Three-Dimensional Spatial Data Types

[1]Sandeep D Bansode, [2]Mrunal D Bhosale
[1]Lecturer , MCA dept., Ratnagiri
[2]Developer, Savy Software sol., Pune
[1]sandeep.bansode86@gmail.com, [2]mrunalbhosale1@gmail.com
[1]9975718997, [2]9921044179

Abstract:-Database management system have significantly changed in the last several years from a system dealing with management of administrative data they have involved to a spatial DBMS providing spatial data types, spatial indexing and extended spatial functionality. Spatial database system as database system that offers spatial data types in its model and query language and supports spatial applications. Spatial database is mainly deal with 2-D dimensionaldata (geometric data). This is sufficient for many spatial application. But on the other hand our world is 3-D and a number of possible application can be identified that would be a greatly benefit from a treatment of 3-D data. Although the provided functionality is limited to the second dimension, various options exist for management of three-dimensional data.

The idea of this topics (Research paper) is 3-D geometric information of spatial database. Our research has led so far to a called abstract model that identifies the essential 3-D spatial data-types.

I. INTRODUCTION

DBMS has traditionally used to handle large volume of data and to ensure the logical consistency and integrity of data, which also have major requirements in handling spatial data.

What is spatial database system?

1) A spatial dB system is a database system
2) It offers spatial data types in its data model and query language.

Evolving to Spatial DBMS: [3]

DBMS have been traditionally used to handle large volumes of data and to ensure the logical consistency and integrity of data, which also have be-come major requirements in handling spatial data. For years, spatial data used to be organized in dual architectures consisting of separated data management for administrative data in a Relational DBMS (RDBMS) and spatial data in a GIS. This is to say spatial data has been managed in single files using proprietary formats. This approach couldeasily result in inconsistency of data, e.g. when deleting a record in the database no mechanism exists to check the corresponding spatial counterpart. A solution to problems of dual architecture was a layered architecture, in which all data is maintained in a single RDMS.

Since spatial data types were at that time not supported at DBMS level, knowledge about spatial data types was maintained in middle ware. Presently, most mainstream DBMS offer spatial data types and spatial functions usually in an object-relational spatial extend to RDBMS Storing spatial data and performing spatial analysis can be completed with SQL queries. Additionally, integrated queries on both spatial and non-spatial parts of features can be executed at database level. The spatial data types and spatial operations reflect only simple two-dimensional features, though embedded in 3D space. This support of 3D/4D coordinates allows for alternatives in management of 3D features. This paper elaborates on current possibilities of DBMS to maintain 3D data. The next section discusses management and visualisation of volumetric objects, 3D lines and 3D points. Then the paper reports on prototype implementations of new data types completed at section GIS. Standardization activities within Open Geospatial Consortium are briefly outlined with respect to resent new initiatives. Last section concludes on demands and expectations to the 3D geometry model.

II. 3-D DATA IN THE DBMS USING CURRENT TECHNIQUES[3]

Providing the spatial extend, DBMS have immediately been challenged by the third dimension. A number of experiments were performed by several researchers to investigate possibilities to store, query and visualize features with their 3D coordinates. The good news is: 3D data can be organized in DBMS, retrieved and rendered by front-end applications. However, there is also a bad news: since no 3D data type is currently

Supported by any DBMS, the user remains self-responsible for the validation of the objects as well as for implementing true 3D functionality. These conclusions, with small variations, are consistent for all main-stream DBMS: Oracle, IBM DB2, Informix, Ingres, and MySQL. All of them offer 2D data types (basically point, line, and polygon) but support 3D/4D coordinates (except Ingres,

which is 2D) and offer a large number of functions more or less compliant with the Open Geospatial Consortium (OGC) standards. Most of the functions are only 2D dimensional (except, which supports few 3D operations), i.e. although not reporting error, they omit the z-coordinate.

A bit frustrating is the implementations of spatial functions and operations: it varies with the DBMS. The statement: select c from b where a < 100, where c, aare numerical data type, can be executed in every DBMS. However if c is a spatial data type and a is a given distance, the SQL statement becomes de-pended on the specific implementation. In some cases (e.g. Oracle Spatial), even the names of the spatial data types are not that apparent. Oracle Spa-tial has one complex data type sdo_geometry composed of several parameters indicating type geometry, dimension, and an array with the x, y, and zcoordinates. The text bellow discusses possible organisation of the 3D real-world features (volumetric, line, point) in current DBMS. Note, the presented approaches are not dependent on the DBMS.

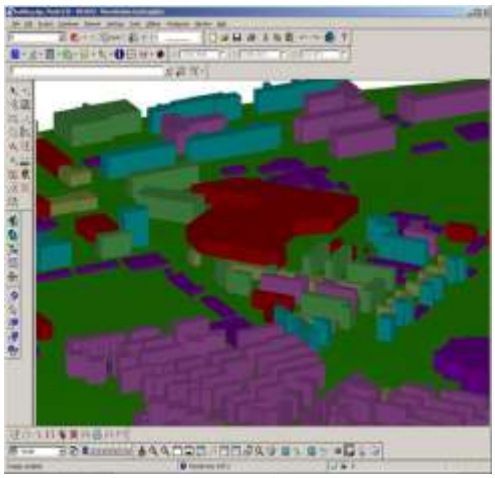

Figure 1: Visualisation of buildings and surface, Represented by simple polygons in Oracle Spatial

1. 3-D volumetric objects

Most discussed 3D features are volumetric objects, which can be used for modelling of man-made objects, such as buildings, and nature-made objects, such as geological formations. To have those managed in the database, the user can choose between:

1) Using DBMS data types polygon and multipolygon

OR

2) Creating a used-defined data type.

The three candidates for a simple volumetric object are polyhedron, triangulated polyhedron and tetrahedron (see for definitions next section) and all three can be easily realized with provided data types. Moreover, there is no practical difference in the implementation of the polyhedron and triangulated polyhedron, since a separate triangle data type does not exist. Tetrahedron would allow for a bit simpler representation since it has only four trianglesthe first option, i.e. defining a 3D feature as a list of polygons can be realized by two columns in one relational table, i.e. feature_ID and a column for the spatial data type (i.e. polygon). If the reality is quite complex, leading to many 3D features with shaped polygons, the DBMS table should be normalized. This means that the polygons have to be organized in a separate table (containing polygon_ID and a column for the spatial data type) and the 3D feature table should contain the indices to the composing polygons. Clearly, the separate relational table for volumetric objects would be simpler if the volumetric object is tetrahedron. It can be organized in a table with finite number of columns: one for the ID of the tetrahedron and four for the composing polygons (triangles). In the second approach, a 3D object is stored using the data type multi-polygon, i.e. all the polygons are listed inside the data type, which is practically one record in the relational database. This case requires only one table, which may contain only two columns: feature_ID and column for the spatial data type. 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 list of polygons) is recognized as one object by front-end applications (GIS/CAD). For example, a 3D multypolygon is visualised as grouped polygons in Bentley Micro station. However, in case of editing, the group still has to be ungrouped into composing polygons i.e. the group is not recognised as 3D shape.

User defined spatial data types can be implemented using different approaches from the simple SQL create data type statement, to more complex implementations, using a Procedural Language (PL), Java, C++, etc. The common drawback of such an implementation is impossibility to apply the native spatial functionality (operations and indexing) of DBMS. Moreover the user-defined spatial data types cannot be stored in the same column of the natively supported spatial data types. Visualisation in front-end applications would be possible only by developing individual connections. User-defined spatial data types, however, are very useful for proto-typing for approval of new concepts.

2. 3-D line objects

Typical examples of 3D line objects are utility networks: pipeline and cable networks. Utility data and systems have been always predominantly two-dimensional. Only recently, investigations have been initiated towards maintaining utility networks in three dimensions. Motivation for this is extended usage of

underground space and therefore the apparent need of more sophisticated mechanisms for visualizing several networks in one environment.

Utility networks (represented as lines with 3D coordinates) can be readily managed in DBMS using the supported spatial data types line or multi-line. If required by an application, some point objects (valves, connectors, etc.) can be separately organized as points. The only trouble of 3D lines is the visualisation in 3D scenes. It is often recommended 3D lines to be substituted with tiny cylinders when rendered. Indeed, such a substitution cannot be justified only for visualisation purposes. Therefore Du and Zla-tanova 2006 suggest keeping the original data as 3D lines and creating 3D cylinders on the fly only for the visualization (Figure 2).

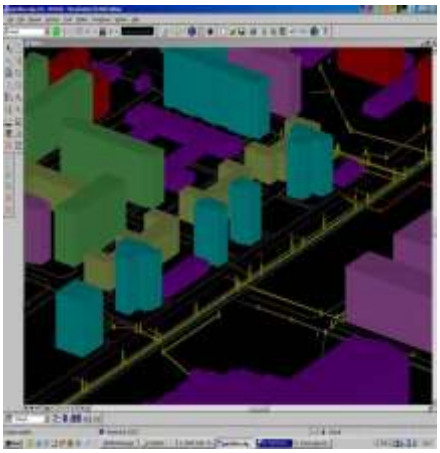

Figure 2: 3D visualisation of pipelines,

Organised as lines in Oracle Spatial

3. 3-D point clouds

Until recently, 3D point objects were relatively rare in real-world data sets and a little attention was given on them. But, with the advances of sensor technology, laser scanning techniques become available, which produce large amounts of very specific 3D point data. Theoretically, these points can also be organised in DBMS by either

1) Using the supported spatial data types point (Figure 3) and multipoint or

2) Creating a user-defined type. Depending on the type of data (row or processed data), the user might decide for either of the representations.

Usually the most common format of processed laser scan data consist of seven parameters: x, y, z-coordinates, intensity and colour represented by r, g, and b-values. The advantage of point data types is possibility to manage all these attributes for each individual point. The major disadvantage refers data storage and indexing, which are very expensive (one record per point). The multipoint data type can be efficiently indexed, but the points lose their identification, which might be important for modelling purposes. Further-more, the number of points in one multipoint has to be carefully considered for an acceptable performance. Depending on the point distribution and size of the point cloud, the operations can become time consuming and thus difficult to handle.

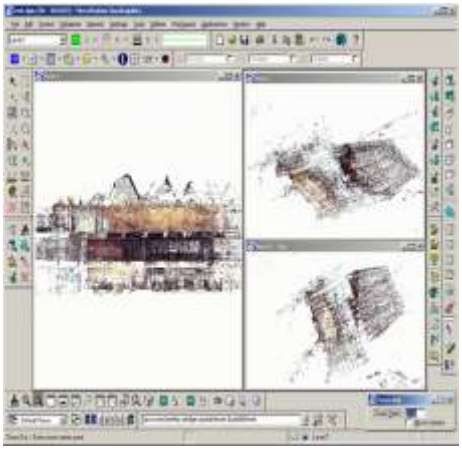

Figure 3: 3D visualisation of point clouds,

Managed as points in DBMS

III. NEW 3D SPATIAL DATA TYPES [3]

As shown above, 3D real-world feature can be stored and indexed in DBMS and retrieved for visualisation and editing in front-end application but they can be analysed only as 2D features. A true 3D geometry data type (polyhedron and/or tetrahedron) and corresponding 3D spatial operations (validation, volume, length, intersection, etc.) are missing in all DBMS. Furthermore, the simple 3D volumetric data type would be still in-sufficient for handling many shapes (cone, sphere) available in AEC/CAD applications.

The sections bellow will briefly present two implementations of new 3D data types, i.e. polyhedron and NURBS surface.

1. 3-D polyhedron

A 3-D spatial data type is the first most important development to be made by DBMS. A simple 3D object can be represented basically in three different ways as a polyhedron (consisting of arbitrary number of planar polygons which have arbitrary number of points), triangulated polyhedron (consisting of an arbitrary number of triangles) or tetrahedron (composed of four triangles). All the three representations have advantages and disadvantages. Tetrahedron is the simplest 3D object consisting of a finite number of points and triangles. While advantageous for computations and consistency check , tetrahedrons are less appropriate for modelling of man-made objects such as buildings, bridges, tunnels, etc., because the interior would require a subdivision into tetrahedrons (which should be omitted for visualisation).

However, tetrahedrons are widely used in modelling geological formations. The polyhedron is the most appropriate representation for man-made objects, but its validation is quite complex. Triangulated polyhedron is compromise between the two ensuring at least the simplicity of the polygons.

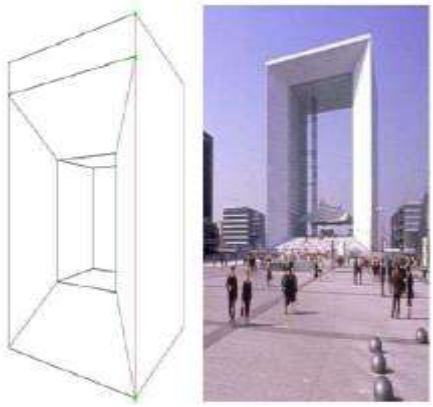

Figure 4: 3D polyhedron (Arens 2003) [1]

Arens 2003, and Arens etal 2005, have selected a polyhedron for implementation, since it is the most complex data type requiring strict validation rules. A polyhedron is defined as a bounded subset of space, enclosed by a finite set of planar polygons (not self-intersecting) such that every edge of a polygon is shared by exactly one other polygon. The polyhedron bounds a single volume, i.e. from every point on the boundary, every other point on the boundary can be reached via the interior. The polyhedron has clearly defined inside/outside space, i.e. it is Orientale. The polyhedron defined in this way can have also cavities (Figure 4). The polyhedron data type is implemented in Oracle Spatial object-oriented data model, but the formalism is generic.

To avoid duplications of point coordinates (As men-tioned above), the description has two sections:

1) A list of all the point co-ordinates and

2) A sequence of polygons, each composed of a list with in-dices to the point coordinates of the first section.

The validity of the new data type is controlled by a specially designed function, which checks the definition rules. Several tests were performed on the new data type and the results were positive.

2. 3-D freeform curves and surfaces

Freeform curves and surfaces such as Bezier, B-spline and NURBS, are becoming progressively important for modelling of buildings, towers, tunnels, etc. Very often these models need to be integrated with 3D GIS for investigations and adjustment of the construction (e.g. for wind resistance). One option for such an integrated environment can be DBMS.

NURBS [2] is the first candidate to be considered. Some of the most important NURBS characteristics are

- NURBS offer a common mathematical form for both, standard analytical shapes (e.g. cone, sphere) and free form shapes,
- The shapes described by NURBS can be evaluated reasonably fast by numerically stable and accurate algorithms,
- Important characteristic for modelling real-world objects is that they are invariant under affine as well as perspective transformations.

The only drawback of NURBS is the extra storage needed to define traditional shapes (e.g. circles). Using NURBS data types, a circle can be represented in different ways but the complexity is much higher compared to its mathematical definition (i.e. radius and centre point). The definition a NURBS curve consists of several quite complex equations, which can be found elsewhere in the literature.

The new data type has been prototyped for Oracle Spatial, but outside the Oracle Spatial SDO_GEOMETRY model, which means it can be readily used for any spatial DBMS (MySQL, Informix, etc.). Besides the validation function, few simple spatial functions (rotation, translation, scale and distance) were developed. Since the data type is much more complex compared to the 3D polyhedron data type, a special care was taken for the visualisation in AEC software, i.e. an engine was developed to map the NURBS data type to the internal representation of Micro station and AutoCAD (Figure 5). Two NURBS models were tested with the developed data type for retrieval, editing and posting.

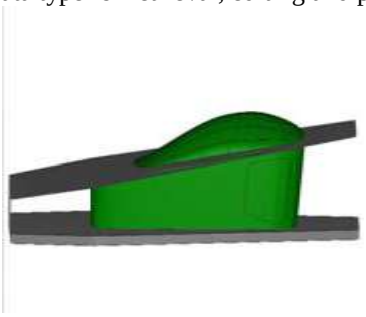

Figure 5: NURBS building retrieved from DBMS

CONCLUDING REMARKS

In the last five years DBMS made a large step toward maintenance of geometries as GIS used to manage them. The support of 2D objects with 3D coordinates is adopted by all mainstream DBMS. The offered functions and operations are predominantly in the 2D domain. The DBMS spatial schemas have to be extended to fully

represent the third dimension (first with simple volumetric object and later with more complex 3D data types). Concepts for 3D objects and prototype implementations are already reported, DBMS have to make the next step and natively support them. 3D operations and functions have to be developed not only for the volumetric object but also for all other objects embedded in 3D space. 3D functionality is next to be considered. The 3D functionality should not be completely taken away from front-end applications such as GIS and CAD/AEC. 3D Spatial DBMS should provide the basic (generic) 3D functions, such as computing volumes and finding neighbours.

Some existing data types are clearly not sufficient for the purpose of some applications. A very typical example is the mutypoint. It was definitely not designed for large amounts of points as from laser scanning. DBMS fail to handle efficiently such amounts of data until now. Such points need a special treatment. Triangle (or TIN) data type is also quite demanded. It is likely that TIN will continue to be widely used for all kinds of complex surface representations in GIS. 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.

Management of texture and mechanism for texture mapping and texture draping is critical for management of realistic 3D City models. 3D objects usually need more attributes for visualisation compared to 2D objects. 3D Spatial DBMS has to offer an appropriate 3D user interface. To date, 3D user interfaces has not yet been exhaustively examined. Future 3D query interfaces should support the formulation of complex SQL-like mixed spatial and non-spatial database queries as well as 3D graphical in-put supporting the intuitive graphical formulation of 3D queries.

REFERENCES

[1].Arens, C, 2003, Maintaining Reality, Modelling 3D spatial objects in a Geo-DBMS using a 3D primitive, Master's Thesis TU Delft, 2003.

[2].Piegl, L. and Tiller W., 1997, the NURBS Book 2nd Edition, Springer-Verlag.

[3].3D geometries in spatial DBMS by
Sisi Zlatanova GISt, Delft University of Technology, the Netherlands.