# Specification and implementation of directional operators in a 3D Spatial Query Language for Building Information Models

André Borrmann [*], Ernst Rank

*Computational Civil and Environmental Engineering, Technische Universität München, Arcisstrasse 21, 80290 Munich, Germany*

**Abstract**

A Spatial Query Language enables the spatial analysis of Building Information Models and the extraction of partial models that fulfill certain spatial constraints. Among other features, the developed spatial query 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 two new directional models for extended 3D objects, the projection-based and the halfspace-based model, by using point set theory notation. It further describes the possible implementation of directional operators using a newly developed 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 enables the user to trade-off between computational effort and the required accuracy. The article also introduces detailed investigations on the runtime performance of the developed algorithms.

*Key words:* Building Information Modeling, Spatial Query Language, Directional Relationships, Slot-tree, Octree

## 1. Introduction

Humans view buildings primarily as an aggregation of physical objects with well-defined geometry and specific spatial relations. 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 relations 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-supported building design has concentrated mainly on the development of a semantic object-oriented building model, also called *Product Model* or *Building Information Model* (BIM) [1–5]. These efforts have resulted in the widely known standards *Industry Foundation Classes* (IFC) [6] and CIS/2 [7].

Building product models, such as the IFC, do not normally describe the geometry of a building component explicitly, i.e. not by using a Boundary Representation (B-Rep) or the Constructive Solid Geometry (CSG) approach, 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* [8] 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

[*] Corresponding author.
*Email addresses:* `borrmann@bv.tum.de` (André Borrmann), `rank@bv.tum.de` (Ernst Rank).

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 furniture contained in a certain room or office, for example.

Other spatial relationships, such as directional relations (e.g. one object above another) are completely ignored by the IFC product model. From the point of view of product modeling this makes absolute sense, because storing all possible spatial relations would (1) result in huge models with many object relations not needed by the majority of applications and (2) introduce even more redundancy, since directional 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 relations are not required 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 AEC domains, such as tunnel, road or bridge engineering, only rudimentary product models exist [9] 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 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 the technical details of efficient data retrieval. 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 our 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 [10,11]. 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. They comprise
– metric (*distance, closerThan, fartherThan* etc.),
– directional (*above, below, northOf* etc.) and
– topological (*touch, within, contains* etc.)
operators.

While the metric operators are presented in [12], this paper discusses the definition and implementation of the directional 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.

## 2. Related work

### 2.1. *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. and provide 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 SQL were also realized in the GIS context. In the late 80's, a multitude of different dialects was developed, including PSQL [13], Spatial SQL [14], GEOQL [15], KGIS [16] and TIGRIS [17]. A good overview of the different dialects and the basic advantages of a SQL-based implementation is provided in [18].

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 [19,20]. There is now a wide range of commercial 2D spatial database systems, the most wide-spread ones being *PostGIS*, *Oracle Spatial* and *Informix Geodetic Datablade*. The majority of available spatial databases complies to 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 [21].

In [22] 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, she stores floor plans of buildings in a GIS database in order to use it's 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 [23], 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 [24–26] 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 [27].

Though [28–31] provide concepts and data structures for storing 3D city models in spatial databases, the definition and implementation of directional operators has been completely omitted in these papers.

[32] 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, which we call *slot-tree*.

## 2.2. *Directional relationships*

Directional operations between point-shaped objects can be defined in a simple and unambiguous way [33,34]. In order to apply these definitions on extended one-, two- or three-dimensional objects a point-based approximation, such as the center of gravity, is normally used. This rough approximation often causes results that do not comply with the intuitive expectations of the user (Fig. 1).

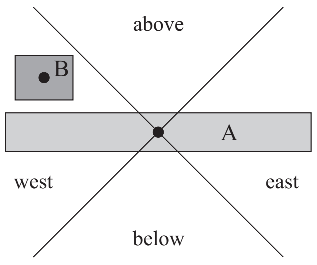

Fig. 1. Approximating target and reference object by their centroids may cause results that do not comply with the intuitive expectations of the users. In this example, $B$ would be classified as being *west* of $A$ and not as being *above* it.

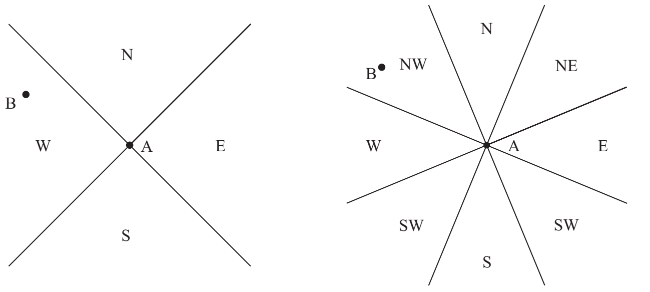

Fig. 2. The cone-based model for point-shaped objects in 2D. Left–hand side: 4-direction model. Right-hand side: 8-direction model.

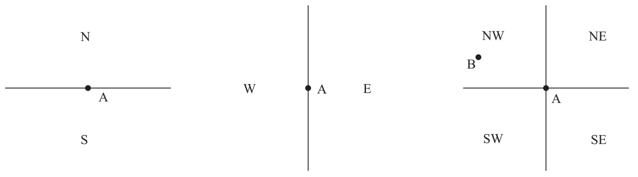

Fig. 3. The projection-based model for point-shaped objects in 2D [33]. The space is dissected horizontally (left) and vertically (right), then the resulting halfspaces are superimposed, thus creating 4 space partitions.

For the two-dimensional space, there are two known models for defining directional relations between points: the *cone-based* and the *projection-based* model. The cone-based model dissects the space around the reference point in either four partitions of 90° (Fig. 2, left-hand side) or eight partitions of 45° (Fig. 2, right-hand side) [33–37]. The direction of the target point with respect to the reference point is defined by the partition in which the target point is located.

The projection-based model [33] dissects the space by means of horizontal and vertical lines that cross at the reference point. While the horizontal line creates a northern and southern halfspace, the vertical line creates the western and eastern halfspace. Superimposing the halfspaces produces four directional partitions, namely *north-west*, *north-east*, *south-east* and *south-west*.

In [38] a framework for representing directional relationships between a line and a point is presented. This model can be employed to assign one of the 15 qualitative distinct locations shown in Fig. 4 to the target point.

Allen [39] uses a set of 13 *interval relations* to model spatial relationships between intervals in one-dimensional

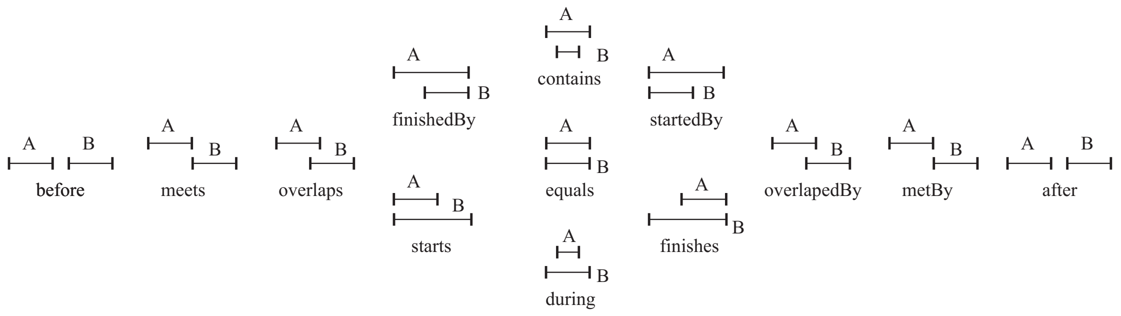

Fig. 5. Allen distinguishes 13 cases to express qualitative relations between intervals in 1D [39].

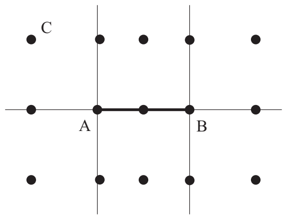

Fig. 4. According to [38] a point C can take one of the 15 illustrated, qualitatively distinct positions with respect to line AB.

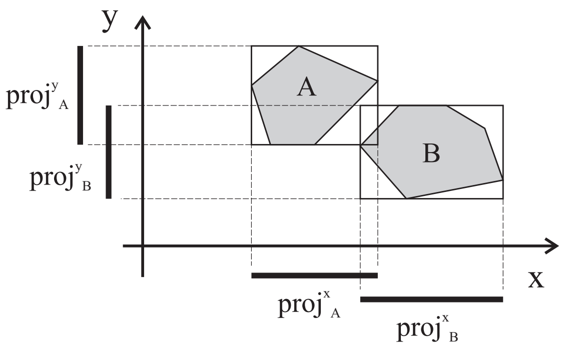

Fig. 6. The model in [40] maps the bounding boxes of target and reference object onto x- and y-axis and then applies Allen's interval relations to classify the spatial relationship. Since MBRs are used to approximate extended objects, the result may differ from the real relation of the exact shapes; the shown configuration would be classified as an *overlap* situation, for example.

space (Fig. 5). Unfortunately, instead of clearly separating topological from directional relations, this model mixes them up. In [40] Allen's intervals are mapped on the x- and the y-axis in order to express directional (and topological) relationships between extended objects in 2D. Only 8 of Allen's 13 intervals are used on each axis, resulting in 64 different relations between 2D objects. The model approximates the geometry of extended objects by means of minimum bounding rectangles (MBR), so the resulting spatial relation may deviate from that of the exact shapes. [41] and [42] follow the same approach, but use a different set of interval relations.

Goyal [43] also partitions the space around the reference object on the basis of projections of the MBR, but introduces a direction-relation matrix that is able to record precisely into which direction tiles a target object falls. But because the model does not assign names to direction relations, it is only of limited use for the application in a spatial query language.

The approach presented in [37] models direction as a unit vector and orientation as a set of directions. The model can not only be used for expressing absolute directional relations, but also for object- and viewer-based orientation. Unfortunately also this model uses the centroids of reference and target object, resulting in the same limitations as stated above. Although relative orientation has important applications in the field of similarity assessment and data mining [43–45], it is of less significance for the area of interest addressed by this paper.

To summarize, no directional model that has been developed so far meets the particular requirements of a spatial query language for 3D building information models. Most models approximate the objects involved either by means of centroids or minimum bounding boxes, producing unexpected results when applied on building components with complex geometry. Very little work has been carried out on directions in 3D space so far [46,37].

## 3. Formal specification of directional operators

Colloquial language is often vague and ambiguous when used to describe directional relationships. Because an unequivocal definition is essential for using directional relationships as conditions in a spatial query language, it is necessary to formally specify their semantics.

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 partitions.

According to [47], three types of reference frames can be distinguished: an *intrinsic* reference frame relies on the inner orientation of the spatial objects, such as that defined by the front side of a building, for example. A *deictic* reference frame is aligned to the position and orientation of the observer. By 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.

In a geographical context, we usually distinguish four *(north, east, south, west)* or eight space partitions *(north,*

*north-east, east, southeast, south, south-west, west, north-west).* In 3D context, normally the additional directional predicates *above* and *below* are used [48], which may also be employed in conjunction with the aforementioned 2D sub-direction, resulting in *north-east-above, east-above*, etc.

To meet 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 precisely two possible relations, only one of which holds at the most: *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. We haven chosen the names of geographical cardinal directions instead of *left, right, in front of, behind* to clearly label our models as observer-independent.

As opposed to the directional models used in [40,41] and [43], 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 two "flavors" of directional operators. Whereas the *strict* directional operators only return *true* if the entire target object falls into the respective directional partition, the *relaxed* operators also return *true* if only parts of it do so.

## 3.1. *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:

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

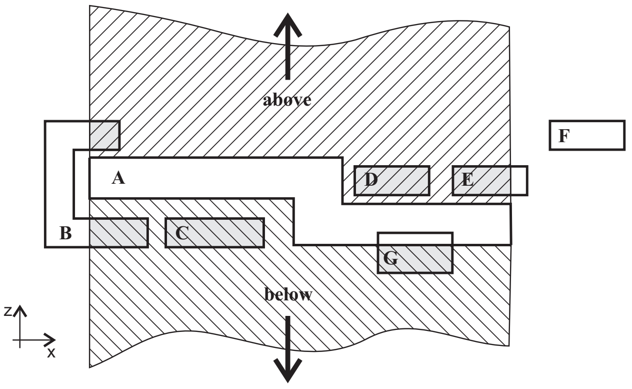

Fig. 7. The projection-based directional model relies on the extrusion of the reference object ($A$) along the respective coordinate axis. In the illustrated example, the relaxed operator *above_proj_relaxed* returns *true* for the target objects $B$, $D$, $E$ and $G$, but *false* for any other target object. By contrast, the strict operator *above_proj_strict* also returns *false* for $B$, $G$ and $E$.

$$\begin{aligned}
&\textit{above\_proj\_relaxed}\ (A,B) \Leftrightarrow \\
&\qquad \exists a,b : a_x = b_x \wedge a_y = b_y \wedge a_z < b_z\ , \\
&\textit{below\_proj\_relaxed}\ (A,B) \Leftrightarrow \\
&\qquad \exists a,b : a_x = b_x \wedge a_y = b_y \wedge a_z > b_z\ .
\end{aligned}$$

The relaxed operators return *true* if there is an intersection between the extrusion body and the target object, 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:

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

Fig. 7 illustrates the consequences of these definitions. In colloquial language, the semantics of the operator

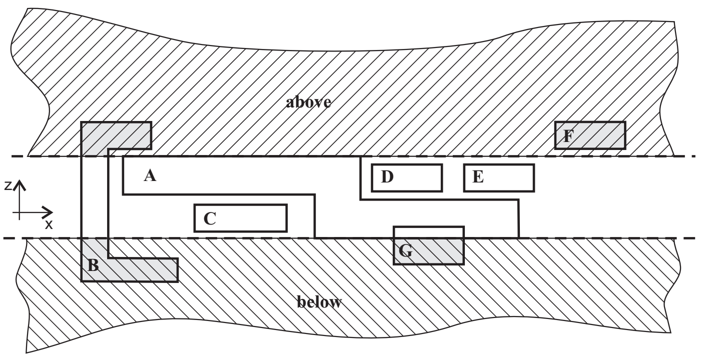

Fig. 9. In the halfspace-based directional model the direction tiles are formed by halfspaces defined by the reference object's axis-aligned bounding box. In the given example, $A$ is the reference object. The *relaxed* operator *above_hs_relaxed* returns *true* for the target objects $B$ and $F$, the *strict* operator *above_hs_strict* only returns *true* for $F$.

*above_proj_strict*, for example, could be described as "directly above" or "exceptionally above". In Fig. 8 the diverging semantics of the different directional operators are illustrated by a practical example.

### 3.2. *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:

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

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 that halfspace. The formal definitions of the strict operators accordingly read:

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

The examples in Fig. 9 illustrate the consequences of these definitions.

## 4. Implementation

### 4.1. *Providing spatial types and operators in SQL*

Our concept for realizing the proposed spatial query language is based on object-relational database techniques implementing the ISO standard SQL:1999 [49] 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) [50–52]. 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 like

```
SELECT *
FROM   buildingcomps comp, Ceilings c1, Ceilings c2
WHERE  comp.above(c1) AND comp.below(c2) AND
       c1.id = 1 AND c2.id =2
```

to extract all building components between Ceiling 1 and Ceiling 2.

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. 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 directional operators as server-side methods.

The spatial types as defined in [10,11] and the directional operators specified in Section 3 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 above_proj_strict(object SPATIALOBJECT)
   RETURN NUMBER
   EXTERNAL NAME 'above_proj_strict(SpatialObjectJ)
                 return int',

  MEMBER FUNCTION below_proj_strict(object SPATIALOBJECT)
   RETURN NUMBER
   EXTERNAL NAME 'below_proj_strict(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. Following its declaration, the user-defined SQL

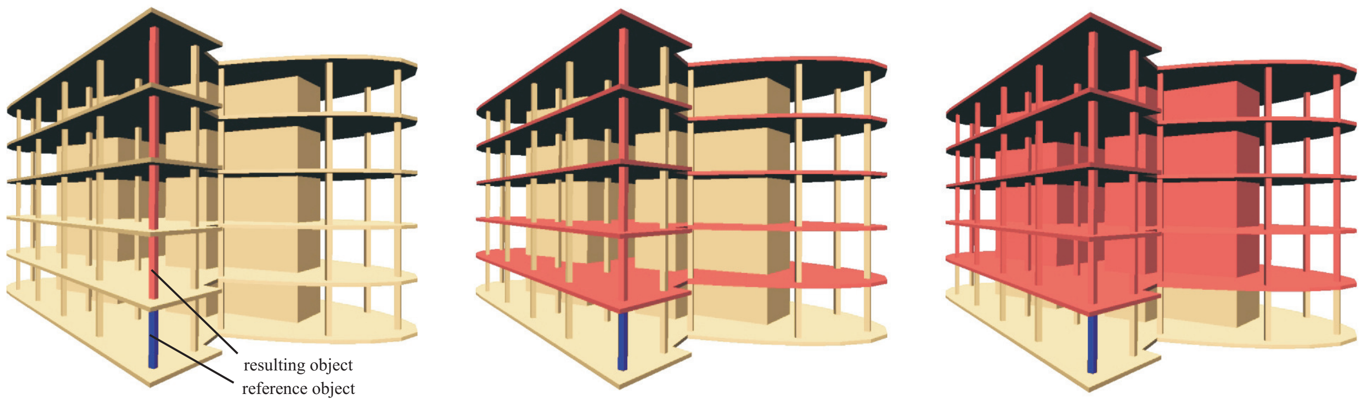

Fig. 8. Example illustrating the diverging semantics of the different directional operators. In each case, the reference object is depicted in blue, whereas the result set identified by the particular operator is depicted in red. Left: *above_proj_strict*. Middle: *above_proj_relaxed*. Right: *above_hs_relaxed*. Please note that, for this example, the result of *above_hs_strict* is equal to that of *above_hs_relaxed*.

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 bc1, buildingcomps bc2
WHERE bc1.id = '58' AND
      bc2.above_proj_strict(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_strict*, the Java stored procedure performs one of the algorithms presented in the next two sections.

## 4.2. *Implementation of the halfspace-based 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. All that needs to be checked is 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 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_relaxed*, 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 $b_{min,z} > a_{max,z}$ holds (Fig. 10).

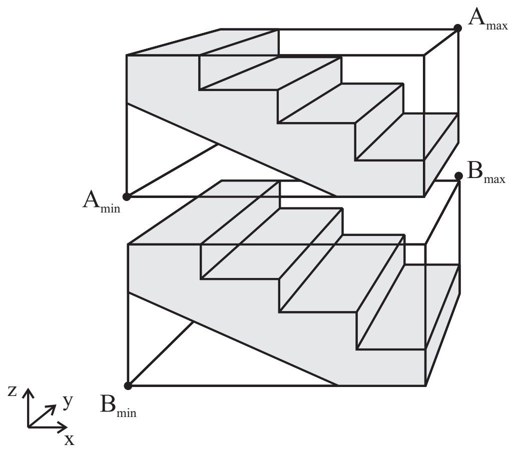

Fig. 10. The implementation of the halfspace-based directional model is based on a comparison between the respective coordinate of the vertices of the reference and target object's AABBs.

## 4.3. *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 called *slot-tree*, that has been developed by our team. It is derived from the octree data structure.

The octree is a space-dividing, hierarchical tree data structure for the discretized representation of 3D volumetric geometry [53–55]. Each node in the tree represents a cubic cell (an octant) and is either *black*, *white* or *gray*, symbolizing whether the octant lies completely inside, outside or on the boundary of the discretized object. Whereas *black* and *white* octants are branch nodes, and accordingly have no children, *gray* 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.

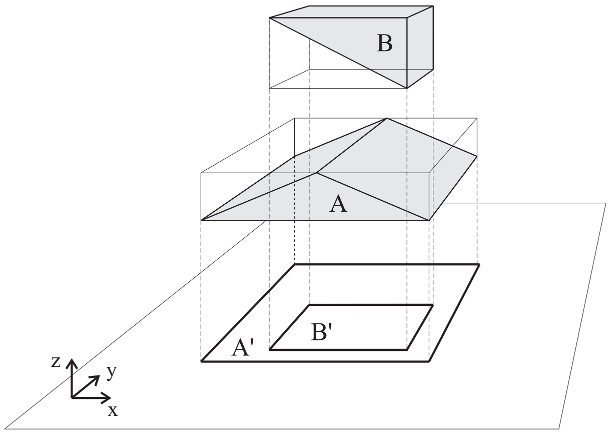

Fig. 11. The implementation of the projection-based operators first checks, whether the projection of the target object's AABB lies completely inside the projection of the reference object's AABB.

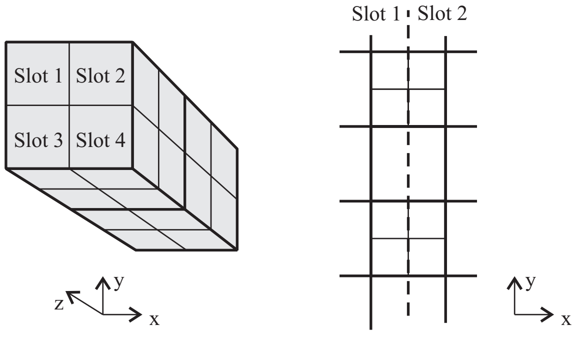

Fig. 12. Slots in 3- and 2-dimensional space, respectively. A slot in $z$-direction contains all the cells that lie above one another.

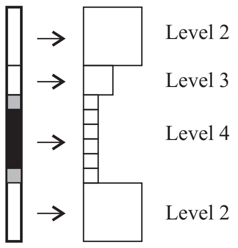

Fig. 13. A slot in 2D that owns cells from different levels of the underlying quadtree (Slot 1212 from Fig. 14).

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 colored *gray*.

For our implementation we use the creation method developed by Mundani [56,57] that is based on the halfspaces formed by the object's bounding faces. In Mundani's approach, the color classification is based on a simple evaluation of the plane equation of each halfspace for the respective octant and a subsequent combination by means of Boolean expressions. Accordingly, the algorithm automatically marks inner cells as *black* without the need to perform an expensive filling algorithm. As described in the next sections, the existence of *black* cells are an important prerequisite for the applicability of many rules that make it possible to abort the recursive algorithm at an early refinement level in many situations.

Before applying the octree-based algorithm, it is wise to conduct a rough test based on the relative position of the operands' AABBs. 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 (Fig. 11). If these initial conditions are not fulfilled, the operators return *false*.

Once the initial test has been passed, it is necessary to conduct a detailed examination based on the exact geometry of the operands. As mentioned above, the proposed algorithms use a newly developed, space-partitioning data structure called *slot-tree*. A slot-tree re-organizes the cells of an octree (octants) with respect to their position orthogonal to the coordinate axis under consideration.

The basic element of a slot-tree is the *slot*. A slot of level $k$ is formed by the extrusion of a level $k$ cell along the examined axis ($x$, $y$ or $z$ according the definitions in Section 3.1). It contains all cells which intersect with this extrusion. If we take a look at the $z$-direction, for example, a slot contains all the cells that lie above one another (Fig. 12). It accordingly possesses 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 (Fig. 13). 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 color and absolute position of the cells contained therein in order to decide whether the examined directional predicate is fulfilled or not.

In analogy to the octree, the slot-tree organizes the slots in a hierarchical manner. Each node in a 3D slot-tree has either 4 or no children, depending on whether the corresponding slot contains *gray* 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 Fig. 14. 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 *gray*. 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.

Due to the differing semantics, strict and relaxed operators are implemented differently. Thus, the corresponding algorithms are explained in two separate subsections. Both algorithms rely on the principle of creating slot pairs with one slot from object $A$ and one from object $B$, both cover-

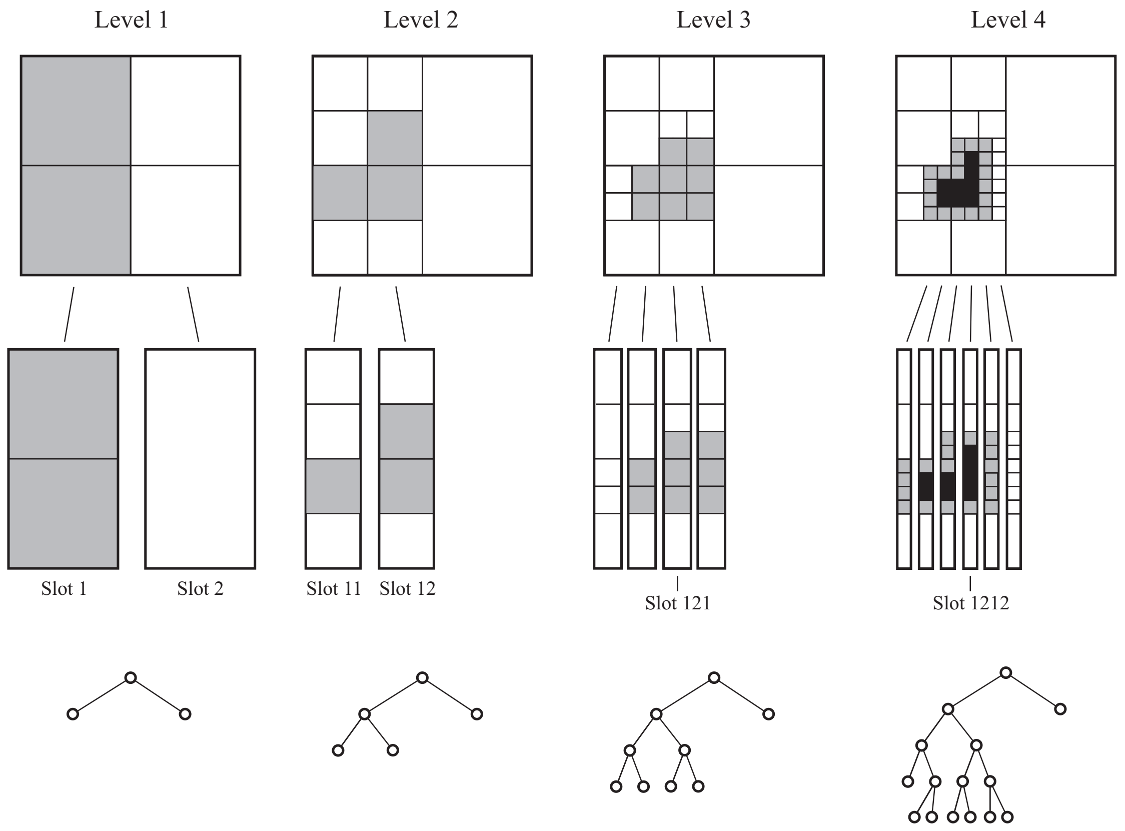

Fig. 14. Generation of a 2D slot-tree up to level 4. A slot will only be refined if it possesses at least one *gray* quadrant. A 2D slot tree can be derived directly from the quadtree presentation of the objects' geometry, a 3D slot tree from an octree representation accordingly.

ing the same subset of space, and performing local tests on these pairs. For the sake of better comprehensibility, but without loss of generality, only the direction *above* is considered here.

4.3.1. *Strict projection-based operators*

The main routine *above_proj_strict()* (Algorithm 1) controls the recursive algorithm. The level of recursion is identical to the slot-tree level that is currently under examination. At the beginning, the level-0 slots of object $A$ and object $B$ are passed to *above_proj_strict()* as parameters. Before jumping to the next level of recursion, all tests for the current hierarchy level are performed (breadth-first traversal). This makes it possible to avoid unnecessary refinement steps, terminate the algorithm at the earliest moment possible and return either *true* or *false*.

The recursion is stopped, if one of the tests for the child pairs returns a negative result (Algorithm 1, lines 3–5), the chosen maximum level of refinement has been reached (line 9) or no pairs for the next recursion level have been created (line 7). The latter occurs if all slot pairs of the current level pass the tests, and thus no refinement is necessary. In this case, *above_proj_strict()* returns *true* (line 15).

The core of the algorithm consists in the slot-wise checking of rules, realized by the subroutine *applyRules_createChildPairs* (Algorithm 1). It is called for each single slot pair. First, general tests based on the slots' colors are performed. The color of a slot is determined by the colors of the octants belonging to it. If at least one of the octants is *gray*, the color of the slot is also *gray*. The same applies, if the slot has both *white* and *black* octants. The slot only obtains the corresponding pure color if there are just white or just black octants, respectively.

The slot color-based rules for *above_proj_strict* are as follows: if the $B$ slot is *black*, the algorithm returns *false* (line 4), because in this case $B$ fills the whole height of the domain, i.e. there is at least one $B$ point that is not above an $A$ point. If slot $B$ is *white*, then the respective slot pair is not relevant for the evaluation of the directional predicate, i.e. no child pairs are created and *true* is returned. If slot $B$ is *gray* and slot $A$ is *white*, then there is at least one point in $B$ that "hangs in the air", i.e. it is not above an $A$ point. Accordingly the algorithm has to return *false* (line 10). If slot $A$ is *gray* and slot $B$ is *black*, then there is no $B$ *point* in this slot that is above all $A$ points. Again, *false* has to be returned (line 13).

Detailed examinations with respect to the position and color of individual cells are only necessary if both slots are *gray*. In this case, the subroutine makes use of the slots' methods *lowestNonWhite()*, *highestNonWhite()*, *highestBlack()* and *lowestBlack()* that return the position of the respective cell as integer value, as well as *hasBlack()* that returns a Boolean value. The implementation of these methods relies on a traversal of the list of cells owned by the slot.

```
boolean above_proj_strict
        (SlotPair[ ] slotpairs, int currentLevel)
 1: for all slotpairs do
 2:   boolean result ← applyRules_createChildPairs( slotpairs[i],
      childPairs )
 3:   if result = false then
 4:     return false
 5:   end if
 6: end for
 7: if number of childPairs > 0 then
 8:   if currentLevel = maxLevel then
 9:     return true
10:   else
11:     currentLevel ← currentLevel + 1
12:     return above_proj_strict(childPairs, currentLevel) // recur-
        sive call
13:   end if
14: else
15:   return true
16: end if
```

**Algorithm 1:** The main routine *above_proj_strict* controls the recursive traversal of the slot-trees by calling itself recursively.

```
boolean applyRules_createChildPairs
        (SlotPair slotPair, SlotPair[ ] childSlotPairs)
 1: slotA ← slotPair.slotA
 2: slotB ← slotPair.slotB
 3: if slotB.color = black then
 4:   return false
 5: else
 6:   if slotB.color = white then
 7:     return true
 8:   else // slotB.color = gray
 9:     if slotA.color = white then
10:       return false
11:     end if
12:     if slotA.color = black then
13:       return false
14:     end if
15:   end if
16: end if
17: if slotB.lowestNonWhite() < slotA.lowestNonWhite() then
18:   return false // Neg1-Rule
19: end if
20: if slotA.hasBlack then
21:   if slotB.highestBlack() ≥ slotA.lowestNonWhite() then
22:     return false // Neg2-Rule
23:   end if
24:   if slotA.highestNonWhite() < slotB.lowestNonWhite() then
25:     return true // Pos-Rule
26:   end if
27: end if
28: if slotB.hasBlack then
29:   if slotB.lowestBlack() ≤ slotA.highestNonWhite() then
30:     return false // Neg3-Rule
31:   end if
32: end if
33: childSlotPairs ← createChildSlotPairs(slotA, slotB)
```

**Algorithm 2:** The sub-routine *applyRules_createChildPairs* performs the slotwise checkings of rules and creates pairs of child slots, if necessary.

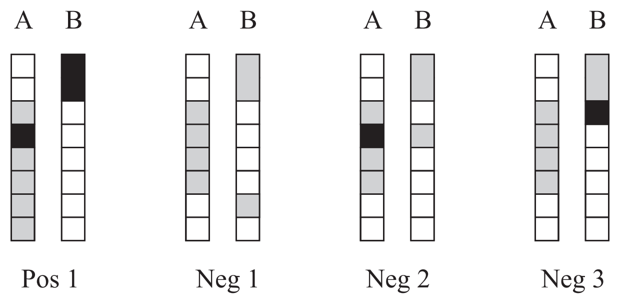

Fig. 15. Examples of constellations where the rules *Pos, Neg1, Neg2* or *Neg3* are applied during the processing of the algorithm *above_proj_strict(A,B)*. The slots shown side-by-side actually occupy the same position in space.

The rules for this exact examination are illustrated in Fig. 15. The first test checks, whether the lowest *non-white* cell in $B$ has a lesser height than the lowest *non-white* cell in $A$. If this is the case, the definition is violated, because $B$ points "hang in the air" (rule *Neg1*, see Fig. 15) and *false* is returned (line 18).

The next two tests are only performed if $B$ has at least one *black* cell. If the highest *black* cell of $A$ is higher than or at the same position than the highest *black* cell of $B$, then *false* has to be returned (line 22), because according to the definition of *above_proj_strict* all *gray* or *black* $B$ cells have to be above the highest *black* $A$ cell (rule *Neg2*). If, however, the highest *non-white* $A$ cell is below the lowest $B$ cell, the definition is entirely fulfilled (rule *Pos1*), i.e. the subroutine returns *true* without creating pairs of child slots (line 25).

The final test is only conducted, if $B$ owns at least one *black* cell. It checks whether the lowest *black* cell in $B$ lies below a *non-white* $A$ cell. If this is the case, then there is at least one point in $B$ that is below a point in $A$, i.e. the definition is violated (rule *Neg3*) and accordingly *false* is returned (line 30).

If none of the tests yields a positive or negative result, there is no definitive statement possible for the current slot pair and a further refinement is required. Accordingly, pairs of child slots are created (line 33) and returned to the main routine *above_proj_strict()*. The creation of pairs of child slots is realized as follows: if both slots are *gray*, i.e. not leaf nodes of the corresponding slot tree, each of the four children of slot $A$ is combined with a child of slot $B$ at the same position, resulting in four pairs of child slots. If one of the slots is either *black* or *white*, i.e. a leaf node without children, it is combined with each child of the other slot, also resulting in four pairs of child slots. Consequently, there may be pairs of slots from different levels.

### 4.3.2. *Relaxed projection-based operators*

The implementation of the relaxed operators differs from that of the strict operators with respect to (1) the rules that are checked during the recursive algorithm and (2) the conditions for stopping the recursion before reaching the maximum refinement level. It follows from the definition of the relaxed operators that, in contrast to the strict opera-

```
boolean above_proj_relaxed
        (SlotPair[] slotpairs, int currentLevel )
 1: for all slotpairs do
 2:   boolean result ← applyRules_createChildPairs( slotpairs[i], ChildPairs )
 3:   if result = true then
 4:     return true
 5:   end if
 6: end for
 7: if number of childPairs > 0 then
 8:   if currentLevel = maxLevel then
 9:     return false
10:   else
11:     currentLevel ← currentLevel + 1
12:     return above_proj_relaxed(childPairs, currentLevel) // recursive call
13:   end if
14: else
15:   return false
16: end if
```

**Algorithm 3:** The main routine *above_proj_relaxed* controls the recursive traversal of the slot-trees by calling itself recursively.

```
boolean applyRules_createChildPairs
        (SlotPair slotPair, SlotPair[] childSlotPairs)
 1: slotA ← slotPair.slotA
 2: slotB ← slotPair.slotB
 3: if slotA.color = white or slotB.color = white then
 4:   return false
 5: else
 6:   if slotA.color = black or slotB.color = black then
 7:     return true
 8:   end if
 9: end if
10: if slotB.hasBlack and (slotB.highestBlack()≥slotA.lowestNonWhite()) then
11:   return true // Pos1-Rule
12: end if
13: if slotA.hasBlack and (slotB.highestNonWhite() ≥ slotA.lowestBlack()) then
14:   return true // Pos2-Rule
15: end if
16: childSlotPairs ← createChildSlotPairs(slotA, slotB)
```

**Algorithm 4:** The sub-routine *applyRules_createChildPairs* performs the slotwise checking of rules and creates pairs of child slots, if necessary.

tors, there is no rule that may force the algorithm to stop because of a negative condition. However, it is possible for the algorithm to stop if only one slot pair fulfills the positive conditions.

This fact is reflected by the main procedure *above_proj_-relaxed()* that controls the recursive breadth-first traversal: as soon as the subroutine *applyRules_createChildPairs()* returns *true* for a slot pair (lines 3–4), *above_proj_relaxed()* stops the recursion and returns *true*. The same behavior occurs whenever no pairs of children have been created on the current refinement level (lines 7, 15). Another difference is that *above_proj_relaxed()* returns *false* if none of the slot pairs has fulfilled the positive conditions on reaching the maximum refinement level (line 9).

As for the strict version of the operators, the rules are tested in *testSlots_createChildPairs* (Fig. 3). Again, simple tests based on the slots' colors are conducted first. If one of the slots is *white*, then this is not a slot pair that has to be subjected to further examination. Accordingly, no pairs of children are created and *false* is returned (line 4).

If none of the slots is *white* and at least one is *black*, then there is at least one point in $B$ that is above a point in $A$ or has the same vertical position. Thus, the formal definition of *above_proj_relaxed* is fulfilled, i.e. no pairs of children are created and *true* is returned (line 7), which results in the abortion of the recursion. If neither the first nor the second rule is fulfilled, then both slots are *gray* and detailed examinations have to be conducted with respect to the color and position of the individual cells.

If slot $B$ owns a *black* cell with the same or a higher position than the highest *non-white* cell of slot $A$, then there is at least one $B$ point above an $A$ point, i.e. the definition is fulfilled and *true* is returned (line 10) without creating pairs of child slots (rule *Pos1*, see Fig. 16).

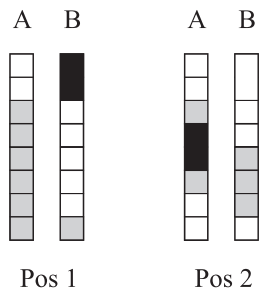

Fig. 16. Examples of constellations where the rule *Pos1* and *Pos2* are applied when processing the algorithm *above_proj_relaxed(A,B)*. The slots shown side-by-side actually occupy the same position in space.

If slot $A$ owns at least one *black* cell and the highest *non-white* cell in $B$ has the same or a higher position than the lowest *black* cell in $A$, then again at least one $B$ point is above an $A$ point. So the definition is fulfilled (rule *Pos2*) and *true* is returned (line 14).

If none of the tests yields a positive or negative result, then either slot $A$ and slot $B$ has exclusively *gray* cells or the *black* cells do not occupy a relevant position. In both cases, a further refinement is necessary, i.e. pairs of child slots are created (line 16) and returned to *above_proj_relaxed()* for the next level of recursion.

### 4.3.3. *Imprecision issues*

If the refinement level is not high enough, the slot-tree based algorithms may produce incorrect results. Due to different interpretations of non-resolved slot pairs when reaching the maximum refinement level, the strict operators may incorrectly return *true* when the definition is actually violated (Fig. 17, left) while, on the other hand, the relaxed operators may return *false* altough the definition is actu-

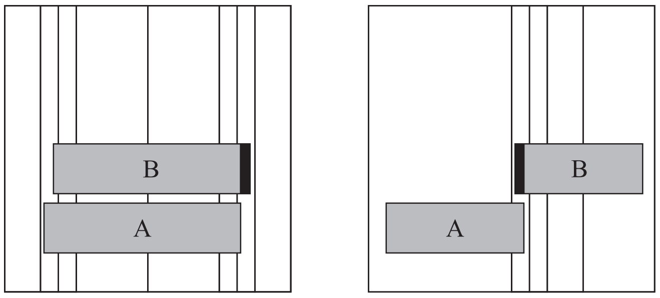

Fig. 17. In the given examples, the critical parts (depicted in black) will not be detected by the slot-based algorithms if the maximum refinement level is not greater than 4. *Left:* The operator *above_proj_strict* will incorrectly return *true*. *Right:* The operator *above_proj_relaxed* will incorrectly return *false*.

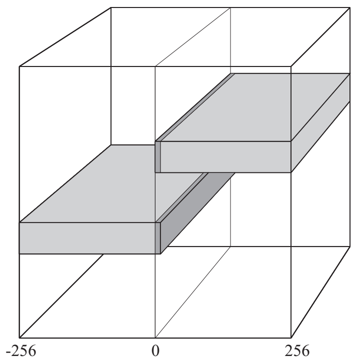

Fig. 18. Geometrical setup 1 for performance measurement of the *strict* operator. The algorithm has to refine an entire edge where the objects overlap vertically.

ally satisfied (Fig. 17, right).

The different treatment of unresolved cases is chosen in this way to reflect the more probable situation: when applying the strict operator, one slot pair that *violates* the definition suffices to stop the algorithm and make the operator return *false*. It can therefore be assumed that the objects in question *fulfill* the definition if the maximum refinement level is reached and no such slot pair has been found. By contrast, when applying the relaxed operator, one slot pair that *fulfills* the definition suffices to stop the algorithm and cause the operator to return *false*. Thus, in this case it is assumed that the objects in question *violate* the definition if the maximum refinement level is reached and no such slot pair has been found.

### 4.3.4. *Performance*

In order to investigate the performance of the developed algorithms, we applied them to extreme geometric constellations that force the algorithms to refine the generated slot-tree up to the maximum level defined by the user. These extreme cases are depicted in Fig. 18 and 19 for the relaxed operator and Fig. 20 for the strict operator.

As can be seen, in the case of the *relaxed* operator, the geometry setup is formed by two plates that slightly overlap when seen from above. Here we have examined two distinct cases; in the first configuration the plates overlap along an entire edge (Fig. 18), in the second one only at one corner

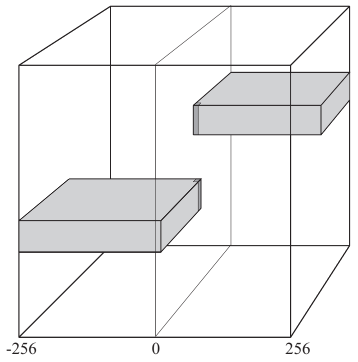

Fig. 19. Geometrical setup 2 for performance measurement of the *strict* projection-based operator. The algorithm only has to refine the corner where the objects overlap vertically.

(Fig. 19). In addition, we varied the thickness of the plates: we ran the tests for thicknesses of 16 and 128 units, respectively.

The results of the performance measurements are depicted in Fig. 22, where the runtime of the algorithm is plotted logarithmically against the maximum refinement level. The diagram clearly shows the exponential behavior of the runtime of the algorithm. Such runtime complexity is doubtlessly undesirable, but it is predetermined by the nature of the refinement approach followed by the slot-tree algorithms.

In the case of the *strict* operator we used two plates that have exactly the same dimensions and lie precisely above one another (Fig. 20). In this case, all four sides of the hexagons have to be refined up to the maximum level. Again, we used two different thicknesses for the plates, 16 and 128 units. The results of these tests are depicted in Fig. 23 and visualized in Fig. 24. As can be seen, we are obliged to observe an exponential behavior of the algorithm in case of the strict operators, too.

It is important to note that the geometric constellations examined here for performance measurements are worst-case scenarios, i.e. constellations that are rarely found in real building models. In most cases, the algorithms will be able to stop at a much lower refinement level, resulting in dramatically reduced processing time for the average object pair.

A fairly realistic scenario is the model of a building's structural framework as presented in Fig. 8, which consists of 216 building components. A specimen query that selects all the building components that are exactly above a ground floor column using the *strict_above_proj* algorithm in conjunction with an AABB-based pre-filter takes no more than 42 ms on average, when refining up to level 7. In this example, the correct set of building components is already identified by the algorithm at refinement level 4.

The presented algorithms are implemented in Java. All performance tests were run using JDK 1.6.0_02 installed on an Intel Pentium 4 3.0 GHz machine, a typical desktop computer in use in many engineering offices today. In order to directly measure the performance of the algorithms presented here, the time for query processing and secondary

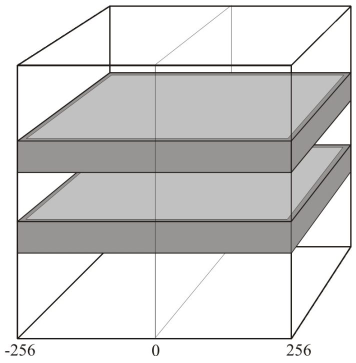

Fig. 20. Geometrical setup for performance measurement of the *relaxed* projection-based operator. The algorithm has to refine all four sides of both hexagons up to the maximum level.

| **max. level** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|
| edge/edge thick | 15 | 60 | 205 | 810 | 3058 | | |
| edge/edge thin | 15 | 41 | 105 | 285 | 1013 | 3341 | |
| corner/corner thin | 5 | 7 | 7 | 7 | 7 | 9 | 11 |
| corner/corner thick | 5 | 7 | 13 | 21 | 41 | 78 | 156 |

Fig. 21. Timings in ms taken for the slot-based implementation of the strict projection-based operators.

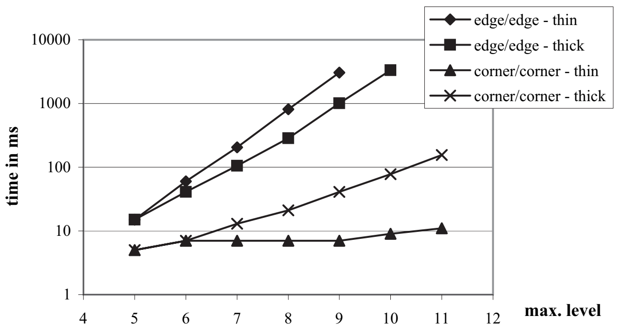

Fig. 22. The timings shown in Fig. **??** plotted logarithmically against the maximum refinement level. The exponential behavior of the runtime is clearly shown.

| **max. level** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|
| thick | 97 | 396 | 1947 | 12302 |
| thin | 97 | 306 | 1078 | 3623 |

Fig. 23. Timings in ms taken for the slot-based implementation of the strict projection-based operators.

storage retrieval has been disregarded.

To summarize, even if the runtime complexity of the algorithms is not optimal for worst-case scenarios, the average runtime of the algorithms – when used for typical building models – is more than acceptable. The sub-optimal runtime behavior has to be assessed in the context that no algorithms for determining precise directional relations between extended 3D objects were available so far. Nevertheless, we intend to pursue our work on improving the runtime performance of the presented algorithms.

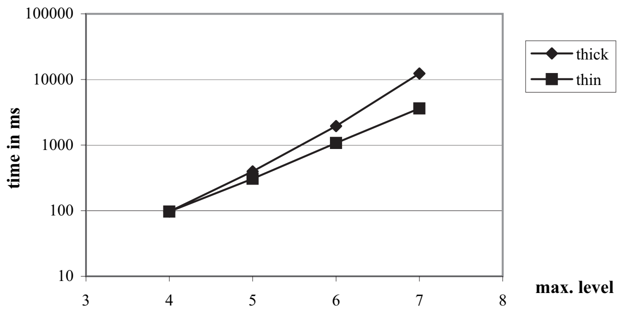

Fig. 24. The timings shown in Fig. 23 plotted logarithmically against the maximum refinement level. The exponential behavior of the runtime is clearly shown.

| **max. level** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|
| time [ms] | 33 | 33 | 35 | 39 | 42 | 90 | 379 | 1261 |

Fig. 25. Timings taken for the evaluation of the strict projection-based operator *strict_above_proj* for the red-marked column of Fig. 8.

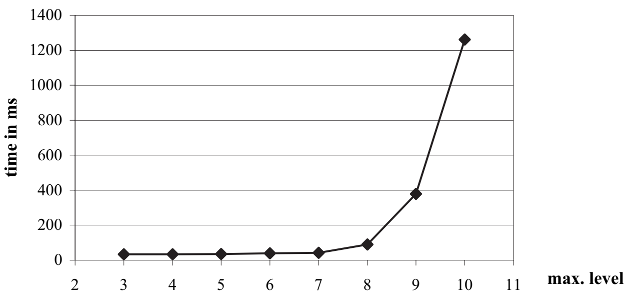

Fig. 26. The timings shown in Fig. 25 plotted linearly against the maximum refinement level. The correct result set is already identified by the algorithm with a maximum refinement level of 4.

## 5. Summary

In this article we have presented in-depth definitions and possible implementations of directional operators in a 3D Spatial Query Language for Building Information Models. By using point-set theory notation, we have formally defined two directional models: the *halfspace-based model* where the directional 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.

Possible implementations of the directional operators have also been discussed. 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 describes in detail a possible implementation by means of so-called *slot-trees*, a new space-partitioning data structure that has been introduced here. The data structure is related to the well-known octree data structure but organizes the cells primarily in directional order. It is created on-the-fly out of the boundary representation of the reference and the target object, respectively. The algorithms traverse the resulting trees in a breadth-first manner, perform local tests based on the color and location of the underlying octants, and either opt for further refinement or for stopping the recursive traversal. With the ongoing recursion, the discrete resolution of the spatial objects employed is successively increased. By choosing the maximum refinement level, the user is able to trade-off between computational effort and the required accuracy.

Finally, the paper has presented various investigations concerning the performance of the presented algorithms. We have studied some worst-case scenarios clearly illustrating the drawback of the refinement approach resulting in exponential runtime behavior. At the same time, we have also been able to show that the algorithms have comparatively short processing times, when applied for real-world scenarios, which makes the entire concept suitable for real engineering problems.

## 6. Future research work

Our current efforts focus on developing an alternative implementation of the projection-based directional operators, which will be based directly on the boundary representation of the reference and target object and will involve traditional geometric approaches, such as ray-triangle intersection tests. We hope to use this approach to overcome the exponential behavior of the slot-tree based algorithm.

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 the 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 [58,59]. 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 [60], and SPARQL, a language for querying RDF ontologies developed in the context of the Semantic Web [61,62].

Finally, we will place emphasis on showing practical applications of the spatial query language by developing usage cases in the context of construction rule checking and partial model creation.

## Acknowledgments

The authors gratefully acknowledge the support for the ongoing project by the German Research Foundation (DFG) under grant Ra 624/17-1. In addition, the authors would like to thank Ms. Stefanie Schraufstetter for her contribution to the implementation of the presented algorithms.

## References

[1] B.-C. Björk, A conceptual model of spaces, space boundaries and enclosing structures, Automation in Construction 1 (3) (1992) 193 – 214.

[2] A. M. Dubois, J. Flynn, M. H. G. Verhoef, G. L. M. Augenbroe, Conceptual modelling approaches in the COMBINE project, in: Proc. of the 1st Europ. Conf. on Product and Process Modeling in the Building Industry, 1995.

[3] F. Tolman, P. Poyet, The ATLAS models, in: Proc. of the 1st Europ. Conf. on Product and Process Modelling in the Building Industry, 1995.

[4] C. Eastman, Building Product Models: Computer Environments Supporting Design and Construction, CRC Press, 1999.

[5] R. Howard, B.-C. Björk, Building information modelling - Experts' views on standardisation and industry deployment, Advanced Engineering Informatics 22 (2) (2008) 271–280.

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

[7] C. M. Eastman, F. Wang, S.-J. You, D. Yang, Deployment of an aec industry sector product model, Computer-Aided Design 37 (12) (2005) 1214–1228.

[8] Y. Adachi, Overview of partial model query language, in: Proc. of the 10th Int. Conf. on Concurrent Engineering, 2003.

[9] N. Yabuki, E. Lebegue, J. Gual, T. Shitani, L. Zhantao, International collaboration for developing the bridge product model IFC-Bridge, in: Proc. of the 11th Int. Conf on Computing in Civil and Building Engineering, 2006.

[10] A. Borrmann, C. van Treeck, E. Rank, Towards a 3D spatial query language for building information models, in: Proc. of the Joint Int. Conf. for Computing and Decision Making in Civil and Building Engineering, 2006.

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

[12] A. Borrmann, S. Schraufstetter, C. van Treeck, E. Rank, An iterative, octree-based algorithm for distance computation between polyhedra with complex surfaces, in: Proc. of the ASCE Int. Workshop of Computing in Civil Engineering, 2007.

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

[14] M. Egenhofer, An extended SQL syntax to treat spatial objects, in: Proc. of the 2nd Int. Seminar on Trends and Concerns of Spatial Sciences, 1987.

[15] B. Ooi, R. Sacks-Davis, K. McDonell, Extending a DBMS for geographic applications, in: Proc. of the IEEE 5th Int. Conf. on Data Engineering, 1989.

[16] K. Ingram, W. Phillips, Geographic information processing using a SQL-based query language, in: Proc. of the 8th Int. Symp. on Computer-Assisted Cartography, 1987.

[17] J. Herring, R. Larsen, J. Shivakumar, Extensions to the SQL language to support spatial analysis in a topological data base, in: Proc. of GIS/LIS '88, 1988.

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

[19] P. Rigaux, M. Scholl, A. Voisard, Spatial Databases with Application to GIS, Morgan Kaufmann, 2002.

[20] S. Shekhar, S. Chawla, Spatial Databases: A Tour, Pearson Eduction, 2003.

[21] OpenGIS Consortium (OGC), OGC Abstract Specification (1999).
URL `http://www.opengis.org/techno/specs.htm`

[22] F. Ozel, Spatial databases and the analysis of dynamic processes in buildings, in: Proc. of the 5th Conf. on Computer Aided Architectural Design Research in Asia, 2000.

[23] G. Gröger, M. Reuter, L. Plümer, Representation of a 3-D city model in spatial object-relational databases, in: Proc. of the 20th ISPRS Congress, 2004.

[24] M. Breunig, T. Bode, A. Cremers, Implementation of elementary geometric database operations for a 3D-GIS, in: Proc. of the 6th Int. Symp. on Spatial Data Handling, 1994.

[25] M. Breunig, A. Cremers, W. Mller, J. Siebeck, New methods for topological clustering and spatial access in object-oriented 3D databases, in: Proc. of the 9th ACM Int. Symp. on Advances in Geographic Information Systems, 2001.

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

[27] N. Paul, P. E. Bradley, Topological houses, in: Proc. of the 16th Int. Conf. of Computer Science and Mathematics in Architecture and Civil Engineering (IKM 2003), 2003.

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

[29] S. Zlatanova, 3D geometries in spatial DBMS, in: Proc. of the Int. Worksh. on 3D Geoinformation 2006, 2006.

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

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

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

[33] A. U. Frank, Qualitative spatial reasoning: Cardinal directions as an example, Int. Journal of Geographic Information Systems 10 (3) (1996) 269–290.

[34] D. Peuquet, C.-X. Zhan, An algorithm to determine the directional relationship between arbitrarily-shaped polygons in the plane, Pattern Recognition 20 (1) (1987) 65–74.

[35] J. Hong, Qualitative distance and direction reasoning in geographic space, Ph.D. thesis, Department of Surveying Engineering, University of Maine, Orono, ME, USA (1994).

[36] A. Abdelmoty, Modelling and reasoning in spatial databases: A deductive object-oriented approach, Ph.D. thesis, Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh, Scotland (1995).

[37] S. Shekhar, X. Liu, S. Chawla, An object model of direction and its implications, GeoInformatica 3 (4) (1999) 357–379.

[38] K. Zimmermann, C. Freksa, Qualitative spatial reasoning using orientation, distance, and path knowledge, Applied Intelligence 6 (1) (1996) 49–58.

[39] J. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843.

[40] H. Guesgen, Spatial reasoning based on allen's temporal logic, Tech. rep., Int. Computer Science Institue, Berkley, CA, USA (1989).

[41] D. Papadias, T. Sellis, Y. Theodoridis, M. Egenhofer, Topological relations in the world of minimum bounding rectangles: A study with r-trees, in: Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, 1995.

[42] J. Sharma, Integrated spatial reasoning in geographic information systems: Combining topology and direction, Ph.D. thesis, Department of Spatial Information science and Engineering, University of Maine, Orono, ME, USA (1996).

[43] R. Goyal, Similarity assessment for cardinal directions between extended spatial objects, Ph.D. thesis, University of Maine (2000).

[44] H. T. Bruns, M. Egenhofer, Similarity of spatial scenes, in: 7th Symp. on Spatial Data Handling, 1996.

[45] P. W. Huang, Y. R. Jean, Using 2D $C^{+}$-string as spatial knowledge representation for image database systems, Pattern Recognition 30 (10) (1994) 1249–1257.

[46] K.-P. Gapp, Basic meanings of spatial relations: computation and evaluation in 3D space, in: Proc. of the 12th Nat. Conf. on Artificial intelligence (AAAI), 1994, pp. 1393–1398.

[47] G. Retz-Schmidt, Various views on spatial prepositions, AI Magazine 9 (2) (1988) 95–105.

[48] T. Fuhr, G. Socher, C. Scheering, G. Sagerer, A three-dimensional spatial model for the interpretation of image data, in: P. Olivier, K. Gapp (Eds.), Representation and Processing of Spatial Expressions, Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1998, pp. 103–118.

[49] International Organization for Standardization, ANSI/ISO/IEC 9075-1:99. ISO International Standard: Database Language SQL. (1999).

[50] J. Melton, Advanced SQL:1999. Understanding Object-Relational and Other Advanced Features., Morgan Kaufmann, San Francisco, USA, 2003.

[51] C. Türker, SQL:1999 & SQL2000. Objekt-relationales SQL, SQLJ & SQL/XML, dpunkt Verlag, 2003.

[52] C. Türker, G. Saake, Objektrelationale Datenbanken, dpunkt Verlag, 2006.

[53] G. Hunter, Efficient computation and data structures for graphics, Phd thesis, Princeton University (1978).

[54] D. Meagher, Geometric modeling using octree encoding, IEEE Computer Graphics and Image Processing 19 (2) (1982) 129–147.

[55] H. Samet, Data structures for quadtree approximation and compression, Communications of the ACM 28 (9) (1985) 973–993.

[56] R.-P. Mundani, H.-J. Bungartz, E. Rank, R. Romberg, A. Niggl, Efficient algorithms for octree-based geometric modelling, in: Proc. of the 9th Int. Conf. on Civil and Structural Engineering Computing, 2003.

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

[58] C. van Treeck, Gebäudemodell-basierte Simulation von Raumluftströmungen, Ph.D. thesis, Technische Universität München (2004).

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

[60] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Simeon, et al., XQuery 1.0: An XML Query Language. W3C Recommendation (2007).
URL `http://www.w3.org/TR/xquery/`

[61] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF. W3C Candidate Recommendation (2007).
URL `http://www.w3.org/TR/rdf-sparql-query/`

[62] J. Beetz, B. de Vries, J. van Leeuwen, RDF-based distributed functional part specifications for the facilitation of service-based architectures, in: Proc. of the 24th CIB-W78 Conf. on Information Technology in Construction, 2007.