# FROM GIS TO BIM AND BACK AGAIN – A SPATIAL QUERY LANGUAGE FOR 3D BUILDING MODELS AND 3D CITY MODELS

A. Borrmann

Computation in Engineering, Technische Universität München, 80290 München, Germany
andre.borrmann@tum.de

**KEY WORDS:** 3D Spatial Query Language, Building Information Model, 3D City Model, Topology, Direction, Octree

**ABSTRACT:**

The article presents the development of Spatial Query Language for 3D building and 3D city models. Inspired by the achievements of the GIS community in developing spatial query functionality for 2D space, the author adopted these concepts and applied them on geometric objects in 3D space. The developed query language provides metric (*closerThan, fartherThan, etc.),* directional (*above, below, northOf, etc.*) and topological operators (*touch, within, contain, etc.*) for use in SQL statements. The operators have been implemented by algorithms which are based on the hierarchical space-partitioning data structure *octree*. The octree allows for the application of recursive algorithms that successively increase the discrete resolution of the spatial objects employed and thereby enables the user to trade off between computational effort and the required accuracy. Additionally, a fuzzy handling of spatial relationships becomes possible. The article describes the available spatial operators and the algorithms developed to implement them.

## 1. INTRODUCTION

In the GIS community, significant scientific research has been invested in developing spatial query functionality for 2D geographic models. This process started in the late 1980's and has resulted in the fact that today 2D spatial query functionality is available in all major software products for the GIS market and standardized by the OpenGIS Consoritum (OGC).

At the same time, the construction informatics community has developed the paradigm of using a semantically rich, object oriented building model as basis of the entire planning process (Eastman 1999, Eastman et al. 2008). This so-called building information model (BIM) does not only allow to create a 3D representation for visualisation purposes, but also acts as perfect basis for simulations and computations required throughout the planning process, including energy, structural and lightning analysis as well as evacuation simulations, for example.

Most of these downstream applications require only a subset of the large data set stored in the BIM. To specify the required subset a declarative query language is usually employed. However, the query languages available today for building information models only allow the application of alphanumeric comparisons on individual attributes of the object-oriented model. Qualitative spatial relationships between building components cannot be used as selection criteria. For this reason the author applied concepts and technologies from the GIS domain on the BIM domain, thus creating a spatial query language for building information models.

This language allows to formulate queries such as

- Get all walls within the first storey.
- Does room 107 contain any heating equipment?
- Get all fire extinguishers within the distance of 40m from a certain door.
- Which columns touch Slab No. 13?
- Are there any gas lines below the footing?

Although originally intended for application in the BIM context, the developed query language can also easily be applied on 3D city models. This closes the circle and takes the technology back to its origins.

After discussing related work in Section 2, the paper gives an overview on the formal definitions of the different spatial types and operators in Section 3. In Section 4, an implementation approach based on the hierarchical space-partitioning data structure octree is introduced. Section 5 describes how the spatial types and operators can be embedded in relational or object-relational SQL. Section 6 discusses possible applications of the query language in the context of 3D building models and 3D city models.

## 2. RELATED WORK

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

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

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

In (Zlatanova et al. 2004; Zlatanova 2006; Coors 2003; Arens et al. 2005) concepts and data structures for storing 3D city models in spatial databases are presented and the suitability of different geometry models for querying topological relationships is discussed. In general, GIS research follows the approach of choosing geometry data structures that implicitly contain topological relationships. Accordingly many of the proposed data structures rely on a simplicial decomposition of the space (Egenhofer et al. 1989; Egenhofer and Herring 1992; Shi et al. 2003). Since building information models are in most case purely geometric representations, we do not assume any pre-defined topological structure in our research.

## 3. SPATIAL TYPES AND OPERATORS

The developed 3D spatial query language relies on a formally defined spatial algebra. 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, such as load points, power lines, plates, slabs etc. All types of spatial objects are subsumed by the super-type *SpatialObject*. The formal definitions of the spatial types provided in (Borrmann 2006) are based on the mathematical frameworks of point set theory and point set topology. The latter serves to unambiguously define the *interior*, the *boundary* and the *exterior* of each of the spatial types, which is required for formally specifying topological relations between spatial objects.

The spatial operators available for the spatial types are the most important part of the algebra. They consist of

- metric,
- directional, and
- topological

operators.

By means of metric operators such as *distance*, *fartherThan* and *closerThan*, distances between spatial objects can be employed as selection criterion in spatial queries (Borrmann et al. 2009). Topological operators such as *touch*, *overlap*, *within, disjoint* can be used to query the relative position of two spatial objects (Borrmann and Rank 2009b). For formally defining the semantics of these operators, Egenhofer's 9-Intersection Model (Egenhofer and Franzosa 1991) has been extended to the 3D space (Figure 1).

Directional operators such as *above, below, eastOf, westOf, northOf* and *southOf* can be used to employ directional relations as selection criterion in a spatial query. Unfortunately, the models for formally defining directional relationships available from literature are applicable only for point-point relations (Frank 1996) or use the bounding box of the reference object as approximation (Goyal and Egenhofer 1997). Approximating extended 3D objects by centroids or bounding boxes may cause results that do not comply with the intuitive expectations of the users. For this reason and 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* (Borrmann and Rank 2009a). Both models differentiate between two "flavours" 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.

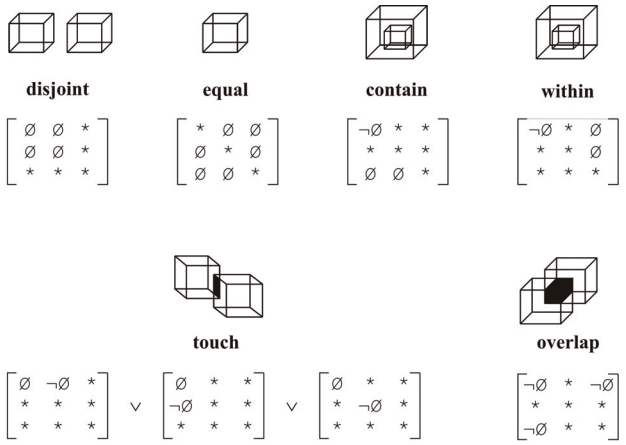


Figure 1: For formally defining the semantics of the topological operators, the 9-Intersection Model has been employed. The resulting matrix captures the intersections between the interior, exterior and the boundary of the involved spatial objects. A star is used at those places where no specification is necessary.

In the projection-based model, the reference object is extruded along the coordinate axis corresponding to the directional operator (Figure 2). The applied operator returns *true* if there is an intersection between the extrusion body and the target object. For the relaxed version it is sufficient if parts of the target object intersect the extrusion body, for the strict version the target object has to be completely within the extrusion body.

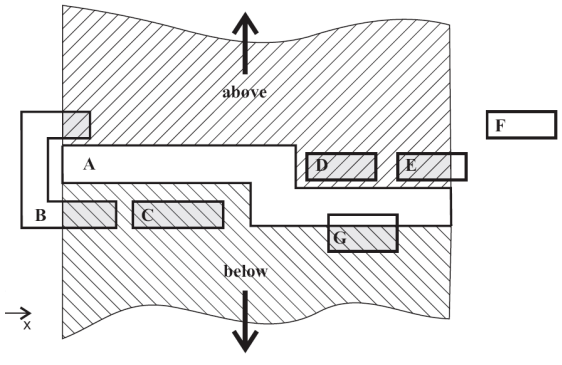


Figure 2: The projection-based directional model is based on the extrusion of the reference object *A* along the respective coordinate axis. In the strict version, the operator above return true only for target object *D*, in the relaxed version also for the objects *B, E*, and *G*.

The second model is based on halfspaces, which are formed by the reference object's bounding box (Figure 3). Each of these halfspaces creates a directional space partition. The directional

operator returns true if the target object is contained within the respective space partition. Again, we distinguish a strict and a relaxed version with the same semantics as for the projection-based model.

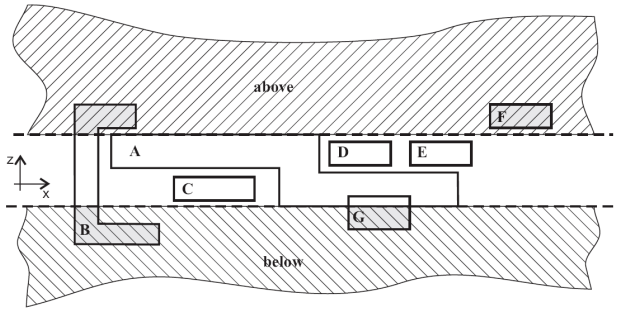


Figure 3: In the model the directional space partitions are created by halfspaces, which are formed by the reference object's bounding box. The strict version of the operator above returns true only for the target objects F, the relaxed version also for target object B.

## 4. IMPLEMENTATION OF SPATIAL OPERATORS

For implementing the spatial operators provided by the query language, algorithms have been developed which are based on an octree representation of the operands. These algorithms are discussed in detail in (Borrmann et al. 2009), (Borrmann and Rank 2009), and (Borrmann and Rank 2009a). Here, only an overview will be given.

### 4.1 Octree and Slot-tree

The octree is a space-dividing, hierarchical tree data structure for the discretized representation of 3D volumetric geometry (Hunter 1978; Jackins and Tanimoto 1980; Meagher 1982). 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 (Figure 4). Whereas black and white octants are branch nodes, and accordingly have no children, gray octants are interior nodes that always have eight children. The union of all child cells is equal to the volume of 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.

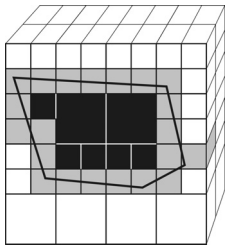

Figure 4. Cross-section through an octree. White cells represent the exterior, black cells the interior and gray cells the boundary of the discretized object. Whereas black and white cells are branch nodes, gray cells always have eight children.

In the implementation concept followed here, 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. those which are coloured *gray*.

For our implementation we use the creation method developed by Mundani (Mundani et al. 2003) that is based on the halfspaces formed by the object's bounding faces. In Mundani's approach, the colour classification is based on a simple evaluation of the plane equation of each halfspace for the respective octant and a subsequent combination using Boolean expressions. Accordingly, the algorithm automatically marks inner cells as *black* without the need to perform a computationally expensive filling algorithm. As described in the next sections, the existence of *black* cells is an important prerequisite for the applicability of numerous rules in the algorithms implementing topological and directional relationships.

To cover dimensionally reduced entities with our algorithms as well, we had to introduce the fourth colour *black/white*. Black/white cells represent space regions where the exterior and the interior of the described object exist, but not its boundary (Figure 5).

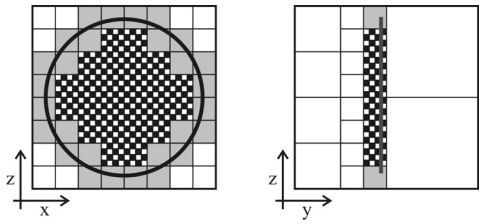


Figure 5: Dimensionally reduced objects like the disc shown here are discretized using the fourth colour *black/white* that represents cells which contain interior and exterior points, but no boundary points.

**Slot-tree.** The algorithms implementing the projection-based directional operators do not use the octree itself, but a newly developed data structure derived from it. This data structure, called a *slot-tree*, organizes the cells of an octree (the octants) with respect to their position orthogonal to the coordinate axis under consideration.

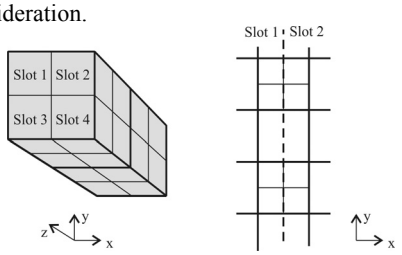


Figure 6: Slots in 3- and 2-dimensional space, respectively. A slot in *z*-direction contains all the cells that lie above one another.

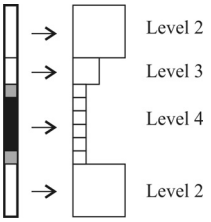


Figure 7: A slot in 2D that owns cells from different levels of the underlying quadtree (Slot 1212 in Figure 6)

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. 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 (Figure 6). 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 (Figure 7). 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 directional predicate under examination is fulfilled, or not.

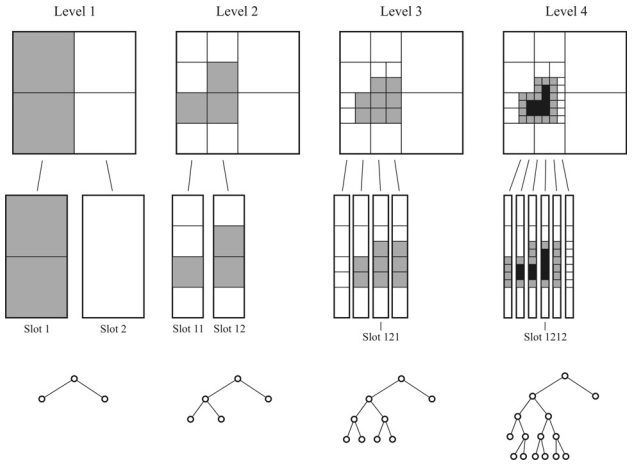


Figure 8: Generation of a 2D slot-tree up to level 4. A slot will only be refined if it possesses at least one *gray* quadrant. A 2D slot tree can be derived directly from the quadtree presentation of the geometry of the objects, a 3D slot tree from an octree representation, respectively.

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 Figure 8. Traversing the octree from the top downwards 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.
In the presented implementation approach, the octree / slot-tree generation is not performed in advance but is coupled with the recursive algorithm presented in the next sections. Thus the octree / slot-tree is built up one level at a time and only at those places that are relevant for verifying or disproving the predicate under examination. This significantly speeds up the query processing.

### 4.2 General Principle

All octree-based algorithms work according the same general principle. As mentioned above, the operands of the spatial operator being processed are encoded in separate octrees /slot-trees. In a first step, the root octants of both octrees are passed as input to the algorithm.
The algorithm consists in a simultaneous breadth-first traversal of both octrees. During the traversal it creates pairs of octants with one member from each octree. In the case of the algorithm implementing topological operators, both octants cover the same partition of the 3D space, whereas in the case of the metric operators the octant pair is among the candidates for the closest proximity.
The algorithm then applies certain operator-specific rules to the pairs of octants. Depending on the result of this test, the algorithm can either stop the recursion and return *true* or *false*, or it has to continue the recursive traversal by creating pairs of child cells, calling itself recursively and thus entering the next level. The user defines a maximum recursion level – if it is reached, the algorithm returns *true*, *false*, or a number representing the knowledge it has gained so far through the breadth-first traversal.
Though the algorithm implementing the directional operators works on slot-trees instead of octrees, it follows the same general principle. Here, the algorithm performs a breadth first-traversal of the slot-trees. During this traversal, pairs of slots are also created that represent the same partition of the 3D space, and rules are subsequently applied to these slot-pairs.
For a detailed description of the algorithms implementing the metric operators the reader is referred to (Borrmann et al. 2009). The algorithms implementing the topological and directional operators are explained in detail in the next subsections.

### 4.3 Implementation of topological operators

For implementing the topological operators, pairs of octants are created on each recursion level with one octant originating from object *A* and one octant from object *B*, both representing the same sector of the 3D space.
Each octant pair provides a colour combination to which specific rules can be applied. These rules may lead to filling a 9-IM *working matrix* that is maintained by the algorithm to keep track of the knowledge gained about the topological constellation. There are 12 positive and 9 negative rules altogether. A positive rule (Figure 9) can be applied when a certain colour combination occurs, and a negative rule (Figure 10) if certain colour combinations do not occur over an entire level. Positive rules lead to empty set entries in the matrix, negative rules to non-empty set entries.

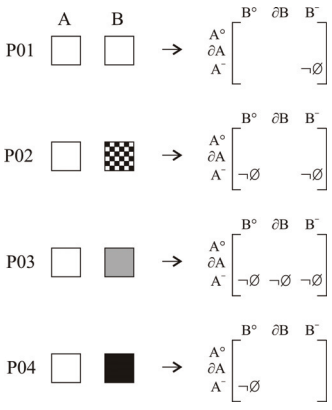


Figure 9: Examples for Positive Rules. If the colour combination on the left-hand side is detected, the 9IM-Matrix can be filled according to the right-hand side.

The rules are derived from the semantics of the colours. A *white* octant, for example, is part of the *exterior* of an operand, and a *black* octant is part of its *interior*. If a *white* octant of the first operand occurs at the same place as a *black* octant of the second operand, it follows that the intersection between the exterior and the interior of the operands is non-empty.

The 9-IM working matrix is successively filled by applying these rules to all octant pairs. When processing the operator *whichTopoPredicate* the working matrix is compared with all predicate matrices of the formal definitions (Section 3). If the working matrix complies fully with one of them, the recursion is aborted and the algorithm returns the respective predicate. If there is any contradiction between the filled matrix and the matrix of a predicate, the respective predicate is precluded. If no unequivocal decision is possible for any of the predicates, a further refinement is necessary, i.e. octant pairs of the next level are created.

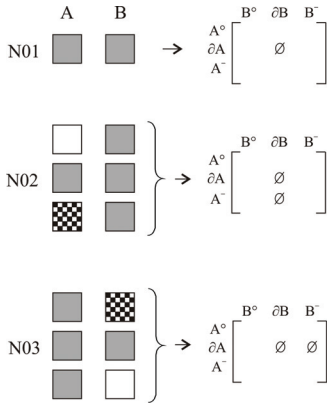


Figure 10: Examples for Negative Rules. If the colour combinations of the left-hand side do not occur across the entire domain, the 9-IM matrix can be filled according to the right hand side.

In the case of the predicate operators the 9-IM working matrix is checked against the corresponding predicate matrix only. If there is a contradiction the algorithm returns *false*, if it completely complies, it returns *true*.

If, after execution of all applicable rules, the current occupancy of the working matrix does not allow for validation or disproval of the/any predicate and the maximum refinement level is not reached, child pairs are created and the algorithm calls itself recursively.

If the algorithm reaches the maximum refinement level and, in the case of *whichTopoPredicate*, none of the predicates is proved or, in the case of a predicate operator, the predicate under examination is neither proved nor disproved, a so-called predicate hierarchy is applied, which again ensures that the most probable situation is detected. This is discussed in the next subsection.

### 4.4 Implementation of directional operators

The halfspace-based directional operators can be implemented by examining the bounding boxes of both the reference and the target object. The algorithms are not explained in detail here, instead the reader is referred to (Borrmann & Rank, 2009).

The core of the algorithm implementing the projection-based directional operators consists of the slot-wise application of rules that are based on the colours of the slots and the octants they contain. First, general tests based on the slots' colours are performed. The colour of a slot is determined by the colours of the octants belonging to it. If at least one of the octants is *gray*, the colour 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 colour if there are just *white* or just *black* octants, respectively.

The occurrence of certain slot colour combinations can lead to a direct validation or disproval of the predicate under examination. In this case, the recursion can be immediately aborted and the algorithm directly returns *true* or *false*.

For example, if *above_proj_strict(A,B)* is evaluated and a black *B* slot occurs, the algorithm returns *false*, because in this case *B* fills the whole height of the domain, and there is accordingly at least one *B* point that is not above an *A* point.

Detailed examinations of the position and the colour of individual cells are only necessary if both slots are *gray*. In this case, the subroutine makes use of the auxiliary functions *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 belonging to the slot concerned.

The rules for this exact examination depend on the direction and the version (strict/relaxed) of the operator that is being processed. They are not explained in detail here, but are shown in Figure 11 for the *strict* version of *above_proj* and in Figure 12 for the relaxed version of *above_proj*.

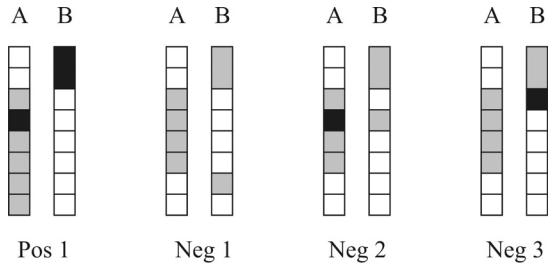


Figure 11: Examples of constellations where the rules Pos, Neg1, Neg2 or Neg3 are applied during the processing of the algorithm *above_proj_strict(A,B)*. The slots shown side-by-side actually occupy the same position in space.

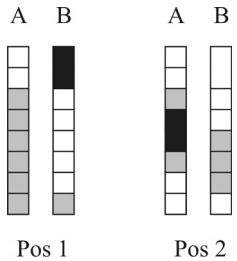


Figure 12: Examples of constellations where the rule Pos1 and Pos2 are applied when processing the algorithm *above_proj_relaxed(A,B)*. The slots shown side-by-side actually occupy the same position in space.

If none of the tests yields a positive or a negative result, no definitive statement can be made with regard to the current slot pair and a further refinement is required. Accordingly, pairs of child slots are created.

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.

The algorithm calls itself recursively until the maximum refinement level is reached. If a decision is still not possible, rules are applied that take the most probable situation into account.

### 4.5 Fuzziness

The octree geometry representation shows a crucial peculiarity for the implementation of spatial operators: The boundary of an object encoded by an octree is not represented sharply, i.e. not as a set of points for each of which a neighbourhood exists that contains both interior and exterior points, but instead in the form of (grey) octants which define a boundary *layer*. The thickness of the layer shrinks with an increasing maximum refinement level (MRL): However, for finite values of the MRL it remains a layer.
This induces a certain fuzziness for all spatial operators. On the one hand such fuzziness results in inaccurate results if the MRL is not chosen high enough. On the other hand it enables the spatial operators to react more "mildly", thus corresponding better to the way human handle qualitative spatial relationships.
A typical example is the relationship *touch*. Even if two building elements "slightly" overlap, in certain application scenarios the user, or some analysis program, might want to treat them as being in touch. The same applies if there is a slight gap between the elements. In the following paragraphs, the impreciseness involved in the octree approach is discussed for the directional and topological operators in more detail.

**Topological Operators.** If, in the case of a predicate operator, the predicate under examination is neither proved nor disproved when reaching the MRL or, in the case of the *whichTopo-Predicate* operator, none of the predicates is fully proved, the predicate hierarchy shown in Figure 13 is applied, i.e. the algorithm returns the highest non-disproved predicate of the hierarchy. The order of the hierarchy is chosen in such a way that, if the actual topological constellation complies with predicate *a*, all predicates above predicate *a* are disproved during successive refinement. On the other hand, the predicates below *a* are not necessarily disproved. In the sense of a "positivistic" approach it is assumed that the highest non-disproved predicate has been proven.

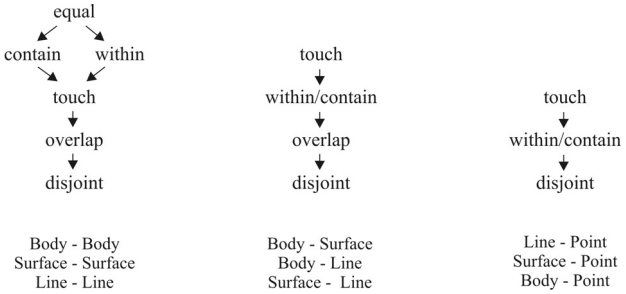


Figure 13: The hierarchy of the topological predicates for different type combinations. The algorithm returns the highest non-disproved predicate. The order of the hierarchy results from the observation that all predicates above a certain predicate *x* are disproved during the ongoing refinement if the actual topological constellation complies with predicate *x*. This hierarchy permits a fuzzy handling of topological relationships.

If both operands have the same dimensionality, *contain* and *within* are equivalent, i.e. for the validation of a "lower" predicate, both *contain* and *within* must be disproved. The equivalence of the predicates results from the fact that when disproving *equal*, either *contain* or *within* is disproved at the same time.
Applying the predicate hierarchy may result in the detection of an incorrect topological predicate if the MRL is too low. However, the hierarchy is chosen in such a way that these errors/misjudgements are acceptable, since they comply with the intuitive human understanding of qualitative spatial relationships.
Using the "positivistic" approach, the requirements of *logical consistency*, *mutual exclusiveness* and *complete coverage* are met by the system of topological operators, since in any case precisely one topological predicate is detected for any topological constellation no matter if the user applies the predicate operators or the *whichTopoPredicate* operator.

**Directional Operators.** The interpretation of non-resolved slot pairs on the final level depends on whether the strict or the relaxed version of the directional predicates is being processed.
The different treatment of unresolved cases is chosen in such a way that it reflects 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 *fulfil* the definition if the MRL is reached and no such slot pair has been found. By contrast, when applying the relaxed operator, one slot pair that *fulfils* 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 MRL is reached and no such slot pair has been found.
According to this interpretation, the strict operators may incorrectly return *true* when the definition is actually violated (Figure 14, left) while, on the other hand, the relaxed operators may return *false* although the definition is actually satisfied (Figure 14, right).

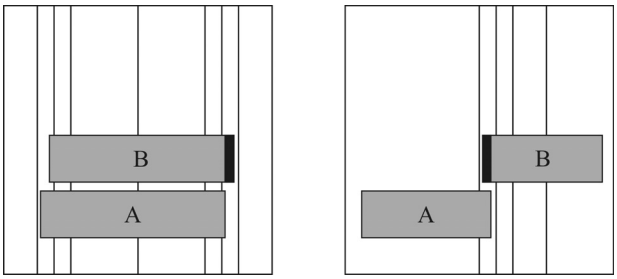


Figure 14: In the given examples, the critical parts (depicted in black) will not be detected by the slot-based algorithms if the maximum refinement level is not greater than 4. Left: The operator above_proj_strict will incorrectly return true. Right: The operator above_proj_relaxed will incorrectly return false.

## 5. EMBEDDING SPATIAL OPERATORS IN A QUERY LANGUAGE

We have based the spatial query support on SQL, since it is one of the most widespread and powerful declarative query languages. Many SQL dialects allow for an extension of the available operators by means of user-defined functions, which may subsequently be used within the WHERE part of an SQL statement. For embedding the spatial operators we experimented with both versions of the SQL standard, the purely relational version SQL-92 and the object-relational version SQL:1999.

**SQL:1999.** SQL:1999 provides the user the possibility to define abstract data types (ADTs), thus extending the database type system in an object-oriented way. These ADTs may not only

possess attributes and references to other ADTS but also member functions (methods) that define the behaviour of the corresponding object instances. Accordingly, the spatial data types defined in Section 3 can be defined as ADTs providing the spatial operators as member functions.
By realizing this, spatial query functionality can be made available to end-users and third-party programmers in an easily manageable manner. A specimen query that retrieves all columns that touch the slab whose ID is *Oid23089* then reads:

```
SELECT   *
FROM     IFCColumn col, IFCSlab slab3
WHERE    col.shape().touch(slab3.shape())
         AND slab3.id = 'Oid23089'
```

For a prototype implementation the authors used the commercially available ORDBMS Oracle 10g. For more detailed information on the integration of spatial operators in SQL using object-relational techniques, the reader is referred to (Borrmann and Rank 2009; Borrmann et al. 2009, Borrmann and Rank 2010).
The most important advantage of using an object-relational approach is the strong type safety provided by the declaration of user-defined types. The declaration of the *touch* member function, for example, forces the passed parameter to be of type *SpatialObject* or one of its sub-types. Thus, type errors may already be detected by the query engine during the interpretation of the SQL statement and more specific error reports can be generated.
**SQL-92.** As for the desired purpose of a declarative spatial query language for BIMs, traditional database functionalities such as concurrency control, rights management and persistency are not of primary interest, the utilization of an in-memory database (IMDB) seems to be most appropriate. These systems, which are normally completely embedded in the final application, usually provide SQL query and data manipulation functionality while avoiding the high overhead of hard-disk access. Unfortunately, there are no in-memory databases available today that provide the full range of the SQL:1999 standard, especially with respect to the possibility of defining ADT's.
We therefore decided in a second approach to base the spatial query functionality on purely relational databases. Here, a semantically weaker way of defining the spatial operators has to be chosen. All spatial operators are defined as global functions whose parameters are *strings* representing the operand's IDs. The specimen query then reads:

```
SELECT col.id
FROM IFCColumn col, IFCSlab slab3
WHERE touch(col.id, slab3.id)
      AND slab3.id = 'Oid23089'
```

## 6. SOFTWARE PROTOTYPE: APPLICATION ON 3D BUILDING MODELS AND 3D CITY MODELS

To prove the feasibility of the developed concepts a software prototype that offers spatial query functionality for 3D building models (Figure 15) and 3D city models (Figure 16) has been implemented. Since it is capable to process VRML models, it is basically able to read-in any 3D model including building models provide in the IFC file format or city models provided as CityGML file. In both cases, a transformation into VRML is possible using standard software tools.
Possible applications of the spatial query functionality for 3D city models include planning processes where existing abvove- and below-ground infrastructure has to be taken into account.

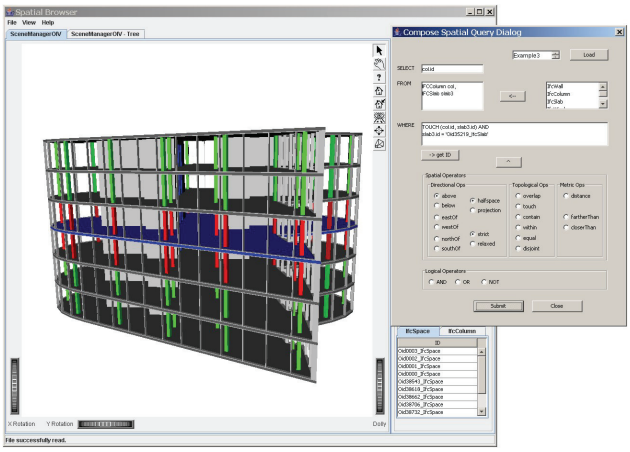

Figure 15: Screenshot of the prototype software applied on a 3D building model. It shows the dialog for composing spatial SQL queries and the 3D viewer highlighting the result set. Here, all columns touching the blue-coloured slab have been found.

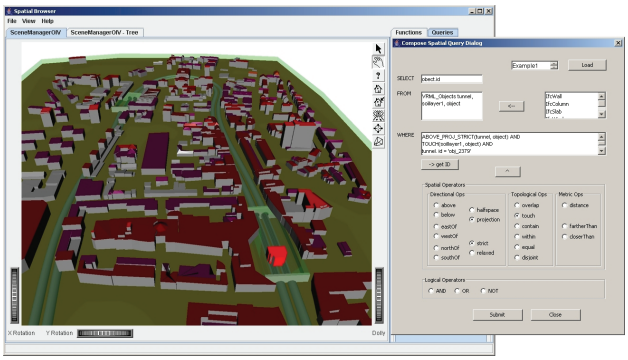

Figure 16: Screenshot of the prototype software applied on a 3D city model. Here the spatial query functionality was used to find all buildings which are located directly above a planned subway tunnel.

## 7. CONCLUSION

The presented spatial query language allows for spatial analysis and partitioning of 3D building and 3D city models. The language provides metric, directional and topological operators which can be used as selection criteria in spatial queries. The operators are processed using octree-based algorithms, which successively increase the discrete resolution of the spatial objects employed and thereby enable the user to trade off between computational effort and the required accuracy. Additionally, a fuzzy handling of spatial relationships becomes possible which complies well with the human recognition of qualitative spatial relationships.
The language has been implemented on top of object-relational SQL:1999 and of purely relational SQL-92. The first option allows for an extension of the type system according to the object-oriented paradigm, thus providing extensive type safety. However, for the second option there are more database systems available, including in-memory databases which perfectly meet the requirements of spatial analysis of 3D building models.
Our future work will concentrate on the development of alternative approaches for implementing the spatial operators. First attempts using algorithms that directly work on the

boundary representation of the operands have shown promising results.

## REFERENCES


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

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

Breunig, M., Cremers, A., Müller, W., Siebeck, J., 2001. 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*.

Borrmann, A., 2006. Extended formal specifications of 3D spatial data types. Technical Report. Technische Universität München, Germany.

Borrmann, A., 2007. Computerunterstützung verteilt-kooperativer Bauplanung durch Integration interaktiver Simulationen und räumlicher Datenbanken. Doctoral dissertation, Lehrstuhl für Bauinformatik, Technische Universität München, Germany.

Borrmann, A., Rank, E., 2009a. Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics,* 23(1), pp. 32–44.

Borrmann, A., Rank, E., 2009b. Topological analysis of 3D building models using a spatial query language. *Advanced Engineering Informatics,* 23(4), pp. 370–385.

Borrmann, A. Rank, E., 2010. Query Support for BIMs Using Semantic and Spatial Conditions. In: Underwood. J. and Isikdag. U. (Eds): Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies, IGI Global

Borrmann, A., Schraufstetter, S., Rank, E., 2009. Implementing metric operators of a spatial query language for 3D building models: Octree and B-Rep approaches. Journal of Computing in Civil Engineering, 23(1), 34–46.

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

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

Eastman, C., Teicholz, P., Sacks, R., Liston, K., 2008. BIM Handbook: A guide to building information modeling for owners, managers, designers, engineers, and contractors. John Wiley & Sons

Egenhofer, M., Frank, A., Jackson, J. P., 1989. A topological data model for spatial databases. In: Proc. of the 1st Int. Symp. on the Design and Implementation of Large Spatial Databases.

Egenhofer, M., Herring, J., 1990. A mathematical framework for the definition of topological relationships. In Proc. of the 4th Int. Symp. on Spatial Data Handling.

Egenhofer, M., Franzosa R., 1991. Point-Set Topological Spatial Relations. *Int. Journal of Geographical Information Systems*, 5(2), pp. 161–174.

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

Goyal, R.K., Egenhofer, M.J., 1997. The direction-relation matrix: A representation of direction relations for extended spatial objects. UCGIS Annual Assembly and Summer Retreat, Bar Harbor, ME

Gröger, G., Reuter, M., Plümer, L., 2004. Representation of a 3-D city model in spatial object-relational databases. In Proc. of the 20th ISPRS congress.

Hunter, G., 1978. Efficient computation and data structures for graphics. Doctoral dissertation, Princeton University.

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

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

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

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

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

Shi, W., Yang, B., Li, Q., 2003. An object-oriented data model for complex objects in three-dimensional geographical information systems. Int. *J. of Geographical Information Science*, 17(5), 411–430.

Zlatanova, S. (2006). 3D geometries in spatial DBMS. In: *Proc. of the Int. Workshop on 3D Geoinformation 2006*.

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