since ArcSDE should only edit tables belonging to ArcSDE, and not tables that also exist without ArcSDE. The influence of a spatial index is also not clear. Without a spatial index a layer can be registered without any problems, however layers without indexes cannot be visualised. Another thing that is not clear is why ArcSDE does not only use the Oracle Spatial metadata table for the dimension and the tolerances of the data. ArcSDE uses a sde-table (sde.dbtune) in which keywords are stored (the –k option in the sde-layer command) together with dimensions and tolerances of the data.

Step 3: define DBMS connection in ArcCatalog
In ArcCAtalog both DBMS-connections need to be added: 'add spatial database connection'. The connections are defined with the name of the machine on which the database is stored, the name of the database, the user and the password of the user. The difference between the 'direct connect' and the SDE client/server connection is the service which specifies the type of connection: 'sde:oracle' for the 'direct connect' and the port number for the SDE client/server connection (usually 5151).

Step 4: Obtain data in ArcMap or ArcScene
The spatial data stored in Oracle can now be visualised in ArcMap or ArcScene by means of the defined database connections. With the 'add data' option all tables that are accessible to the user are checked for geometries columns (also tables that are not registered with sde). This means that also tables owned by other users who have granted select privileges are checked. This is not optimal because for this query different sde tables need to be queried, which is time consuming. A more optimal option would be to just query the Oracle Spatial metadata table of the specific user. This is only one table and in this way the user can choose which data should be available for ArcGIS.

The available layers are shown in the 'add data' dialog-window. The icons preceding the table-names show if the layer has been registered in ArcSDE and has therefore a geometry column that can be visualised in ArcScene or ArcMap (see figure 5). Also the data type (point, line or polygon) are indicated by this icon.

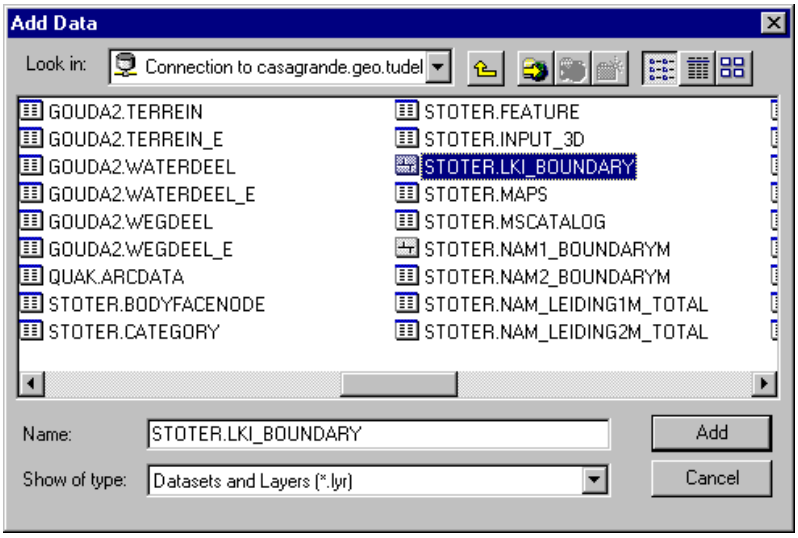


Figure 5: Dialog window in ArcMAP showing the tables stored in Oracle. Note that the tables of the users stoter, quak and gouda2 are all shown

Findings of visualising spatial data stored in Oracle in ArcGIS

2D data that is stored as an sdo_geometry type can be visualised in ArcGIS with both connections, without modifying the geometry tables (e.g. adding columns) although there are some minor problems. A table cannot contain more than one geometry column (for example when a bounding box is stored together with the geometries in one table). Also only one geometry type per column is possible (so lines and polygons cannot be stored in the sdo_geometry column of the same table). When there is a geometry stored in the table that is not valid according to Oracle it might happen that none of the geometries in the table can be visualised. A last remark is that the primary key should be of type number(38) to avoid visualisation problems. This solution (adding primary key of type number(38)) is not straightforward and for an ignorant user hard to find in the manual.

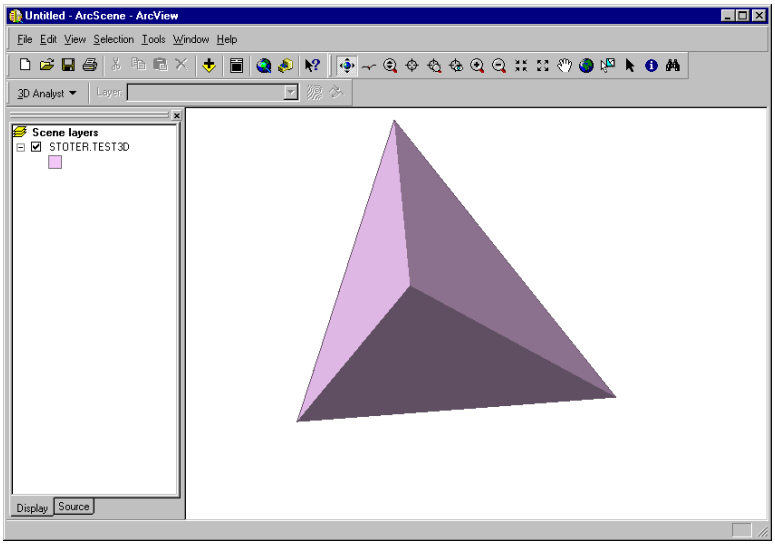


Figure 6: A tetrahedron defined by four separate polygons in Oracle, visualised in ArcScene.

Concerning 3D data, experiences showed that the z-values that are stored in Oracle are recognized by ArcScene. Therefore it is possible to visualise 3D polygons (see figure 6), although it was not possible to visualise the multipolygon variant of the tetrahedron in figure 6. Another problem is that 'vertical'

polygons (polygons that are perpendicular with the x,y plane) are not supported. This is a major problem in urban modelling since vertical polygons are important when modelling buildings.

3.3 Virtual Reality Modelling Language (VRML)

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 a DBMS can be visualised and queried with VRML. For the conversion to VRML we use spatial data that is topologically structured in the DBMS according to the Simplified Spatial Model (SSM, Zlatanova 2000). The reason for this is that the IndexFacedSet of VRML that describes a polyhedron, is similar to the SSM (a body is described by the faces it consists, the faces are defined by references to the nodes, and the coordinates of the nodes are stored only once).

In this section, we examine three methods to view and query 3D data that is stored in a DBMS with VRML. The first method is to use VRML both for the spatial data and the attribute data, the second method is to use VRML for the spatial data and HTML for the attribute data. The last method is to create VRML or HTML (or both) on the fly with respect to the result of the query (text or geometry).

Using VRML for spatial data and attribute data

In the first case, only one VRML file is created, which contains both attributes and geometries. The attributes become visible (as texts in the VRML browser) on 'mouse-click' or 'mouse-on' the object of interest. Figure 7 shows an example of viewing the attributes of an object (a building, represented as a cube). The text becomes visible when the user places the cursor on the building.

Since a VR browser is not a complete GUI, e.g. the point-and-click operation is not a responsibility of the browser, the interaction has to be explicitly described in the VRML. This can be organised by two additional VRML nodes. First, a particular *sensor* (e.g. TouchSensor) has to be attached to the object (a Shape), which will monitor whether the cursor interacts with the object. Second, a billboard node has to be introduced to visualise the attributes in text format. In our example, we have designed a new PROTO node. The node is basically a TouchSensor extended with a javascript code (included in the VRML file), which controls the text that is visualised (attribute information). The code provides a link between the attributes and the geometry. The VRML code for the example of Figure 7 is listed bellow:

```
PROTO TOUCH [                                                    -----definition TouchSensor
sensor
   field       SFInt32        object 0
   eventOut    MFString       string_changed
            ]
{
   DEF SENS TouchSensor {}
   DEF NODE Script {
      url "javascript:                                          -----definition of javascript
         function set_boolean (bool)
             {
              if ((bool == true)&&(object == 30))
                        string_changed [0] = 'BUILDING /34';     ----object with id=30 and its
                                                                 ----attribute info "building /34"
              if ((bool == false)||(object == 0))
                        string_changed [0] = '';}"
   eventIn SFBool set_boolean
   eventOut MFString string_changed IS string_changed
   field SFInt32 object IS object
   }
ROUTE SENS.isOver TO NODE.set_boolean
}#TOUCH                                                           ----end definition Thouch sensor

Transform {
    translation 0 0 -30
    children [
       DEF Box30 TOUCH {
         object 30}
       Shape {
```

```
        appearance Appearance {
            material Material {
            diffuseColor 0.60 0 0}}
        geometry Box {                                          ----geometry of the object
            size 4 4 4}}
    Transform {
         translation 4 4 4
         children[                                              ----billboard to visualise the
                                                                ----attributes
              Billboard {
                  children [
                      Shape {
                            appearance Appearance {
                                material Material {
                                diffuseColor 0 0 0}}
                            geometry DEF TEXT30 Text {          ----attributes
                                length [5,120]}}]
                  axisOfRotation 0.0 1.0 0.0}]}]}
```

ROUTE Box30.string_changed TO TEXT30.string_changed

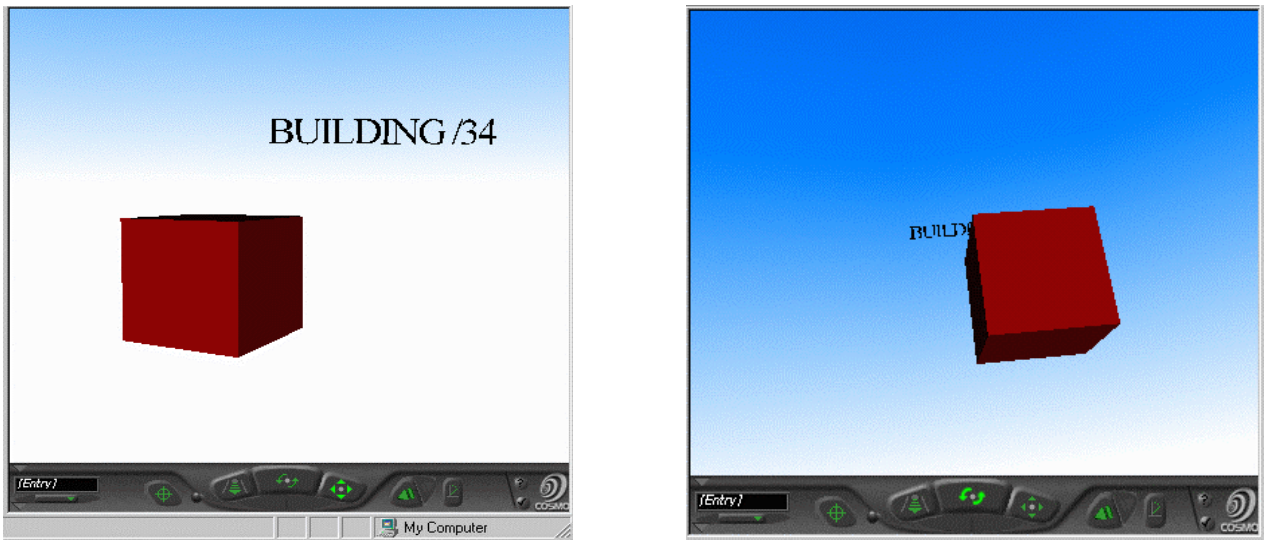


Figure 7: VRML containing geometries and attributes

A java program converts the attribute and geographical information to the VRML file. The protonode can contain more than one object. The major drawbacks of this approach are related to the size of the file and the visualisation of the text. Depending on the size of attribute information for visualisation and the number of spatial objects with attributes, the VRML file can become between 5 to 10 times larger compared to the VRML containing only the geometries. In case of large VRML models this can result in longer time needed to create the VRML file and display it on the screen. Since the attributes are visualised on a billboard (i.e. another 3D shape in VRML), they may be occluded by the object (Figure 7, right) or even invisible (if a user observes the billboard shape from a direction perpendicular to the axis of rotation of the billboard).

VRML for spatial data and HTML for attribute data

The problem of getting very large (and consequently slow-working) VRML files when including the attribute data in the VRML files, can be overcome by using HTML files for the attribute data. A VRML file is generated containing the geometries in the Oracle table. Every object in the VRML file is represented as an IndexedFaceSet and an anchor is attached to very object. An anchor is used to link an url to an object. The anchor contains fields specifying the anchor.

For every object a single HTML file is generated containing the attributes of the object that are stored in the Oracle table. When you click on the object the corresponding url (which indicates the specific

HTML file) is opened in a frame defined in the parameter field of the anchor with the keyword "target=<frame>" (see figure 8).

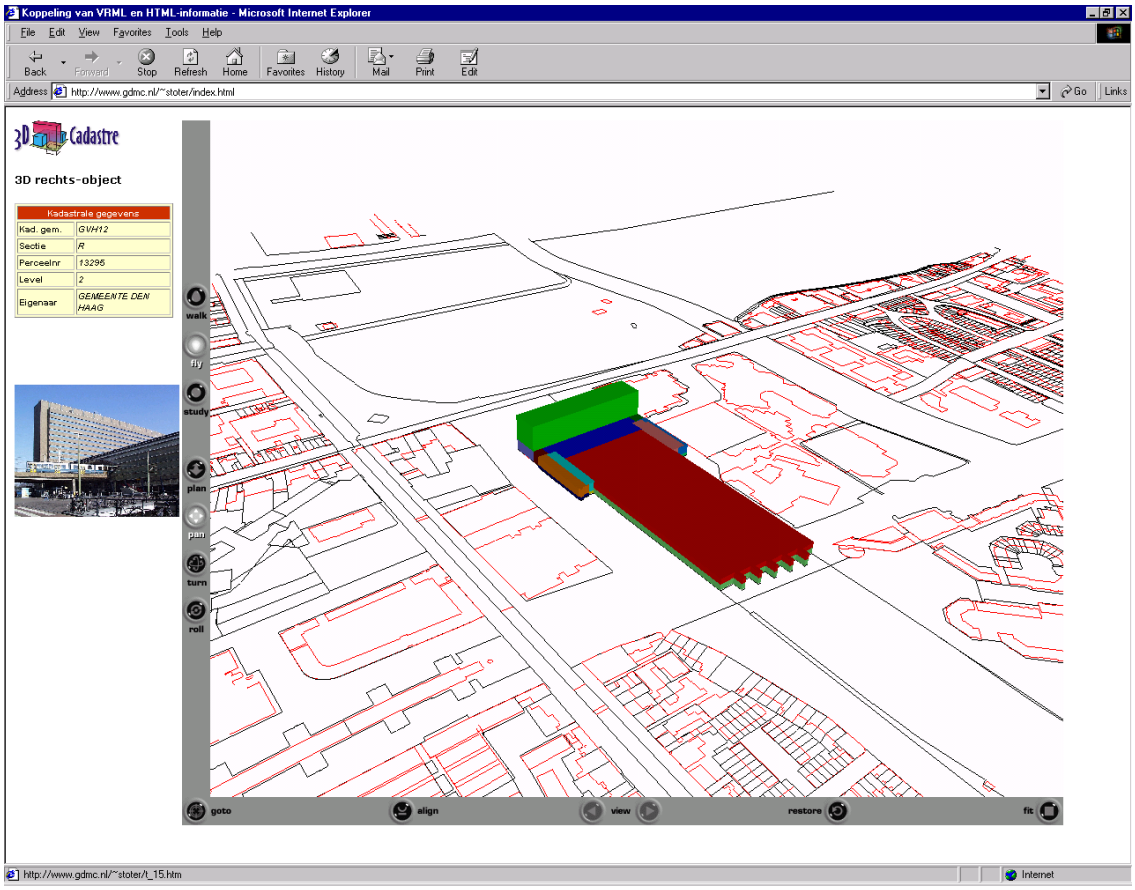


Figure 8: Querying objects in a VRML file with attributes stored in HTML files. The files are generated on top of the database.

A conversion tool has been written in java which converts the geometry of objects to one VRML file and which converts the accompanying attribute data to separate HTML files for every object. The urls of the HTML files are specified for every object in the VRML file. For one object this looks as follows (with the attributes stored in t_1.htm):

```
Anchor {
      parameter "target=leftframe"
      description "Test 3D Cadastre"
      url "t_1.htm"
      children [
      Shape {
        appearance Appearance {
          material Material {
            diffuseColor 1.0 0.0 0.0
            specularColor 0.8 0.8 0.8
            transparency 0.0
          }
        }
        geometry IndexedFaceSet {
          convex FALSE
          solid FALSE
          coord Coordinate {
            point [
              82.14005  455.38986       0.0,
              82.1244   455.37831       0.0,
              82.10310000000001 455.36196       0.0,
              82.03691  455.31116       0.0,
              ------------------------------   82.16185  455.36911       0.012,
              82.15977000000001 455.37179       0.012,
              82.14372  455.39257000000003      0.012,
```

```
  ]
}
coordIndex [
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, -1,
   0, 11, 12, 1, 0, -1,
   1, 12, 13, 2, 1, -1,
   2, 13, 14, 3, 2, -1,
   3, 14, 15, 4, 3, -1,
   4, 15, 16, 5, 4, -1,
   5, 16, 17, 6, 5, -1,
   6, 17, 18, 7, 6, -1,
   7, 18, 19, 8, 7, -1,
   8, 19, 20, 9, 8, -1,
   9, 20, 21, 10, 9, -1,
   10, 21, 11, 0, 10, -1,
   11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 11, -1,

                    ]
                           }
 }
    ]
    }
```

Currently the conversion is not performed on the fly, which will be the next step as will be described in the next section.

VRML and/or HTML on the fly

The last approach is a combination of HTML and VRML that are created on the fly dependent on the query of the user (attribute data, geographical data or both). Zlatanova 2000 discusses an example of such a system. The prototype was realised in another DBMS, i.e. MySQL (MySQL, 2002) (does not support spatial data types), as the attribute and geometric information was organised in a topological data structure (i.e. SSM). The information about any object (e.g. a building in a town) can be collected by a series of user actions routing to java and CGI scripts. The scripts are included in the body of a starting file (VRML or HTML). Thus the next action can be either a new query to the database (on the server) or a query of the already delivered VRML file (on the client station). For example, pointing to the object with the mouse, the user can get specific information about the building stored in the VRML file, e.g. the name of the institution using it (see also above). The first click on the building may deliver a new HTML form with optional menu (e.g. the year of construction, the name of the architect). The second click on the same building may visualise an HTML form where an SQL statement can be formulated. A click on the send button (within this SQL form) may execute a spatial query (e.g. 'show the neighbouring houses'), which will be visualised in a VRML file. Thus, one pointing and three following actions by the user allow query of semantic and spatial information about a building. The same mechanism can be applied in a more sophisticated way to explore complex spatial objects, which can be seen as an approach for introducing dynamics.

In the prototype system, CGI scripts take decisions what the resulting event has to be. The decision is taken during the dynamic creation of the files. Clearly, the functionality of such a system relies on the CGI scripts, which may lead to the following problems:

- a drastic increase in CGI scripts in the case of complex sequential queries, which may complicate their management and slow down the performance on the server
- sophisticated algorithms that may become time-expensive with respect to the dynamic composition of VRML documents
- long VRML documents, which may cause negative effects in two directions: occupation of the server and long delivery time.

Further experiments are needed toward both, i.e. storing some of needed scripts together with the objects in the DBMS or performing some of the operations on the client side.

4. Conclusion

What is the state–of-the-art when storing 3D objects in a DBMS and when visualising these data in GIS and CAD front-ends? What is possible from a technological point of view? In this paper we looked at Oracle Spatial 9i to store 3D objects, and we examined three front ends for visualising the data.

In current DBMSs 3D primitives are not supported, which means 3D objects have to be modelled by means of polygons defined in 3D. Future research is needed for implementing a 3D primitive in DBMSs (see also Arens et al, 2003).

In addition visualising 3D objects stored in a DBMS is not always straightforward. With Bentley (MicroStation Geographics) it is possible to visualise 3D objects rather easily, but if you want to give the objects separate colours or if you want to perform an 'identify' on the objects you have to create features and attach features to the objects, which is not simple. This process is also needed when posting data to the DBMS. The main problem with ESRI ArcGIS is that it does not support vertical polygons. We reported this problem to ESRI and it is currently being solved by ESRI with a workaround. VRML offers the best possibilities to visualise 3D data stored in a DBMS. At the moment the VRML-HTML method seems the most optimal one, since it performs the best. However, the process of converting Oracle tables to VRML and HTML files could be improved when doing this on the fly, which can be done with the VRML file that evokes CGI scripts. This could be a solution in the future, when performance becomes a less critical issue because of the increasing power of computers. This solution has not yet been implemented on Oracle Spatial, but will be focused at in future research. X3D, the successor of VRML will also be taken into account in future research.

References

Arens, C., J.E. Stoter, and P.J.M. van Oosterom (2003), Modelling 3D spatial objects in a GeoDBMS using a 3D primitive, under review for AGILE 2003.

Ames A.L., D.R. Nadeau and J.L. Moreland (1997), The VRML 2.0 Sourcebook

Bentley (2002), Bentley MicroStation GeoGraphics iSpatial edition (2002). URL: http://www.bentley.com/products/geographics/

ESRI (2002), ArcGIS, URL: http://www.esri.com

IBM (2000): IBM DB2 Spatial Extender User's Guide and Reference. special web release edition.

Informix (2000): Informix Spatial DataBlade Module User's Guide. Part no. 000-6868, URL: www.informix.com, 2002

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

Laser-Scan Radius Topology (2002), URL: http://www.radius.laser-scan.com/

MapInfo (2002), URL: www.mapinfo.com, 2002

MySQL (2002), URL: www.mysql.com, 2002

Molenaar, M. (1990): A formal data structure for 3D vector maps, in: Proceedings of EGIS'90, Vol. 2, Amsterdam, The Netherlands, pp. 770-781

OGC (2001(1)): OpenGIS specifications, 2001, available on http://www.opengis.org/techno/specs.htm.

OGC (2001(2)): Request Number 12, Geometry Working Group, A request for Proposals: OpenGIS Feature Geometry, September, 2001.

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

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

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

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

PostGIS (2002), URL: postgis.refractions.net, 2002

Stoter, J.E. and P.J.M. van Oosterom (2002): Incorporating 3D geo-objects into a 2D geo-DBMS, FIG, ACSM/ASPRS, April 19-26 2002, Washington D.C. USA

Stoter, J.E. and S. Zlatanova (2002): 3D large scale modelling, Workshop on 3Dcadastres and Large scale Modelling, organised during UDMS 2002, October 2002, Prague, Tsjech Republic

Tempfli, K. (1998): 3D topographic mapping for urban GIS, ITC Journal 3/4, pp. 181-190

Zlatanova, S. (2000): 3D GIS for urban development, PhD thesis, TUGraz, Austria, ITC publication, The Netherlands, 222 p.

Zlatanova, S., A. Rahman, M. Pilouk, (2002): 3D GIS: current status and perspectives, in: Proceedings of ISPRS, 8-12 July, Ottawa, Canada, CDROM, 8p.