Revised paper for AGDM'94, Delft, The Netherlands, 12–14 September'94

INTEGRATED 3D MODELLING WITHIN A GIS

PETER VAN OOSTEROM, WILCO VERTEGAAL, MARCEL VAN HEKKEN
TNO Physics and Electronics Laboratory,
P.O. Box 96864, 2509 JG The Hague, The Netherlands,
Email: oosterom@fel.tno.nl.

and

TOM VIJLBRIEF
TNO Human Factors Research Institute,
P.O. Box 23, 3769 ZG Soesterberg, The Netherlands.
Email: tom@tm.tno.nl

ABSTRACT. This paper describes the architecture of a truly integrated 3D GIS. The system has two main components: an extensible database, Postgres, and an extensible geographic front-end, GEO^{++}. The database is extended with 3D geometric data types, functions, and a 3D access method. The implementation of the 3D visualization is based on the graphics standard PEX. The resulting system is again extensible and can be used as a framework for modelling and analyzing many kinds of 3D applications: geology, air-space management, terrain modelling, etc.

1. INTRODUCTION

The relational approach has proven to be very effective for developing various types of information systems. Methodologies and advanced tools are available for designing and implementing relational data models and associated applications; e.g. CASE and 4GL tools. However, the relational databases are not capable of handling spatial information, as encountered in a Geographic Information System (GIS), efficiently. We have therefore designed a 2D spatial extension (van Oosterom & Vijlbrief, 1991) and implemented this within the Postgres eRDBMS[1]. Together with a geographic front-end, called GEO^{++}(Vijlbrief & van Oosterom, 1992), this system is a complete GIS.

For certain applications the current capabilities of GEO^{++} are not sufficient; a few examples are:

1. *terrain modelling* (Douglas, 1986; Gold, 1984; Gold, Charters, & Ramsden, 1977):

[1]eRDBMS stands for extensible Relational DataBase Management System (Stonebraker & Rowe, 1986; Stonebraker, Rowe, & Hirohama, 1990). Note that the functionality in Postgres is similar to the future SQL3 standard (Robinson & Tom, 1993), as used in for example *Illustra*, the commercial successor of the Postgres research DBMS. Though we use Postgres examples in this paper, the method is general and can be applied in combination with any eRDBMS.

the GIS contains elevation data of the Earth surface (2.5D data model[2]), which can be displayed and analyzed; e.g. compute which areas are visible from a certain location;

2. *geology* (Barrera & Vázquez-Gómez, 1989; Calkins, Xia, & Guan, 1993; Carlson, 1987; Jones, 1989): the layers below the Earth surface (rock, sand, oil, water, ...) have to be modelled; e.g. for exploration by an oil company;

3. *air-space management*: the space above the Earth surface is subdivided into 3D corridors and is allocated to airplanes;

4. *fishery*: monitoring or simulating the fish population and movements (lat/long position and depth) in the Oceans;

5. *cadastre*: the cadastral boundaries in tall buildings can be true 3D; e.g. in the city of Amsterdam;

6. *cartography* (Kraak, 1988, 1992): certain cartographic techniques require 3D capabilities to visualize thematic attribute information; e.g. a PRISM map.

It is clear that we need a 3D GIS for these types of applications. There are a number of different systems available for this purpose, but they usually have one or more of the following drawbacks:

- they are limited to 2.5D data models;
- they are separate modules, hard to integrate with the GIS;
- if they are true 3D, then they are usually quite specific programs: not general purpose GIS based on a DBMS.

Therefore, we decided to design and implement our own 3D GIS-modelling environment based on the GEO^{++} system. Section 2 describes the overall architecture of this system. The subsequent Sections 3 and 4 give additional details w.r.t. the Abstract Data Types (ADTs) for 3D geometric primitives and w.r.t. the topological relationships in 3D-space. Examples of complex 3D spatial analyses are listed in Section 5. The Graphic User Interface (GUI) and visualization are the topics of Section 6. The paper is concluded in Section 7, which also contains possible future work.

2. 3D GIS ARCHITECTURE

As described in (Vijlbrief & van Oosterom, 1992), an *integrated* GIS-architecture has several advantages over a *dual* or *layered* GIS-architecture: consistency and efficiency. Both thematic and spatial data are integrated within objects (or features) and stored in the same DBMS. Further, reality as captured in the data model is translated directly

[2]In a 2.5D or Digital Terrain Model (DTM), for every location (x, y), an unique elevation z can be obtained. There are three usual ways of representing these data: contours (Watson, 1992) (in analogy with normal paper maps), TIN (Peuker, Fowler, Little, & Mark, 1976; Floriani, 1987) (Triangular Irregular Network), and grid (Mark, 1978) (with height values at regular distances; as can be found in DLMS-DTED (DMA, 1986)).

to the physical DBMS in the integrated architecture. The advantages of the integrated architecture are valid for both 2D and 3D modelling.

Weibel (Weibel, 1993) defines three levels of integration of the Digital Terrain Model (DTM)-module with the GIS:

1. *loose coupling:* GIS and DTM are separate and only exchange data sets,
2. *functional integration:* DTM-functions can be used through GIS-interface. Further, files with a common data format are used, and
3. *full database integration:* DTM and GIS are based on same DBMS and the user interface is also integrated.

Since the advantages of full database integration are also clear for true 3D-applications (not only for DTMs), we decided to follow this approach, in which both 3D data and functions are available within the same DBMS and GIS environment.

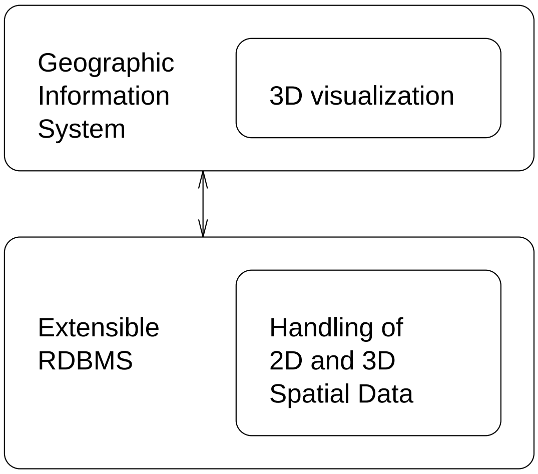

Figure 1: Architecture of an integrated 3D GIS based on an eRDBMS.

The system architecture consists of major main components: 1. an extensible RDBMS, in our case Postgres, and 2. a geographic front-end, GEO++. The 3D geometric data types and functions are implemented as ADTs within the database; see the next section. Further, 3D spatial indexing is added in order to efficiently answer proximity queries. The geographic front-end is extended with 3D visualization tools. Thus we obtain an open, but integrated 3D GIS environment; see Figure 1.

3. 3D ABSTRACT DATA TYPES

The Abstract Data Types (ADTs) for the 3D Geometric Primitives are comparable to the basic 2D ADTs (van Oosterom & Vijlbrief, 1991). The 3D equivalents are: POINT3, POLYLINE3, and POLYGON3. Further, due to the nature of the 3D-space, one new type is needed: POLYHEDRON3, a true volume or body. In comparison with the 2D ADTs, there are a few additional operators; e.g. gradient and azimuth of a POLYLINE3/ POLYGON3, volume and centroid of a POLYHEDRON3 (Lee & Requicha, 1982).

Instead of 2D spatial indexing (quadtree, R-tree, etc.), 3D spatial indexing is required to enable efficient query processing. It was decided to use a 3D variant of the R-tree

(Guttman, 1984). Supporting this purpose, the type BOX3 was introduced together with an R-tree operator class for this type. The 3D data types are listed in Table 1 (Appendix A) and some examples are depicted in Figure 2.

These ADTs are sufficient for a lot of simple spatial analyses (van Oosterom & Vijlbrief, 1994). For example: "Give all regions with a surface gradient greater than 5% and that are at an elevation of 1000 meters or higher" or "Give all sand and clay soils whose surface orientation is *south*". Subsection 3.1 describes the data types in more detail and Subsection 3.2 describes the associated operators and functions.

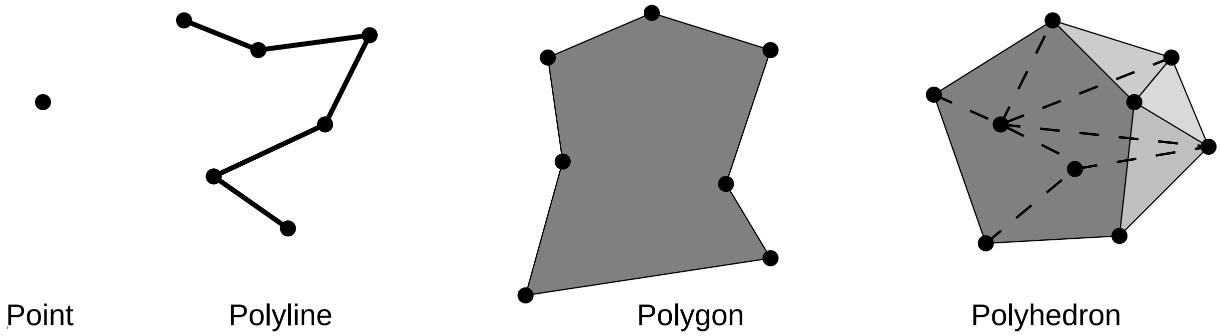

Figure 2: Point, polyline, polygon, and polyhedron data types in 3D.

3.1 Internal/external representation

In addition to 'real' user functions, auxiliary functions are needed to read a user's input, and to show the contents of objects as output. The output should be formatted in a logical way, so that the user can understand what it represents. For example, the Postgres functions **Point3_in()** and **Point3_out()** convert, respectively, a POINT3 from and to an ASCII string. This string has the following format: three floating point numbers, separated by commas, and surrounded by brackets. For instance, the string `(1.0,2.5,-3.7)` would represent a POINT3 with an x coordinate of 1.0, a y coordinate of 2.5 and a z coordinate of -3.7.

For POLYLINE3 and POLYGON3, the formats are the same: `(nr_of_points: x1,y1,z1, x2,y2,z2, ...)`. A problem that arises with polygons is that the points that form the polygon boundary should all be in one plane. Furthermore, there should be no crossing edges. A function, named **Check3Pgn()**, has been provided to test the validity of a POLYGON3 object.

The POLYHEDRON3 data type is the most complicated of the 3D Postgres data structures. A polyhedron consists of a number of faces (polygons) that form the boundary of the object. Since a polyhedron can have any number of faces (at least 4), and all polygons can consist of any number of points larger than 2, a variable length array of variable length arrays is used to store this information. Furthermore, some representation possibilities should be considered. It may be convenient to store not only the faces, but also the points that they are built from, and/or the edges of the faces.

As for the representation problem, the decision has been made to store both the points and the faces of a polyhedron, but not the edges. This is because the edges are not really useful. They could be used to avoid redundancy (otherwise edges are implicitly defined twice) and to check if a polyhedron is valid[3]. Therefore, it is better to compute the edges only when they are really needed. Storing the points, however, does have some benefits. In the first place, storage space is saved when the points are stored as POINT3s, and the faces consist of indexes to these, since each POINT3 consists of three `float`s, whereas an index needs only one `int`. Beyond this, it seems more logical to store a point only once and make a number of references to it, than to store this point every time it is a corner of a face. It makes the coherence between the faces clearer.

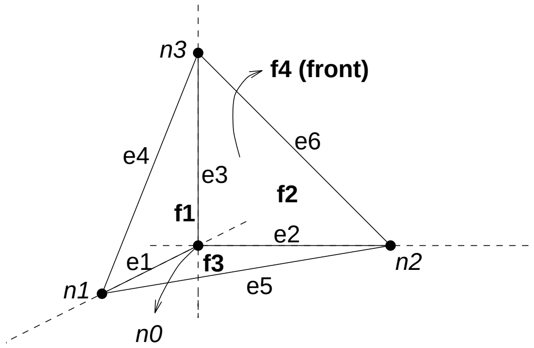

Figure 3: A simple 3D polyhedron with four faces.

The format of input and output character strings has been chosen as follows: $((n : p_1x, p_1y, p_1z, ..., p_nx, p_ny, p_nz); (m : (f\_1 : v_{1,1}, ..., v_{1,f\_1}), ..., (f\_m : v_{m,1}, ..., v_{m,f\_m}))$ where n is the number of points, m is the number of faces, and f_i is the number of point indexes in face i. Between parentheses, a list of points and a list of faces are enclosed. They are separated by a semicolon. The point list looks exactly like the POLYLINE3 and POLYGON3 format: an opening parenthesis, the number of points, a colon, all coordinates, and a closing parenthesis. The list of faces, which is also enclosed by parentheses, consists of the number of faces, a colon, and all faces, separated by commas. Each face starts with a parenthesis, followed by the number of point indexes, a colon, the indexes (separated by commas), and a closing parenthesis. For example, the polyhedron in Figure 3 would be represented by the string `((4:0,0,0, 1,0,0, 0,1,0, 0,0,1); (4:(3:0,1,2), (3:1,2,3), (3:2,3,0), (3:3,0,1)))`.

The most important reason for including a BOX3 data type is to have an economic way to access the 3D objects in a GIS using a 3D-rtree. For this purpose, the minimum and maximum x, y, and z coordinates of an object are needed. Another reason is that there are a number of functions and operators that do the same job on all or a number of the four basic 3D data types. Instead of providing these functions for all data types, they are now first converted to BOX3 and then a BOX3-function is used. Examples are minimum and maximum x, y, and z of an object. Further, intersections and unions of two objects can be approximated by using the BOX3 type.

3.2 Functions and Operators

In this subsection the functions and operators that are of interest to GIS-applications are described. Functions are the routines that actually work on the data, while operators are symbolic names given to them, that can be used within Postgres instead of

[3] A polyhedron is valid if every edge of any face is shared with exactly one other face, if no faces cross or overlap, and if all faces are valid polygons.

the full function name. For instance, a function called **Distance3PntPnt** (calculating the distance between two points) could have an operator defined as $< - >$. Instead of **Distance3PntPnt(p1, p2)** the expression **p1**$< - >$**p2** could be used, which would have the same result. These operators are also used for creating queries by means of the graphical query editor in GEO++. Further, query optimization techniques are associated with operators.

The functions and operators are subdivided into two categories: those that are explicitly suited for one data type (see Tables 2, 3, 4, and 5 in Appendix A), and those that are more general and applicable to almost any combination of two objects. The general functions can be subdivided into: the distance functions (see Table 6) and the topology functions (see next section and Table 7). In the GIS context, distances between various objects are often required. The only case where a unique distance can be obtained is with point–point operands. In all other cases one can get the minimal or maximal distance between the two objects. All distance functions return value of type `float`.

4. TOPOLOGY

Topology (Pullar & Egenhofer, 1988; Egenhofer & Franzosa, 1991; de Hoop & van Oosterom, 1992) is always an interesting aspect in a GIS and this is certainly also true in 3D. In this section two aspects are considered:

1. *classification of topology* relationships in 3D: five relationships (in, touch, cross, overlap, and disjoint) together with their boundary operators can be proven to be sufficient. This is in analogy with the 2D situation (Clementini, Felice, & van Oosterom, 1993), with only a slight modification of one definition. More details are given in Subsection 4.1.

2. *storage of topology* relationships (Molenaar, 1990; Pigot, 1992) in the data model: The POLYHEDRON3 example in Figure 4 shows two alternatives for representing the 3D body of Figure 3: either 1. store everything within the type itself (all coordinates of edges and faces) or 2. use "topological" references to faces, which in turn may contain references to edges[4]. The latter is very useful in the situation of a partitioning, in which space is subdivided into a set of non-overlapping regions. A similar solution for storing 2D topology information in a 2D GIS has been described in (van Oosterom & Vijlbrief, 1994). The complete modelling freedom is with the users of GEO++, because they can choose which model is best suited for their purpose.

4.1 Formal Topological Relationships

The spatial relationships in the previous section often are not adequate to describe the relationship between two objects with respect to their relative configuration in space. It may be important to know if two objects share a piece of space, and in what way they are connected. In (Clementini *et al.*, 1993), it is proven that only five separate topological relationships are needed to describe all possible relationships between any combination of

[4]Note that in this example the topological structure may be taken one step further (add the class nodes with a POINT3 attribute, and replace POLYLINE3 attribute in the class edges by references to nodes) or one step back (drop the class edges, and replace references to edges by POLYGON3 in the class faces).

Figure 4: Data models for 3D bodies with and without topology

```
/* without topology */
create rocks(rock_id=int4, owner=text, location=POLYHEDRON3)
append rocks(rock_id=1, owner=Shell,
   location="((4:0,0,0, 1,0,0, 0,1,0, 0,0,1);
   (4:(3:0,1,3),(3:0,2,3),(3:0,1,2),(3:1,2,3)))"::POLYHEDRON3)
    /* First the nodes, then the faces referring to these nodes */

/* with topology */
create edges(edge_id=int4, line=POLYLINE3)
create faces(face_id=int4, edge_ids=int4[])
create rocks(rock_id=int4, face_ids=int4[])
/* note that edge_ids and face_ids are both variable
   length arrays with references */

append edges(edge_id=1, line="(2:0,0,0, 1,0,0)"::POLYLINE3)
append edges(edge_id=2, line="(2:0,0,0, 0,1,0)"::POLYLINE3)
append edges(edge_id=3, line="(2:0,0,0, 0,0,1)"::POLYLINE3)
append edges(edge_id=4, line="(2:1,0,0, 0,0,1)"::POLYLINE3)
append edges(edge_id=5, line="(2:1,0,0, 0,1,0)"::POLYLINE3)
append edges(edge_id=6, line="(2:0,1,0, 0,0,1)"::POLYLINE3)

append faces(face_id=1, edge_ids={1,3,4})
append faces(face_id=2, edge_ids={2,3,6})
append faces(face_id=3, edge_ids={1,2,5})
append faces(face_id=4, edge_ids={4,5,6})

append rocks(rock_id=1, face_ids={1,2,3,4})
```

two objects from the point, polyline, and polygon types. When dropping one of the five relationships, some expressive power would be lost; adding more relationships would result in redundancy. Although the paper is about 2D objects, it almost completely covers the 3D situation, extending the relationships for use with polyhedrons without any additions and hardly any changes. For the 3D case, the five relationships are the following:

- The *touch* relationship:

  $$\langle \lambda_1, touch, \lambda_2 \rangle \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ = \emptyset) \wedge (\lambda_1 \cap \lambda_2 \neq \emptyset) \ ;$$

  (where the interior of an object, λ°, is defined as the difference between the object λ and its boundary $\partial\lambda$.);

- The *in* relation:

  $$\langle \lambda_1, in, \lambda_2 \rangle \Leftrightarrow (\lambda_1 \cap \lambda_2 = \lambda_1) \wedge (\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset) \ ;$$

- The *cross* relationship:

  $$\begin{aligned} \langle \lambda_1, cross, \lambda_2 \rangle \ \Leftrightarrow \ & (dim(\lambda_1^\circ \cap \lambda_2^\circ) < max(dim(\lambda_1^\circ), dim(\lambda_2^\circ))) \wedge \\ & (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2) \ ; \end{aligned}$$

- The *overlap* relationship:

$$\begin{aligned}\langle \lambda_1, overlap, \lambda_2 \rangle \;\;\Leftrightarrow\;\; & (dim(\lambda_1^\circ) = dim(\lambda_2^\circ) = dim(\lambda_1^\circ \cap \lambda_2^\circ)) \wedge \\ & (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)\ ;\end{aligned}$$

- The *disjoint* relation:

$$\langle \lambda_1, disjoint, \lambda_2 \rangle \Leftrightarrow \lambda_1 \cap \lambda_2 = \emptyset\ .$$

With respect to the definitions given in (Clementini *et al.*, 1993), one change was made. The original definition of the cross relation was as follows:

$$\begin{aligned}\langle \lambda_1, cross, \lambda_2 \rangle \;\;\Leftrightarrow\;\; & (dim(\lambda_1^\circ \cap \lambda_2^\circ) = max(dim(\lambda_1^\circ), dim(\lambda_2^\circ)) - 1) \wedge \\ & (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)\ .\end{aligned}$$

This no longer holds for 3D objects. For example, the shared area of a crossing line and plane can be a point in 3D, where in 2D situations it would always be a line (assuming that the objects share any area).

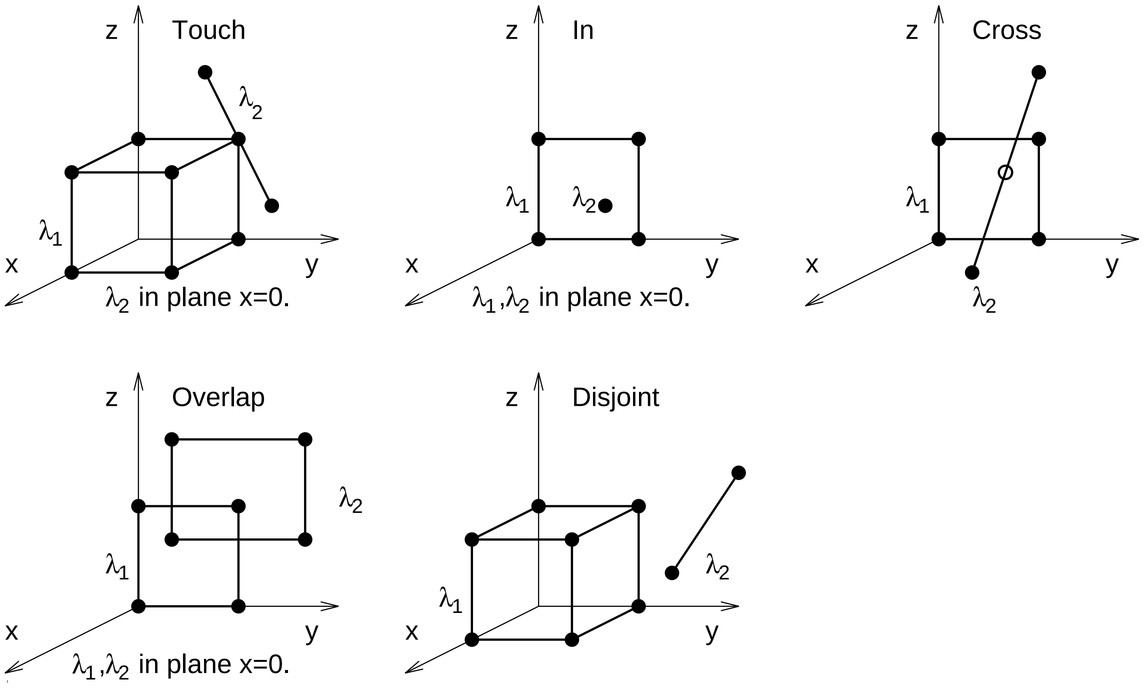

Figure 5: Examples of the five topological relationships in 3D.

With the definitions given above, a set of Boolena functions can be implemented, returning whether two objects *do* or *do not* meet the given relationship. These are given in Table 7. A sixth relation has been added, the *equal* relationship, which tests if two objects are equal. Although this could be derived from the *in* relation (two objects are equal if and only if the first object is in the second, and the second object is in the first), it is useful to create a separate function for this, which require less calculations than when using the *in* functions.

5. COMPLEX 3D SPATIAL ANALYSES

Simple analyses have already been described in Section 3. In contrast to simple analyses, complex analyses are usually done outside the DBMS. Some characteristics of complex analyses are that the computations are quite expensive and the input and output may have complex relationships. A few examples of 3D (or 2.5D) complex analyses are:

1. computation of visibility diagrams/ maps (Fisher, 1993; Floriani & Magillo, 1993; Katz, Overmars, & Sharir, 1992);
2. conversion 2.5D models, e.g. from TIN to contours (Watson, 1992);
3. 3D generalization (Ware & Jones, 1992; Jones, 1989; Floriani, 1989; Scarlatos & Pavlidis, 1991; Polis & McKeown, Jr., 1993; de Cambray, 1993), hierarchical/ multi-scale DTM's;
4. computation of watershed runoff and drainage basins;
5. finding the shortest path in a hilly terrain (Mitchel, Mount, & Papadimitriou, 1987; van Bemmelen, Quak, van Hekken, & van Oosterom, 1993);
6. compute a slice of a set of real 3D objects in order to display the internal structure; e.g. in geology.

Currently, only a few 3D analysis functions are available within GEO^{++}. However, the system is extensible and new functions can be integrated in a seamless manner within the 3D user interface of GEO^{++}; see the next section.

6. VISUALIZATION

As described in earlier papers (van Oosterom & Vijlbrief, 1991; Vijlbrief & van Oosterom, 1992), GEO^{++} is extensible. We have added several new data types before; e.g. raster and ARC2 (arc's and circles). The Postgres DBMS is extended with these ADTs implemented in C. The visualization in GEO^{++} of a new Postgres ADT is then achieved by defining a QueryShape, which is implemented in C^{++}. The core of GEO^{++} can remain unchanged, because the new type is dynamically loaded.

However, in 3D the case is quite different. The transfer from 3D-space to 2D-space, a mathematical projection, has to be defined in the "viewing pipeline" (Foley & van Dam, 1982; Singleton, 1986). The basic ET^{++} (Weinand, Gamma, & Marty, 1989) X11 toolkit, on which GEO^{++} is based, is not sufficient and has to be extended. This is done using PEX[5] (Gaskins, 1992; Rost, 1987), which is a standard 3D graphics environment on workstations; e.g. on our SparcStation under Solaris 2.3.

The 3D user interface of GEO^{++} (see Figure 6) is implemented using the extended version of ET^{++} and the BUILDER language[6]. Further, GEO^{++} itself has been modified and extended with several hooks for the 3D user interface:

[5]PEX is Phigs (ISO, 1989) 3D graphics within a X11 windows (Scheifler & Gettys, 1986).

[6]The BUILDER language (van Oosterom & Vijlbrief, 1994) can be used for efficiently developing user interfaces without "low level" programming. The BUILDER itself is also based on ET^{++}, which assures the uniform look.

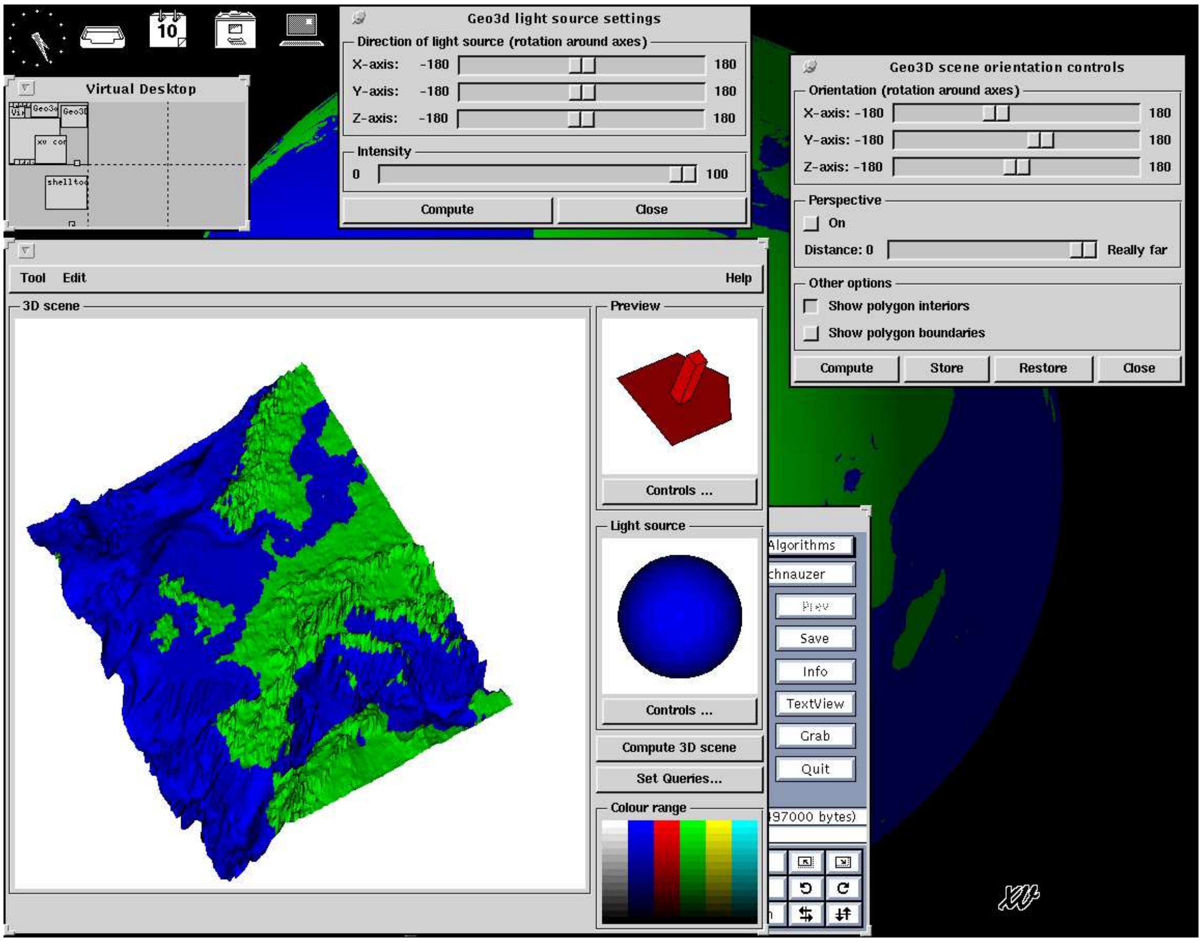

Figure 6: The 3D Geographic User Interface.

- Tuple Editor, Table Browser, and Query Composer are made available;
- Notifications of Area Changed / Query Changed in order to adjust the 3D display;
- Data model retrieval in order to obtain the specified classes with their attributes.

The 3D user interface has direct manipulation tools for setting the rotation (model orientation) and light source (shading). The *preview object* is designed so that it gives the user a good feeling of the current rotations (orientation); see Figure 7. Further, it is very simple, so it can be rotated in real time on displays without 3D graphics hardware. The user can rotate the scene by pointing at the top, bottom, left, or right part of the preview object: a *roll-and-pitch view*. Alternatively, the object can be manipulated through 3 sliders: for rotating around the x, y, and z axis.

For specifying the light source angle, a sphere shows in a consistent way from which direction the light illuminates it; see Figure 8. By using the sliders or roll-and-pitch view, the direction of the light source can be manipulated. Further, the user can select either parallel or perspective projections, and switch boundaries and interior display on/off (wire-frame or solid display); see Figure 6. More details w.r.t. the implementation of the 3D-user interface can be found in (Vertegaal, 1994).

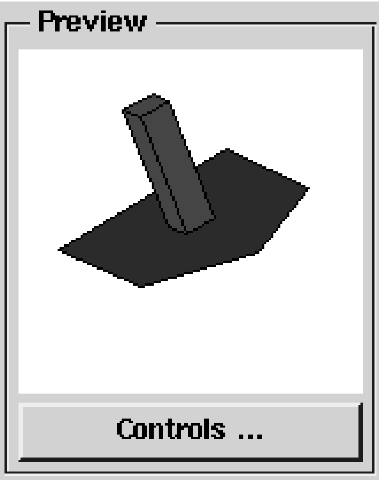

Figure 7: The preview object.

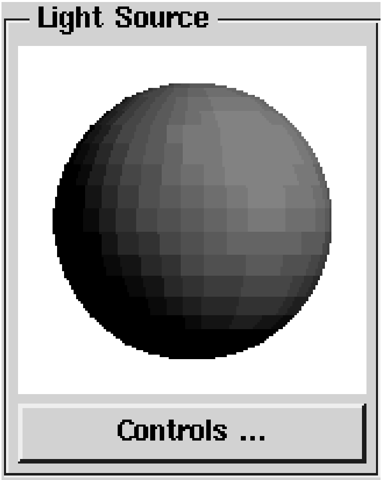

Figure 8: The reflecting sphere.

7. CONCLUSION AND FUTURE WORK

An integrated, extensible, and portable 3D GIS has been created. Note that this environment is also suitable for other 3D spatial applications, such as CAD (Computer Aided Design) (Mortenson, 1985; Mäntylä, 1988). The plans for future work will focus on GIS and include:

- coupling with virtual reality environment;
- implementation of more complex analysis functions;
- scientific visualization; e.g. of dynamic flows, or of real 3D bodies which are difficult to visualize (using transparent interiors);
- monitoring or simulating a 3D phenomenon, that is, also taking time into account;
- multi-scale DTM models supported by reactive data structures (van Oosterom, 1994);
- draping of Remotely Sensed data (or scanned paper maps) over a DTM;
- support for area patches and voxel data models.

Acknowledgements

Many valuable comments and suggestions on a preliminary version of this paper were made by Paul Strooper.

References

Barrera, R., & Vázquez-Gómez, J. (1989). A shortest path method for hierarchical terrain models. In *Auto-Carto 9*, pp. 156–163.

Calkins, H. W., Xia, F. F., & Guan, W. (1993). GIS based 3d segmentation for water quality modeling. In *GIS/LIS Proceedings, Minneapolis, MN*, pp. 92–101.

Carlson, E. (1987). Three-dimensional conceptual modeling of subsurface structures. In *Auto-Carto 8*, pp. 336–345.

Clementini, E., Felice, P. D., & van Oosterom, P. (1993). A small set of formal topological relationships suitable for end-user interaction. In *SSD'93: The Third International Symposium on Large Spatial Databases, Singapore*, pp. 277–295 Berlin. Springer-Verlag.

de Cambray, B. (1993). Three-dimensional (3d) modelling in a geographical database. In *Auto-Carto 11*, pp. 338–347.

de Hoop, S., & van Oosterom, P. (1992). Storage and manipulation of topology in postgres. In *Proceedings EGIS'92: Third European Conference on Geographical Information Systems*, pp. 1324–1336. EGIS Foundation.

DMA (1986). Product specifications for digital terrain elevation data (DTED). Defense Mapping Agency, Aerospace Center, St Louis, Mo.

Douglas, D. H. (1986). Experiments to locate ridges and channels to create a new type of digital elevation model. *Cartographica*, *23(4)*, 29–61.

Egenhofer, M. J., & Franzosa, R. D. (1991). Point-set topological spatial relations. *International Journal of Geographical Information Systems*, *5(2)*, 161–174.

Fisher, P. F. (1993). Algorithm and implementation uncertainty in viewshed analyses. *International Journal of Geographical Information Systems*, *7(5)*, 331–347.

Floriani, L. D., & Magillo, P. (1993). Computing visibility maps on a digital terrain model. In *COSIT'93, Elba Island, Italy*, pp. 248–269 Berlin. Springer-Verlag.

Floriani, L. D. (1987). Surface representation based on triangular grid. *The Visual Computer*, *3(1)*, 27–50.

Floriani, L. D. (1989). A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics & Applications*, *9(2)*, 67–78.

Foley, J. D., & van Dam, A. (1982). *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading, Mass.

Gaskins, T. (1992). *PEXlib Programming Manual.* O'Reilly & Associates, Inc.

Gold, C. M., Charters, T. D., & Ramsden, J. (1977). Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *ACM Computer Graphics*, *11(2)*, 170–175.

Gold, C. M. (1984). Common-sense automated contouring, some generalizations. *Cartographica*, *21*, 121–129.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *ACM SIGMOD*, *13*, 47–57.

ISO (1989). Information processing systems – computer graphics – programmers hierarchical interactive graphics system (phigs) – part 1: functional description. Tech. rep. ISO IEC 9592-1, International Organization for Standardization.

Jones, C. B. (1989). Data structures for three-dimensional spatial information systems in geology. *International Journal of Geographical Information Systems*, *3(1)*, 15–31.

Katz, M. J., Overmars, M. H., & Sharir, M. (1992). Efficient hidden surface removal for objects with small union size. *Computational Geometry: Theory and Applications*, *2*, 223–234.

Kraak, M. J. (1988). *Computer-Assisted Cartographical Three-Dimensional Imaging Techniques.* Ph.D. thesis, Delft University of Technology.

Kraak, M.-J. (1992). Tetrahedrons and animated maps in 2d and 3d space. In *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 63–71 Columbus, OH. International Geographical Union IGU.

Lee, Y. T., & Requicha, A. G. (1982). Algorithms for computing the volume and other integral properties of solids. i. known methods and open issues. *Communications of the ACM*, *25(9)*, 635–641.

Mäntylä, M. (1988). *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Md.

Mark, D. M. (1978). Concepts of data structures for digital terrain models. In *Proceedings of the DTM Symposium, ASPRS-ACSM, St. Louis, Missouri*, pp. 24–31.

Mitchel, J. S. B., Mount, D. M., & Papadimitriou, C. H. (1987). The discrete geodesic problem. *SIAM J. Comput.*, *16(4)*, 647–668.

Molenaar, M. (1990). A formal data structure for three dimensional vector maps. In *4th International Symposium on Spatial Data Handling, Zürich*, pp. 830–843 Columbus, OH. International Geographical Union IGU.

Mortenson, M. E. (1985). *Geometric Modeling.* John Wiley, New York.

Peuker, T. K., Fowler, R. J., Little, R. J., & Mark, D. M. (1976). Digital representation of three dimensional surfaces by triangulated irregular network. Tech. rep. 10, ONR Contract N00014–75–C–0886.

Pigot, S. (1992). Topological models for 3d spatial information systems. In *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 344–360 Columbus, OH. International Geographical Union IGU.

Polis, M. F., & McKeown, Jr., D. M. (1993). Issues in iterative tin generation to support large scale simulations. In *Auto-Carto 11*, pp. 267–277.

Pullar, D. V., & Egenhofer, M. J. (1988). Toward formal definitions of topological relations among spatial objects. In *Proceedings of the 3rd International Symposium on Spatial Data Handling, Sydney, Australia*, pp. 225–241 Columbus, OH. International Geographical Union IGU.

Robinson, V. B., & Tom, H. (1993). Towards sql database language extensions for geographic information systems. Tech. rep. NISTIR 5258, National Institute of Standards and Technology.

Rost, R. J. (1987). PEX PROTOCOL SPECIFICATION, VERSION 3.00. , Digital Equipment Corporation.

Scarlatos, L., & Pavlidis, T. (1991). Adaptive hierarchical triangulation. In *Auto-Carto 10*, pp. 234–246.

Scheifler, R. W., & Gettys, J. (1986). The x window system. *ACM Transactions on Graphics*, *5(2)*, 79–109.

Singleton, K. (1986). An implementation of the gks-3d/phigs viewing pipeline. In Requicha, A. A. (Ed.), *Eurographics '86*, pp. 325–355.

Stonebraker, M., & Rowe, L. A. (1986). The design of postgres. *ACM SIGMOD*, *15(2)*, 340–355.

Stonebraker, M., Rowe, L. A., & Hirohama, M. (1990). The implementation of postgres. *IEEE Transactions on Knowledge and Data Engineering*, *2(1)*, 125–142.

van Bemmelen, J., Quak, W., van Hekken, M., & van Oosterom, P. (1993). Vector vs. raster-based algorithms for cross country movement planning. In *Auto-Carto 11*, pp. 304–317.

van Oosterom, P., & Vijlbrief, T. (1991). Building a GIS on Top of the Open DBMS "Postgres". In *Proceedings EGIS'91: Second European Conference on Geographical Information Systems*, pp. 775–787. EGIS Foundation.

van Oosterom, P., & Vijlbrief, T. (1994). Integrating complex spatial analysis functions in an extensible gis. In *Proceedings of the 6th International Symposium on Spatial Data Handling, Edinburgh, Scotland*. Accepted.

van Oosterom, P. (1994). *Reactive Data Structures for Geographic Information Systems.* Oxford University Press, Oxford.

Vertegaal, W. (1994). A 3D modelling extension to Postgres/GEO++. Tech. rep. FEL-94-S164, FEL-TNO Divisie 2.

Vijlbrief, T., & van Oosterom, P. (1992). The GEO++ System: An Extensible GIS. In *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, pp. 40–50 Columbus, OH. International Geographical Union IGU.

Ware, J. M., & Jones, C. B. (1992). A multiresolution topographic surface database. *International Journal of Geographical Information Systems*, *6(6)*, 479–496.

Watson, D. F. (1992). *Contouring: A Guide to the Analysis and Display of Spatial Data.* Pergamon Press, New York.

Weibel, R. (1993). On the integration of digital terrain and surface modeling into geographic information systems. In *Auto-Carto 11*, pp. 257–266.

Weinand, A., Gamma, E., & Marty, R. (1989). Design and Implementation of ET++: A Seamless Object-Oriented Application Framework. *Structured Programming*, *10(2)*, 63–87.

APPENDIX A: 3D SPATIAL ADT FUNCTIONS

This appendix contains the functions specific for one of the following 3D spatial data types: POINT3, POLYLINE3, POLYGON3, and POLYHEDRON3; see Table 1. In Tables 2, 3, 4, and 5, the in- and output functions are omitted. Table 7 contains an overview of the *topological* predicates and Table 6 lists the *distance* functions.

data type	Postgres name	components
point	POINT3	x, y, z coordinates (floats)
polyline	POLYLINE3	list of POINT3s
polygon	POLYGON3	list of POINT3s
polyhedron	POLYHEDRON3	list of POINT3s, list of faces
box	BOX3	2 POINT3s (opposing corners)

Table 1: The 3D geometric data types.

name	arguments	return value	description
Pnt2ToPnt3	POINT2	POINT3	converts point from 2D to 3D
Pnt3ToPnt2	POINT3	POINT2	converts point from 3D to 2D
Box3Pnt	POINT3	BOX3	converts POINT3 to BOX3

Table 2: Point functions.

name	argument	return value	description
Pln2ToPln3	POLYLINE2	POLYLINE3	converts polyline from 2D to 3D
Pln3ToPln2	POLYLINE3	POLYLINE2	converts polyline from 3D to 2D
Box3Pln	POLYLINE3	BOX3	converts POLYLINE3 to BOX3
Length3Pln	POLYLINE3	float	length of polyline
MinGradient3Pln	POLYLINE3	float	minimum gradient of polyline
MaxGradient3Pln	POLYLINE3	float	maximum gradient of polyline
AvgGradient3Pln	POLYLINE3	float	average gradient of polyline
MinAzimuth3Pln	POLYLINE3	float	minimum azimuth of polyline
MaxAzimuth3Pln	POLYLINE3	float	maximum azimuth of polyline
AvgAzimuth3Pln	POLYLINE3	float	average azimuth of polyline

Table 3: Polyline functions.

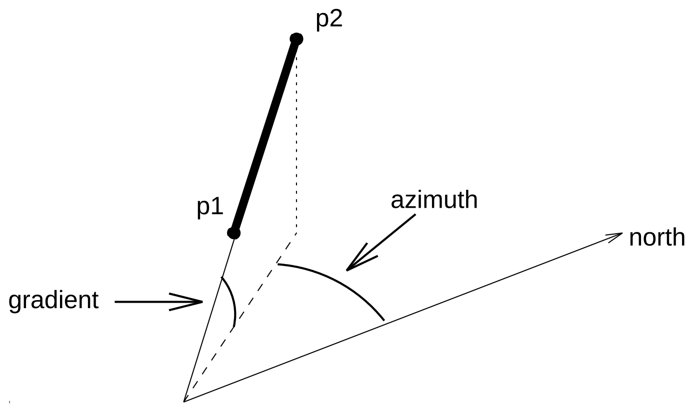

Figure 9: Definition of gradient and azimuth of a line segment.

name	arguments	return value	description
Pgn2ToPgn3	POLYGON2	POLYGON3	converts polygon from 2D to 3D
Pgn3ToPgn2	POLYGON3	POLYGON2	converts polygon from 3D to 2D
Box3Pgn	POLYGON3	BOX3	converts POLYGON3 to BOX3
Gradient3Pgn	POLYGON3	float	gradient of polygon
Azimuth3Pgn	POLYGON3	float	azimuth of polygon
GravCentre3Pgn	POLYGON3	POINT3	center of gravity
Centroid3Pgn	POLYGON3	POINT3	centroid of polygon
Area3Pgn	POLYGON3	float	surface area of polygon
Perim3Pgn	POLYGON3	float	perimeter of polygon
Check3Pgn	POLYGON3	boolean	tests if polygon is valid

Table 4: Polygon functions.

name	argument	return value	description
Box3Phn	POLYHEDRON3	BOX3	converts POLYHEDRON3 to BOX3
GravCenter3Phn	POLYHEDRON3	POINT3	center of gravity of polyhedron
Centroid3Phn	POLYHEDRON3	POINT3	centroid of polyhedron
Area3Phn	POLYHEDRON3	float	surface area of polyhedron
Content3Phn	POLYHEDRON3	float	content of polyhedron
Check3Phn	POLYHEDRON3	boolean	tests if POLYHEDRON3 object is valid

Table 5: Polyhedron functions.

name-prefix	input combinations	description
Distance3	PntPnt	distance between two points
MinDist3	all except PntPnt	minimal distance between two objects
MaxDist3	all except PntPnt	maximal distance between two objects

Table 6: Distance functions (return value is float).

prefix	object combinations	description
Equal3	PntPnt, PlnPln, PgnPgn, PhnPhn	the objects are equal
Touch3	all except PntPnt	the objects share boundary, but no interior areas
In3	all possible combinations	the first object is completely contained by the second object
Cross3	PlnPln, PlnPgn, PlnPhn, PgnPgn, PgnPhn	the objects share part(s) of their interiors, but the dimension of the shared area's boundary is lower than the higher of the dimensions of the two objects' boundaries
Overlap3	PlnPln, PgnPgn, PhnPhn	the objects share part(s) of boundaries, and the dimension of the shared area's boundary is equal to the dimension of the objects' boundaries
Disjoint3	all combinations	the objects share neither parts of boundaries, nor any area

Table 7: Topological relationships (return value is boolean).