An octree-based implementation of directional operators in a 3D Spatial Query Language for Building Information Models

André Borrmann, Stefanie Schraufstetter, Christoph van Treeck, Ernst Rank
Computational Civil Engineering
{borrmann,schraufstetter,treeck,rank}@bv.tum.de
Technische Universität München, Arcisstrasse 21, 80290 Munich, Germany

ABSTRACT
In a current research project, our group is developing a 3D Spatial Query Language that enables the spatial analysis of Building Information Models and the extraction of partial models that fulfil certain spatial constraints. Among other features, the spatial language includes directional operators, i.e. operators that reflect the directional relationships between 3D spatial objects, such as *northOf, southOf, eastOf, westOf, above* and *below*. The paper presents in-depth definitions of the semantics of these operators by means of point set theory. It further gives an overview on the possible implementation of directional operators using a new space-partitioning data structure called *slot-tree*, which is derived from the objects' octree representation. The slot-tree allows for the application of recursive algorithms that successively increase the discrete resolution of the spatial objects employed and thereby offer the possibility for a trade-off between computational effort and required accuracy.

KEY WORDS: Spatial Query Language, Building Information Modelling, Direction, Octree

1. INTRODUCTION

The current project develops a 3D Spatial Query Language that allows for the spatial analysis of Building Information Models (BIMs) and for the extraction of partial models that fulfil certain spatial constraints. In existing query languages for BIMs, such as the *Product Model Query Language* of the EuroStep Model Server[1] and the *Partial Model Query Language* of the IFC Model Server (Adachi, 2003), the utilization of spatial relations within a query is limited to simple containment relationships predefined in the product model. This is mainly due to the structure of the underlying BIM which does not incorporate the explicit geometry of the building components.

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). 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*. 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. They comprise metric (Borrmann et al., 2007), directional and topological operators. This paper focuses on the directional operators.

Our concept for realizing the proposed spatial query language is based on object-relational database technique implementing the ISO standard SQL:1999 (ISO, 1999) which allows the extension of the database type system in an object-oriented way, especially by abstract data types (ADTs), which may include methods (Melton, 2003). By using an ORDBMS, spatial data types and spatial operators can be made directly available to the end-users, enabling them to formulate queries as shown in Figure 1. As can be seen in the example, spatial operators such as *above* are implemented as methods of spatial data types and can be used in the WHERE part of an SQL statement. In contrast to purely object-oriented databases, these

[1] http://www.eurostep.com/prodserv/ems/ems.html

methods are stored and processed server-side, resulting in dramatically reduced network traffic compared to a client-side solution. This paper discusses the formal definition of directional operators and their implementation as server-side methods.

```
SELECT *
FROM BuildingComponents comp, Ceilings c1, Ceilings c2
WHERE comp.isAbove(c1) AND comp.isBelow(c2) AND c1.id=1 AND c2.id=2
```

Figure 1. Example of a spatial SQL query returning all building components located above ceiling 1 and below ceiling 2.

Developing a spatial query language for BIMs is a first step towards higher spatial concepts directly available in computer-aided engineering tools. The 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. It is expected that spatial modeling and processing will play an increasing role in future engineering systems.
Especially interesting is the combination of the proposed spatial query language for BIMs with techniques for the extraction of air volumes from 3D models developed by our group (van Treeck & Rank, 2004), (van Treeck & 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 entities such as rooms and floors.
Although many of the methods and techniques developed within this project are applicable to *Spatial Reasoning,* this field of study is not within the scope of the currently conducted research.

2. FORMAL DEFINITION

Direction is a binary relation of an ordered pair of objects *A* and *B*, where *A* is the *reference object* and *B* is the *target object*. The third part of a directional relation is formed by the *reference frame*, which assigns names or symbols to space. According to (Retz-Schmidt, 1988), three types of reference frames can be distinguished: An *intrinsic* reference frame relies on the inner orientation of the spatial objects, such as defined by the front side of a building, for example. A *deictic* reference frame is based on the position and orientation of the observer. In contrast, an *extrinsic* reference frame is defined by external reference points. In geographical applications, for example, these external reference points are the earth's north and south pole.
The models for the representation of directional relations between spatial objects developed so far only work in 2D space (Peuquet & Zhan, 1987) or simply use points as reference objects, like the cone-based and the projection-based model (Hong, 1994) (Frank, 1996), and are accordingly not suitable for engineering purposes. To fulfil the requirements of different application scenarios, we developed two new models for representing directional relationships between 3D objects: the projection-based model and the halfspace based model. Both models use an intrinsic reference frame that is determined by the orientation of the coordinate system chosen by the user.
The proposed directional models are appropriate for arbitrary combinations of spatial types and are based on a separate examination of directional relationships with respect to the three coordinate axes. For each axis, there are exactly two possible relations, of which at most one holds: *eastOf* and *westOf* in the case of the *x*-axis, *northOf* and *southOf* for the *y*-axis and *above* and *below* for the *z*-axis. As opposed to the directional models used in (Guesgen, 1989), (Papadias et al., 1995) and (Goyal, 2000), the directional relationships of the relevant axis are not superimposed. Accordingly, the relationship between two spatial objects is not *north-east*, for example, but *northOf* **and** *eastOf*.

Both models distinguish between two groups of directional operators. Whereas the *strict* directional operators only return *true* if the entire target object falls into the respective direction partition, the *relaxed* operators also return *true* if only parts of it do so.

2.1 THE PROJECTION-BASED DIRECTIONAL MODEL

In the *projection-based model*, the reference object is extruded along the coordinate axis corresponding to the directional operator. The target object is tested for intersection with this extrusion. Let *reference object A* and *target object B* be spatial objects of type *SpatialObject* and $a \in A$, $b \in B$. Then the formal definitions of the relaxed projection-based operators read:

$$
\begin{aligned}
eastOf\_proj(A,B) &\Leftrightarrow \exists a,b \text{ with } a_y = b_y \wedge a_z = b_z : a_x < b_x \ , \\
westOf\_proj(A,B) &\Leftrightarrow \exists a,b \text{ with } a_y = b_y \wedge a_z = b_z : a_x > b_x \ , \\
northOf\_proj(A,B) &\Leftrightarrow \exists a,b \text{ with } a_x = b_x \wedge a_z = b_z : a_y < b_y \ , \\
southOf\_proj(A,B) &\Leftrightarrow \exists a,b \text{ with } a_x = b_x \wedge a_z = b_z : a_y > b_y \ , \\
above\_proj(A,B) &\Leftrightarrow \exists a,b \text{ with } a_x = b_x \wedge a_y = b_y : a_z < b_z \ , \\
below\_proj(A,B) &\Leftrightarrow \exists a,b \text{ with } a_x = b_x \wedge a_y = b_y : a_z > b_z \ .
\end{aligned}
$$

The *relaxed* operators return *true* if there is an intersection, otherwise *false*. By contrast, the *strict* projection-based operators only return *true* if the target object is completely within the extrusion body. Accordingly, the formal definitions of the strict operators are:

$$
\begin{aligned}
eastOf\_proj\_strict(A,B) \Leftrightarrow \forall b : &(\exists a : a_y = b_y \wedge a_z = b_z \wedge a_x < b_x) \wedge \\
&(\nexists a : a_y = b_y \wedge a_z = b_z \wedge a_x \geq b_x) \ , \\
westOf\_proj\_strict(A,B) \Leftrightarrow \forall b : &(\exists a : a_y = b_y \wedge a_z = b_z \wedge a_x > b_x) \wedge \\
&(\nexists a : a_y = b_y \wedge a_z = b_z \wedge a_x \leq b_x) \ , \\
northOf\_proj\_strict(A,B) \Leftrightarrow \forall b : &(\exists a : a_x = b_x \wedge a_z = b_z \wedge a_y < b_y) \wedge \\
&(\nexists a : a_x = b_x \wedge a_z = b_z \wedge a_y \geq b_y) \ , \\
southOf\_proj\_strict(A,B) \Leftrightarrow \forall b : &(\exists a : a_x = b_x \wedge a_z = b_z \wedge a_y > b_y) \wedge \\
&(\nexists a : a_x = b_x \wedge a_z = b_z \wedge a_y \leq b_y) \ , \\
above\_proj\_strict(A,B) \Leftrightarrow \forall b : &(\exists a : a_x = b_x \wedge a_y = b_y \wedge a_z < b_z) \wedge \\
&(\nexists a : a_x = b_x \wedge a_y = b_y \wedge a_z \geq b_z) \ , \\
below\_proj\_strict(A,B) \Leftrightarrow \forall b : &(\exists a : a_x = b_x \wedge a_y = b_y \wedge a_z > b_z) \wedge \\
&(\nexists a : a_x = b_x \wedge a_y = b_y \wedge a_z \leq b_z) \ .
\end{aligned}
$$

Figure 2 illustrates the consequences of these definitions. In colloquial language, the semantics of the operator *above_proj_strict*, for example, could be described as *"directly above"* or *"exceptionally above"*.

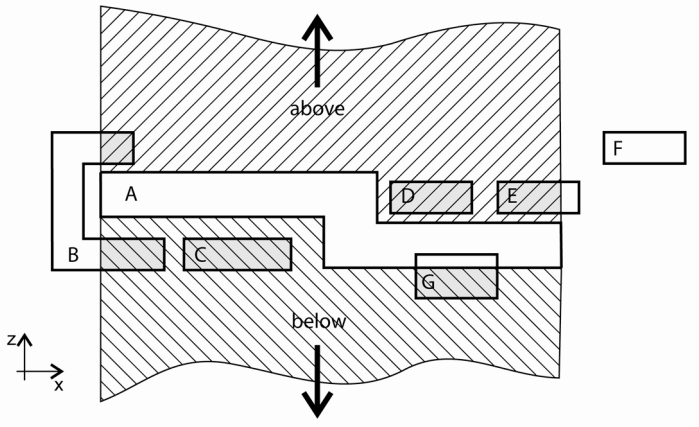

Figure 2. The projection-based directional model relies on the extrusion of the reference object (*A*) along the respective coordinate axis. In the illustrated example, the relaxed operator *above_proj* returns *true* for the target objects *B, D, E and G*, but *false* for any other target object. By contrast, the strict operator *above_proj_strict* also returns *false* for *B, G* and *E*.

2.2 THE HALFSPACE-BASED MODEL

The second model is based on halfspaces that are described by the reference object's axis aligned bounding box (AABB). In this model, the target object is tested for intersection with the halfspace corresponding to the directional predicate. In analogy to the projection-based model, we distinguish strict and relaxed operators. The formal definitions of the relaxed operators are:

$$
\begin{aligned}
eastOf\_hs(A,B) &\Leftrightarrow \forall a : \exists b : a_x < b_x \ , \\
westOf\_hs(A,B) &\Leftrightarrow \forall a : \exists b : a_x > b_x \ , \\
northOf\_hs(A,B) &\Leftrightarrow \forall a : \exists b : a_y < b_y \ , \\
southOf\_hs(A,B) &\Leftrightarrow \forall a : \exists b : a_y > b_y \ , \\
above\_hs(A,B) &\Leftrightarrow \forall a : \exists b : a_z < b_z \ , \\
below\_hs(A,B) &\Leftrightarrow \forall a : \exists b : a_z > b_z \ .
\end{aligned}
$$

For the *relaxed* operators to return *true* it is sufficient if parts of the target object are within the relevant halfspace. By contrast, the *strict* operators only return *true* if the target object is completely within the halfspace. The formal definitions of the strict operators accordingly read:

$$
\begin{aligned}
eastOf\_hs\_strict(A,B) &\Leftrightarrow \forall a,b : a_x < b_x \ , \\
westOf\_hs\_strict(A,B) &\Leftrightarrow \forall a,b : a_x > b_x \ , \\
northOf\_hs\_strict(A,B) &\Leftrightarrow \forall a,b : a_y < b_y \ , \\
southOf\_hs\_strict(A,B) &\Leftrightarrow \forall a,b : a_y > b_y \ , \\
above\_hs\_strict(A,B) &\Leftrightarrow \forall a,b : a_z < b_z \ , \\
below\_hs\_strict(A,B) &\Leftrightarrow \forall a,b : a_z > b_z \ .
\end{aligned}
$$

The examples in Figure 3 illustrate the consequences of these definitions.

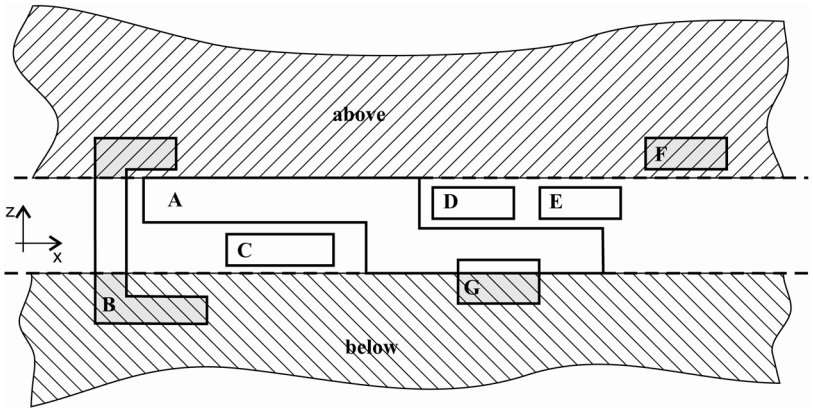

Figure 3. The halfspace-based directional model relies on the halfspaces formed by the reference object's axis-aligned bounding planes. In the example shown, *A* is the reference object. Accordingly, the relaxed operator *above_hs* returns *true* for the target objects *B* and *F*, but *false* for any other target object. The strict operator *above_hs_strict* also returns *false* for B. In contrast to the projection-based model, the halfspace-based model cannot assign a direction to target object C, D or E.

3. IMPLEMENTATION

3.1 PROVIDING SPATIAL TYPES AND OPERATORS IN SQL

The spatial types as defined in (Borrmann, 2006) and the directional operators specified in Section 1 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[2]:

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

    MEMBER FUNCTION below_proj(object SPATIALOBJECT)  RETURN NUMBER
      EXTERNAL NAME 'below(SpatialObjectJ) 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. After 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[3] in the WHERE part:

```
SELECT *
FROM buildingcomponents bc1, buildingcomponents bc2
WHERE bc1.id = '58' and bc2.above_proj(VALUE(bc1) = 1
```

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

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 the future, we will 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. Additionally, we will store semantic information, such as BIM classes and non-geometric attributes to allow the usage of such information within the selection predicate.

3.2 IMPLEMENTATION OF THE HALFSPACE-BASED DIRECTIONAL MODEL

The halfspace-based directional model can be implemented easily and efficiently by using the axis-aligned bounding boxes of both the reference and the target object. It merely has to be checked whether the vertices of the target object's bounding box are within the respective halfspace with regard to the reference object. To this end, only the coordinate associated with

---

[2] Note that Oracle SQL does not support the datatype BOOLEAN. Hence, the return value of the directional operators is of type NUMBER, representing FALSE by 0 and TRUE by 1.

[3] Note that the intended integration of a spatial indexing method will require the declaration of SQL operators.

the examined direction has to be tested. In the case of the relaxed operators, in order to return *true*, it is sufficient for the coordinate of one of the vertices of the target's AABB to be smaller/greater than that of all the vertices of the reference's AABB.
Let $A_{min} = (a_{min\_x}, a_{min\_y}, a_{min\_z})$ and $A_{max} = (a_{max\_x}, a_{max\_y}, a_{max\_z})$ be the vertices of the reference object's AABB and B_{min} and B_{max} the vertices of the target object's AABB accordingly. Then the relaxed operator *above_hs*, for example, checks whether $b_{max\_z} \geq a_{max\_z}$ is fulfilled. The strict operator *above_hs_strict*, on the other hand, checks whether $b_{min\_z} > a_{max\_z}$ is fulfilled (Figure 4).

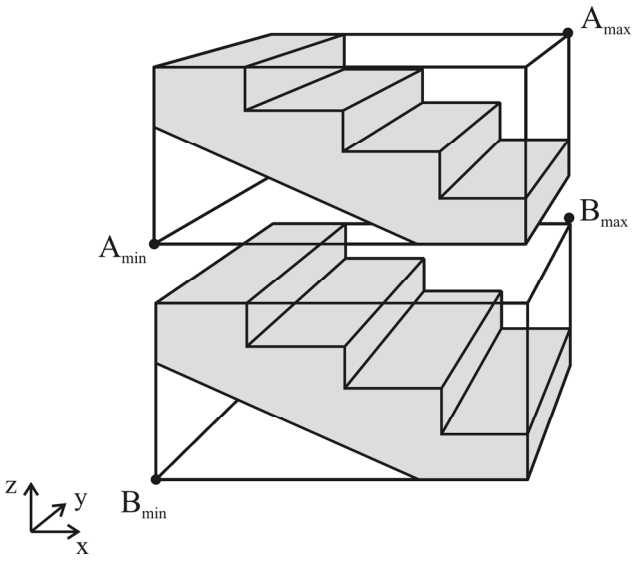

Figure 4. The implementation of the halfspace-based directional model is based on a comparison between the respective coordinate of the vertices of the reference and target object's AABBs.

3.3 IMPLEMENTATION OF THE PROJECTION-BASED DIRECTIONAL MODEL

The implementation of the projection-based model is much more complex than that of the halfspace-based model. The proposed algorithm is based on a hierarchical space-partitioning data structure *slot-tree*, which is related to the octree data structure.
The octree is a space-dividing, hierarchical tree data structure for the discretized representation of 3D volumetric geometry (Meagher, 1982). Each node in the tree represents a cubic cell (an *octant*) and is either *black*, *white* or *grey*, symbolizing whether the octant lies completely in the *interior*, in the *exterior* or on the *boundary* of the discretized object. Whereas *black* and *white* octants are branch nodes, and accordingly have no children, *grey* octants are interior nodes that have exactly eight children. The union of all child cells is equal to the parent cell, and 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*.
In our implementation concept for projection-based directional operators, each spatial object is represented by an individual octree. There are several different approaches for generating an octree out of the object's boundary representation, most of which are based on a recursive algorithm that starts at the root octant and refines those cells that lie on the boundary of the original geometry, i.e. which are coloured *grey*. A very efficient creation method based on halfspaces formed by the object's bounding faces is presented in (Mundani, 2003) and is used in our implementation. But before applying the octree-based algorithm, it is wise to conduct a rough test based on the relative position of the operands' AABBs.

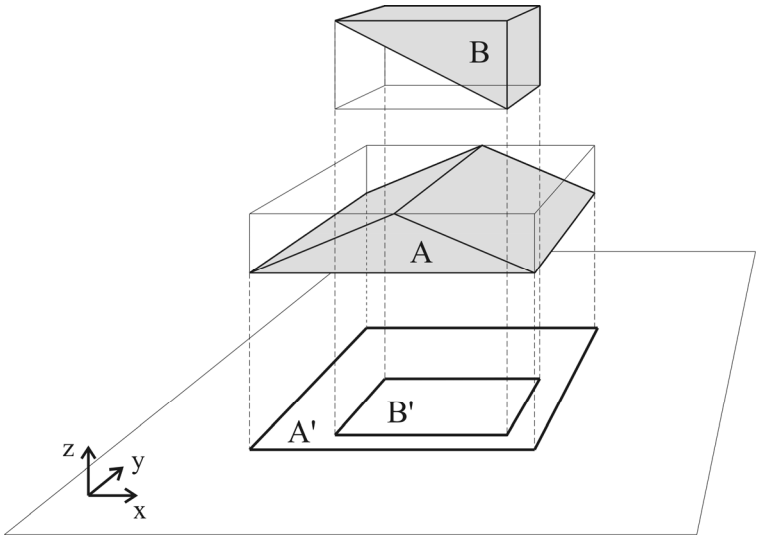

Figure 5. Implementing the initial check to determine whether the projection of the target object's AABB lies completely within the projection of the reference object's AABB.

In the case of the relaxed operators, it is necessary for the projections of the AABBs on the plane orthogonal to the direction under examination to overlap. In the case of the strict operators, the projected AABB of the target object has to lie completely inside the projection of the reference object's AABB (Figure 5). If these initial conditions are not fulfilled, the operators return *false*.

Once the initial test has been passed, a detailed examination based on the exact geometry of the operands has to be conducted. As mentioned above, the proposed algorithms use a space-partitioning data structure called a *slot-tree*, which is introduced here. A *slot-tree* re-organizes the cells of an octree with respect to their position orthogonal to the considered coordinate axis.

The basic element of a slot-tree is the *slot*. If we take a look at the *z*-direction, for example, a slot contains all cells that lie above each other (Figure 6, left). It accordingly contains a list of octants in the order of their appearance. The octants may stem from different levels of the octree, and consequently may have different sizes (Figure 6, right). This also means that one octant might appear in the list of different slots. Introducing the *slot* data structure allows for the application of simple tests based on the colour and absolute position of the cells contained therein in order to decide whether the examined directional predicate is fulfilled or not.

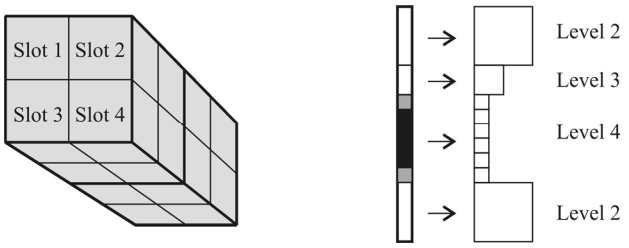

Figure 6. Left: Slots in 3D. Right: A slot in 2D that contains cells from different levels of the underlying quadtree. Slot 1212 from Figure 7.

In analogy to the octree, the slot-tree organizes the slots in a hierarchical way. Each node in a 3D slot-tree has either 4 or no children, dependent on whether the corresponding slot contains grey octants. A slot-tree may be directly derived from an existing octree representation, or

generated on-the-fly while processing the algorithm of the directional operator. The procedure is illustrated in Figure 7. Traversing the octree top-down in a breadth-first manner, we proceed to build up the slot-tree, generating child slots and inserting them into the slot-tree, as required. Such a refinement is necessary if at least one cell in the current slot is *grey*. By coupling the generation of octree and slot-tree with the processing of the directional operator, it is possible to avoid unnecessary refinements at places of no relevance for the operator's results.

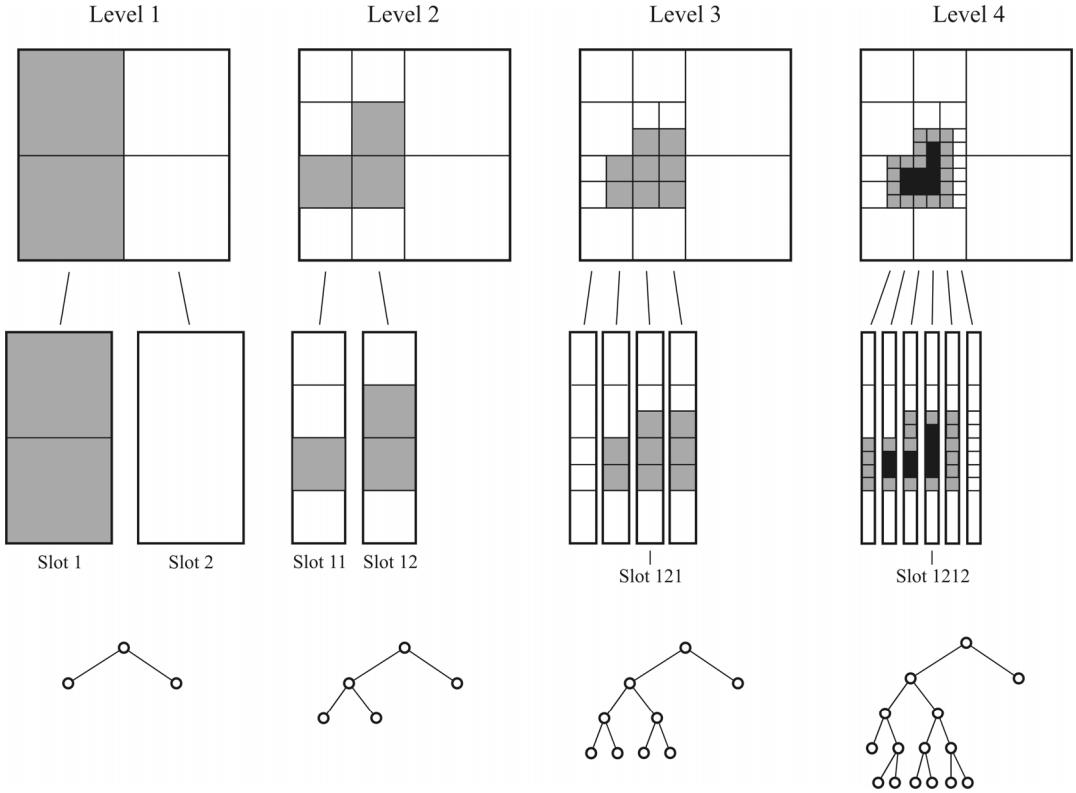

Figure 7. Generation of a 2D slot-tree up to level 4. A slot is only refined if it contains at least one grey quadrant. A 2D slot-tree can be derived directly from the quadtree presentation of the object's geometry, a 3D slot-tree from an octree representation accordingly.

Due to the differing semantics, strict and relaxed operators are implemented differently. Both algorithms rely on the principle of creating slot pairs with one slot from object *A* and one from object *B*, both covering the same subset of space, and performing local tests on these pairs. Due to the limited space available, a detailed description of these algorithms will be presented in follow-up publications.

4. CONCLUSION

This paper presents in-depth definitions of directional predicates in 3D space by introducing two directional models: The *halfspace-based model* where the direction partitions are formed by the reference object's axis aligned bounding planes, and the *projection-based model* that relies on the extrusion of the reference object in the respective direction. The notions of *strict* and *relaxed* predicates have been defined for both models.

Whereas the halfspace-based model can be implemented by simple tests using the axis aligned bounding boxes of both the reference and the target object, the algorithms for implementing the projection-based model are much more complex. The paper gives an overview on a possible implementation by means of slot-trees, a new space-partitioning data structure that is introduced here. It can be derived from the octree representation and allows for the appli-

cation of local tests based on the colour and location of the underlying octants. A detailed description of the recursive algorithms on the basis of the slot-tree data structure will be presented in follow-up publications. Figure 8 shows the prototypical client application used to submit a spatial query and visualize its results.

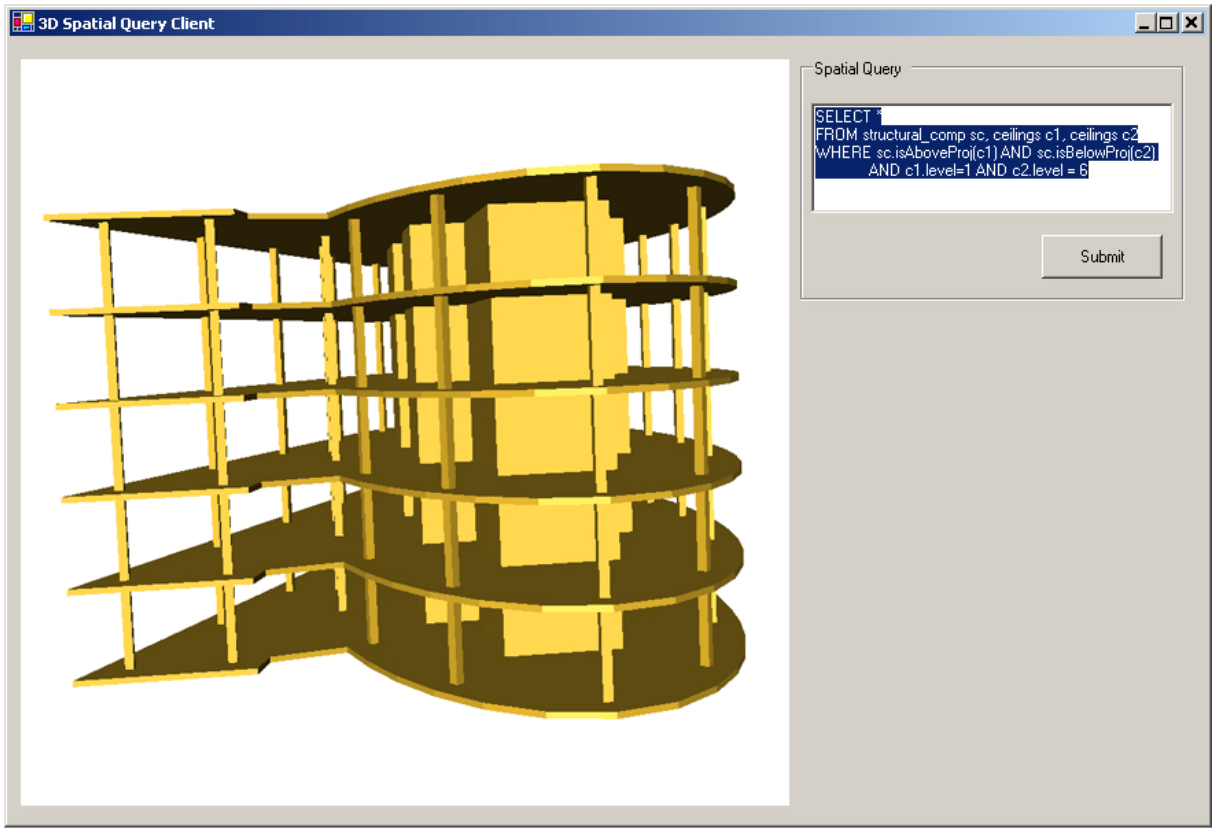

Figure 8. The client application used to submit spatial queries und visualize the results. The example shows a partial model that was extracted from the structural model of the building. It contains all components above the first and below the sixth ceiling and was created by the use of an SQL query whose conditional statement (WHERE part) contains the directional operators *above_proj* and *below_proj*.

AKNOWLEDGMENTS

The project presented here is sponsored by the German Research Foundation (DFG) under grant Ra 624/17-1.

REFERENCES

Adachi, Y. (2003). ”Overview of partial model query language.“ Proc. of the 10th Int. Conf. on Concurrent Engineering.

Borrmann, A., Schraufstetter, S., van Treeck, C., Rank, E. (2007). “An iterative, octree-based algorithm for distance computation between polyhedra with complex surfaces.” Proc. of the Int. ASCE Workshop on Computing in Civil Engineering. In press.

Borrmann, A., van Treeck, C. and Rank, E. (2006). “Towards a 3D Spatial Query Language for Building Information Models.” Proc. Joint Int. Conf. of Computing and Decision Making in Civil and Building Engineering (ICCCBE-XI).

Frank, A. (1996). “Qualitative Spatial Reasoning: Cardinal Directions as an Example.” *Int. Journal of Geographic Information Systems,* 10(3) 269-290

Guesgen, H. (1989). “Spatial Reasoning based on Allen’s Temporal Logic.” Technical Report. International Computer Science Institute, Berkley, CA, USA.

Goyal, R. (2000). Similarity Assessment for Cardinal Directions between Extended Spatial Objects. Ph.D. thesis, University of Maine.

Hong, J. (1994). Qualitative Distance and Direction Reasoning in Geographic Space. Ph.D. thesis, University of Maine.

ISO (1999). ISO International Standard 9075-99: Database Language SQL.

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.*

Mundani, R. P., Bungartz, H.-J., Rank, E., Romberg, R. and Niggl, A. (2003). “Efficient Algorithms for Octree-Based Geometric Modelling,” Proc. of 9th Int. Conf. on Civil and Structural Engineering Computing.

Papadias, D., Theodoridis, Y. and Sellis, T. (1994). “The Retrieval of Direction Relations using R-Trees.” Proc. of the 5th Int. Conf. on Database and Expert Systems Applications.

Peuquet, D. & Zhan, C. (1987). An Algorithm to Determine the Directional Relationship Between Arbitrarily-Shaped Polygons in the Plane. *Pattern Recognition,* 20(1), 65-74

Retz-Schmidt, G. (1988). “Various Views on Spatial Prepositions.” *AI Magazine* 9(2), 95-105

van Treeck, C. & Rank, E. (2004) "Analysis of Building Structure and Topology Based on Graph Theory." *Proc. of the 10th Int. Conf. on Computing in Civil and Building Engineering (ICCCBE-X)*

van Treeck, C. & Rank, E. (2007) "Dimensional reduction of 3D building models using graph theory and its application in building energy simulation." *Engineering with Computers,* Accepted for publication. DOI: 10.1007/s00366-006-0053-7