Implementing metric operators of a Spatial Query Language for 3D Building Models: Octree and B-Rep approaches

André Borrmann[1], Stefanie Schraufstetter[2], Ernst Rank[3]

Abstract

A Spatial Query Language for Building Information Models enables the spatial analysis of buildings and the extraction of partial models that fulfill certain spatial constraints. Among other features, the developed spatial query language includes metric operators, i.e. operators that reflect distance relationships between spatial objects, such as *mindist, maxdist, isCloser* and *isFarther*. The paper presents formal definitions of the semantics of these operators by using point set theory notation. It further describes two possible implementation methods: the first one is based on a discrete representation of the operands' geometry by means of the hierarchical, space-partitioning data structure octree. The octree allows for the application of recursive algorithms that successively increase the discrete resolution of the spatial objects employed and thereby enables the user to trade-off between computational effort and the required accuracy. By contrast, the second approach uses the exact boundary representation (B-Rep) of both spatial objects resulting in precise distance measurements. Here the bounding facets of each operand are indexed by a so-called axis-aligned bounding boxes tree (AABB tree). The algorithm uses the AABB-tree structure to identify candidate pairs of facets, for which an exact but expensive distance algorithm is employed. The article compares both

[1] Dr.-Ing. André Borrmann, Computation in Engineering, Technische Universtät München, Arcisstrasse 21, 80290 Munich, Germany, borrmann@bv.tum.de
[2] Ms. Stefanie Schraufstetter, Computation in Engineering, Technische Universität München, Arcisstrasse 21, 80290 Munich, Germany, schraufstetter@bv.tum.de
[3] Prof. Dr. rer.-nat. Ernst Rank, Computation in Engineering, Technische Universität München, Arcisstrasse 21, 80290 Munich, Germany, rank@bv.tum.de

approaches by means of detailed investigations on the runtime performance of the developed algorithms.

Keywords: Building Information Model, Spatial Query Language, Metric relationships, Distance, Octree, AABB-Tree

Introduction

This paper presents partial results of the research project "A 3D Spatial Query Language for Building Information Models". The proposed language provides directional, topological and metric operators for specifying spatial conditions. While the directional operators are presented in (Borrmann and Rank, 2008a) and the topological operators are addressed in (Borrmann and Rank, 2008b), this paper discusses the definition and implementation of the metric operators.

Humans view buildings primarily as an aggregation of physical objects with well-defined geometry and specific spatial relationships. In most cases, the architectural and/or structural function of a particular building component is closely related to its shape and its position in relation to other building components. For architects and engineers involved in designing buildings, geometric properties and spatial relationships between building components accordingly play a major role in finding solutions for most of the design and engineering tasks. However, software tools that allow for a sophisticated spatial analysis of digital building models are not yet available.

The current lack of building model management software supporting geometric-topological analysis can be explained by the fact that, over the last decade, research in the field of

computer-aided building design has concentrated mainly on the development of a semantic object-oriented building model, also called Product Model or Building Information Model (BIM) (Dubois et al., 1995; Scherer, 1995; Tolman and Poyet, 1995; Eastman, 1999). These efforts have resulted in the widely known ISO standard Industry Foundation Classes (IFC) (International Organization for Standardization, 2005).

Building product models, such as the IFC, do not normally describe the geometry of a building component explicitly, i.e. not by using the boundary representation (B-Rep) or the Constructive Solid Geometry (CSG) method, but by object attributes that have a geometric meaning. The main motivation behind this "attribute-driven geometry" approach is the scope that an abstract description of this kind provides for deriving both full three-dimensional models and two-dimensional drawings with partly symbolized representations, as stipulated in national building regulations and construction contracts.

Unfortunately, the existing product model servers that are utilized to store and manage building information models are unable to interpret the attribute-driven geometric information that is implicitly contained in the building model, since they are not familiar with the spatial semantics of particular attributes and relationships. Accordingly, the expressiveness of the query languages provided by the product model servers, such as the *Partial Model Query Language* (Adachi, 2003) of the Secom IFC Model Server or the *Product Model Query Language* of the EuroStep Model Server, is limited to numerical comparisons and tests on those spatial relationships that are predefined in the product data model.

In the case of the IFC, some examples of these predefined relationships are *IfcRelFillsElement, IfcRelVoidsElement* and *IfcRelContainedInSpatialStructure*.

Unfortunately, many product modeling tools do not fill the entire set of spatial relations with appropriate data when exporting a building model into the IFC format. In a recently conducted test, we used the commercial CAD tool *Autodesk Revit 2008* to model a high-rise building completely equipped with interior fittings. The analysis of the exported IFC file showed that, while the *IFCRelFillsElement* and *IFCRelVoidsElement* relationships between walls and windows were set correctly, no *IfcRelContainedInSpatialStructure* relationships had been set. Accordingly it was not possible to query the product model for the heating equipment contained in a certain room or office, for example.

Other spatial relationships, such as metric relationships (one object being at a certain distance from another, for example) are completely ignored by the IFC product model. From the point of view of product modeling this makes sense, because storing all possible spatial relationships would (1) result in huge models with many object relations not needed by the majority of applications and (2) introduce even more redundancy, since metric relationships are already implicitly defined by the shape and position of the respective objects. For this reason, we follow an analytic approach here, where spatial relationships do not need to be set by the modeling tool in use, but are derived from the objects' shapes and positions, i.e. the explicit geometry of the building's components, instead.

Another issue to be considered in the context of spatial analysis is that for many construction domains, such as tunnel, road or bridge engineering, only rudimentary product models exist (Yabuki et al., 2006) and have not been used in practice so far. On the other hand, pure 3D modeling is gaining more and more importance in these areas and, with it, the potential benefits of using spatial analysis tools.

In order to fill the technological gaps mentioned above, we have developed a spatial query language for 3D building and infrastructure models. We propose employing a query language as an interface to the spatial analysis facilities, because it allows a user or application developer to formulate spatial analysis problems in a declarative way while hiding all internal complexity. Furthermore, using a query language is the usual way to access database systems, which – thanks to their inherent multi-user and persistence capabilities – will naturally be the instrument of choice for hosting building models in the near future. The concept of spatial query languages is well established in the field of Geographic Information Systems (GIS), but was so far limited to two-dimensional models.

Possible applications for the developed 3D Spatial Query Language for Building Information Models range from the selection of specific building components to the verification of construction rules and the extraction of partial models that fulfill certain spatial constraints. Such a partial model resulting from a spatial query may serve as input for a numerical simulation or analysis, or might be made exclusively accessible to certain participants in a collaborative scenario.

The proposed 3D Spatial Query Language relies on a spatial algebra that is formally defined by means of point set theory and point set topology (Borrmann et al., 2006; Borrmann, 2007). Besides fully three-dimensional objects of type *Body*, the algebra also provides abstractions for spatial objects with reduced dimensionality, namely by the types *Point, Line* and *Surface*. This is necessary because building models often comprise dimensionally reduced entities. All types of spatial objects are subsumed by the super-type *SpatialObject*.

The spatial operators available for the spatial types are the most important part of the algebra. As mentioned above, they comprise metric (*distance, closerThan, fartherThan* etc.), directional (*above, below, northOf* etc.) and topological (*touch, within, contains* etc.) operators.

We see the development of a spatial query language for BIMs as a first step towards making higher spatial concepts directly available in computer-aided engineering tools. We expect that spatial modeling and processing will play an increasing role in future engineering systems.

Related work

Spatial query languages

The overall concept of providing a Spatial Query Language for analyzing Building Information Models is closely related to concepts and technologies developed in the area of Geographic Information Systems (GIS). Such systems maintain geographical data, such as the position and shape of cities, streets, rivers etc., as well as providing functionalities for the spatial analysis of this data. Due to the nature of this domain most GI systems only support spatial objects in two-dimensional space.

The first implementations of spatial query languages on the basis of the standard query language of relational databases SQL were also realized in the GIS context. In the late 80's, a multitude of different dialects were developed, including Spatial SQL (Egenhofer, 1987), KGIS (Ingram and Phillips, 1987), PSQL (Roussopoulos et al., 1988), TIGRIS (Herring et al., 1988), and GEOQL (Ooi et al., 1989). A good overview of the different dialects and the basic advantages of a SQL-based implementation is provided in (Egenhofer, 1992).

The GIS research community also coined the phrase Spatial Database to describe database management systems (DBMS) that provide spatial data types and spatial indexing techniques and thus allow for an easy and efficient access to spatial data (Rigaux et al., 2002; Shekhar and Chawla, 2003). There is now a wide range of commercial 2D spatial database systems, the most widespread ones being *PostGIS, Oracle Spatial* and *Informix Geodetic Datablade*. The majority of available spatial databases comply with the standard developed by the OpenGIS consortium that defines a common interface for accessing 2D spatial data and accordingly enables the exchangeability of the database component in an overall GI system (OGC, 1999).

In (Ozel, 2000) the potential benefits of using GI systems for the analysis of dynamical processes in buildings are discussed. The author states that, even if component-oriented CAD systems provide sophisticated functionality for geometric modeling, they normally lack comprehensive spatial analysis capabilities. For this reason, Ozel stores floor plans of buildings in a GIS database in order to use its 2D spatial analysis facilities. The author underlines that 3D spatial analysis would be an even more powerful tool for analyzing processes in buildings.

Up to now, spatial database systems that support 3D spatial analysis are only to be found in a research context. The investigations set out in (Gröger et al., 2004), for example, clearly show that the spatial analysis capabilities of the commercial database system *Oracle Spatial* are limited to 2D space, even though it is possible to store simple 3D geometry.

As far as GIS is concerned, the main interest lies in the 3D modeling of the ground surface, buildings and infrastructure as well as the subsoil layers. The most important works in this

area include (Breunig et al., 1994, 2001; Balovnev et al., 2004) which report on the development of GeoToolkit, an object-oriented framework for efficiently storing and accessing 3D geographic and geologic data. The main disadvantage of using the framework for analyzing building models is the need to model all spatial entities according to the mathematical concept of simplicial complexes. The obligatory conversion of a boundary representation, as used in CAD tools, to a simplicial complex representation is expensive and, in some special cases, absolutely unfeasible. A more flexible, yet theoretic approach for applying algebraic topology on building models is presented in (Paul and Bradley, 2003).

Though (Coors, 2003; Arens et al., 2005; Zlatanova et al., 2004; Zlatanova, 2006) provide concepts and data structures for storing 3D city models in spatial databases, the definition and implementation of metric operators has been completely omitted in these papers.

(Kriegel et al., 2003) introduces a database system that allows for the spatial analysis of 3D CAD models. It provides simple volume, collision and distance queries, but supports neither topological nor directional predicates. The implementation of the system relies on a voxel approximation of the CAD parts stored in the database and a special index structure optimized for this representation. We follow a similar approach here but use a dynamically created, hierarchical data structure: the octree.

Metric operators

Metric operators are well established in 2D Geographic Information Systems and the corresponding spatial databases. For example, in (Güting, 1988) and (van Oosterom et al., 1994) three different distance operations are proposed: while the operation *dist* calculates the distance between two points, the operations *mindist* and *maxdist* calculate the minimum and maximum distance between two spatially extended objects, such as lines and regions. Another

metric operator that is often found in GIS literature is called *diameter*. It is used to determine the maximum distance between any two points of one spatial object.

Also, in the much smaller 3D GIS community metric operators are commonly used. For example, (Breunig, 1995) defines operators for calculating the *minimum distance*, the *maximum distance*, the *centroid distance* and the *Hausdorff distance* between two so-called e-complexes, a geometry description based on the concepts of algebraic topology. Unfortunately the implementation is not explained in detail. In most of the available GI systems metric operators are implemented on the basis of the operand's bounding boxes yielding rather imprecise results not acceptable for the construction domain in scope here.

However, most of the known approaches for implementing metric operators in 3D have not been developed in the context of GIS, but of Computational Geometry, mostly to implement fast collision detection between moving objects. A well-known solution for calculating the distance between polyhedra with convex surfaces is the GJK algorithm (Gilbert et. al, 1988), which relies on the Minkowski sum of the two objects and the iterative search for the vertex closest to the origin.

However, in complex 3D scenes with a multitude of objects consisting of a large number of facets, it is far too laborious to perform a GJK computation for every conceivable pair of facets. Therefore, bounding volumes have been introduced to filter out irrelevant facet pairs beforehand. Organizing bounding volumes in a hierarchical way allows for recursive processing and yields an even higher performance. In (van den Bergen, 1998) the utilization of an Axis Aligned Bounding Box (AABB) tree for collision detection was introduced. The second algorithm presented here is based on this approach, but applies it to distance computation. Other developments are bounding volume hierarchies that consist of spheres

(Quinlan, 1994), Oriented Bounding Boxes (OBBs) (Gottschalk et al., 1996, Johnson and Cohen, 1998) and Swept Sphere Volumes (Larsen et al., 1999). The latter paper provides a good overview of the different approaches and a detailed comparison of their performance.

The algorithms described above are based on a pre-set finite tree depth and a final calculation step to determine the distance between two polygonal or triangular facets (using GJK, for example), resulting in a precise distance value. By contrast, the octree-based algorithm proposed in this paper avoids any final calculation step and runs recursively until the desired accuracy is obtained. A similar approach is followed by (Jung and Gupta, 1997) but is again used with the aim of collision detection. Mundani (2005) shows how gaps between building components can be efficiently detected using octree representations, but does not extend the method to the more general case of distance computation.

Formal definitions of metric operators

All metric operators of the spatial query language rely on the Euclidean metric defined in 3D space. Let p (x_p, y_p, z_p) and q(x_q, y_q, z_q) □ □ í³, then the Euclidian distance between p and q is defined as:

$$\mathrm{d}(p,q) := \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2 + (z_p - z_q)^2}$$

The operator *distance* returns the minimal distance between two spatial objects as a real number. Let A and B be objects of type *Spatial* and $a \square A$, $b \square B$. Then *distance* is formally defined as follows:

$$\mathrm{distance}(A,B) := \min_{a,b}(\mathrm{d}(a,b))$$

distance returns 0, if the operands touch or penetrate each other.

The operator *maxdist* can be used to determine the maximum distance between two spatial objects. It also returns a real value and is defined as follows:

$$\text{maxdist}(A,B) := \max_{a,b}(\text{d}(a,b))$$

The operators *isCloser* and *isFarther* are based on the minimal distance. Operands of both operators are two spatial objects A and B as well as a positive real value c. The operators return a Boolean value and are formally defined as follows:

$$\text{isCloser}(A,B,c) \Leftrightarrow \min_{a,b}(\text{d}(a,b)) < c$$

$$\text{isFarther}(A,B,c) \Leftrightarrow \min_{a,b}(\text{d}(a,b)) > c$$

These operators can be used to select objects that are inside or outside a buffer zone around the reference object.

The operator *diameter* returns the maximum distance between two points of one individual object. The operand is a spatial object A, the return value a real number. Let $a, b \in A$. Then the diameter is defined as:

$$\text{diameter}(A) = \max_{a,b}(\text{d}(a,b))$$

Figure 1 illustrates the semantics of the definitions by means of an example.

Implementation

Providing spatial types and operators in SQL

Our concept for realizing the proposed spatial query language is based on an object-relational database technique implementing the ISO standard SQL:1999 (International Organization for Standardization, 1999) which allows the extension of the database type system in an object-oriented way, especially by providing abstract data types (ADTs) which may possess member functions (methods) (Melton, 2003; Türker, 2003; Türker and Saake, 2006). By using an object-relational database management system (ORDBMS), spatial data types and spatial operators can be made directly available to the end-users, enabling them to formulate queries such as

```
SELECT   *
FROM     buildingcomps comp, columns col
WHERE    comp.isCloser(20, col) AND
         col.id = 'UID09ZhXa'
```

to find all building components that have a distance of less than 20 cm to the column with the specified ID.

As can be seen in the example, spatial operators, such as *isCloser*, are implemented as methods of spatial data types and can be used in the WHERE part of an SQL statement. As opposed to purely object-oriented databases, these methods are stored and processed server-side, resulting in dramatically reduced network traffic compared to a client-side solution. This section discusses the implementation of the metric operators as server-side methods.

The spatial types defined in (Borrmann et al., 2006; Borrmann, 2007) and the metric operators specified in this paper are integrated in the object-relational query language SQL:1999 in the following way: the supertype *SpatialObject* and its subtypes *Body, Surface, Line* and *Point* are

declared as complex, user-defined types and the available spatial operators as member functions of these types. For the commercial ORDBMS Oracle the declaration reads:

```
CREATE OR REPLACE TYPE SPATIALOBJECT AS OBJECT
  EXTERNAL NAME 'SpatialObjectJ'
  LANGUAGE JAVA USING ORAData
(
        ...
        MEMBER FUNCTION distance(object SPATIALOBJECT)
        RETURN NUMBER
        EXTERNAL NAME 'distance(SpatialObjectJ) return double',

        MEMBER FUNCTION isCloser(object SPATIALOBJECT, dist NUMBER)
        RETURN NUMBER
        EXTERNAL NAME isCloser (SpatialObjectJ, double) return int',
      ...
);
```

The SQL type is bound to a corresponding Java type stored within the database, accordingly the declared SQL member functions are bound to specific Java methods of this type. Following its declaration, the user-defined SQL type may be used to create object tables, i.e. tables that exclusively host instances of the given type.

```
CREATE TABLE buildingcomponents OF BODY;
```

As soon as the table is filled with instances, the user is able to perform queries on them that may contain calls of member functions in the WHERE part:

```
SELECT   *
FROM     buildingcomps comp1, buildingcomps comp2
WHERE    comp1.id = 'UIDahjoik' AND
         comp2.isCloser(VALUE(comp1)) = 1
```

The processing of a spatial operator is forwarded to the specified Java routines stored within the database. In the case of a metric operator, such as *isCloser*, the Java stored procedure performs one of the algorithms presented in the next section.

Implementation of metric operators

Octree-based approach

Octree encoding

In the first implementation approach, we use the octree-encoded discretized geometry of the operands for distance calculation (Figure 2, left). The octree is a space-dividing, hierarchical tree data structure (Hunter, 1978; Jackins & Tanimoto, 1980; Meagher, 1982; Samet, 1985). Each node in the tree represents a cubic cell and has exactly one father and either eight children if it is an inner node, or zero children if it is a branch node. The aggregation of all child cells results in the parent cell. The ratio of the child cell's edge length to that of its father is always 1:2. The equivalent of the octree in 2D is called quadtree. Data structures of this kind are usually referred to as *space partition trees*.

The impact of the octree encoding's primary parameters, the size of the maximum and the minimum octant, on the resulting precision of the geometry description of typical building components is extensively discussed in (Shapira, 1993).

There are different methods for generating the octree representation of an object's geometry. The approaches differ according to whether only the body's boundary (Wenisch and Wenisch, 2004) or also its interior (Mundani et al., 2003; Mundani, 2005) is represented by the resulting octree (Figure 3). What is common to all approaches is the principle that, starting at the root node, only those children that are neither entirely inside the body nor entirely outside of it are refined. In the case of a two-colored tree, the cells that are cut by the boundary are marked *black* and all others *white*. In the case of a three-colored tree, interior cells are marked *black*, boundary cells *grey* and exterior cells *white*.

The octree encoding has several advantages over the cell enumeration method. Firstly, the amount of storage space is reduced by one order (from $\boldsymbol{O}\,(n^3)$ to $\boldsymbol{O}\,(n^2)$), because only the boundary has to be fully resolved. Secondly, the recursive nature of the data structure allows for the application of recursive algorithms.

In the proposed concept for the implementation of spatial operators, the geometry of each single building component is discretized in a separate octree. As regards generation time for the corresponding octree, there are two possibilities: the octree can either be created when the geometric object (building component) is inserted into the database (generation in advance) or when the spatial operation is performed (on-the-fly generation). The first approach requires more processing time for insertion and a lot of memory space needed for the continual storage of the octree.

The decision to adopt one or the other approach mainly depends on the interval between insertion/update operations compared with the frequency of queries containing spatial operators. In the application domain discussed here, the frequency of insert/update operations

is expected to be much higher than that of spatial queries. Accordingly, the approach of on-the-fly generation is considered to be more appropriate in this case. It offers the additional advantage – particularly for the implementation of the *distance* operator – that the octree generation can be coupled with the recursive algorithm, i.e. only those parts of the octrees need to be refined that are regarded as being closest to each other.

The octree-based distance computation algorithm

The input for the algorithm is the octree representation of both objects for which the distance is to be calculated, plus the maximum refinement level. It returns an interval in $\mathbb{i}^+$ determining the upper and lower bound between which the exact distance lies. This can be interpreted as a fixed value (the midpoint of the interval) combined with a maximum error.

The core of the algorithm is based on the calculation of distance values for pairs of octree cells (a cell pair is composed of one cell that belongs to octree *A* and another cell that belongs to octree *B*) on each octree level and the principle that those cell pairs whose distance is definitely higher than that of any other cell pair can be excluded from further refinement. The algorithm is implemented by two functions *findClosestCells()* and *createCandidateList()* which are shown schematically in Figure 8 and Figure 9. *FindClosestCells()* steers the traversal of the octrees by recursively calling itself. Additionally it creates cell pairs for the next refinement level and calls *createCandidateList()* which filters out all irrelevant pairs.

The algorithm is based on the fact that the exact position of the boundary of the objects is unknown when using an octree encoding (Figure 5). Therefore an upper and lower bound of the distance value has to be calculated for each cell pair (Figure 9, lines 6-14). These values reflect the interval in which the real distance lies. They are derived from the coordinates of the

octants in the coordinate system of the current level. It is important to bear in mind that these values have a local nature bound to the current octree level that need to be scaled to real-world coordinates to allow for conclusions about the real distance.

To compute lower and upper bound, each Cartesian direction is evaluated separately by applying the following rules: if the coordinate value of both cells is equal, then the lower bound is 0 and the upper bound is 1 (Figure 9, line 12). In any other case, the lower bound results from the difference between the coordinate values minus 1 and the upper bound from the difference between the coordinate values plus 1. The cell pairs' lower and upper bound can now be computed by summing up the squared values (Figure 9, lines 9-10). The expensive calculation of the square root is not yet necessary, because in this step only the global order (closer, farther) of the cell pairs has to be determined.

By computing the upper and lower bounds for all cell pairs, it is possible to identify the candidates for the closest cell pair ranking. To this end, the lowest upper bound of all pairs of the current level is determined (Figure 9, lines 15-17) and all cell pairs whose lower bound is higher than this value are excluded (Figure 9, lines 22-26).

All other pairs are candidates. For them, the algorithm is recursively repeated (Figure 8, line 10). They are refined, i.e. pairs of the respective child cells are composed (Figure 8, 2-5), and the filtering algorithm is applied to the resulting pairs of children (Figure 8, line 10), i.e. distance values are calculated, candidates are chosen, and so on. The recursion is aborted when the maximum refinement level is reached (Figure 8, line 7-9).

By descending both octrees in this way (which is a breadth-first traversal – see Figure 6), the precision of the calculated distance is successively increased: the calculated distance can be

expressed on each level by means of an interval, whose endpoints are calculated from the square root of the upper and lower distance values determined for the cell pairs on the level in question. By letting the user choose the termination level, he is able to balance the accuracy of the result obtained with the time needed for its computation.

The interval in which the real distance lies is calculated after the recursion has finished. The final lower bound results from the square root of the lowest lower bound on the final octree level multiplied by the edge length of an octant on this level. The final upper bound is derived from the square root of the lowest upper bound on the final level, again multiplied by an octant's edge length. The return value of the algorithm is either a tuple of two real numbers representing upper and lower bound of the distance, or a single real number calculated as the arithmetic mean of upper and lower bound. The latter version can be integrated more easily into a spatial query language.

It is possible to couple the generation of the octree with the distance algorithm. This further increases the overall performance, because then the octree is built up only at points that are relevant for the distance computation (Figure 7). The proper place for integrating such an on-the-fly octree generation in the algorithm is the *refine()* function call (Figure 8. 1, line 3).

By slightly modifying the basic algorithm for calculating the *distance*, it can also be applied to implement the *maxdist, diameter, isCloser* and *isFarther* operators. The implementation of *isCloser* and *isFarther* is almost identical to the *distance* algorithm. As soon as the distance algorithm is entirely within the range defined by *isCloser* or *isFarther*, the algorithm can abort the recursion and return *true*. In the case of *isCloser* the algorithm can also stop and return *false* when the lower bound of the distance interval exceeds the given distance. In the

case of *isFarther* it can stop and return *false* when the upper bound of the distance interval drops below the given distance.

For the implementation of *maxdist* a somewhat greater modification of the basic algorithm is required: instead of choosing those pairs of octants for further refinement whose lower bound distance is smaller than the smallest upper bound distance, here the octant pairs are candidates whose upper bound distance is smaller than the largest lower bound distance.

The operator *diameter* has also been implemented on the basis of the *maxdist* algorithm. To do this, the octree of the object to be examined is passed as the first and second parameter to the *maxdist* function. This accordingly determines the maximum distance between the points of the object, which corresponds exactly to the definition of *diameter*. The *maxdist* algorithm can also be used to determine the penetration depth of two overlapping objects. For this purpose, the Boolean intersection of both octree-encoded geometries first has to be computed, by the algorithm proposed in (Mundani et al., 2003), for example. The *maxdist* algorithm is subsequently applied to the resulting octree.

Time complexity of the octree-based algorithm

Contrary to traditional distance algorithms, the computational time needed to calculate the distance is independent of the number of facets the objects' surfaces are composed of, but depends on the desired accuracy. It corresponds to the width of the resulting interval and increases with each recursion level by the order $\boldsymbol{O}\ (n^3)$.

The time complexity depends directly on the number of cell pairs that need to be compared. To estimate this number, we examine the worst case, which results from two objects facing

each other with parallel (plane) surfaces. For the sake of simplicity, we first examine the 2D case. As stated above, all cell pairs satisfying the *distance condition* $d_{lower} < d_{upper,min}$ are refined. The first step is to count those cell pairs where cell *A* has a fixed *x* coordinate and cell *B* fulfills the distance condition. The number of such pairs multiplied by the number of cells on the flat surface in *y* direction yields the number of candidate pairs. Any side effects are disregarded, in other words, we assume an infinitely extended surface. For pairs of cells that satisfy the distance condition, the distance in *y* direction is *a*-1, *a* or *a*+1 (Figure 10). No *y* distance smaller than *a*-1 is possible due to the extent of the surfaces. Nor is a *y* distance greater than *a*+1 possible in view of the distance condition, because for a cell pair with a *y* distance of *a+2* and the smallest possible *x* distance of *0* $d_{lower} = (a+2)^2 + 0^2 > d_{upper,min} = (a+1)^2 + 1^2$ already holds. For cell pairs with a *y* distance of *a* or *a+1* there is additional scope in *x* direction, i.e. cell *B* can be shifted by 1or 2 places respectively (Figure 11).

To determine the maximum distance *b* in *x* direction we have to distinguish between three cases. In the first case, the y distance is *a*-1. The distance condition accordingly states $(a-1)^2 + b^2 < (a+1)^2 + 1^2$, yielding $b_{1,max} = \sqrt{4a+1}$. In the second case, the *y* distance is *a*. The distance condition accordingly states $a^2 + b^2 < (a+1)^2 + 1^2$, yielding $b_{2,max} = \sqrt{2a+2}$. In the third case, the *y* distance is *a*+1. The distance condition accordingly states $(a+1)^2 + (b+1)^2 < (a+1)^2 + 1^2$, yielding $b_{3,max} = 0$. The cell pairs may have the distance *b* in either the positive or negative direction. In addition, there are always 3 cell pairs with a *y* distance of 0, so that in each case 3 + 2*b* cell pairs satisfy the distance condition. The number of cell pairs accordingly results in

$$N_1 = (3 + 2\sqrt{4a+1}) + 2 \bullet (3 + 2\sqrt{2a+2}) + 3 \bullet 3 \approx \boldsymbol{O}(\sqrt{a})$$

To obtain the total number of cell pairs requiring refinement, it is necessary to multiply N_1 by the number of cells in *x* direction (length *l*): $N = N_1 \bullet l$.

Both the distance a and the length l depend on the current refinement level. Let a_0 be the distance at level 0. In each recursion step, a range of length x_d is mapped to a range of length x_{d+1} with $x_{d+1} = 2x_d$. After d refinement steps, a can be estimated using $a \approx a_0 \bullet 2^d$ and l by $l \approx l_0 \bullet 2^d$. Accordingly, the number of cell pairs that have to be compared on level d is

$$N = \boldsymbol{O}\,(\sqrt{2^d} \bullet 2^d) = \boldsymbol{O}\,(2^{3/2d})$$

for the 2D case.

In 3D, the minimum upper distance is $d_{upper,min} = (a+1)^2 + 1^2 + 1^2$. Because it is 1 greater than in the 2D case, $b_{1,max}$, $b_{2,max}$ and $b_{3,max}$ change slightly: $b_{1,max} = \sqrt{4a+2}$ $b_{2,max} = \sqrt{2a+3}$ and $b_{3,max} = 1$. The number of cells within the corresponding shell segments of an imaginary sphere can be approximated by

$$N_1 \approx (3 + 2b_{1,\max})^2\,\pi / 4 + 2\,(3 + 2b_{2,\max})^2\,\pi / 4 + 3\,(3 + 2b_{3,\max})^2\,\pi / 4 \approx \boldsymbol{O}\,(a)$$

Let l_x and l_y be the lengths of the plate in x and y direction. Accordingly, the number of cell pairs that need to be compared on level d is

$$N = N_1 \bullet l_x \bullet l_y = \boldsymbol{O}\,(2^d \bullet 2^d \bullet 2^d) = \boldsymbol{O}\,(2^{3d})$$

for the 3D case.

The AABB-based approach

The second implementation approach is based on the exact shapes of the geometric objects (the operands) given by their boundary representation (B-Rep). However, the aim in this case is likewise to avoid the high effort for computing the distances between all possible facet pairs. Therefore we employ axis-aligned bounding box trees (AABB-trees) to organize the facets of each of the operands hierarchically. We can then traverse both trees in a similar fashion as in the octree-based implementation, i.e. compute an upper and lower bound for each pair of AABBs and accordingly prune all irrelevant pairs and their children.

An AABB tree is a binary tree that is built up by recursively subdividing a given mesh into axis-aligned bounding boxes. Every internal node has the property that its AABB encompasses the AABBs of its children and thus all primitives assigned to the two child nodes. In contrast to octrees, AABB trees can only model the surface but not the interior of an object.

AABB trees are constructed in a top-down manner (Figure 11). Given a triangulation of an object's surface, the corresponding AABB tree is built up as follows: First, the smallest AABB containing the set of all primitives is computed. After that, the set is split into two subsets, i.e. a left and a right one, according to a partition rule. In our case, the projection onto the longest axis of the AABB is decisive: if the midpoint of a primitive's projection is smaller than the midpoint of the AABB's projection, the primitive is allotted to the left subset. Otherwise it is attached to the right subset. Once all the primitives of the AABB under consideration have been classified, we continue recursively with both the left and the right subset until an AABB contains only one primitive or the maximum level has been reached. In the latter case, the box is a leaf of the AABB tree.

An alternative partition rule sorts the primitives in ascending order with respect to the midpoint of the respective projection onto the longest AABB axis and then splits the set into two equal subsets. In contrast to the first partition rule, it produces a balanced AABB tree but, due to the wider distribution of the box size, the maximum extent of the boxes on a certain level may be larger.

The idea of the distance algorithm applied to this hierarchical model representation is similar to that of the octree-based algorithm. In a breadth-first traversal on each level, all AABB pairs that are not candidates for the minimum distance are excluded. In contrast to the octree-based distance algorithm, calculations cannot be realized as integer operations and, in addition, leaf nodes of the AABB trees are handled differently.

If an AABB pair is a potential candidate for minimal distance and both AABBs of the pair are leaf nodes, we build the cross product of all primitives assigned to the two AABBs and, in a final calculation step, compute the exact minimal distance for all resulting pairs. The exact distance between two primitives under consideration is computed by means of the GJK algorithm for convex polyhedrons. Since an AABB generally encompasses a few triangles only, the computational effort is limited.

The computed distances between triangles may lead to a new minimum upper distance that can be used to exclude other AABB pairs in the same way as in the octree-based approach. It is possible that an AABB pair is a potential candidate for the minimal distance, but only one box of the pair is a leaf and, therefore, only the second bounding box has two children. In this case, the two children of the second AABB are paired with the first AABB, respectively. Finally, the global minimum distance results as the lowest of all the distances from one triangle to another that were computed with GJK during the algorithm.

Johnson and Cohen (1998) report a time complexity of $\boldsymbol{O}(n)$ for the OBB tree based algorithm, where n is the number of facets. Because the algorithmic nature of the AABB tree approach is identical, we can assume the same time complexity in this case.

Performance comparison

For comparing the performance of the different algorithms we used two different setups – a theoretical and a practical real-world example. In the theoretic setup we computed the distance between two spheres with radius 1, one located at (0,0,0) and the other one at (2,2,2). To model surfaces with different complexities, the spheres were triangulated using 8, 16, 32 and 64 segments resulting in 48, 224, 960 and 3968 facets, respectively. Tables 1 and 2 depict the results of the performance measurements. The timings include the creation of the octrees and AABB trees respectively. In the case of the octree based algorithm, the tree is only generated during the recursion where it is necessary, i.e. only candidate cells are refined.

We implemented both algorithms as pure Java code, because this simplifies the server-side integration into an object-relational database. All intersection tests between facets and octants / bounding boxes are performed using the Akenine-Möller approach (Akenine-Möller, 2001). All timings were taken on an Intel PM-770 2.13GHz machine.

Tables 1 and 2 clearly show that the performance of the AABB algorithm is superior to that of the octree based algorithm, particularly if one takes into account that it delivers the precise distance. But, as depicted in the diagram of Figure 13, the processing time of the AABB algorithm is highly dependent on the chosen maximum depth of the AABB tree. The optimal depth of the AABB tree depends on the size and distribution of the facets and can hardly be predicted for the general case. By contrast, the processing time of the octree based algorithm scales continuously with $\boldsymbol{O}\,(\,2^{3d}\,)$, as deduced theoretically above.

The superiority of the AABB approach becomes even more obvious when looking at a real-world example. We use the building model shown in Figure 14, which consists of 216 building components, and use three different algorithms for calculating the distance between

one of the columns and all other objects. The results presented in Table 3 and Figure 15 clearly show that the fact that most components of "standard" buildings have a rectangular, axis-aligned shape, renders the AABB algorithm even more suitable: the algorithm runs in less than 10% of the time the octree algorithms taken for a maximum level of 6. In addition, it returns the precise distance instead of an interval (not taking into account the impreciseness resulting from numerical issues here).

Summary

In this article we have presented definitions and possible implementations of directional operators in a Spatial Query Language for 3D Building Models. By using point-set theory notation, we have formally defined the metric operators *distance, maxdist, isCloser, isFarther* and *diameter*. The paper describes in detail two possible implementations of these operators. The first one relies on the octree representation of the geometric objects. The recursive algorithm calculates upper and lower bounds for all octant pairs of one level and excludes all irrelevant pairs from further refinement. By traversing the octrees, the resulting distance interval becomes narrower, and thus the preciseness of the calculated is successively increased.

The second approach employs trees of Axis Aligned Bounding Boxes (AABBs). Here the bounding facets of the original objects are organized hierarchically. Also in this case, lower and upper bound distances are calculated during a traversal of the trees and irrelevant branches are excluded from further examination. As opposed to the octree approach, there is a final level that contains a subset of the original facets (the candidates), for which an exact distance computation is performed.

The performance comparisons show that using AABB trees coupled with executing the GJK algorithm for the final candidates is a good solution for calculating the distance between two polyhedra. Even for objects with complex surfaces, it displays a better performance than the octree-based algorithm. Moreover, the distance returned by the AABB algorithm is an exact value, not an interval, as in the case of the octree algorithm. The fact that most components of "standard" buildings have a rectangular, axis-aligned shape, renders the AABB algorithm even more suitable.

Although we discovered that the performance of the octree-based distance algorithm falls short of that of an AABB-based one, the former has a primal advantage: for a particular level of precision, the time for processing the distance algorithm has an upper limit, regardless of the number and arrangement of the facets. This property is very useful for integrating the algorithm in a spatial database, because the user is not normally aware of the complexity of the stored objects when submitting a query. By contrast, the processing time of the AABB-based algorithm largely depends on the chosen tree-depth and the distribution of the facets. Moreover, the octree encoding provides an opportunity for an easy, yet robust implementation not only of the *distance* operator but also of other metric operators, such as *volume* or *area*.

Outlook

Our future work in the context of metric operators will include further research into the runtime performance of our algorithms for real-world building models. We also plan to test alternative implementations, such as algorithms based on oriented bounding boxes.

Within the scope of the overall Spatial Query Language project we intend to enhance the performance of the database access further still by implementing R-tree indexing structures

within the database management system. In addition, we want to analyze the potential use of so-called In-Memory Databases to avoid secondary storage access while retaining the benefits of a declarative query language.

In the current phase of our project we store the explicit geometry of all building components of a BIM in the database by means of a simple vertex-edge-face data structure. In future, we want to upgrade to a more comprehensive boundary representation, such as Winged-Edge or Radial-Edge, which will make it possible to use the results of a spatial query for further processing in the end-user's CAD system. We also intend to store semantic information, such as BIM classes and non-geometric attributes, to make it possible to employ such information within the selection predicate.

Of particular interest is the combination of the proposed spatial query language for building models with techniques for the extraction of air volumes from 3D models that have been developed by our group (van Treeck and Rank, 2007). This combination will enable the user to not only query spatial relationships between building components, such as walls and columns, but also to include non-physical spatial entities such as rooms and floors. Another promising research direction is the evaluation of an alternative query language as a basis for the extension by spatial operators. Possible candidates are XQuery, a query language for XML data, and SPARQL, a language for querying RDF ontologies developed in the context of the Semantic Web (Beetz et al., 2007).

Finally, we will place emphasis on showing practical applications of the spatial query language by developing forms of usage in the context of checking construction rules and creating partial models.

Acknowledgements

The authors gratefully acknowledge the support for the ongoing project by the German Research Foundation (DFG) under grant Ra624/17-1.

References

Adachi, Y. (2003). “Overview of partial model query language.” *Proc. of the 10th Int. Conf. on Concurrent Engineering*. ISPE, Madeira, Portugal. A. A. Balkema Publishers, 549-555.

Akenine-Möller, T. (2001). ”Fast 3D triangle-box overlap testing,” *J. of Graphics Tool*s, 6, 29–33.

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

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

Beetz, J., de Vries, B., van Leeuwen, J. (2007). “RDF-based distributed functional part specifications for the facilitation of service-based architectures.” *Proc. of the 24th CIB-W78 Conf. on Information Technology in Construction,* CIB-W78, Maribor, Slovenia.

Borrmann, A. (2007). “Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken.” Ph.D. thesis, Lehrstuhl für Bauinformatik, Technische Universität München.

Borrmann, A., C. van Treeck, and E. Rank (2006). “Towards a 3D spatial query language for building information models.” *Proc. of the Joint Int. Conf. for Computing and Decision Making in Civil and Building Engineering*. ISCCBE, Montreal, Canada.

Borrmann, A. and Rank, E. (2008a). “Specification and implementation of directional operators in a 3D spatial query language for building information models.” *Advanced Engineering Informatics*. accepted.

Borrmann, A. and Rank, E. (2008b). “Topological operators in a 3D Spatial Query Language for Building Information Models.” *Proc. of the 12th Int. Conf. on Computing in Civil and Building Engineering,* ISCCBE, Beijing, China.

Breunig, M., Bode, T. and Cremers, A. (1994). “Implementation of elementary geometric database operations for a 3D-GIS.” *Proc. of the 6th Int. Symp. on Spatial Data Handling*. IGU, Edinburgh, Scotland.

Breunig, M., Cremers, A., Müller, W., and Siebeck, J. (2001). “New methods for topological clustering and spatial access in object-oriented 3D databases.” *Proc. of the 9th ACM Int. Symp. on Advances in Geographic Information Systems*. ACM, Atlanta, Georgia, USA.

Coors, V. (2003). “3D-GIS in networking environments.” *Computers, Environment and Urban Systems,* 27 (4), 345–357.

Dubois, A. M., Flynn, J., Verhoef, M. H. G. and Augenbroe, G. L. M. (1995). “Conceptual modelling approaches in the COMBINE project.” *Proc. of the 1st Europ. Conf. on Product and Process Modeling in the Building Industry*. EAPPM, Dresden, Germany.

Eastman, C. (1999). Building Product Models: Computer Environments Supporting Design and Construction. CRC Press.

Egenhofer, M. (1987). "An extended SQL syntax to treat spatial objects." *Proc. of the 2nd Int. Seminar on Trends and Concerns of Spatial Sciences*. Fredericton, NB, Canada.

Egenhofer, M. (1992). "Why not SQL!" *Journal of Geographical Information Systems,* 6(2), 71–85.

Gröger, G., Reuter, M. and Plümer, L. (2004). "Representation of a 3-D city model in spatial object-relational databases." *Proc. of the 20th ISPRS Congress,* ISPRS, Istanbul, Turkey.

Güting, R. H. (1988). "Geo-relational algebra: A model and query language for geometric database systems." *Proc. of the Int. Conf. on Extending Database Technology,* EDBT Assoc., Venice, Italy.

Gottschalk, S., Lin, M., and Manocha, D. (1996). "OBB-Tree: A hierarchical structure for rapid interference detection." *Proc. of 23rd Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*, ACM, New Orleans, LA, USA.

Herring, J., Larsen, R., and Shivakumar, J. (1988). "Extensions to the SQL query language to support spatial analysis in a topological data base." *Proc. of GIS/LIS '88,* ACSM, San Antonio, Texas.

Hunter, G. “Efficient computation and data structures for graphics.” PhD thesis. Princeton University.

Ingram, K. and Phillips, W. (1987). “Geographic information processing using a SQL-based query language.” *Proc. of the 8th Int. Symp. on Computer-Assisted Cartography*. CaGIS, Baltimore, MD, USA.

Jung, D. and Gupta, K.K. (1997). “Octree-based hierarchical distance maps for collision detection.” Journal of Robotic Systems 14(11), 789–806.

International Organization for Standardization (1999). ANSI/ISO/IEC 90751:99. ISO International Standard: Database Language SQL.

International Organization for Standardization (2005). ISO/PAS 16739:2005 Industry Foundation Classes, Release 2x, Platform Specification.

Jackins, C. L., and Tanimoto, S. L. (1980). “Oct-trees and their use in representing three-dimensional objects.” *IEEE Computer Graphics and Image Processing*, 14(3), 249–270.

Johnson, D., and Cohen, E. (1988). “A framework for efficient minimum distance computation.” *Proc. IEEE Int. Conf. on Robotics and Automation,* IEEE, Philadelphia, PA, USA.

Gilbert, E. G., Johnson, D.W., and Keerthi, S. S. (1988). “A fast procedure for computing the distance between complex objects in three-dimensional space.” *IEEE Journal of Robotics and Automation*, 4(2),193–203.

Gottschalk, S., Lin, M., and Manocha, D. (1996). ”OBB-Tree: A hierarchical structure for rapid interference detection.” *Proc. of 23rd Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*, ACM, New Orleans, LA, USA.

Kriegel, H.-P., Pfeifle, M., Pötke, M., Renz, M. and Seidl, T. (2003). “Spatial data management for virtual product development.” *Lecture Notes in Computer Science* 2598, 216–230.

Lin, M. C. and Canny, J. F. (1991). “A fast algorithm for incremental distance computation.” *Proc. IEEE Int. Conf. on Robotics and Automation,* IEEE, Sacramento, CA, USA.

Meagher, D. (1982). “Geometric modeling using octree encoding.” *IEEE Computer Graphics and Image Processing,* 19 (2), 129–147.

Melton, J. (2003). “Advanced SQL:1999. Understanding Object-Relational and Other Advanced Features.” Morgan Kaufmann, San Francisco, USA.

Mundani, R.-P. (2005). “Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben.” Ph.D. thesis, Universität Stuttgart.

Mundani, R.-P., Bungartz, H.-J., Rank, E., Romberg, R. and Niggl, A. (2003). “Efficient algorithms for octree-based geometric modelling.” *Proc. of the 9th Int. Conf. on Civil and Structural Engineering Computing*. B.H.V. Topping (ed.), Egmond aan Zee, The Netherlands.

Ooi, B., Sacks-Davis, R. and McDonell, K. (1989). “Extending a DBMS for geographic applications.” *Proc. of the IEEE 5th Int. Conf. on Data Engineering*, IEEE, Los Angeles, CA, USA.

OpenGIS Consortium (OGC) (1999). OGC Abstract Specification.

Ozel, F. (2000). “Spatial databases and the analysis of dynamic processes in buildings.” *Proc. of the 5th Conf. on Computer Aided Architectural Design Research in Asia*. CAADRIA, Singapore, Malaysia.

Paul, N. and Bradley, P. E. (2003). “Topological houses.” *Proc. of the 16th Int. Conf. of Computer Science and Mathematics in Architecture and Civil Engineering (IKM 2003).* Weimar, Germany.

Rigaux, P., Scholl, M., and Voisard, A. (2002). “Spatial Databases with Application to GIS.” Morgan Kaufmann.

Roussopoulos, N., Faloutsos, C., and Sellis, T. (1988). “An efficient pictorial database system for PSQL.” *IEEE Transactions on Software Engineering* 14 (5), 639–650.

Scherer, R. J. (1995). “EU-project COMBI – objectives and overview.” *Proc. of the 1st Europ. Conf. on Product and Process Modeling in the Building Industry*. EAPPM, Dresden, Germany.

Shapira, A. (1993). “Octree subdivision of building elements”, *Journal of Computing in Civil Engineering*, 7(4), 439–457.

Shekhar, S. and Chawla S. (2003). “Spatial Databases: A Tour.” Pearson Education.

Tolman, F. and Poyet P. (1995). “The ATLAS models.” *Proc. of the 1st Europ. Conf. on Product and Process Modelling in the Building Industry.* EAPPM, Dresden, Germany.

Türker, C. (2003). “SQL:1999 & SQL2000. Objekt-relationales SQL, SQLJ & SQL/XML.” dpunkt Verlag.

Türker, C. and Saake, G. (2006). “Objektrelationale Datenbanken.” dpunkt Verlag.

van Oosterom, P., Vertegaal, W., van Hekken, M., and Vijlbrief, T. (1994). “Integrated 3D modelling within a GIS.” *Proc. of Advanced Geographic Data Modelling (AGDM'94) - International GIS Workshop*. Netherlands Geodetic Commission, Delft, The Netherlands.

van Treeck, C. and Rank, E. (2007). “Dimensional reduction of 3D building models using graph theory and its application in building energy simulation.” *Engineering with Computers* 23 (2) 109–122.

Wenisch, P. and Wenisch, O. (2004). “Fast octree-based voxelization of 3D boundary representation-objects.” Technical report, Computation in Engineering, Technische Universität München.

Yabuki, N., Lebegue, E., Gual, J., Shitani, T. and Zhantao, L. (2006). “International collaboration for developing the bridge product model IFC-Bridge.” *Proc. of the 11th Int. Conf. on Computing in Civil and Building Engineering.* ISCCBE, Montreal, Canada.

Zlatanova, S. (2006). “3D geometries in spatial DBMS.” *Innovations in 3D Geo Information Systems*, A. Abdul-Rahman, S. Zlatanova and V. Coors, eds., Springer, Berlin, 1–14.

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

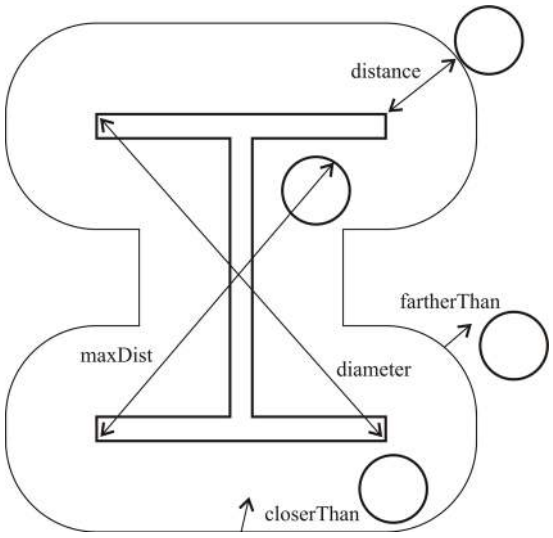


Figure 1.

1-column, height: 69.5mm

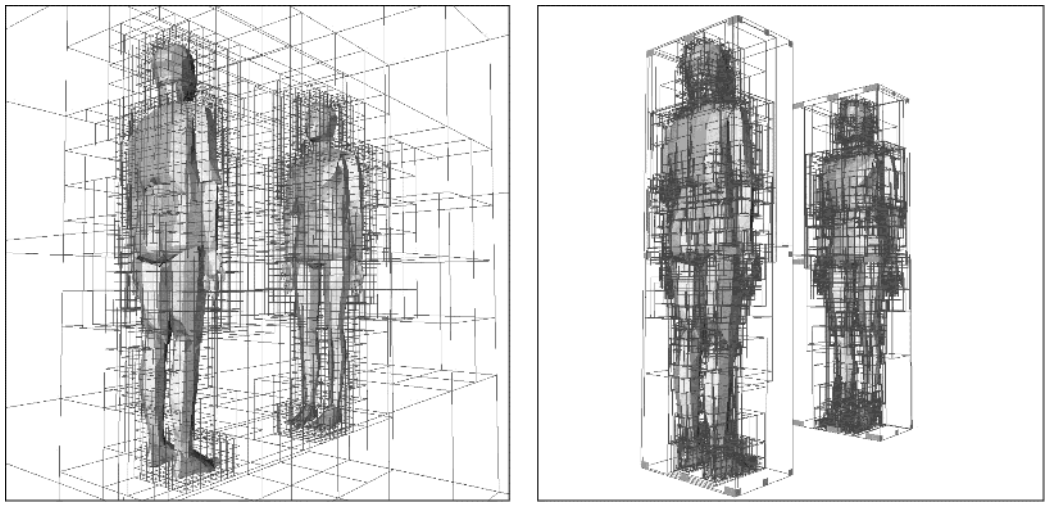


Figure 2.

2 colums, height: 65.7 mm

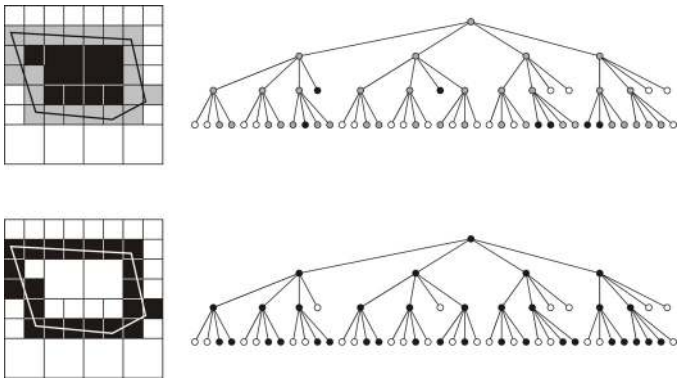


Figure 3.

1 column, height: 49.1mm

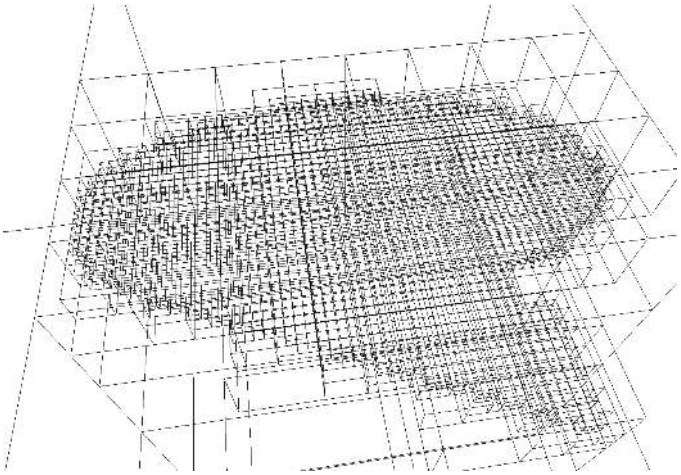


Figure 4.

1 column, height: 61.63mm

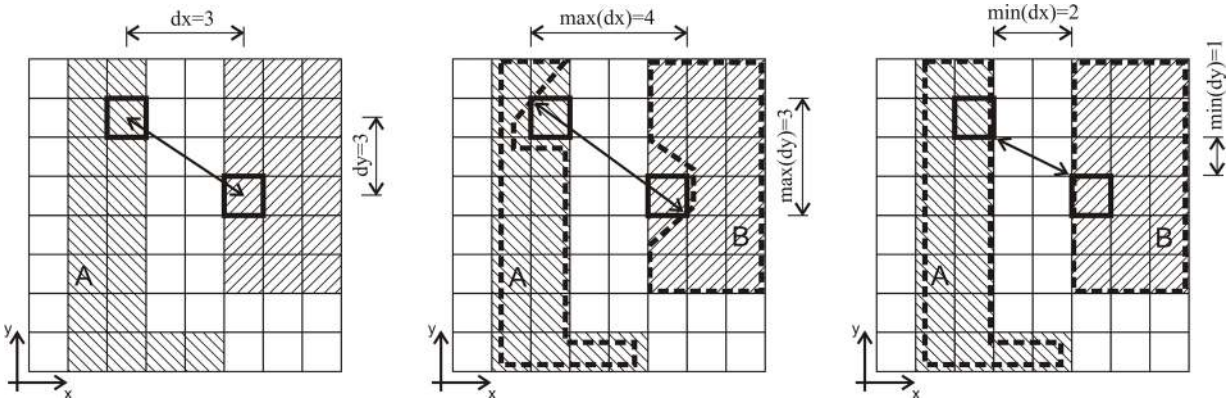


Figure 5.

2 columns, height: 52,2mm

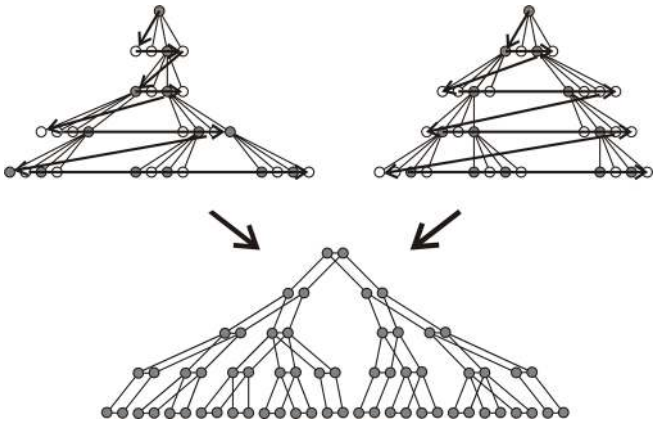


Figure 6.

1 column, height 54.5mm

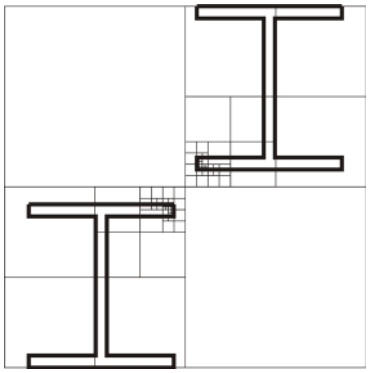
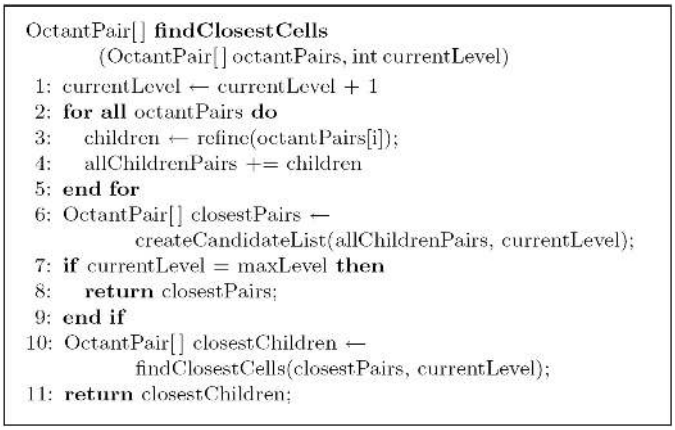


Figure 7.

Height: 48.1 mm



```
OctantPair[] findClosestCells
        (OctantPair[] octantPairs, int currentLevel)
 1: currentLevel ← currentLevel + 1
 2: for all octantPairs do
 3:    children ← refine(octantPairs[i]);
 4:    allChildrenPairs += children
 5: end for
 6: OctantPair[] closestPairs ←
            createCandidateList(allChildrenPairs, currentLevel);
 7: if currentLevel = maxLevel then
 8:    return closestPairs;
 9: end if
10: OctantPair[] closestChildren ←
            findClosestCells(closestPairs, currentLevel);
11: return closestChildren;
```

Figure 8.

1 column, height: 55.9 mm

```
OctantPair[] createCandidateList(OctantPair[] octantPairs)
 1: int lowestUpperBound ← domainSize;
 2: int lowestLowerBound ← lowestUpperBound;
 3: for all octantPairs do
 4:    lowerBound ← 0
 5:    upperBound ← 0
 6:    for i = 0 to 2 do
 7:       calculate dist[i] // 0 ≅ x, 1 ≅ y, 2 ≅ z
 8:       if dist[i] > 0 then
 9:          lowerBound ← lowerBound+(dist[i]−1)^2
10:          upperBound ← upperBound+(dist[i]+1)^2
11:       else
12:          upperBound ← upperBound+1
13:       end if
14:    end for
15:    if upperBound < lowestUpperBound then
16:       lowestUpperBound ← upperBound;
17:    end if
18:    if lowerBound < lowestLowBound  then
19:       lowestLowerBound ← lowerBound;
20:    end if
21: end for
22: for all octantPairs do
23:    if lowerBound ≤ lowestUpperBound then
24:       candidateList += octantPairs;
25:    end if
26: end for
27: return candidateList
```

Figure 9.

1 column, height: 99.6 mm

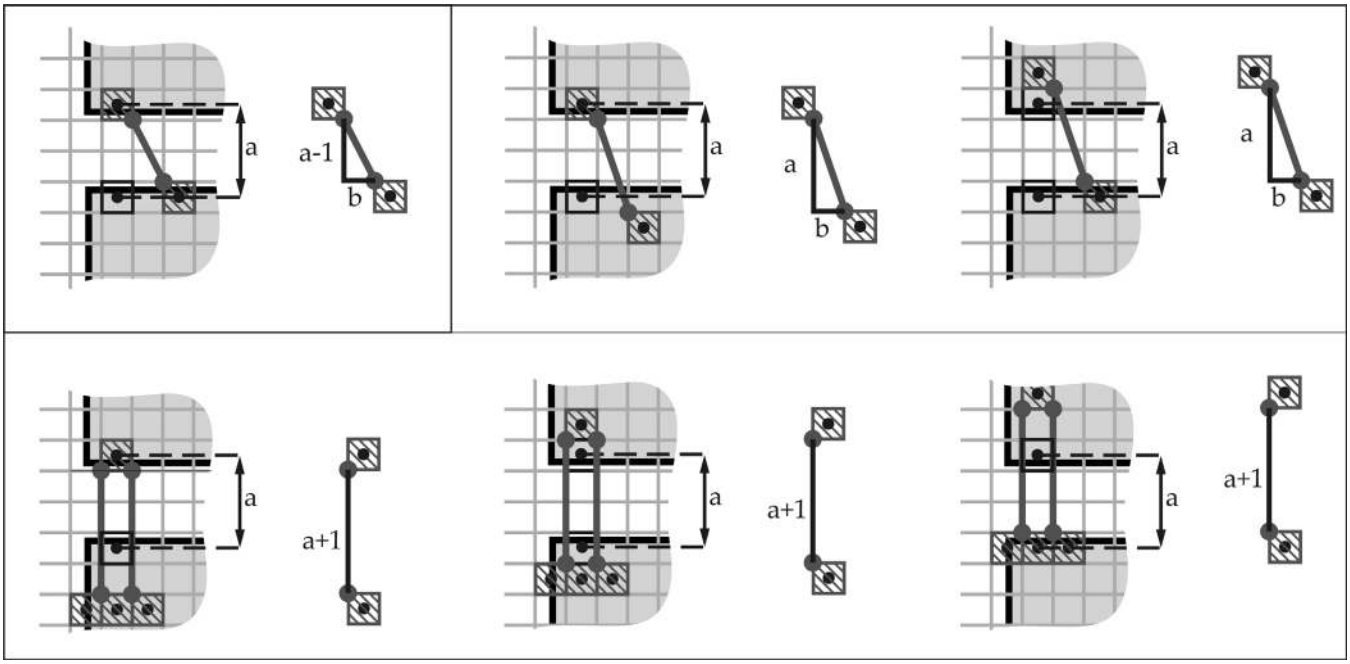


Figure 10.

2 columns, height 86.8 mm

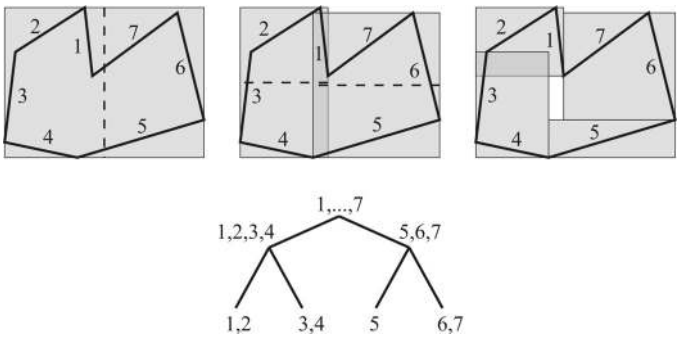


Figure 11.

1 column, height 43.4 mm

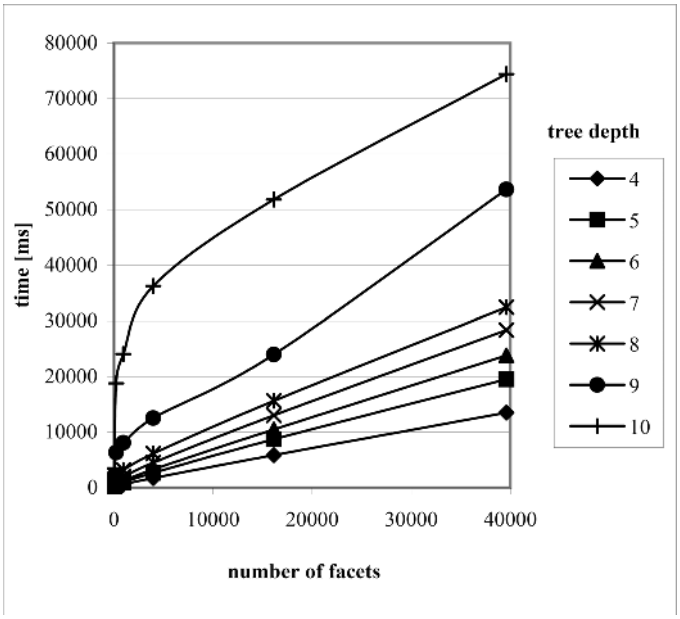


Figure 12.

1 column, height: 80.6mm

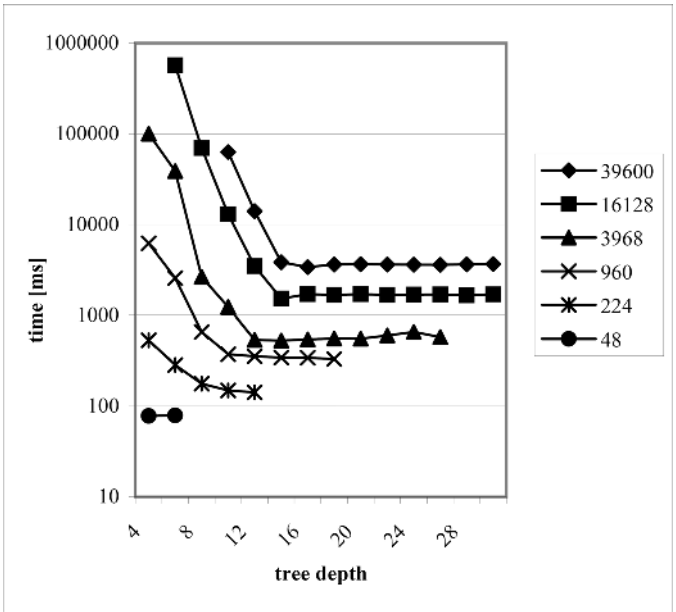


Figure 13.

1 column, height: 80.4mm

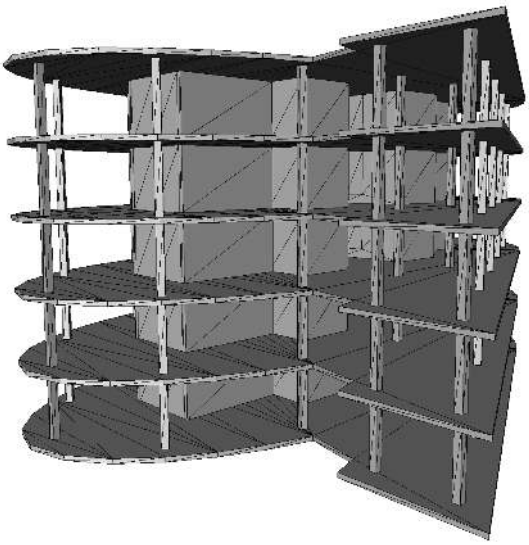


Figure 14.

1 column, height: 74.9 mm

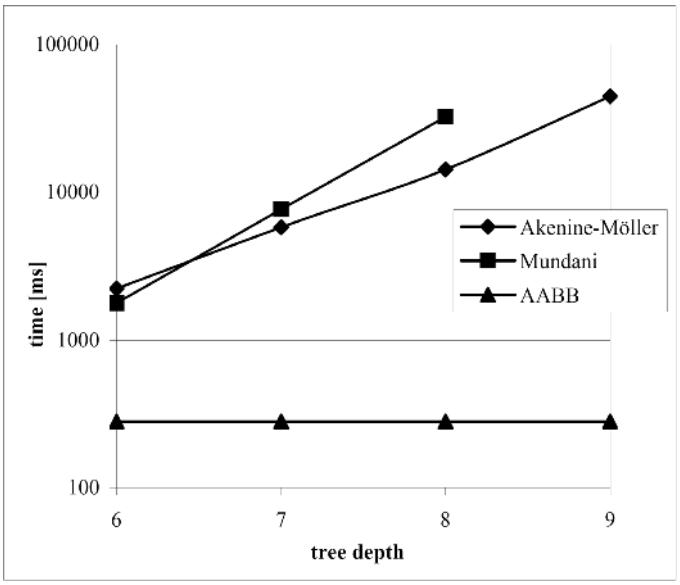


Figure 15.

1 column, height 76mm

max depth	number of facets per sphere					
	48	**224**	**960**	**3968**	**16128**	**39600**
4	112	221	600	1730	5853	13531
5	245	428	972	2665	8718	19539
6	503	608	1285	3306	10484	23798
7	909	1087	1952	4495	13039	28383
8	1364	2322	3198	6149	15622	32447
9	2108	6324	8094	12538	23967	53639
10	3487	18791	24055	36273	51821	74429

Table 1.

1 column, height 42.5mm

max depth	number of facets per sphere					
	48	**224**	**960**	**3968**	**16128**	**39600**
4	78	532	6196	99941		
6	79	284	2555	38775	570905	
8		177	659	2646	69586	
10		149	373	1229	12978	
12		142	354	537	3474	14002
14			343	527	1522	3825
16			342	544	1702	3386
18			331	556	1672	3615
20				554	1704	3639
22				601	1671	3616
24				659	1680	3607
26				578	1690	3591
28					1655	3637
30					1698	3666

Table 2.

1 column, height 75.0 mm

max depth	Akenine-Möller	Mundani	AABB
6	2234	1782	281
7	5812	7703	281
8	14296	32485	281
9	44778		281

Table 3.

1 column, height 30.5 mm

Figure 1. Illustration of the semantics of the metric operators. As can be seen in the example, *closerThan* and *fartherThan* can be interpreted in a way that is called *buffer zoning* in the GIS context (Shekhar and Chawla, 2003).

Figure 2. Two different approaches for implementing metric operators: While the left-hand diagram depicts the octree representations of the operands, the right-hand diagram shows their AABB tree representations.

Figure 3. The difference between a three-colored quadtree (above), where the interior of described object is marked black, and a two-colored quadtree (below), that only represents the boundary of the object.

Figure 4. The octree representation of one of the slabs of the building model shown in Figure 14.

Figure 5. Since the exact position of the boundary of the objects is unknown when using octree encodings, an upper (middle diagram) and lower distance value (right-hand diagram) has to be determined for each cell pair based on the distance between the cells' midpoints (left-hand diagram).

Figure 6. While processing the algorithm, the octrees of both operands are traversed in a breadth-first manner. On each level, pairs of octants are created with one octant from octree A and one octant from octant B. This figure shows a corresponding quadtree example.

Figure 7. If the octree generation is coupled with the distance algorithm only refinements at relevant places are necessary.

Figure 8. The function *findClosestCells* identifies the set of cell pairs that might be closest together. It calls itself recursively to enter the next level of refinement.

The function *findClosestCells* identifies the set of cell pairs that might be closest together. It calls itself recursively to enter the next level of refinement.

Figure 9. The helper function *createCandidateList()* filters out all irrelevant cell pairs by calculating the lower and the upper bound of the distance for each cell pair and comparing it with the lowest upper bound on this level.

Figure 10. All possible combinations of cells that satisfy the distance condition. The tiles correspond to different distances in y direction (a-1, a and a+1).

Figure 11. 2D example for the generation of an AABB tree for a given polygon. On each level, the current AABB is divided along the longest axis into two subspaces. By checking the edges' midpoints for containment, each edge is assigned to one of the two subspaces. Subsequently, a bounding box containing all edges of one subspace is created and inserted into the AABB tree as a child of the current AABB. This is repeated recursively until the maximum tree level is reached. Note that the leaf nodes may contain more than one edge.

Figure 12. The diagram plots the scaling of the octree algorithm with respect to the number of facets per sphere.

Figure 13. The diagram shows that the performance of the AABB-based algorithm depends largely on the chosen tree-depth.

Figure 14. Real-world example: the 3D model of the structural framework of a 5-storey building consisting of 216 elements. Three different algorithms were employed for calculating the distance between one of the columns and all other building components.

Figure 15. The diagram shows the performance of the diverse algorithms for the real-world example from Figure 14. Please note the logarithmic scaling of the time-axis in the diagram.

Table 1. The results of the performance measurements of the octree-based algorithm. All timings are in milliseconds.

Table 2. The results of the performance measurements of the AABB-based algorithm. All timings are in milliseconds.

Table 3. Timings taken for a real-world example from Figure 14 using the octree and the AABB tree approach. In the case of the octree algorithm, a tree generation method based on

triangle-octant intersection tests (Akenine-Möller, 2001) was employed in conjunction with the approach proposed by Mundani (2005). All timings are in milliseconds.