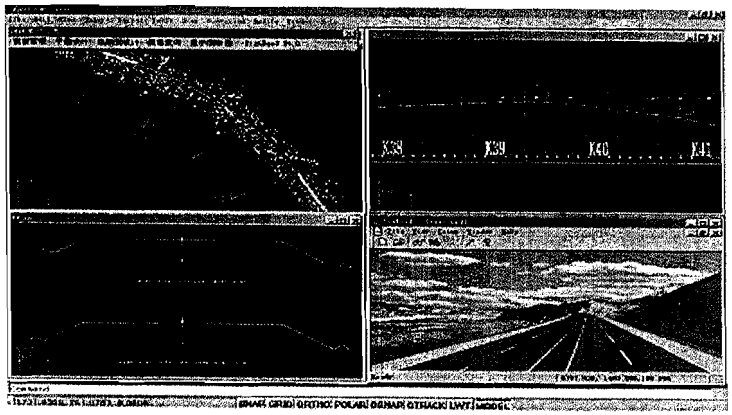
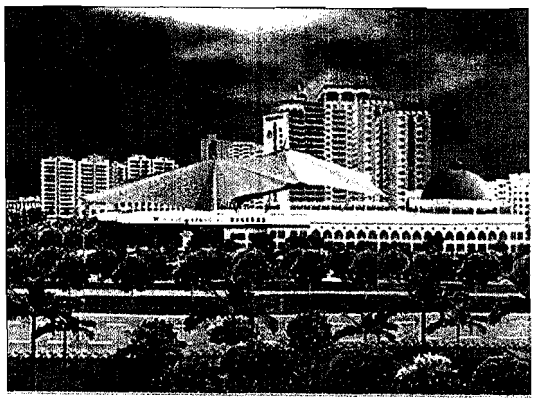


COLOR FIGURE 9.4 Multiview based informative road design.



COLOR FIGURE 9.8 Blended scene of image-based and geometry-based representations.

4 3D Geo-DBMS

Martin Breunig and Sisi Zlatanova

CONTENTS

4.1 GEO-DBMS: HISTORICAL DEVELOPMENT AND STATE OF THE ART

Historically, one can distinguish between two ways of incorporating database management system (DBMS) in GIS. The first way is that GIS vendors use DBMS mainly to store thematic (i.e., nonspatial) data. The spatial data are managed by single files only, i.e., no DBMS support is provided for spatial data. In this case, the analysis of the data takes place in the GIS, and only database queries on thematic attributes are executed by the DBMS. If data analysis concerns spatial *and* nonspatial data, the geo-objects have to be composed explicitly by their spatial and nonspatial parts. Few GIS vendors (e.g., ESRI) pursued storing both nonspatial and spatial data into DBMS. Both solutions are "top-down approaches," because, from an architectural point of view, the DBMS functionality has been constructed "under" the GIS application and the GIS application accesses "top-down" to the geodata stored in the DBMS.

The second way to integrate DBMS within GIS is DBMS offering support of our spatial data types. This solution is called the "bottom-up approach," because it extends "low level" DBMS data types and indexes to use them in the upper level of GIS applications. Data analysis is then performed by DBMS during the execution of database queries. Data analysis on the spatial *and* nonspatial parts of objects can be executed. Since the 1990s, more and more commercial DBMSs provide such spatial extensions to offer support of spatial objects.

In the field of AEC, the use of DBMS functionality is even more restrictive than in the field of GIS. As the spatial modeling features in DBMS are restricted to single points, edges, and polygons (triangles), there are no complex objects that could be stored consistently in a database. It should be noticed, however, that an increasing number of AEC systems (e.g., GeoGraphics, MicroStation) have already developed extensions that make use of spatial models and functionality provided by Geo-DBMS. We believe the significance of Geo-DBMS in both GIS and AEC worlds will continue to increase.

4.2 BENEFITS OF USING DBMS FOR GIS AND AEC APPLICATIONS

Clearly, there are a lot of reasons to use Geo-DBMSs in both GIS and CAD/AEC systems: multiuser control on shared data and crash recovery, automatic locks of single objects while using database transactions, advanced database protocol mechanisms to prevent the loss of data, data security, data integrity and operations that comfortably retrieve, insert, and update data. This section will provide an argumentation for wider utilization of geo-DBMS in GIS and AEC applications.

4.2.1 Providing DBMS Standard Functionality

Most GIS and CAD system users would be concerned if there was no multiuser access possible on data during design processes. There is no question that the results of their work being constructed on shared data should be automatically stored within the database periodically. Furthermore, unpredictable events, such as power breakdown,

wrong operation by the user, hardware error, etc., should not cause the loss of data. Fortunately, multiuser control and crash recovery are two of the standard functions in today's DBMSs. The user does not have to care about obtaining exclusive data access. The DBMS automatically locks single objects or tables using database transactions so that several users can access the same objects, reading or even updating them at the same time. Furthermore, advanced database protocol mechanisms prevent the loss of data by writing the data on disk beforehand.

Other useful DBMS functions concern data security and the checking of data integrity. Thus the correctness and the consistency of thematic and spatial data can be automatically checked during the input of the data and during the execution of database queries. Typical examples are the checking of the data types for input data and the checking of their domains. Finally, DBMSs offer operations for the comfortable retrieval, insertion, and update of nonspatial data. Therefore, standard relational or object-relational DBMSs are well suited to store and retrieve the values of nonspatial attributes of geo-objects.

4.2.2 Extending the DBMS to a Geo-DBMS (2D and 3D)

Many DBMSs have evolved to offer spatial functionality, which brings further benefits for GIS and AEC applications. The spatial functionality concerns the support of spatial data types in the data model, the implementation of spatial access methods, and the execution of spatial database queries (see Güting, 1994). Therefore, a Geo-DBMS manages both thematic *and* spatial data. Typical geo-database queries, such as "select the names of all buildings which are higher than 20 m and whose distance is smaller than 1 km from Frankfurt Airport" concern thematic *and* spatial attributes and functions of geo-objects.

Geo-DBMSs provide spatial *data types* and spatial *functions (and operations)* on them that define the *spatial functionality* of a Geo-DBMS. A Geo-DBMS knows primitive (simple) and composed geometric data types in the same way as its standard data types such as character, string, integer, real, etc. Currently, many DBMSs offer support of spatial data types, but most of them are only 2D types, i.e., point, line, and polygon. In contrast to the nonspatial data types, spatial data types have to be organized in a model (topological or geometrical), where a number of spatial rules have to be fulfilled. For example, "lines should not self-intersect," "two points on one line cannot be the same," etc. (see next section).

Following this order of thoughts, a Geo-DBMS is a 3D Geo-DBMS if it:

- Supports 3D data types, i.e., point, line, surface, and volume in 3D Euclidean space
- Maintains 3D (topological and/or geometrical) models
- Offers 3D spatial functionality, i.e., spatial operations and functions that can operate with the 3D data types

Having in mind the higher complexity of 3D objects, a 3D Geo-DBMS may need to support even different geometric and topological models needed in different GIS and AEC application classes. The primitive and composed 3D geometric data

types differ between GIS and AEC applications, because the requirements of these two application fields vary from standardized geometries of buildings to complex nature-formed objects in the geosciences.

An important aspect of 3D Geo-DBMS is the amount of data to be processed. The access problem can be resolved by extending well-known spatial access methods like the R-tree (Guttman, 1984) or the R*-tree (Beckmann et al., 1990) for the third spatial dimension (z-coordinates). The procedure is the same as in the two-dimensional case: in the first step (filter or approximation step), the data set is reduced by using a spatial search function of the spatial access method computing the intersection between approximated objects — in most cases minimal circumscribing 3D-boxes of the object geometries — and a 3D query box. In the second step (refinement-step) the real geometric operation (e.g., intersection between the exact geometries of 3D objects and the 3D query box) is executed on the reduced data set. At present, Geo-DBMSs support several types of 3D spatial indexing.

3D Geo-DBMS has to offer an appropriate 3D user interface. To date, 3D user interfaces have not yet been exhaustively examined. Future 3D query interfaces should support the formulation of complex SQL-like mixed spatial and nonspatial database queries as well as 3D graphical input supporting the intuitive graphical formulation of 3D queries. Existing VRML/X3D interfaces are not flexible enough to support the manipulation and database update of 3D objects. Potential solutions are to be sought in coupling AEC software with Geo-DBMS. AEC applications offer a rich set of 3D modeling and visualization tools that can be further extended toward specifying 3D spatial (SQL) queries.

4.2.3 Geo-DBMS Models: Geometry vs. Topology

Geometry and topology are often used in Geo-DBMS terminology (in contrast to Open Geospatial specifications) as synonyms to denote the two ways of describing spatial data types. Geometry data types are defined by the x, y, z coordinates of the points composing a data type. Topology data types have references to the unique identifiers of low-dimensional data types.

Apparently, for some period of time, two models (topology and geometry) will be maintained in the Geo-DBMS. Although there is belief that a topological model might be sufficient (geometry can be derived from topology), in the same way, one could argue that the same is true for a geometry model, because topology can be derived from geometry. In this respect, one can choose between three different options:

- Storing the topological model in the 3D Geo-DBMS (and deriving geometry from it)
- Storing the geometric model in the 3D Geo-DBMS (and deriving topology from it)
- Storing both models in the 3D Geo-DBMS.

In approach 1, the topological relationships between object parts, such as "all surfaces belonging to the boundary of a 3D volume object" are stored in the topology

model of the DBMS. The location of objects, i.e., the x, y, z coordinates of its defining points, are not part of the topology model. The advantage of this approach is that objects can quickly be identified by their topological properties, such as the Euler characteristics for triangle nets. However, geometric database queries, such as "return the intersecting geometry of two objects," cannot profit from the explicit knowledge of the objects' topology. What they need is the geometry, i.e., the "shape" of the objects given by plane equations, individual x, y, z coordinates of the vertices, etc.

Approach 2 prevents the disadvantage of approach 1; however, topological database queries cannot be answered efficiently by providing only a geometric model. As the topological relationships between single polygons (surfaces) or polyhedra (volumes) are missing, 3D spatial queries have to be completed using computational algorithms (which might become very complex and, thus, slow). For example, constructing a new triangulated irregualr network (TIN) object (geometrically computed by the intersection between two TIN objects) can profit from the neighborhood information of triangles while testing for intersecting triangles between the two TIN objects.

Approach 3 still seems to be the best solution: it allows flexible topological and geometric database queries executed efficiently by accessing the topology and geometry model in the 3D Geo-DBMS. With these two models, all relevant types of spatial database queries can be executed, including:

- Topological database queries (e.g., neighborhoods)
- Geometric database queries (e.g., spatial search inside a 3D box)

Topologically based analysis will be beneficial for consistency checks and all kinds of operations making use of neighborhood relationships. Geometric-based analyses will be necessary for constructing new objects (e.g., buffer, aggregations), to build the topology and perform metric operations (distance, area, spatial search). CAD and AEC applications, being more visualization- than analysis-oriented, will largely benefit from geometry models, which will allow fast retrieval of coordinates (having the coordinates stored with the data types). GIS applications will be ensured with consistent models (topology).

4.2.4 Object-Relational vs. Object-Oriented DBMS

There are two different database system architectures to be considered for Geo-DBMS: object-relational and object-oriented. The third possibility, i.e., XML-DBMS, does not seem to be appropriate, because most 3D geodata are well structured, containing complex geometric and topological structures. XML-DMBSs are specialized to manage semi-structured and text data instead. Both object-relational and object-oriented DBMSs allow their users to model data as objects. Of course, object-databases can export their objects also as XML-structures by using appropriate adapters.

In contrast to object-relational database management systems (ORDBMSs), an object-oriented database management system (OODBMS) does not only support

objects as its data model, but it also physically stores the objects on disk. In contrast to ORDBMSs, OODBMSs are not based upon existing relational database technology. OODBMSs are newly developed DBMSs for the management of objects.

Most of today's object databases such as Oracle®, Informix®, etc., are solutions on top of relational DBMS (ORDBMS), but there are also some native OODBMS on the market, such as ObjectStore®, FastObjects®, etc.

The conceptual data modeling in object-relational and object-oriented DBMSs is the same: data are modeled as classes, which are the generalization of objects with the same or similar properties and operations. ORDBMSs are based on the well-proven technology of 30 years of relational DBMS experience. They are using sophisticated query optimization strategies based on the well-founded relational algebra. What makes the difference is the internal physical data model. ORDBMSs store their data in tables that are coupled by referential integrity, with relationships between primary and secondary keys of different tables. This change of the data model from user-defined classes to internal "flat" tables leads to the impedance mismatch, which must be paid by a performance loss during the "object" retrieval of very large data sets.

OODBMSs allow the "native" storage of objects in their data model. This is mostly done by an overriding of the new operator of object-oriented programming languages like C++ or Java. Furthermore, cluster strategies for the optimized storage of objects in one class are provided. OODBMSs are flexible and can be extended by new spatial data types and spatial indexes. However, 15 years after their birth, OODBMSs still are a research subject and the idea of having standardized OODBMSs (ODMG, 1993) seems to be far in the future. Unfortunately, they do not provide comfortable data modeling and I/O-tools. OODBMSs have not succeeded on the market yet, in spite of their undisputed advantages concerning the management of complex 2D and 3D geo-objects.

GIS and AEC applications are currently offering support to data types provided by object-relational DBMS. One of the reasons is, of course, that most of the mainstream Geo-DBMS are object-relational; the other is that the objects can be defined in a different way by OODBMS.

4.3 TOWARD 3D GEO-DBMS

3D Geo-DBMS has to be able to provide the necessary data types to maintain as many as possible representations, to be able to serve both GIS and ACE applications. Several possibilities will be discussed here.

4.3.1 Geometry

4.3.1.1 Simple Nature-Formed Objects

Depending on the system architecture of the DBMS (see Section 4.4), the 3D geo-objects are internally stored as tables or as objects by overriding the new operator of the object-oriented application programming interface of the Geo-DBMS.

We pick up one of the representations to model simple 3D (vector) objects demonstrating their design and implementation in Geo-DBMS. The following primitive

3D geometric data types are appropriate to be implemented in Geo-DBMSs, if *nature-formed objects* shall be modeled:

```
- type Point3D = (x_i:double, y_i:double, z_i:double)
with (i = 0);

- type Segment3D = (x_i:double, y_i:double,
z_i:double) with (0 ≤ i ≤ 1);

- type Triangle3D = (x_i:double, y_i:double, z_i:double)
with (0 ≤ i ≤ 2);

- type Tetrahedron3D = (x_i:double, y_i:double,
z_i:double) with(0 ≤ i ≤ 3).
```

We summarize the special properties of these primitive 3D geometric data types as follows:

They are defined in 3D Euclidean space and are pairwise independent, i.e., the following consistency conditions hold: the two points of a segment must not have the same coordinates, the three points of a triangle must not be on one line, and the four points of a tetrahedron must not lie on one plane.

Each primitive 3D geometric data type with dimension i (0 i 3) contains those spatial objects of dimension i with the most simple geometry in dimension i. Notice that there is a primitive 3D geometric data type — Tetrahedron3D — for the explicit modeling of volumetric objects. Thus, thematic information can be attached to single or groups of tetrahedron objects.

4.3.1.2 Complex Nature-Formed Objects

Complex nature-formed objects can be easily constructed by composing adjacent primitive objects. Thus, the following composed 3D geometric data types are appropriate:

```
- type Line3D = (x_i:double, y_i:double, z_i:double) with
(0 ≤ i ≤ 1);

- type Surface3D = (tri_i:Triangle3D) with (0 ≤ i ≤ p);

- type Volume3D = (tet_i:Tetrahedron3D) with
(0 ≤ i ≤ s).
```

The left side of the data type definition is the abstract interface (line, surface, volume). The right side, however, gives a possible implementation that can be exchanged by other implementations. In the given example, Surface3D is implemented as a list of triangles and Volume3D as a list of tetrahedrons, respectively.

4.3.1.3 Simple Man-Made Objects

In many cases, this approach is not appropriate due to data collection procedures (e.g., many real-world objects are measured only from the outside), modeling considerations (e.g., unnecessary subdivisions), or volume of data. The problems are mostly in the 3D primitive. A simple box (e.g., representing a building) will

require six tetrahedrons to model it. Furthermore, CAD software can model a spatial object very realistically with curved surfaces, but curved surfaces produce a large number of tetrahedrons. Such arguments give preference to the polyhedron option.

The data types in Geo-DBMS will then become point3D, line3D, polygon3D, and polyhedron3D. However, extending the freedom in the shape requires rules defining which shapes are allowed in the model. Examples of widely implemented rules are related to constraints on self-intersection and planarity of polygons. The polyhedron is restricted to a polyhedron composed of flat faces.

The representations of the data types then will be:

```
- type Point3D = (x_i:double, y_i:double, z_i:double)
with (i = 0);

- type Line3D = (x_i:double, y_i:double, z_i:double)
with (0 ≤ i ≤ 1);

- type Polygon3D = (x_i:double, y_i:double, z_i:double)
with (0 ≤ i ≤ p);

- type Polyhedron3D = (x_i:double, y_i:double,
z_i:double) with (0 ≤ i ≤ s).
```

The need of a rule for such data types is apparent. Any polygon with $i > 2$ is not planar by default. A polyhedron is defined as "a bounded subset of 3D coordinate space enclosed by a finite set of flat polygons (called faces) such that every edge of a polygon is shared by exactly one other polygon." The polyhedron should bound a single volume, i.e., from every point (can be on boundary), every other point (can be on boundary) can be reached via the interior. The characteristics of a polyhedron primitive are given in Arens et al. (2005) as:

- Flatness: The polygons that make up the polyhedron have to be flat. This means that all points that make up the polygon must be in the same plane (Figure 4.1).
- 2-Manifold: The polyhedron should bound only one volume. This means that from every point on the boundary, one should be able to reach every other point on the boundary via the interior. For the object to be valid, the faces where the hole starts and ends have to be modeled as a face with one or more inner rings.
- Simplicity: The polyhedron has to be composed of simple features, i.e., closed polygons that are not self-intersecting and have no inner rings. The faces of a polyhedron, however, are allowed to have inner rings, if the polygons together form a closed polyhedron.
- The inner rings of polygons are not allowed to interact with the outer ring, except for touching boundaries. The vertices that span a face are not allowed to lie all on a straight line, i.e., the face has to have an area. A face has exactly one outer ring, each edge has exactly two vertices. Only edges with two points are allowed. Note that two or more (but not all) edges are allowed to lie on a straight line, if this is more convenient for modeling an object.

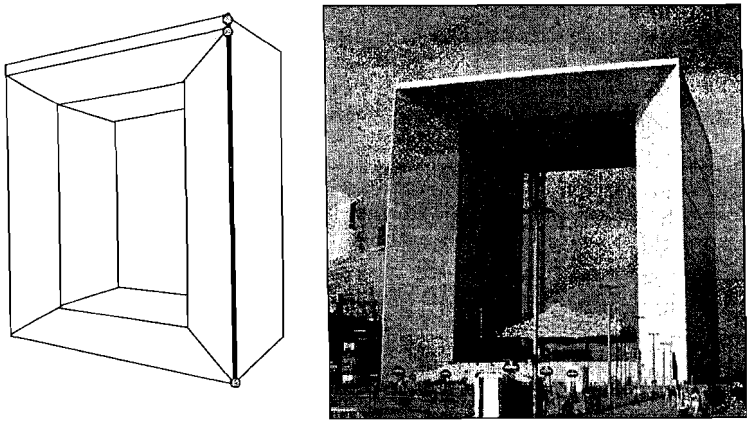


FIGURE 4.1 Two edges in a line (Arens et al., 2003).

- Orientable: The outside and inside of the polyhedron has to be specified. Basic rules in computer graphics related to the normal vector are used for orientation. This means that the vertices in a face must be specified in counter-clockwise order seen from the outside of the object. The vertices in inner rings of faces need to be ordered in the opposite direction (clockwise).

Implementation of such a data type is given in Section 4.4.2.2.

4.3.1.4 Complex Man-Made Objects

Representations with only flat polygons are usually insufficient for modeling many AEC 3D free-form curves and surfaces. Introduction of complex data types based on free-form mathematical curves and surfaces is required.

The representations of the data types will then be:

```
- type Point3D = (x_i:double, y_i:double, z_i:double)
with (i = 0);

- type Curve3D = (x_i:double, y_i:double, z_i:double,
r_ij: double) with (0 ≤ i ≤ k) and (0 ≤ j ≤ k);

- type Surface3D = (x_i:double, y_i:double, z_i:double,
r_ij: double) with (0 ≤ i ≤ 1) and (0 ≤ j ≤ k);

- type Polyhedron3D = (x_i:double, y_i:double, z_i:double,
r_ij:double) with (0 ≤ i ≤ m) and (0 ≤ j ≤ k);
```

where r_i is a parameter denoting a property (one or many) of a free-form shape.

NURBS can be an option for representing man-made objects. NURBS are approved as industry standards for the representation and design of geometry. Important characteristics of NURBS are listed below (Rogers and Earnshaw, 1991).

- NURBS offer a common mathematical form for both standard analytical shapes (e.g., cones, spheres) and free-form shapes.
- They provide a flexible way of designing a large variety of shapes.
- The shapes described by NURBS can be evaluated reasonably fast by numerically stable and accurate algorithms.

Important characteristics for modeling real-world objects is that they are invariant under affine as well as perspective transformations.

The general drawback of NURBS is the extra storage needed to define traditional shapes (e.g., circles). See also the discussion of NURBS in Chapter 3. NURBS shapes are defined by control points, weights associated with each control point, and knots. A NURBS curve $C(u)$, is defined as (Piegl, 1991):

$$C(u) = \frac{\sum_{i=0}^{n} \{w_i * P_i * N_{i,k}(u)\}}{\sum_{i=0}^{n} \{w_i * N_{i,k}(u)\}}$$

where

w_i: = weights
P_i = control points (vector)
$N_{i,k}$ = normalized B-spline basis functions of degree k.

These B-splines are defined recursively as:

$$N_{i,k}(u) = \frac{u - t_i}{t_{i+k} - t_i} * N_{i,k-1}(u) + \frac{t_{i+k+1} - u}{t_{i+k+1} - t_{i+1}} * N_{i+1,k-1}(u)$$

and

$$N_{i,0}(u) = 1 \text{ if } t_i <= u < t_{i+1}$$

and

$$N_{i,0}(u) = 0 \text{ else}$$

where t_i are the knot points forming a knot vector $U = \{t_0, t_1, ..., t_m\}$ (see Figure 4.2).

Then the data type will be:

```
- type NURBS3D = (xi:double, yi:double, zi:double,
wi:double, tij:double) with (0 ≤ i ≤ k) and
(0 ≤ j ≤ k);
```

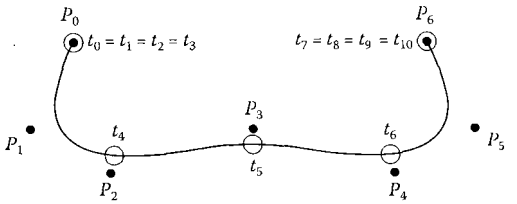



FIGURE 4.2 B-spline (courtesy of http://mathworld.wolfram.com/B-Spline.html).

NURBS surface can be defined in a similar way:

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} * R_{i,k,j,l}(u,v)$$

where

$$R_{i,k,j,l}(u,v) = \frac{w_{i,j} * N_{j,k}(u) * N_{j,l}(v)}{\sum_{r=0}^{n} \sum_{s=0}^{m} w_{r,s} * N_{r,k}(u) * N_{s,l}(u)}$$

The definition of NURBS requires a number of parameters to be included in the data type. Examples of such parameters are knots sequence, number of control points, coordinates of control points, etc. In case of surfaces, the parameter doubles.

4.3.2 Topology

Similar to geometry data types, topology data types can be different with respect to the objects to be modeled. Discussing topological data structures, many application-related issues have to be taken into consideration, e.g., the space partitioning (full, embedding), the object components (volumes, faces), the construction rules (planarity, intersection constraints, etc.). The 3D topological data structures reported currently in the literature can be subdivided into two large groups: structures maintaining objects and those maintaining relationships. While in the first group (object-oriented), most of the relationships between the objects have to be derived, in the second group (topology-oriented), the representation of the objects has to be derived. Many structures that are typical examples of the explicit storage of objects also maintain explicit storage of relationships, i.e., singularities.

4.3.2.1 Nature-Made Geo-Objects

The kind of geometric or topological data model presented here is known as "cell decomposition" (Requicha and Voelcker, 1982). Cell decomposition is taken here as an example to show the design and implementation of topology and geometry

in Geo-DBMS. Being a special case of a cell decomposition, a TIN is known as "a collection of adjacent triangles with the topological consistency requirement that two pairwise adjacent triangles must completely touch in their adjacent edges by their whole length." There are no "dangling" triangles allowed in the TIN. Thematic attributes can be attached to whole TIN objects as well as to its parts, i.e., to single points, edges, and triangles. Analogously, tetrahedron nets can be joined together by adjacent tetrahedra with the topological consistency requirement that two pairwise tetrahedra are touching completely at their adjacent triangle faces. Thematic attributes can be attached to whole tetrahedron net objects or to its parts, i.e., to single points, edges, triangles, and tetrahedra.

The theory behind the implementation of topological or composed data types like "point set," "polylines," "surfaces," and "volumes" are the simplicial complexes that have been introduced into the field of GIS by Egenhofer (1989) and Egenhofer et al. (1990). The advantages of implementing simplicial complex data types in Geo-DBMSs are the following:

Unified treatment for 0-, 1-, 2-, and 3-dimensional objects, i.e., all objects are defined in one unified topology and geometry model.

Simplicial complexes are composed objects so that thematic attributes can also be attached to single points, edges, surfaces, and volumes of simplicial complex objects. Simplicial complexes provide a good approximation for the shape of complex nature-formed 2D and 3D objects, like the earth's surface or geological strata. Geometric database queries like the intersection query between a set of complex 3D objects executed on simplicial complexes internally profit from using the relatively simple intersection algorithms between points, segments, triangles, and tetrahedra, all being members of the introduced primitive 3D geometric data types.

To choose a suitable 3D primitive, some criteria have to be evaluated. The implementation should lead to valid objects. And once an object is modeled, there cannot be any ambiguities. A representation of an object should make clear how the object looks in reality. It should be easy to create and enable efficient algorithms. Furthermore, the size and redundancy of storage (conciseness) should be taken into consideration.

Figure 4.3 shows a conceptual 3D Geo-DBMS topology and geometry model for complex nature-formed objects. The simple topological data types are *Point, Segment, Triangle,* and *Tetrahedron.* Notice the following topological extension of the "traditional" simplicial complex approach: Each segment internally knows its start and end point, each triangle is aware of the list of its maximum three neighboring triangles, and each tetrahedron knows its maximum four neighboring tetrahedra.

The complex objects *PolyLine, TriangleNet,* and *TetraNet* are composed by a list of segments, triangles, and tetrahedra, respectively. However, a segment, a triangle, and a tetrahedron can belong to one or more polylines, triangle nets, or tetra nets to avoid data redundancy of primitive topological objects.

GeoPoint, GeoSegment, GeoTriangle, and *GeoTetrahedron* are the corresponding 3D geometric data types that contain the x-, y-, z-coordinate information and additional thematic information.

Finally, *GeoObject3D* is a polyline, triangle net, or tetra net attached with thematic information for application-specific thematic information. A *Group* is a collection of GeoObject3D objects that are allowed to have different dimensions. Group objects are

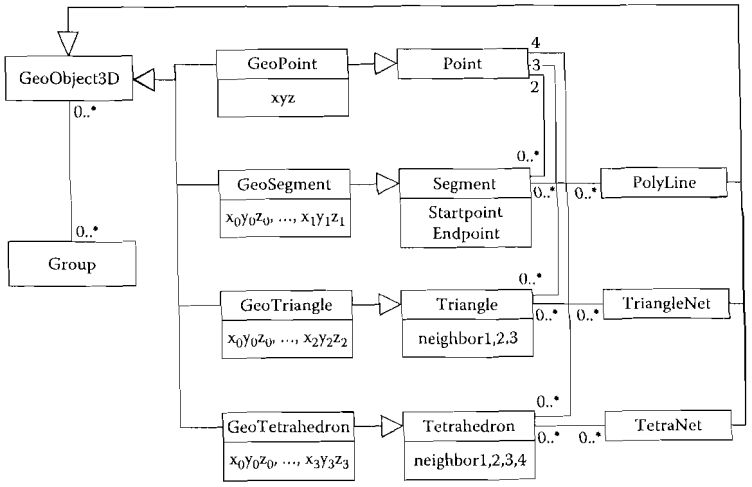


FIGURE 4.3 Conceptual 3D Geo-DBMS topology and geometry model for complex nature-formed objects.

generated during the execution of geometric 3D operations, e.g., if the intersection between two intersecting and touching tetra nets is computed as mixed triangle (two-dimensional) and tetra nets (three-dimensional objects).

In the future, it must be discussed whether advanced topology models such as the "GMaps" (Lienhardt, 1994; Levy, 1999) being used in the gOcad® 3D geological modeling software (Mallet, 1992, 2002) should be implemented in 3D Geo-DBMS. Obviously GMaps serve as an abstract framework as well as a flexible instrument to attach thematic and geometric attributes to 3D geo-objects.

4.3.2.2 Man-Made Geo-Objects

The existence of many topological models for man-made objects clearly indicates the complexity of the issue. A large number of topological structures have been developed through the years (Zlatanova et al., 2004). To distinguish between geometry and topology, often different names are used for the primitives, (e.g., TEN):

- type Node3D = ($pointID_i$, x_j, y_j, z_j) with ($j = 0$);
- type Arc3D = ($arcID_i$, $pointID_{i,j}$) with ($0 \leq j \leq 1$);
- type Face3D = ($faceID_i$, $arcID_{i,j}$) with ($0 \leq j \leq 2$);
- type Body3D = ($bodyID_i$, $faceID_{i,j}$) with ($0 \leq j \leq 3$);

Furthermore, the type of primitive varies from simplexes as described above (Pilouk, 1996; Coors 2003) to more free representations (Molenaar, 1992, Zlatanova 2000, de la

Losa and Cervelle 1999). Clearly, advantages of a topological representation in one of the aspects occur as disadvantages in another aspect. For example, the arbitrary number of nodes per face can be seen as advantage and disadvantage for different applications. It is very convenient for modeling complex 3D objects (e.g., buildings), since an inappropriate partitioning (into triangles) is not required, but the operators for consistency checks become very complex.

The subdivision into triangles furnishes the data needed to display graphic information in the most appropriate way. In this respect, TEN and UDM (Coors, 2003) are perhaps the optimal models for visualization of surfaces. Maintenance of triangles solves other modeling problems as holes or explicit storage of relationships (such as arc-on-face and node-on-face). An additional disadvantage for TEN is the much larger database compared to other representations and the need for special processing of the tetrahedrons that are not needed for visualization.

Some of the primitives (e.g., arcs) are suggested to be left out of the model to accelerate the visualization (Zlatanova, Holweg, and Coors, 2003). Those usually stored with Arcs BeginNode EndNode can be derived from ordered nodes in faces. However, some topological queries, such as navigating trough surfaces (e.g., "follow shortest path"), can become time-consuming.

The relationships stored per object also differ: a body can be explicitly described by faces, but body can also be implicitly derived from references stored with the faces (i.e., "left" and "right" body). "Left" and "right" body stored per face is a very convenient manner for navigating through 3D objects, but requires reference to a body "open air." The major problem with TEN (see above) refers to the modeling stage. Since the space is completely subdivided into tetrahedrons, the interiors of objects (e.g., buildings), as well as the open space, are also decomposed into tetrahedrons. Such subdivision is rather inconvenient for 3D man-made objects. Pilouk 1996 suggests a combination of TEN and TIN when appropriate.

Pigot (1995) elaborates on a cell tuple data structure, which provides the largest spectrum of topological relations between cells and complex cells. From a database point of view, this model promises an easy maintenance, due to the solid mathematical foundations and the simple representation. In the visualization respect, the extraction of faces and points is a simple operation, due to the explicitly stored link between the cells. In addition to the tuples, some supplementary information is needed, such as order (clockwise or counter-clockwise) of cells (note that the cycle is ensured). Assuming a relational implementation, the entire tuple information is available in one relational table, which has advantages and disadvantages. On the one hand, there is no need to perform JOIN operations to select any data. On the other hand, the size of the table grows tremendously, which slows down the speed of SELECT operations. For example, the records for a simple box occupy double space compared to other representations.

Presently, no agreement on a 3D topological representation exists. Van Oosterom et al. (2002) suggest an alternative approach. Instead of looking for an appropriate 3D topological representation, it is suggested to extend the DBMS kernel with meta information describing different topological structures in the DBMS.

This topological meta information can then be used both within the DBMS and outside the DBMS. In general, meta information (or system catalogs) of a DBMS

contains descriptions of the data stored in the database: tables, attributes, and types, and also contains descriptions of the available types and operators. The different topological structures can be characterized by the following "parameters":

- Dimension of the embedding space: 2D, 2.5D, 3D, time added
- Topological primitives are used: node, edge, face, volume
- If the elements considered are directed (oriented) or not
- Explicit topological relationships (part_ of, in, on) are stored
- Topological "rules" — crossing edges allowed? dangling elements allowed? same topological primitive on both sides of boundary allowed? etc.

In general, such an approach may resolve many 3D modeling and visualization problems but requires robust consistency management. Such an approach is not completely unfamiliar for Geo-DBMS. For example, Oracle Spatial also uses a specific meta data table to describe geometric attributes in more detail: USER_SDO_GEOM_METADATA, which contains information about the number of dimensions, the extent of the domain, and the resolution.

Similarly Geo-DBMS supporting topology management has to store (and supply to the applications) the topological information: e.g. topological layer name, which table plays the role of the boundary table, which table plays the role of the area table, and how are the relevant attributes, with metrical and topological information called, within these tables. A drawback of this solution is that the topology elements (object ids, references and also the metric attributes) have fixed names. A better approach is storing this information in a meta data table. Again, somewhere it must be declared which tables and which attributes carry the topological information. An example of the extension of the meta information of the DBMS is given in Vijlbrief and van Oosterom, 1992.

4.3.3 3D Spatial Access Methods

3D spatial access methods implemented in a Geo-DBMS have the task to speed up database queries referring to the spatial position of objects in 3D Euclidean space. They support the 3D region query (3D-box query), i.e., all objects containing or intersecting a given 3D query box are selected. The SQL-like formulation of the 3D region query runs as follows:

```
SELECT *
  FROM GeoObject3D
  WHERE 3DQUERYBOX intersects GeoObject.boundingBox;
```

To support the spatial search query efficiently, spatial index structures such as the R-tree (Guttman, 1984) or the R*-tree (Beckmann et al., 1990) are used. Such index structures are based on the hierarchical decomposition of the data space. The objects in the database are divided into subsets (buckets) that correspond to a partition of the data space, respectively. The buckets are accessed by a search tree. Typically all buckets

belonging to an R-tree or R*-tree node are stored together on one database page. That is why a spatial preclustering is guaranteed in the database. For each object in the database, a minimal 3D axe-directed bounding box must exist, because the spatial index structure only takes these "approximated objects" to determine a possible intersection between a given 3D query box and the minimal bounding boxes of the objects in the database.

4.3.4 3D Spatial Predicates, Functions, and Operations

An important question is which spatial functionality should be allocated to a Geo-DBMS. Apparently, the functionality will be distributed between DBMS and front end, but how? In general, GIS functionalities that are generic for the geodata should be provided by the DBMS. Arguments for this are maintenance of logical consistency and data integrity. Following these arguments, we appreciate the development of generic geo-database kernel systems and front ends. They provide 3D data types, 3D access methods, and a 3D query user interface. First prototype systems have been developed with the DASDBS-Geokernel (Schek and Waterfeld, 1986; Waterfeld and Breunig, 1992), GEO++ (Vijlbrief and van Oosterom, 1992), Geo_2 (David et al., 1993), OMS (Breunig et al., 1994), GeoToolKit (Balovnev et al. 2004), and others. Front ends can build a further specific functionality. On the other hand, 3D topological analyses are of higher computational complexity compared to 2D. If performed at object-relational DBMS level — e.g., on top of SQL — this might have a negative effect on the database performance. This would speak for the direct embedding into the programming language of an OODBMS instead.

Moreover, the question arises whether the Geo-DBMS is an appropriate place to organize level of detail (LOD). ORDBMSs or OODBMSs could be extended by new spatial access methods supporting LOD.

Obviously, there are many specialized 3D geometric functions and operations needed in GIS and AEC applications. Therefore, a Geo-DBMS should provide the possibility to embed user-defined procedures for the computation of such operators (see Günther, 1991; Breunig et al., 1994). Of course, the efficiency of the computing in these operators depends on the geometry and topology model, as well as on the spatial access methods internally used in the geometric operators (Günther, 1988; Güting, 1994).

4.3.5 3D Extensions for Spatial Query Languages

To support 3D spatial database queries, query languages like SQL have to be extended by 3D predicates, functions, and operations. We give two examples for 3D database queries in geology using the 3D spatial function and operation introduced in the last section:

```
SELECT *
  FROM drilling
    WHERE distance3D(DRILLING, drilling.geometry) <
100m;
```


Query 1 uses the *distance3D* function. It selects all drillings in 3D Euclidean space that have a distance of less than 100 m to the specified DRILLING. One could also think of an additional geometric 3D query that projects the drillings selected by query 1 on a vertical plane (Breunig et al., 2004). Such projections are very useful for the construction of geological profile sections.

```
DEFINE VIEW AS
  SELECT intersection3D(fault.geometry)
  FROM fault
  WHERE STRATUM intersects3D fault.geometry ;
```

Query 2 uses the spatial predicate *intersects3D* and the spatial operation *intersection3D*. It creates new objects being computed by the intersection between the current geometry of STRATUM and the geometry of all faults in the database. In each step of the query execution the intersection is only computed if the *intersects3D* predicate returns the Boolean value TRUE.

4.4 3D IN PRESENT DBMS

The currently offered functionality by Geo-DBMS is not 3D with respect to the background provided in Section 4.3, but 3D objects can be stored and a number of spatial operations can be performed on 3D objects. The following sections provide examples.

4.4.1 3D Objects in Object-Oriented DBMS

Kay, the inventor of Smalltalk object-oriented programming language, summarized five essential characteristics of pure object-oriented modeling and programming (Eckel, 2002):

- Everything is an object.
- A program is a bunch of objects telling each other what to do by sending messages.
- Each object has its own memory made up of other objects.
- Every object has a type.
- All objects of a particular type can receive the same messages.

The requests that may be sent to an object are defined by its *interface*. The data type determines the interface. The idea that the data type is quasi equivalent to the interface is fundamental to object-oriented modeling. Thus, the implementation, i.e., all internal structures are hidden from the user and are separated from the interface. In the database context, the interface is called *database schema*.

We speak of an OODBMS if the DBMS completely "understands" objects as its data model. Thus, the database schema is defined by classes (collections of objects) and the DBMS is in the position to store extensions of a class, i.e., arbitrary objects (instances). The OODBMS can return the instances by its object query

language (OQL). This implies facilities for the description of the schema and for the hierarchical navigation between objects in classes.

The design of the topology and geometry model (see Figure 4.3) can be directly transferred to the physical database schema OODBMS is used. The structure of the spatial data type hierarchy consisting of primitive and composed data types can be directly maintained by inheritance relationships between the corresponding classes. The attributes of the 3D data types are implemented as properties of the classes (class variables), and their geometric and topological operations are realized as class methods. Therefore, every 3D object knows to which class it belongs and which properties and methods it owns. The multiplicities of the relationships between two classes (such as n:m-relationship) are implemented as properties of classes that again are allowed to be of type "class." Furthermore, every class is responsible for checking the input data if generating a valid geometry of this class (see *valid* method in the following example).

Example — part of class code demonstrating some methods for class Tetrahedron3D:

```
PUBLIC MEMBERS:

static int valid (const pointRep& P1, const pointRep&
P2,

 const pointRep& P3, const pointRep& P4);

// Description: this function checks if the points
P1, P2, P3 and P4 form a valid tetrahedron.

// Parameter: the points of the tetrahedron.

// Return value: 1 if the tetrahedron is valid, else 0.


Tetrahedron3D();

// Description: generates a "default-" Tetrahedron3D
with the points (0,0,0), (1,0,0), (0,1,0), (0,0,1).


Tetrahedron3D (const pointRep& P1, const pointRep&
P2,

 const pointRep& P3, const pointRep& P4);

// Description: generates a Tetrahedron3D with the
points P1, P2, P3 and P4.

// Parameter: the four points of the tetrahedron.

// Integrity constraint: The four points must define
a tetrahedron, i.e. they must not be

// equal and they must not be situated on a single
plane.
```


In the above example, we have only shown the methods for the integrity check of the geometry and two constructors generating a new Tetrahedron3D object. Beyond the presented code, the Tetrahedron3D class provides other methods such as cloning and geometric intersection with other Tetrahedron3D objects.

From an architectural point of view, we can distinguish between two types of OODBMS: page-based OODBMS vs. object-based OODBMS. Page-based OODBMS allows the loading of all objects that are located on one database page in one call. Therefore, they use the advantage of an object cached in main memory instead of reloading single objects. Against that, object-based OODMBS allow the loading of single objects from a disk. This type of OODBMS is well suited for the retrieval of individual objects or small sets of objects.

4.4.2 3D Objects in Object-Relational DBMS

Object-relational approach is based on defining new objects (data types) that can be stored in one record of relational tables. DBMS knows the meaning of the new data types and threads them in the same way as the simple data types. The spatial data types are complex, easily resulting in the relationship 1:m. This multiplicity can be represented in different ways, e.g., Oracle Spatial offers *Varrays* and *Nested tables*.

The data types and the validity rules are only the first step in providing spatial functionality. Geo-DBMS offers a set of spatial functions. For example, Oracle Spatial, SDO_RELATE operator implements the 9-intersection model for categorizing binary topological relations between points, lines, and polygons. The 9-intersection model investigates the intersection between the interiors, boundaries, and exteriors of two objects. The intersections are recorded in a 3×3 matrix, which results in 9 intersections. In general, Oracle, IBM DB2, Informix, and PostGIS support geometric functions defined by OGC — and often more functions than these.

The major problem of the implementations of spatial features and operators in mainstream DBMSs is that they differ from each other. The statement "select attribute_a from table_b where a < 100" is the same in every DBMS. However, if geometries have to be found within a certain distance, different types of queries have to be executed in the different DBMSs. For example, Oracle Spatial has implemented data types that do not have explicit names such as point, line, and polygon. There is one complex data type, sdo_geometry, composed of several parameters indicating type geometry, dimension, and an array with the x, y, z coordinates.

3D objects can be organized in Geo-DBMS in different ways: using existing 2D data types or defining a new 3D data type.

4.4.2.1 Using 2D Data Types

Mainstream DBMSs maintain 2D data types (point, line, polygon) but with their 3D coordinates. Using 3D polygons, 3D objects can be represented as polyhedrons in two ways: as a list of data type *polygons* or as data type *multipolygon/collection*.

The first option (defining a 3D objects as a list of 3D polygons) will be completed by creating two tables: a table BODY and a table FACE. In the table BODY the 3D

spatial object is defined by a set of records containing a unique pointer to the faces closing the volume of a body. The table FACE contains geometry of faces stored as 3D polygons. This model is partly a topological model; since the body is defined by references to the faces (faces share two bodies).

In the second representation, a body is stored, using the data type *multipolygon/collection*, as one record. In this case, only one table is necessary, i.e., BODY. Note that the type of geometry used is not visible in the *create* statement.

An apparent advantage of the 3D multipolygon approach is the one-to-one correspondence between a record and an object. Furthermore, the 3D multipolygon (compare to a list of polygons) is that it is recognized as one object by front-end applications (GIS/CAD). For example, a 3D multipolygon is visualized as a "group" of objects in Microstation Geographics. However, in case of editing the objects, it still has to be ungrouped into composing faces (Figure 4.4).

A disadvantage of both representations is the redundant storage of coordinates (also in the case of 3D multipolygons). The coordinates of vertex of the body are stored at least three times either in the 3D multipolygon or in the FACE table.

Moreover, both representations are not recognized by DBMS as a volumetric object, i.e., they are still polygons and, thus, the 3D objects cannot be validated (as specified in Section 4.3.1). The objects can be indexed as 3D polygons but not as 3D volumetric objects.

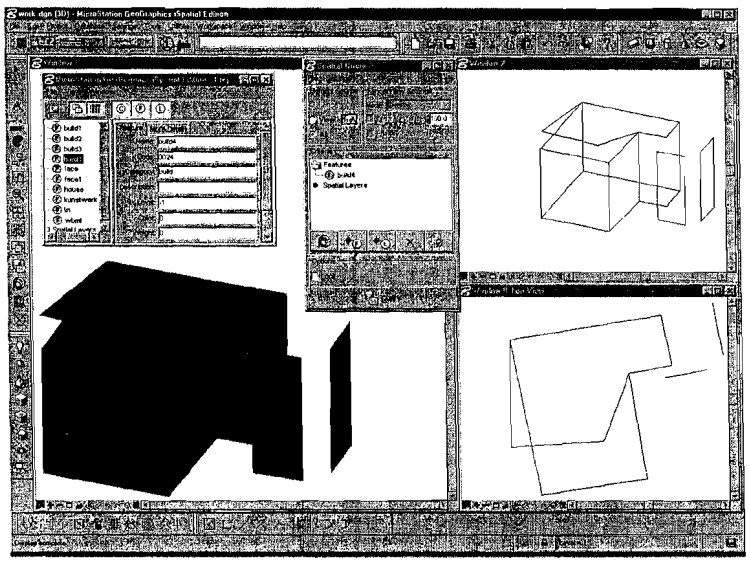


FIGURE 4.4 Visualization of 3D object stored in the Oracle Spatial.

4.4.2.2 Using a New 3D Data Type

The only way to resolve these drawbacks is by defining a true 3D data type. Arens at el. (2003) implemented a 3D data type polyhedron (as described in Section 4.3.1.3) using the Oracle Spatial object relational model. The new data is given sdo_gtype = 3008. Instead of listing all the coordinates composing a face, the structure of mdsys.sdo_ordinate array is suggested to have two parts — one with the coordinates of all the vertices (listed only once) and, second, a list with references to the triple coordinates.

The rules for validity of polyhedra are used as basis for a validation function. This function as well as some other interesting functions such as 3D area, volume, etc., are implemented and tested with different data sets. More details can be found in Arens (2003).

4.5 CASE STUDIES

Two case studies from man-made objects and nature-made objects reveal some of the problems with maintaining large-scale 3D objects, which are used to draw conclusions and requirements for 3D developments.

4.5.1 AEC Case Study, "Modeling the Interior of the Aula, the Congress Center of TUDelft"

3D modeling of man-made objects usually results in large amounts of data and has always required careful consideration of details (with or without texture), thematic and spatial structuring of individual sections of large public buildings, etc. With the advances of technology (as discussed in Chapter 2), new sensors and 3D reconstruction methods are becoming available, which allows complex construction to be modeled with centimeter accuracy. This progress brings new opportunities for creating high-resolution models of real-world objects, which look as detailed as the construction plans at design stage.

For the purpose of experimenting with 3D indoor navigation, a 3D interior model of the congress Centrum (the Aula) at TUDelft has to be created (Figure 4.5). The entire Aula building was scanned from inside. This resulted in a 25 million point cloud obtained from 237 scans (Figure 4.6). Such a data set poses new challenges to the database. It was decided to organize the point cloud in Oracle Spatial for the purpose of simplifying the modeling process and preserving the points for later use. It should be noticed, that for the purpose of 3D navigation, only walls, floors, ceilings, windows, and doors were modeled. Many other objects such as tables, chairs, plants, curtains, etc., were left out. However, the points from these objects still can be appropriately organized in the database for future consideration. Furthermore, a proper organization of points (segmentation into scans or rooms, floors) would allow many users to have access to only a part of the point cloud, which will simplify the modeling process. Finally, each modeled object (wall, room, etc.) will know the points it was created from.

Tests with this point cloud revealed drawbacks of the offered data types. The supported data type for a point is very expensive; one point is stored in one record.

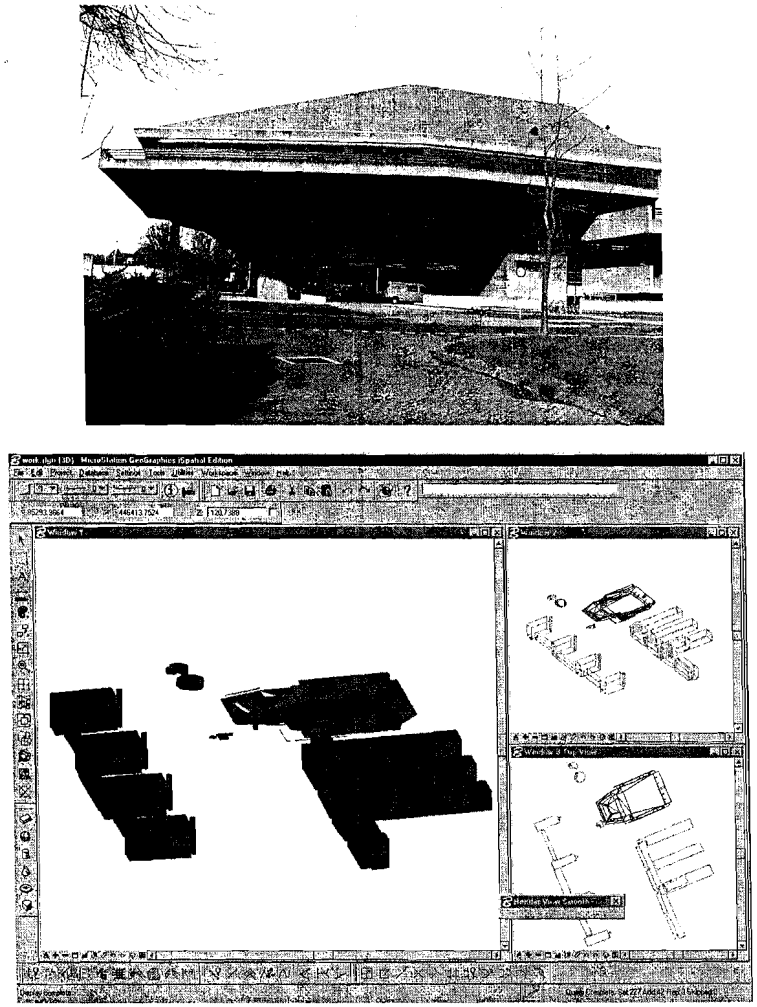


FIGURE 4.5 TUDelft, the Aula: photo (above) and 3D model (below).

Operations on such a point cloud are time-consuming (loading, 10 hours; indexing, 13 hours; etc.). Spatial indexing is not optimal for point data types. If multiple point data types are used, the identity of the points are lost. Apparently, point data type has to be reconsidered, or a new point data type has to be created.

FIGURE 4.6 (See color insert after page 86.) Point cloud of the scanned interior of the Aula.

Furthermore, many sections of the interior can be modeled in CAD software (Cyclone, CloudWorx) with predefined primitives such a cones, cylinders, spheres, etc. Such objects, however, cannot be organized later in the database. Data storage of each representation can be based on free-form shapes, NURBS, Bezier, or B-splines as suggested in Section 4.3.1.4. Storage of NURBS requires space nearly twice as large as Bezier or B-spline. However, certain costs in data storage should be acceptable in order to get more realistic results.

4.5.2 GIS Case Study, "3D Geological Modeling of an Open Cast Mine in the Lower Rhine Basin"

The following sections summarize the description first given by Thomsen and Siehl (2002). Geological objects are, by nature, three-dimensional. In the case of almost undisturbed, and not much inclined strata, 2.5-D "flying carpet" models using a stack of stratum boundary surfaces as basic elements of a geometry model may be sufficient. But in the general case of folded and faulted strata, or of intrusionary orebodies, salt domes, etc., only a true 3D model is adequate. As an example, the 3D modeling of the Bergheim open cast mine near Cologne, Germany, is presented (Figure 4.7). For further details of this application, we refer to Thomsen and Siehl (2002).

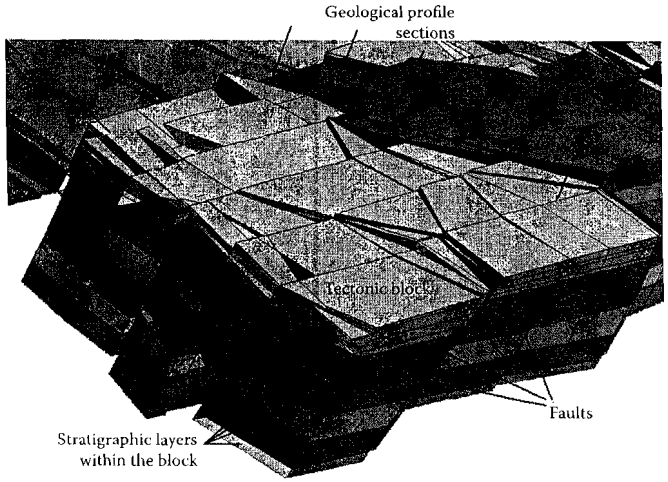


FIGURE 4.7 Structure of the Bergheim 3D geometry model.

This 3D model has been constructed to support a new kind of geological map. The objective is to gain deeper insight into the structural evolution of the Lower Rhine Basin from the Tertiary period until today, by modeling the kinematics of the Bergheim lignite mine, a small and intensely faulted region with an extension of about 2 km and a maximal depth of 500m.

The Bergheim open cast mine 3D model was designed to accommodate about 400 fault blocks, with about 2000 stratum surfaces, 800 fault surfaces, and 800 artificial boundary surfaces, each surface consisting of a small number of triangles.

For future spatio-temporal analysis (Breunig, 2001), an estimated total of 10,000 time-dependent surfaces have to be managed with an estimated 50,000 small timestep surfaces. Whereas the Bergheim kinematic model may serve as an example of a very discontinuous space-time model, experience from other static 3D-models of parts of the Lower Rhine Basin showed that individual stratum surfaces may comprise as many as 100,000 triangles.

Tectonic blocks are separated by layer boundaries, fault surfaces, and artificial boundaries at the location of the geological sections. Each block combines three types of triangulated surfaces, namely layer boundaries, faults, and artificial boundaries, following the original geological profile sections.

Starting with 20 parallel geological sections in SW–NE direction at a scale of 1:2000 supplied by Rheinbraun AG, Cologne, the gOcad 3D® modeler (Mallet, 1992) was used to construct the 3D volume model. For the data management, an extension of the 3D Geo-DBMS kernel GeoToolKit (Balovnev et al., 2004) was used, based on OODBMS ObjectStore®.

The geoscientific 3D geometry model must obey certain *consistency constraints,* whether strict geometrical and topological (e.g., the exclusion of gaps and overlaps between neighboring objects) or weaker and more problem specific (e.g., inverse correspondence of age and vertical position of strata, with certain exceptions). Therefore the Geo-DBMS must enforce *geometric and topological consistency* between strata, faults, and other geological objects. It should also permit formulating integrity checks based on application specific constraints.

The 3D topology and geometry model of the Bergheim open cast mine are implemented in GeoToolKit, using an extension with 3D data types in boundary representation. Thus, the topology and geometry of the geological profile sections, faults, and tectonic blocks are implemented by GeoToolKit classes, inherited by persistent ObjectStore classes.

The 3D geometric operations are realized as methods of GeoToolKit's 3D topological and geometric classes. This means that they are narrowly coupled with the OODBMS, also using ObjectStore's clustering and object management facilities. The methods are embedded into calls of 3D database queries realized with the C++ API of ObjectStore. Until now, no optimization rules determining the order of their execution have been implemented. This would be a difficult task, because the execution of arbitrary 3D geometric operations would lead far beyond the rules known from relational algebra applied in relational or object-relational DBMSs. To support the efficient execution of 3D database queries, the 3D geometric intersection operations, for example, are using an R*-tree index internally. Thus, the search of the "entry point" in the intersection algorithm, i.e., searching the first two intersecting primitive geometries (triangles or tetrahedra), is executed efficiently.

As GeoToolKit does not provide a complete spatial query language, the 3D database queries are implemented in C++ code supported by the object browser and the 3D visualization interface. Typical 3D database queries being implemented for this application scenario are:

- "Return the spatial part of the subsurface model specified by the 3D query box."
- "Compute the distance between all faults and the geological profile section No. 10A."
- "Return the fault surfaces intersecting the tectonic block No. 4."

To support a flexible client–server connection between geoscientific applications and the Geo-DBMS, a client interface was developed in Java, using CORBA technology (OMG, 1998) and an extension to a commercial VRML browser for visualization of geometry (Shumilov, et al., 2002; Thomsen and Siehl, 2002).

4.6 SUMMARY AND OUTLOOK: TOWARD BRIDGING AEC AND GIS

In the last several years, many changes affected DBMS toward providing spatial functionality. Still, many developments at the DBMS level have to be completed, e.g., a 3D geometrical model should be fully supported by DBMS, based on standardized

specifications. The native support of a polyhedron by DBMS will have significant consequences for 3D modeling in the coming years. Many questions related to CAD-like features have to be addressed: how to maintain curved faces, how to store parametric shapes (cone, sphere, etc.), how to store texture and other components used to create realistic 3D scenes, mechanisms to maintain levels of detail in DBMS, etc.

Maintenance of topology (especially 3D topology) at the DBMS level is still in an infant stage. In both application fields, GIS and AEC, there are no agreements on a standard 3D topological model. The current implementations are mostly 2D with small exceptions (e.g., radius topology, which uses z-tolerance). The problems with the unification of 3D topology models are relatively high compared to the 2D topology. It is likely that several topologies have to be maintained in the database. The description of the models and transition from one to another may be described in meta-data tables.

Learning from the case studies, we can summarize the following requirements to 3D Geo-DBMS:

- 3D object management of 0D-, 1D-, 2D-, and 3D-objects, such as point clouds of buildings, drillings, profile sections, faults, and tectonic blocks
- 3D data visualization for whole scenes and for results of 3D database queries
- Database support for spatio-temporal planning and processes such as city planning and geological analysis of kinematic modeling

Obviously, most of the requirements cannot be accomplished by today's standard DBMSs.

Clearly DBMS can be used as a bridge between AEC and GIS applications. Currently, many AEC vendors (Bentley, AutoCAD), as well as many GIS vendors (ESRI, MapInfo, Intergraph), use the spatial models of Geo-DBMSs. Although many interoperability issues (maintenance of color, texture, attributes) still have to be developed, the experiments show promising results, i.e., models stored on Geo-DBMS can be visualized and edited from both GIS and AEC applications.

To be able to provide a stronger management of objects from AEC and GIS, Geo-DBMSs have to extend their spatial support to accommodate design objects (coming from AEC) and real-world objects (considering new data collection and modeling techniques). Some of the required developments are listed below:

- Real 3D geometry types and corresponding 3D validity operations: Many objects coming from AEC applications are well-defined closed spaces and their integrity has to be recognized by Geo-DBMSs.
- Free-form curves and surfaces: AEC applications have a large number of free-form curves and surfaces, basically not used in GIS. To be able to combine 3D design models (modeled in CAD) with existing situations (modeled in GIS), specialized data types for free-form surfaces are needed. Among a large amount of mathematical representations of 3D space, NURBS could be one possible solution. NURBS are approved as industry standards for the representation and design of geometry.

- Point cloud data types: With the progress of data collection techniques, existing data types and spatial indexing appeared to be insufficient. Today, laser-scanned technology easily generates millions of points for a single building. DBMSs fail to efficiently handle such amounts of data.
- TIN data type: Most of the terrain representations presently maintained in GIS, as well as many CAD designs (meshes), are TIN representations. TINs can be stored in DBMS using the polygon data type. This data type is generally assumed for multiple vertices and thus contains many attributes. Thus, a simpler adapted data type is required.
- Maintenance of multiple representations to be used as LOD: The management of multiple representations is still far from formalized. Users take care of their own implementations, which, however, cannot benefit from the indexing mechanisms in the Geo-DBMS.
- Management of texture and mechanism for texture mapping and texture draping: 3D objects usually need more physical attributes compared to 2D objects. Often 3D objects are textured with images from the real world. As AEC and GIS applications come together, the question of linking textures to geometrics will appear. Textures have to serve as attributes of 3D objects decoded in the data types.

Questions such as 3D functionality are the next to be considered. It should not be forgotten that Geo-DBMS, in the first place, is a DBMS, i.e., the location for storage and management. The 3D functionality should not be completely taken away from GIS and CAD/AEC applications. 3D Geo-DBMSs should provide the basic (simple) 3D functions, such as computing volumes and finding neighbors. Complex analysis has to be attributed to the applications.

REFERENCES

Aguilera, A., *Orthogonal polyhedra: study and application*, Ph.D. Thesis, Universitat Politechica de Catalunya, Barcelona, Spain. 2001.

Arens, C.A., Maintaining reality, modelling 3D spatial objects in a Geo-DBMS using a 3D primitive, M.Sc. Thesis, TU Delft, The Netherlands, available at http://www.gdmc.nl/, 2003.

Arens, C., Stoter, J., and van Oosterom, P., Modelling 3D spatial objects in a Geo-DBMS using a 3D primitive, *Computers & Geosciences*, 31 (2), 165–177, 2005.

Balovnev, O., Bode, T., Breunig, M. Cremers, A.B., Müller, W., Pogodaev, G., Shumilov, S., Siebeck, J., Siehl, A., and Thomsen, A., The story of the GeoToolKit — an object-oriented geodatabase kernel system. *GeoInformatica*, 8 (1), 5–47, 2004.

Beckmann, N., Kriegel, H.P., Schneider, R., and Seeger, B., The R-tree: An efficient and robust access method for points and rectangles, *SIGMOD Conference*, 322–331, 1990.

Breunig, M., *On the Way to Component-Based 3D/4D Geoinformation Systems. Lecture Notes in Earth Sciences*, Springer, Heidelberg, 2001.

Breunig, M., Thomsen, A., and Bär, W. Advancement of Geoservices — Services for Geoscientific Applications Based on a 3D-Geodatabase Kernel. Geotechnologies Science Report No. 4, Information Systems in Earth Management, Potsdam, Germany, 35–39, 2004.

Breunig, M., Bode, T., and Cremers, A.B., Implementation of elementary geometric database operations for a 3D-GIS, in *Proceedings of the 6th International Symposium on Spatial Data Handling SDH*, Edinburgh, Scotland, 604–617, 1994.

Coors, V., 3D GIS in Networking Environments, CEUS, 27 (4), 345–357, 2003.

David, B., Rayal, L., Schorter, G., and Mansart, V., GeO2: why objects in a geographical DBMS? in *Proceedings of the 3rd International Symposium on Large Spatial Databases*, Singapore, 264–276, 1993.

de la Losa, A. and Cervelle, B., 3D topological modelling and visualisation for 3D GIS, *Computer&Graphics*, Vol. 23, 1999.

Eckel, B., *Thinking in Java*, 3rd Edition, Prentice Hall, free electronic book, http://www.BruceEckel.com, 2002.

Egenhofer, M.J., A formal definition of binary topological relationships, in *Foundations of Data Organisation and Algorithms, Proceedings FODO*, Paris, Lecture Notes in Computer Science No. 367, Springer, Berlin, 457–472, 1989.

Egenhofer, M.J., Frank, A.U., and Jackson, J.P., A topological data model for spatial databases, in *Proc. 1st Symp. on Design and Implementation of Large Spatial Databases*, Vol. 409 of Lecture Notes in Computer Science, Buchmann, A., Günther, O., Smith,T.R., and Wang, Y.-F., Eds., Springer, Berlin, 271–286, 1990.

Günther, O. Efficient structures for geometric data management. *Lecture Notes in Computer Science* No. 337, Springer, Berlin, 1988.

Günther, O. Spatial databases (in German), *Informatik Spektrum*, 14 (4), 218–220, 1991.

Güting, R.H., An introduction to spatial database systems, *VLDB Journal*, 3 (4), 357–399, 1994.

Guttman, A., R-trees, a dynamic data structure for spatial searching, in *ACM SIGMOD* 13, 47–57, 1984.

Levy, B., *Modélisation à base topologique: Combinatoire et Plongement*, Ph.D. Thesis, Institut National Polytechnique de Lorraine, Nancy, France, 1999.

Lienhardt, P., N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *Journal on Computational Geometry and Applications* 4 (3), 261–274, 1994.

Mallet, J.-L., GOCAD — a computer aided design program for geological applications, in *Three-Dimensional Modeling with Geoscientific Information Systems. NATO ASI Vol. 354*, Turner, A.K., Ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, 123–142, 1992.

Mallet, J.-L., *Geomodeling*, Oxford University Press, Oxford, U.K., 599 S, 2002.

Molenaar, M., A formal data structure for 3D vector maps, in *Proceedings of EGIS"90*, Vol. 2, Amsterdam, The Netherlands, 770–781, 1990.

ODMG, *Cattell RGG, the Object Database Standard: ODMB-93*. Morgan Kaufman Publishers, San Mateo, CA, 1993.

OMG, Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 2.2., 1998.

Piegl, L, On NURBS: a survey, *IEEE Computer Graphics and Applications*, 11 (1), 55–71, 1991.

Pigot, S., *A topological model for a 3-dimensional, Spatial Information System*, Ph.D. Thesis, University of Tasmania, Australia, 1995.

Pilouk, M., *Integrated modelling for 3D GIS*, Ph.D. Thesis, ITC, The Netherlands, 1996.

Requicha, A.A.G. and Voelcker, H.B., Solid Modeling: A Historical Summary and Contemporary Assessment. *IEEE Computer Graphics and Applications*, 9–24, 1982.

Rogers, D.F and Earnshaw, R.A., Eds., *State of the Art in Computer Graphics — Visualization and Modeling*, Springer-Verlag, New York, 1991.

Schek, H.-J. and Waterfeld, W., A database kernel system for geoscientific applications. in *Proceedings of the 2nd Symposium on Spatial Data Handling SDH*, Seattle, 1986.

Shumilov, S., Thomsen, A., Cremers, A.B., and Koos, B., Management and visualisation of large, complex and time-dependent 3D objects in distributed GIS, in *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems*, McLean, VA, 2002.

Stoter, J. and Zlatanova, S., Visualisation and editing of 3D objects organised in a DBMS, *Proceedings of the EuroSDR Com V. Workshop on Visualisation and Rendering*, 22–24 January 2003, Enschede, The Netherlands, 2003.

Thomsen, A. and Siehl, A., Towards a balanced 3D kinematic model of a faulted domain — the Bergheim open pit mine, Lower Rhine Basin, *Netherlands Journal of Geosciences/ Geol. Mijnbouw*, 81 (2), 241–250, 2002.

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

Vijlbrief, T. and van Oosterom, P., The Geo++ System: an extensible GIS, in *Proceedings of the 5th International Symposium on Spatial Data Handling SDH*, Charleston, SC, Vol. 1, 44–50, 1992.

Waterfeld, W. and Breunig, M., Experiences with the DASDBS Geokernel: Extensibility and Applications, in *From Geoscientific Map Series to Geo-Information Systems*, Geolog. Jahrbuch, A(122), Hannover, Germany, 77–90, 1992.

Zlatanova, S., Holweg, D., and Coors, V., Geometrical and topological models for real-time GIS, *International Workshop on Next Generation Geospatial Information*, 19–21 October 2003, Cambridge, MA, 2003.

Zlatanova, S., Rahman, A.A., and Shi, W., Topological models and frameworks for 3D spatial objects, *Journal of Computers & Geosciences*, 30 (4), 419–428, 2004.