Topological analysis of 3D Building Models using a Spatial Query Language

André Borrmann [*], Ernst Rank

Computation in 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 topological operators, i.e. operators that reflect the topological relationships between 3D spatial objects. The paper presents definitions of the semantics of the topological operators *within*, *contain*, *touch*, *overlap*, *disjoint* and *equal* in 3D space by using the 9-intersection model. It further describes a possible implementation of the topological operators by means of an octree-based algorithm. The recursive algorithm presented in this article relies on a breadth-first traversal of the operands' octree representations and the application of rules that are based on the color of the octants under examination. Because it successively increases the discrete resolution of the spatial objects employed, the algorithm enables the user on the one hand to handle topological relationships in a fuzzy manner and on the other hand to trade-off between computational effort and the required accuracy. The article also presents detailed investigations on the runtime performance of the developed algorithm.

Key words: Building Information Modeling, Spatial Query Language, Topology, 3D, Octree, 9-Intersection model

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 *Building Product Model* or *Building Information Model* (BIM) [1–4]. These efforts have resulted in the widely known standards *Industry Foundation Classes* (IFC) [5] and CIS/2 [6].

Central *model management servers* or *product model servers* are seen as the most important IT infrastructure component for the future, as they promise to solve many of the issues involved in the highly collaborative design and engineering of buildings [7,8]. The main task of these model servers is to store the building information model centrally and manage all accesses to it. Product model servers specialized in handling IFC data that are already on the market include the Secom IFC Model Server, the Jotne EDMServer and the EuroStep Model Server, for example.

A very important prerequisite for asynchronous collaborative engineering is the creation of partial models. To allow the user to extract parts of the full building model, the product model servers provide query languages which make it possible to formulate conditions that need to be fulfilled by the resulting set of building components.

Unfortunately, the existing product model servers are unable to interpret the geometric information that is implicitly or explicitly contained in the building models, since they are not familiar with the spatial semantics of particular attributes and relationships. Accordingly, the expressive power of the query languages provided by the product model servers, such as the *Partial Model Query Language* [9] of the Secom IFC Model Server or the *Product Model*

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

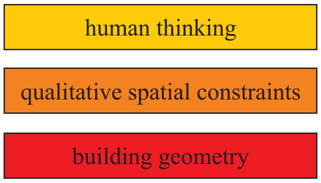


Fig. 1. The qualitative spatial operators provided by the Spatial Query Language form an intermediate level of abstraction.

Query Language of the EuroStep Model Server, is limited to numerical comparisons and tests on those spatial relationships that are predefined in the product data model.

This has to be seen as a major deficiency, since spatial relations between building components play a significant role in most of the design and engineering tasks of the AEC domain. To fill this technological gap we have developed concepts and techniques for a 3D Spatial Query Language for Building Information Models. The technology we have devised makes it possible to select specific building components by means of qualitative spatial constraints. These constraints form an intermediate level of abstraction between the technical view on building geometry using vertex coordinates, for instance, and the way humans think about buildings and the relations between their components (Fig. 1).

Possible applications for this innovative 3D Spatial Query Language for Building Information Models range from verifying construction rules to extracting partial models that fulfill particular 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. Also, the spatial query language can provide basic knowledge required for qualitative spatial reasoning [10].

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 [11–13]. 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 spatial operators available for the spatial types are the most important part of the algebra. They consist of

- metric (*distance, closerThan, fartherThan* etc.),
- directional (*above, below, northOf* etc.) and
- topological (*touch, within, contains* etc.)

operators.

While the metric operators are presented in [14] and the directional operators in [15], this paper discusses the definition and implementation of the topological operators.

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 realized in the GIS context. In the late 1980's, a multitude of different dialects were developed, including PSQL [16], Spatial SQL [17], GEOQL [18], KGIS [19] and TIGRIS [20]. A good overview of the different dialects and the basic advantages of a SQL-based implementation is provided in [21].

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 [22,23]. There is now a wide range of commercial 2D spatial database systems, the most widespread ones being *PostGIS*, *Oracle Spatial* and *Informix Geodetic Datablade*. The majority of available spatial databases comply to the standard developed by the OpenGIS consortium that defines a common interface for accessing 2D spatial data and accordingly enable the exchangeability of the database component in an overall GI system [24].

The potential benefits of using GI systems for the analysis of dynamical processes in buildings are discussed in [25] . The author states that, even if component-oriented CAD systems provide sophisticated functionality for geometric modeling, they normally lack comprehensive spatial analysis capabilities. For this reason, Ozel stores floor plans of buildings in a GIS database in order to use its 2D spatial analysis facilities. The author underlines that 3D spatial analysis would be 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 [26], 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.

[27] 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 employ the hierarchical data structure *oc-*

tree, which is dynamically created while processing a topological query.

2.2. *Topological operators*

Topological operators are used to query the topological relationship between two spatial entities. Since topological operators return a Boolean value, they are also denominated *topological predicates.*

Topological relationships can be formally described as follows [28]: Let X and Y be topological spaces. A mapping $f : X \rightarrow Y$ is continuous if for each open subset V of Y the set $f^{-1}(V)$ is an open subset of X. If the mapping f is a bijection and both f and f^{-1} are continuous, then f is called a *topological isomorphism.* Topological isomorphisms maintain neighborhood relationships between points during the mapping. Typical isomorphisms are translation, rotation and scaling ($scaleFactor \neq 0$) as well as any combination of these transformations. Topological relationships are those relationships that are invariant under a topological isomorphism.

Topological relations are among the most intensively investigated spatial relationships in the context of spatial query languages. Soon it became obvious that the impreciseness and ambiguity of colloquial language demands a formal definition of topological relationships. The main challenge is to find a set of qualitatively distinct relationships that is large enough to allow for a suitable classification and at the same time small enough to keep it manageable for the user.

The first substantial step towards a formalization of topological relationships was the development of the *4-intersection model* by Egenhofer et al. [29,30]. To formally specify the semantics of topological predicates the model determines the intersections between the interior and the boundary of the first object and the interior and the boundary of the second one as an empty or non-empty set. In theory there are 16 possible configurations but, depending on the dimensionality of the geometric object in question, only a subset can be found in reality.

The concept was first employed on intervals in one-dimensional space [31] and later extended to relations between simple regions in $\mathbb{R}^2$ [30]. In both cases 8 different relations have been distinguished to which the natural language denominations *disjoint, touch, equals, inside, contains, covers, coveredBy* and *overlap* could be assigned.

To resolve topological relations between line elements in $\mathbb{R}^2$ more precisely, the 4-intersection model has been upgraded to the 9-intersection model (9-IM) by incorporating the exteriors of both operands [30]. The resulting nine intersections are recorded in a 3 × 3 matrix:

$$\mathbf{I} = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}.$$

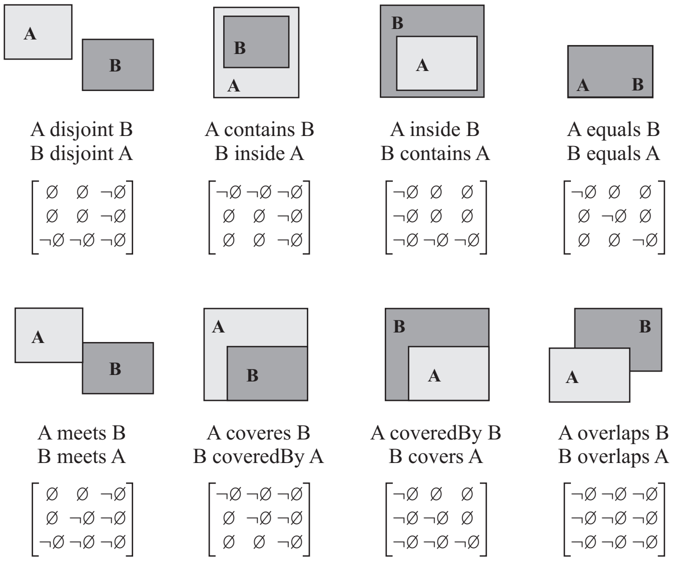


Fig. 2. The 9-IM matrices originally defined by Egenhofer for 2D space [32].

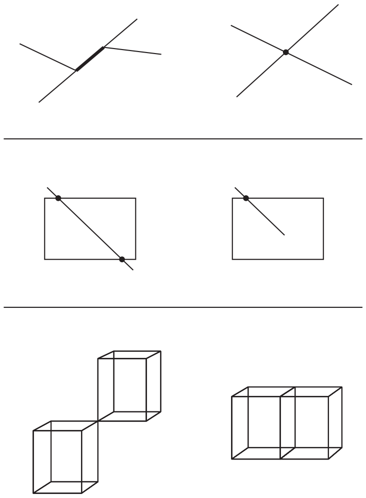


Fig. 3. Topological constellations that are intuitively different result in the same 9-IM matrix.

A° denotes the *interior*, ∂A the *boundary* and A^- the *exterior* of the spatial object A (see Section 3). The 9-IM can also be applied to combinations of spatial objects with a different dimensionality [32]. Fig. 2 shows the 9-IM matrices of the eight topological predicates defined by Egenhofer and depicts examples for 2D regions.

One drawback of the 9-IM is that some topological configurations that are intuitively different result in the same 9-IM matrix (Figure 3) while others that are intuitively identical are treated as being different. The first problem is partially solved by the *Dimensionally Extended 9-Intersection Model* (DE-9IM) which also records the dimensionality of the intersection set [28].

The DE-9IM forms the basis for the formal definitions of topological relationships in the OGC standard [24]. Here, *F* (false) is used in the matrices to denote an empty set, *T* (true) to denote an non-empty set, numbers may be used to define the dimensionality of the intersection set and, in addition, the wildcard ($*$) may be used at certain places in the matrix that are not relevant for the particular predicate, thereby solving the second of the aforementioned problems. Using this extended set of symbols, the OGC defines the predicates *contains, within, cross, disjoint, equals, intersect, touch* and *overlaps* for arbitrary combinations of (simple) *point*, *line* and *polygon* objects in 2D space.

An important pre-requisite for applying the 9-IM or its derivates is the formal specification of the *interior/boundary/exterior* of spatial objects. In general, we can distinguish two different approaches to realize this: The first approach relies on algebraic topology using *cellular complexes* [32] or *simplicial complexes* [33,34] to model spatial entities. This implies a complete partitioning of the entire space in a rigorously formal manner and accordingly calls for appropriate modeling tools, since conventional B-Rep modelers do not provide these capabilities.

The second approach relies on point-set topology [30,35,11]: Here, *interior, boundary* and *exterior* are understood as point sets. The boundary point set is formed by points whose neighborhood (a well-defined concept of point-set topology) contains both exterior and interior points. This concept can be easily applied to conventional B-Rep models [36]. Special care has to be taken in the case of dimensionally reduced entities in order to avoid that all points become boundary points.

2.3. *3D-GIS*

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 [34,37,38] 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 [39].

[40–43] provide concepts and data structures for storing 3D city models in spatial databases and discuss the suitability of different geometry models for querying topological relationships. 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 [33,32,44].

However, as mentioned above, the existing building information models use more general B-Rep models to describe geometry. For this reason, we pursue a different approach here: Instead of storing the topological relationships, we derive them on-the-fly from the position and shape of the geometric objects involved. This avoids redundancy (position and shape of the objects determine their topological relationship) while simultaneously supporting the use of conventional B-Rep approaches to describe the buildings' geometry.

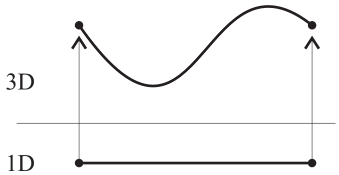


Fig. 4. If 3D *Line* objects are defined as mappings from 1D intervals, the notion of *boundary* can be preserved for the endpoints.

3. Formal specification of topological operators

Since the definitions of topological operators found in literature are unsatisfactory for the intended use in the Building Model context, we have set up our own definitions.

We use the pure *9-Intersection Model* instead of the dimensionally extended version, because the *dimension* operator cannot be realized by means of the octree implementation technique presented in Section 4. In order to avoid an unmanageably large number of different topological predicates, we apply the clustering method, as proposed in [45], which makes it possible to place wildcards (*) at those places in the 9-IM matrix that are not decisive for assigning a predicate to a certain constellation.

To be able to apply the 9-IM it is necessary to define the *interior*, the *boundary* and the *exterior* for each of the 4 spatial types defined within the Spatial Query Language. These definitions have been given in [12,36] and are not repeated here. In summary, the purpose of the definitions is to transfer the specification of the *interior* and *exterior* for each of the spatial types from the world of algebraic topology into the world of point-set topology. For example, we intended to define the endpoints of a *Line* element as its *boundary*, and all other points as belonging to its *interior*. Using the neighborhood concept from point-set topology in 3D space would result in all points of a Line belong to the boundary. Accordingly we defined all *Line* objects as mappings from 1D to 3D space, specified the boundary points in 1D, and assigned "boundary" also to their mappings (Fig. 4) .

Figures 5 and 6 show the topological predicates provided within our Spatial Query Language, present the corresponding 9-IM matrices and illustrate their semantics for different combinations of types by means of pictograms. The given system of topological predicates satisfies the requirements of *completeness* and *mutual exclusiveness*, i.e. we assign to any topological constellation exactly one of the predicates. This makes it possible to introduce an ad-

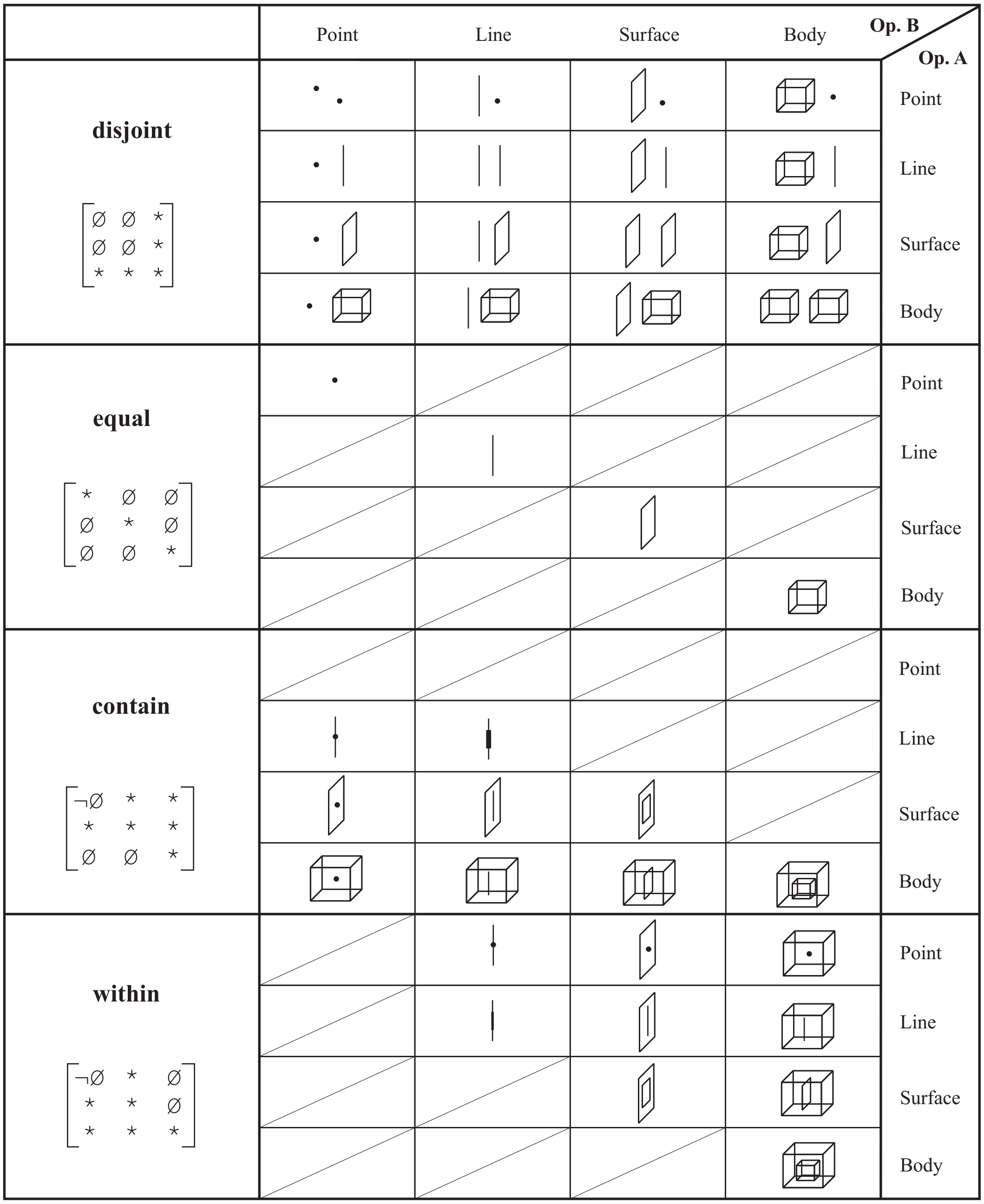


Fig. 5. The topological predicates provided by the Spatial Query Language (part 1).

ditional operator which returns the topological predicate for any given pair of spatial objects. This operator is called *whichTopoPredicate*.

Note that using the pure 9-IM without the dimension operator does not allow for any distinction between an *overlap* and a *cross* situation, as proposed in [11]. Nor is it possible to realize the proposed refinements of *touch* (*meet* and *onBoundary*).

There are small distinctions between the definitions in [45] with respect to the clustering of predicates: The predicates *coverBy* and *cover* have not been adopted, because in the application domain considered here, it is normally irrelevant whether only the two operands' interiors overlap or their boundaries as well. Accordingly, these two constellations are subsumed under *within* and *contains*, respectively. In addition, we use the designation *touch* instead of *meet* in order to achieve maximum compliance with the OGC standard.

4. Octree-based implementation

4.1. *Octree representation*

Our implementation technique is based on the octree representation of the spatial objects involved in the topological query. The octree is a space-dividing, hierarchical tree

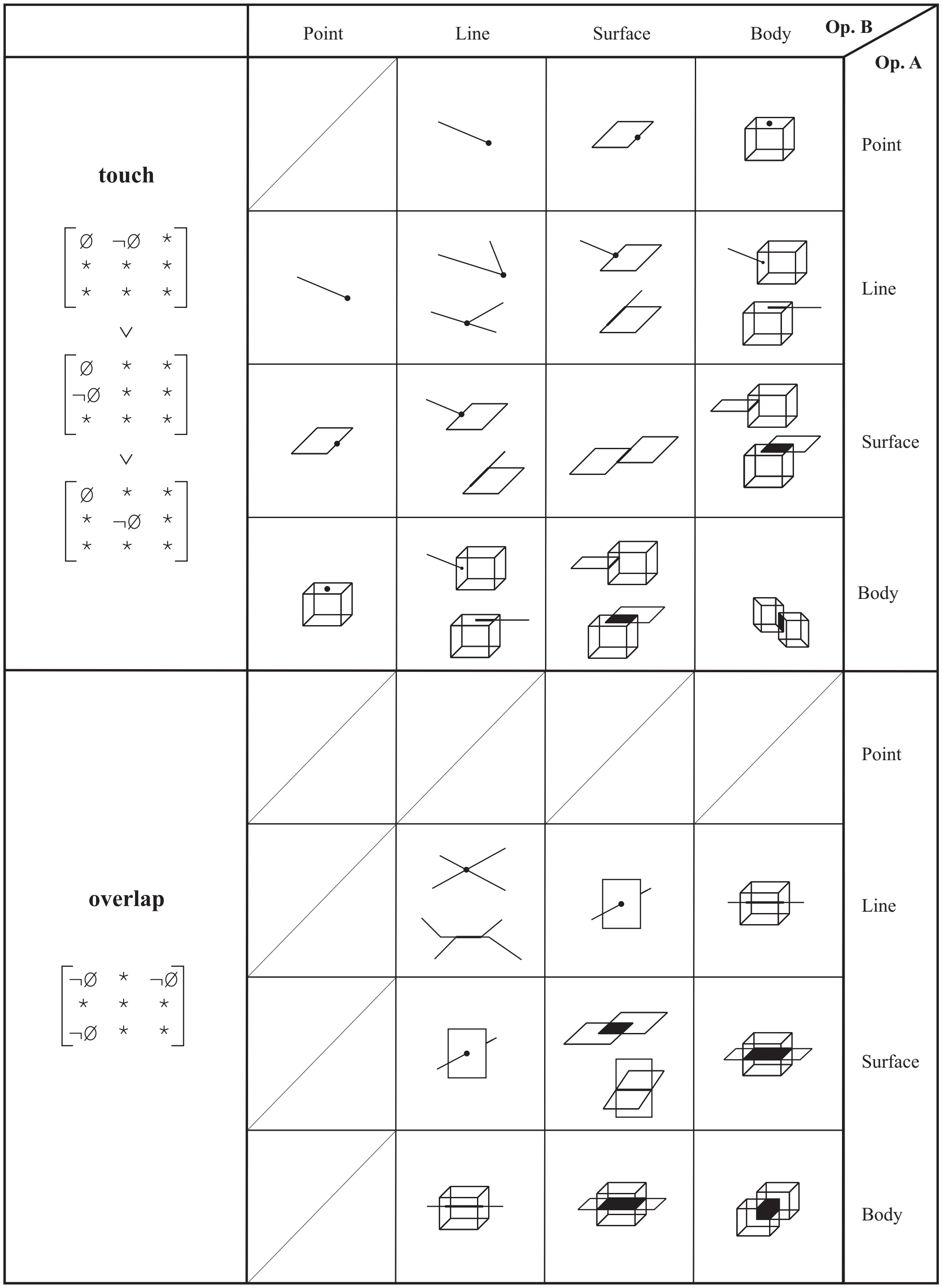

Point
Line
Surface
Body
Op. B
Op. A
Point
Line
Surface
Body
touch
$\begin{bmatrix} \emptyset & \neg\emptyset & \star \\ \star & \star & \star \\ \star & \star & \star \end{bmatrix} \vee \begin{bmatrix} \emptyset & \star & \star \\ \neg\emptyset & \star & \star \\ \star & \star & \star \end{bmatrix} \vee \begin{bmatrix} \emptyset & \star & \star \\ \star & \neg\emptyset & \star \\ \star & \star & \star \end{bmatrix}$
Point
Line
Surface
Body
overlap
$\begin{bmatrix} \neg\emptyset & \star & \neg\emptyset \\ \star & \star & \star \\ \neg\emptyset & \star & \star \end{bmatrix}$

Fig. 6. The topological predicates provided by the Spatial Query Language (part 2).

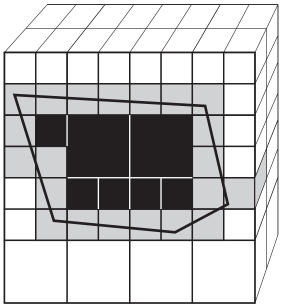


Fig. 7. Cross-section through an octree. *White* cells represent the *exterior*, *black* cells the *interior* and *gray* cells the *boundary* of the discretized object. Whereas *black* and *white* cells are branch nodes, *gray* cells always have eight children.

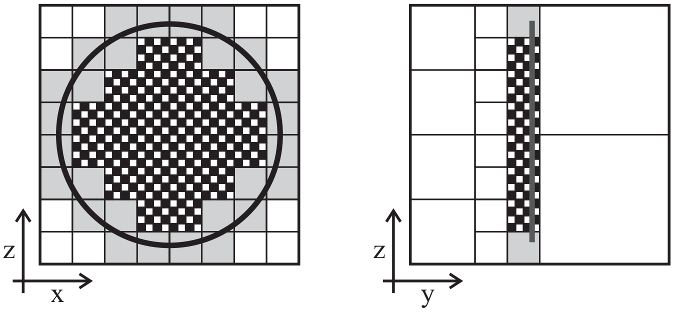


Fig. 8. Dimensionally reduced objects such as the illustrated disc are discretized using the fourth color *black/white* that represents cells which contain interior and exterior points, but no boundary points.

data structure for the discretized representation of 3D volumetric geometry [46–49].

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 (Fig. 7). 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.

In our implementation concept, each spatial object is represented by an individual octree. There are several different approaches for creating an octree from 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 that are colored gray.

For our implementation we use a highly efficient creation method developed by Mundani [50] that is based on processing the halfspaces formed by the object's bounding faces. The most important advantage of Mundani's approach for our purposes is that it automatically marks inner cells as black without performing a laborious filling algorithm. As described in the next sections, the existence of black cells is 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.

To cover also dimensionally reduced entities with our algorithm, we have to introduce the fourth color *black/white*. Black/white cells represent areas where the exterior and the interior of the described object exist, but not its boundary (Fig. 8).

In our approach the octree generation is not performed in advance, but coupled with the recursive algorithm presented in the next section. Thus the octree 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. *Overview*

For a better understanding, we first give an overview on the functioning of the algorithm before describing the details. This subsection describes the algorithm implementing the *whichTopoPredicate* operator.

Both spatial objects for which the topological relationship is to be determined are encoded in a separate octree. Then, the recursive algorithm performs a synchronized breadth-first traversal of both octrees. On each level, pairs of octants are created 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 color combination for the specific rules that 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 acquired about the topological constellation. There are 12 positive and 9 negative rules altogether (Fig. 9, 10 and 11). A positive rule can be applied when a certain color combination occurs, and a negative rule if certain color 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.

The rules are derived from the semantics of the colors. 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 operand A occurs at the same place as a *black* octant of operand B, it follows that the intersection between the exterior of A and the interior of B is non-empty.

The 9-IM working matrix is successively filled by applying these rules for all octant pairs. Each time a new entry is made, the matrix is compared with the matrices of the formal definitions (Section 3). If it completely complies with one of these matrices, 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.

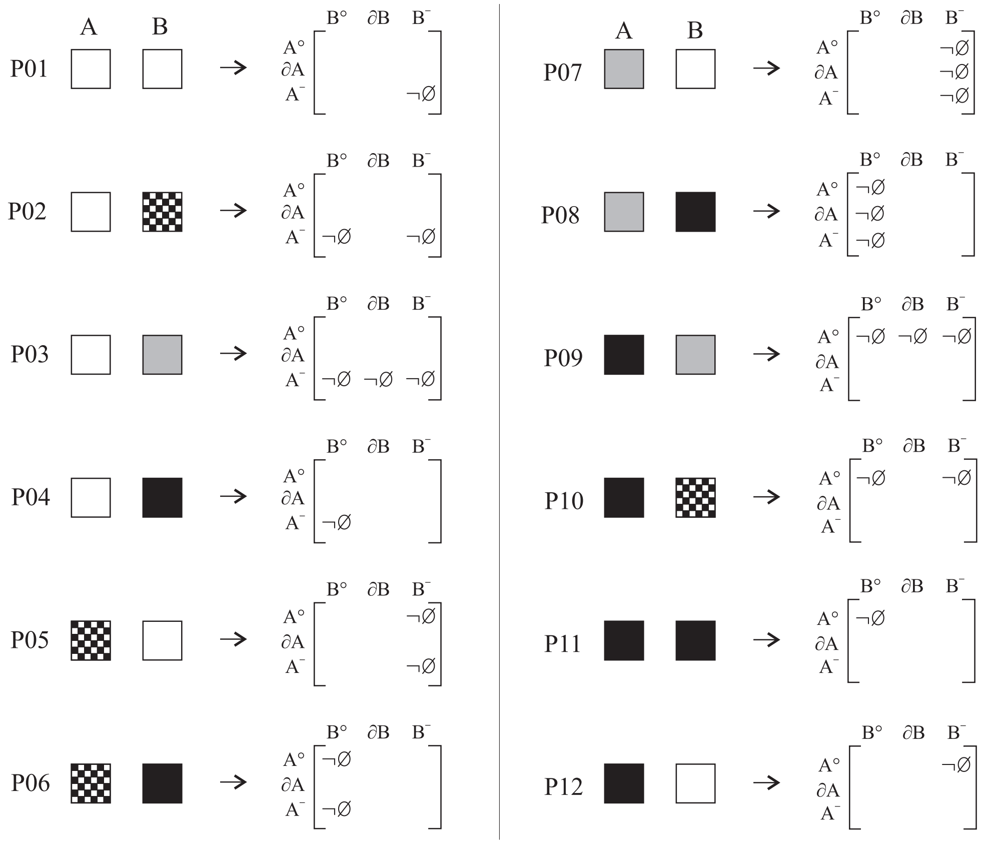


Fig. 9. Positive Rules (Part 1). If the color combination on the left-hand side is detected, the 9IM-Matrix can be filled according to the right-hand side. Combinations of mixed color cells (gray and/or checkered) never lead to the 9IM matrix being filled, since they do not allow for a statement about the exact boundary position. For the same reason there is no color combination from which $\partial A \cap \partial B = \neg\emptyset$ could be derived.

4.3. *The recursive algorithm*

In a first step, we perform tests based on the bounding boxes of both operands for a fast decision in simple cases. These tests check preconditions that are necessary but not sufficient for assigning the topological predicate under investigation. To fulfill the *equal* predicate, the bounding boxes of both operands have to be equal, to fulfill *touch* the bounding boxes may either touch each other or overlap, but are not allowed to be disjoint. To fulfill *overlap*, however, the bounding boxes of the operands also have to overlap. In order to satisfy *contain*, the bounding box of the first operand has to contain that of the second one, and in the case of *within* it is vice-versa.

If the prerquisits test based on the bounding-boxes is successful, the octree-based algorithm for a detailed investigation can commence. The principle of the algorithm is the successive refinement of the octrees at “critical” places and the continuous filling of a 9-IM working matrix which finally leads to the validation or falsification of the predicate under investigation.

The algorithm is identical for all operators representing topological predicates: *disjoint*, *equal*, *within*, *contain*, *touch* and *overlap*. The algorithm works recursively, equally to the algorithms presented for implementing metric and directional operators [15,14]. Figures 15 to 17 illustrate the functioning of the algorithm for different topological constellations.

First, the main routine *testTopoPredicate* (Alg. 1) is called, passing as parameters the root octants of the operands, an integer value representing the predicate to be tested *(predicate)*, and a 9-IM matrix occupied exclusively by wildcards *(workingMatrix)*. The latter is successively filled with values during the recursion. The three possible values *empty set*, *non-empty set* and *undefined* are encoded by the integer values 0, 1 and −1, respectively.

The core of the algorithm lies in the execution of rules. The occurrence of specific color combinations results in filling the 9-IM working matrix with non-empty set symbols (positive rules, Fig. 9). On the other-hand, the absence of certain color combinations leads to one or more empty-set entries in the matrix (negative rules, Fig. 10 and 11).

The appearance of a certain color combination is recorded in the field *color_combinations* consisting of 16 Boolean variables (Alg. 1, line 2). Positive rules are applied directly for each cell pair (line 5). Once this has been accomplished, we check whether the resulting matrix already allows a validation or falsification of the predicate. If this is the case, the *testTopoPredicate* function returns *true* or *false*, respectively, the recursion is stopped and the main

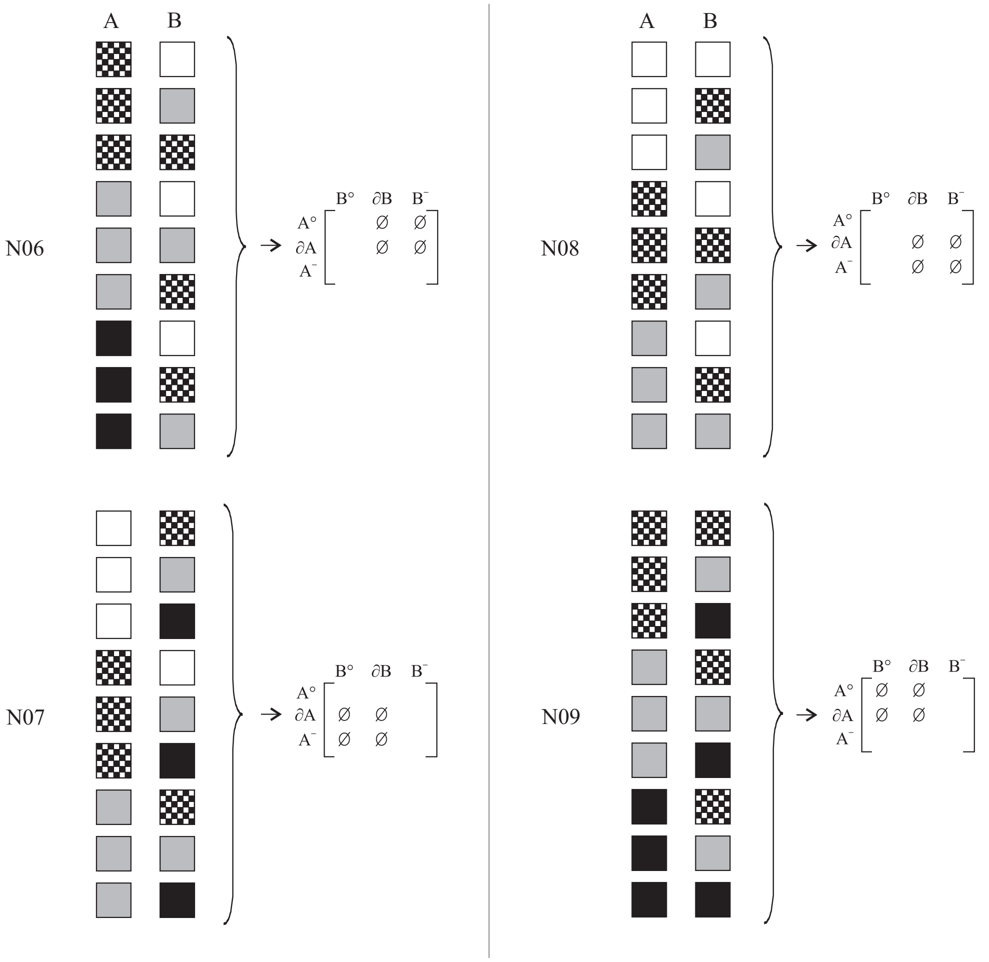


Fig. 11. Negative Rules (2). If the color combination of the left-hand side do not occur in the entire domain, the 9-IM matrix can be filled according to the right-hand side.

routine returns *true* or *false* as final result. The same happens if it was possible to validate a different topological predicate from the one that was tested (line 12–16).

On the other hand, negative rules are applied after all color combination of the current recursion level have been recorded (line 18). The existing negative rules depend on each other in a hierarchical manner (Fig. 12): If the prerequisites for applying a rule are not fulfilled, they automatically remain unfulfilled for its successors. Hence, the rules are tested in an appropriate order. If a rule is applicable, we test whether a final validation (line 21, 22) or falsification (line 24, 25) is possible by comparing the current matrix with the target matrix.

If after execution of all applicable rules, the current occupancy of the working matrix does not allow for validation or falsification of the predicate and the maximum refinement level has not been reached, child pairs are created using the sub-routine *createChildrenPairs* (Alg. 4.3) (Alg. 1, line 43) and passed to the next level of recursion by calling *testTopoPredicate* (Alg. 1, line 45)

4.4. *The predicate hierarchy*

If the predicate under examination is neither proved nor disproved when reaching the maximum refinement level, the predicate hierarchy shown in Fig. 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 we assume that the highest non-disproved predicate as proved.

If both operands have the same dimensionality, *contain* and *within* are equivalent: 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.

As discussed in detail later on, the application of the predicate hierarchy may result in the detection of an incorrect topological predicate, if the maximum refinement level is too low. However, the hierarchy is chosen in such a

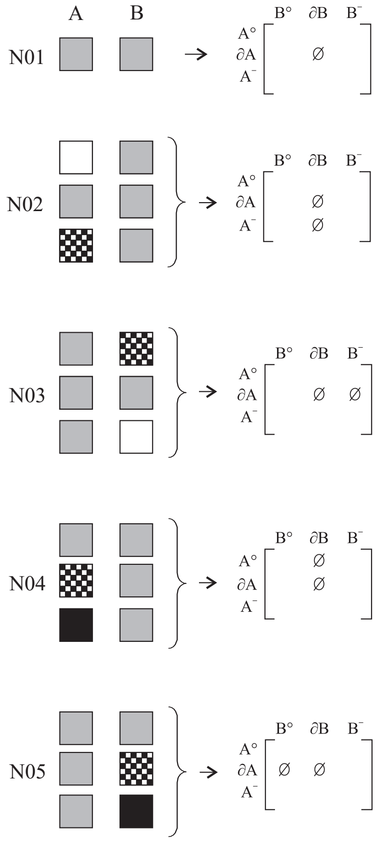


Fig. 10. Negative Rules (1). If the color combinations of the left-hand side do not occur in the entire domain, the 9-IM matrix can be filled according to the right-hand side.

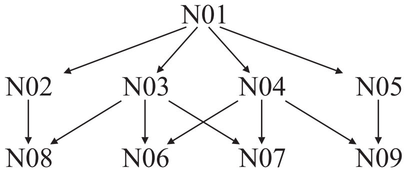


Fig. 12. Dependencies between the negative rules. If the preconditions of a rule are not fulfilled, they remain unfulfilled for any of its successors.

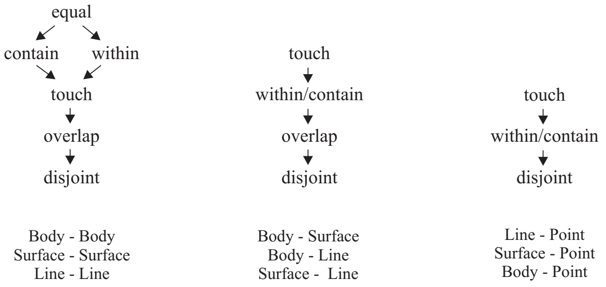


Fig. 13. The hierarchy of the topological predicates for different type combinations. The algorithm returns the highest non-disproved predicate. The order of the hierarchy results from the observation that all predicates above a certain predicate a are disproved during ongoing refinement if the actual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

```
boolean testTopoPredicate
    (OctantPair[] pairs, int predicate, int[] workingMatrix, int currentLevel)
currentLevel ← currentLevel + 1
int[16] color_combinations
for all pairs do
    add 1 to accordant position in color_combinations
    fill workingMatrix according to positive rules
    if workingMatrix mismatches 9IM-matrices[predicate] then
        return false
    end if
    if workingMatrix matches 9IM-matrices[predicate] then
        return true
    end if
    for all predicates i≠predicate do
        if workingMatrix matches 9IM-matrices[i] then
            return false
        end if
    end for
end for
for all negativ rules do
    if rule is applicable then
        fill workingMatrix
        if workingMatrix matches 9IM-matrices[predicate] then
            return true
        end if
        if workingMatrix mismatches 9IM-matrices[predicate] then
            return false
        end if
        for all predicates i≠predicate do
            if workingMatrix matches 9IM-matrices[i] then
                return false
            end if
        end for
    end if
end for
if currentLevel = maxLevel then
    for all i < predicate do
        if workingMatrix does not mismatch 9IM-matrices[i] then
            return false
        end if
    end for
    return true
else
    for all pairs do
        childrenPairs += createChildrenPairs(pair[i])
    end for
    return testTopoPredicate(childrenPairs, workingMatrix, currentLevel)
end if
```

Algorithm 1: The function *testTopoPredicate* determines the occurrence of a specific topological situation by traversing the octrees in breadth-first manner and simultaneously filling the 9-IM working matrix. To realize the octree traversal, the functions calls itself recursively.

way that these errors/misjudgements are acceptable, since they correlate with the intuitive human understanding of fuzziness. Fig. 14 illustrates constellations where the application of the predicate hierarchy results in the detection of an incorrect topological predicate.

Consequently, the algorithm *testTopoPredicate* works as follows: If the predicate under examination is neither proved nor disproved when reaching the maximum refine-

```
boolean **createChildrenPairs**(OctantPair pair)
if  (pair.A.color = gray) or (pair.A.color = black/white) then
  if (pair.B.color = gray) or (pair.B.color = black/white) then
    for all A.children do
      for all B.children do
        create pair(A.child, B.child)
        childrenPairs += pair
      end for
    end for
  else // B is white or black
    for all A.children do
      create pair(A.child, B)
      childrenPairs += pair
    end for
  end if
else // A is white or black
  if (pair.B.color = gray) or (pair.B.color = black/white) then
    for all A.children do
      for all B.children do
        create pair(A, B.child)
        childrenPairs += pair
      end for
    end for
  end if
end if
return childrenPairs
```

Algorithm 2: The subroutine *createChildrenPairs* generates child cell pairs, that are used for the tests on the next recursion level.

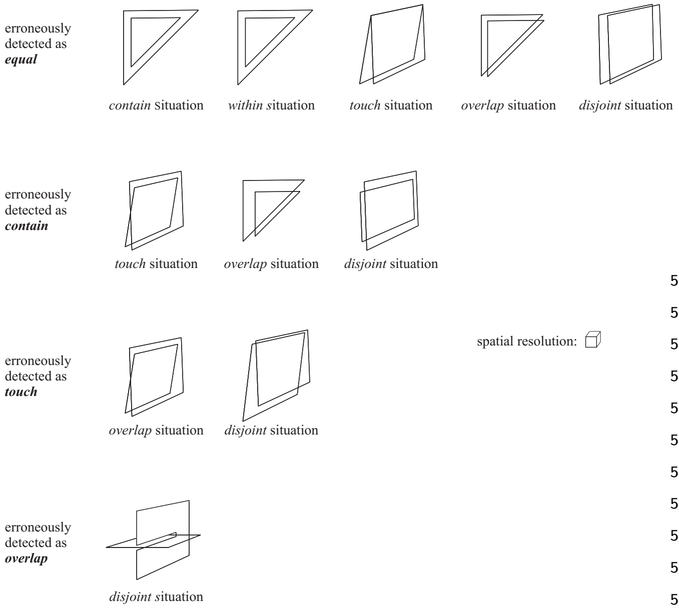


Fig. 14. Misjudgments caused by a resolution that is too rough in the case of *Surface–Surface* constellations. The predicate erroneously assigned to the depicted situation results directly from the chosen predicate hierarchy.

ment level (Alg. 1, line 34), it checks whether all predicates which are higher in the hierarchy are disproved (line 35–39). If this is the case it returns *true*, if not, *false* (line 37). For this purpose, the 9-IM matrices of the predicates are stored in the array in the order of the hierarchy. The special case of the predicates *contain* and *within* which are on the same level of the hierarchy is not described here.

```
int **whichTopoPredicate**
    (OctantPair[ ] pairs, int[ ] workingMatrix, int currentLevel)
currentLevel ← currentLevel + 1
int[16] color_combinations
for all pairs do
  add 1 to accordant position in color_combinations
  fill workingMatrix according to positive rules
  for all predicates i do
    if workingMatrix matches 9IM-matrices[i] then
      return i
    end if
  end for
end for
for all negativ rules do
  if rule is applicable then
    fill workingMatrix
    for all predicates i do
      if workingMatrix matches 9IM-matrices[i] then
        return i
      end if
    end for
  end if
end for
if currentLevel = maxLevel then
  for  i = 0 to 5  do
    if workingMatrix does not mismatch 9IM-matrices[i] then
      return i
    end if
  end for
else
  for all pairs do
    childrenPairs += createChildrenPairs(pair[i])
  end for
  return whichTopoPredicate(childrenPairs, workingMatrix,
  currentLevel) // recursive call
end if
```

Algorithm 3: The function *whichTopoPredicate* determines the topological situation for two spatial objects.

The algorithm *whichTopoPredicate* works in a similiar way (Alg. 4.4): It successively refines the octrees till either the 9-IM working matrix fully complies with one of the predicates, or the maximum refinement level is reached. In the latter case, the first non-disproved predicate of the hierarchy is returned.

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 exactly one topological predicate is detected for any topological constellation, regardless of whether the user applies the predicate-operators or *whichTopoPredicate*.

The positivistic approach is especially important for topological constellations that can never be fully proved, even not at an arbitrary high resolution. Among them are the predicates *touch* and *disjoint* for Body-Body combinations. The reason is that, if the geometric objects under examination really comply with one of these topological situations, the algorithm will detect an overlapping of gray A cells and gray B cells at any refinement level (see Figs. 16 and 17).

So in the case of *touch*, we cannot apply rules that fill the 9-IM working matrix with $A^\circ \cap B^\circ = \emptyset$ which would be necessary for the working matrix to fully comply with the *touch* predicate matrix. And in the case of *equal* it is not possible to apply any negative rule that would generate the required $\emptyset$ entries of the *equal* matrix.

If we have a closer look at these issues we will find that *touch* "competes" with *overlap* and *disjoint.* A slight displacement of the objects would result in one of the other predicates being returned. However, while a real *overlap* or *disjoint* situation would result in fully provable working matrices at a certain refinement level, a real *touch* situation will not do so. At the same time, *overlap* can not be disproved in a real *touch* situation, since gray cells meet at any level. These observations resulted in introducing the predicate hierarchy that is based on the positivistic approach, assuming the most probable state: *touch* is returned if we reach the maximum level and neither *overlap* nor *disjoint* are fulfilled.

A consequence of the positivistic approach is that potentially erroneous results have to be accepted. If the maximum level is not chosen high enough and the final resolution is consequently too rough, two slightly overlapping objects might be classified as being in touch, for example. However, the overlap that is not detected by the algorithm has an upper limit that is directly related to the size of an octant at the maximum refinement level. Thus, the user can choose the appropriate accuracy by defining a corresponding refinement level. We can study a similar behavior of the algorithm assuming two objects that are "almost" equal, but in fact slightly overlap: If the maximum refinement level is not high enough these two objects are classified as being equal.

These "errors" can be seen as a disadvantage of our approach, but we consider it to be its main advantage: The octree-based algorithm allows for a *fuzzy handling* of topological relationships that complies to the way humans think about topological relationships. In many applications it is desirable that two almost equal geometric objects are treated as equal. The same is true for objects that slightly overlap or have a small gap between them - in many cases it is appropiate to classify them as touching. However, the algorithm prsented here also allows us to "take a closer look" at the topological situation by choosing a higher maximum refinement level and thus limiting the divergence of the objects classified as being equal, or the gap / overlap of the objects classified as being in touch.

Another implementation approach currently under examination is to work with ternary data types returned by the topological operators, thus offering three different values *true*, *false* and *unknown.* In this case, errors would be completely avoided and the interpretation of the result would be left to the users. This approach will be discussed in more detail in future publications.

In any case, by using the octree-based implementation the user can balance the desired accuracy with the required computational effort by choosing a suitable maximum refinement level.

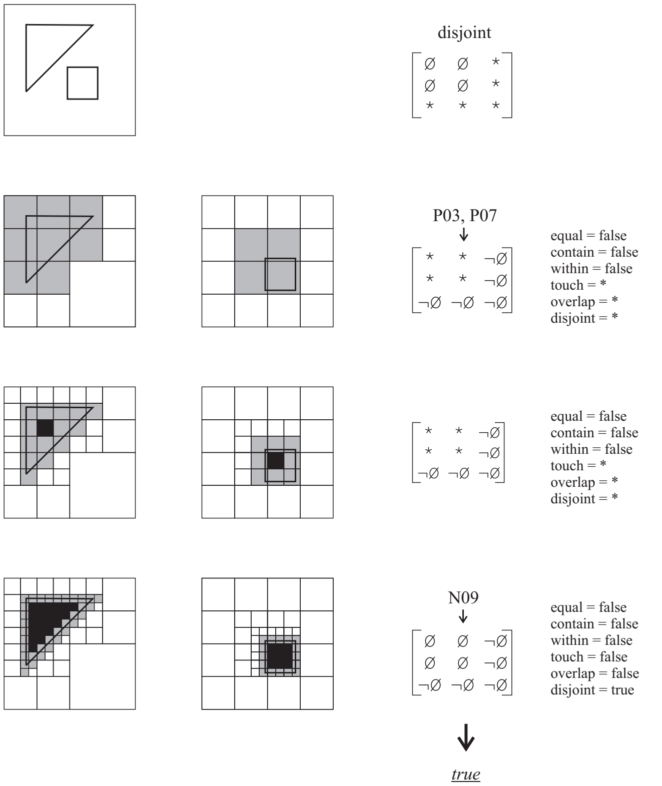


Fig. 15. 2D illustration of the examination of a *disjoint* constellation of n-dimensional objects in n-dimensional space. The *disjoint* situation is correctly detected on level 4. If the chosen maximum refinement level is lower, the algorithm will detect a *touch* constellation by applying the predicate hierarchy. There are analogous examples in 3D space for *Body* objects.

5. Query language implementation base: Object-relational vs. relational SQL

We decided to base the declarative spatial query support on SQL, since it is one of the most widespread and powerful declarative query languages with strongly formalized foundations in relational algebra [51]. 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.

We compared two different versions of the standard: the purely relational version known as SQL-92 and the object-relational version known as SQL:1999 [52,53]. The latter enables an extension of the database type system in an object-oriented way, especially by providing abstract data types (ADTs) which may possess member functions (methods) [54–56]. 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   IFCColumn col, IFCSlab slab3
WHERE  col.touch(slab3) AND slab3.id = 'Oid23089'
```

to extract all columns that touch the slab whose ID is

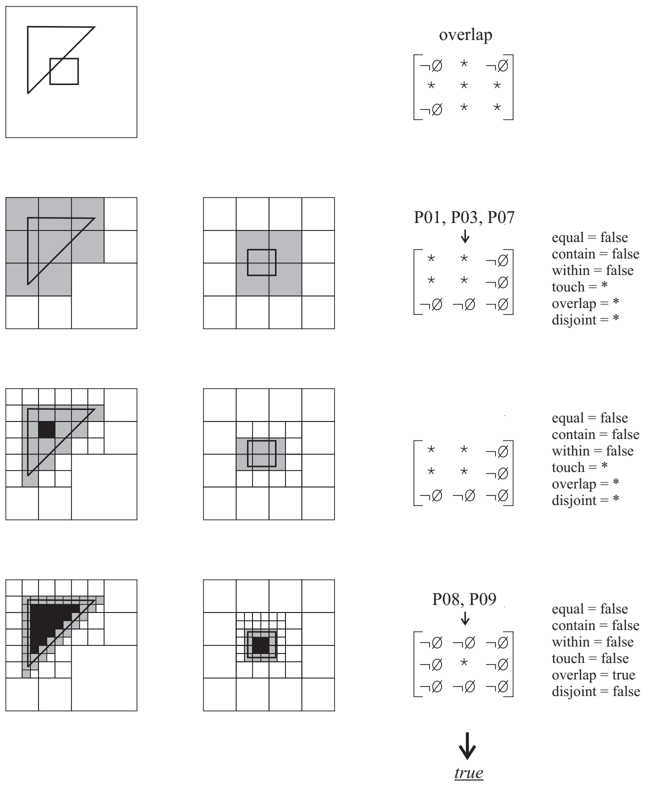


Fig. 16. 2D illustration of the examination of an *overlap* constellation of *n*-dimensional objects in *n*-dimensional space. The *overlap* situation is correctly detected on level 4. If the chosen maximum refinement level is lower, the algorithm will detect a *touch* constellation by applying the predicate hierarchy. There are analogous examples in 3D space for *Body* objects.

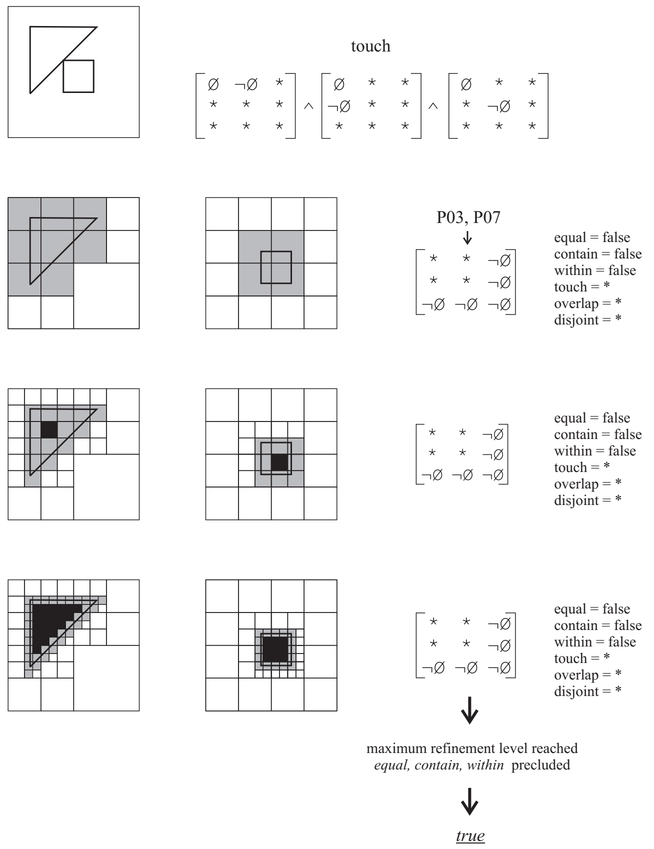


Fig. 17. 2D illustration of the examination of a *touch* constellation of *n*-dimensional objects in *n*-dimensional space. Also with an arbitrary high refinement, *touch* can not be fully validated: The existence of *gray-gray* pairs may result in *disjoint* or *overlap* constellations when refining further. For this reason, the predicate hierarchy is introduced. Its application makes it possible to assign *touch* when the maximum refinement level is reached and the predicates *equal*, *contain* and *within* have been precluded.

Oid23089.

As can be seen in the example, spatial operators, such as *touch*, 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.

For a prototype implementation we 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 should refer to [15,14].

The most important advantage of using an object-relational approach is the strong type safety provided through the declaration of user-defined types. The declaration of the *touch* operator, for example, forces the operands to be of type *SpatialObject* or one of its sub-types *Body*, *Surface*, *Line* or *Point*. Thus, type errors may already be detected by the query engine during the interpretation of the SQL statement and a more specific error report can be generated for the user.

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 required, 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 for 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 new complex data types, also denoted as Abstract Data Types (ADT's).

So it is necessary to choose a semantically weaker way of defining the spatial operators: The operands are defined as *strings* representing the object's IDs, and not as being of type *SpatialObject*. Nor are the operands not defined as member functions, but as global functions. The specimen query then reads:

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

Because the syntactic differences are comparatively small on the one hand, and the query processing time is considerably reduced on the other hand, we decided to continue

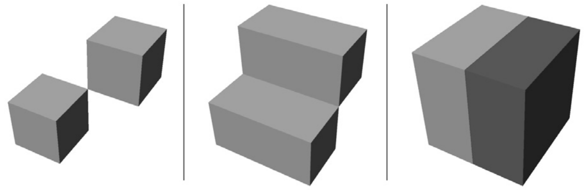


Fig. 19. The scenarios “cubes meet in vertex”, “cubes meet at edge”, “cubes meet on a face” for the theoretical performance tests.

our developments using an In-Memory Database accepting the loss of type-safety. For this purpose we chose the freely available, open-source database engine H2 which displays excellent performance and offers the required support for user-defined stored procedures [57].

In both implementation approaches, the user-defined functions representing the topological operators are linked to the algorithms presented in Section 4 that perform the actual processing of the spatial operator.

6. Software prototype

To prove the feasibility of the developed concept we implemented a software prototype that offers spatial query functionality for building information models. It is capable of reading-in IFC-VRML files generated by the *IFC-StoreyView* program developed by Karlsruhe Institute of Technology [58]. It stores all the building elements contained in the file in database tables, such that there is one table per found element type. It further offers the possibility of composing SQL queries containing spatial predicates. After processing the query, the resulting set of building elements is highlighted in the 3D viewer. A screenshot of the prototype is shown in Fig. 18.

7. Performance tests

To test the performance of the octree-based implementation of topological operators in combination with an in-memory SQL query engine we not only applied them to theoretical scenarios representing extreme cases, but also to real-world examples that show the runtime behavior under typical conditions.

7.1. *Theoretical scenarios*

For the theoretical scenarios we chose the operator *touch* and applied it on three different constellations: In the first scenario two cubes meet in a vertex (case 1), in the second scenario, these two cubes meet at an edge (case 2) and in the third one they meet on a face (case 3). The constellations are depicted in Fig. 19. In a fourth test, we chose the operator *equal* and applied it on two identical cubes (case 4). All timings are in milliseconds.

max. level	vertex	edge	face	equal
2	4	4	4	4
3	8	10	22	36
4	15	27	61	89
5	15	55	163	372
6	20	106	487	1158
7	21	150	1685	4421
8	27	219	5816	41641
9	27	354	-	-
10	28	737	-	-
11	34	1323	-	-
12	36	2622	-	-
13	37	5067	-	-
14	42	15705	-	-

Fig. 20. Performance measurement results for testing on *touch* for the scenarios “cubes meet in vertex”, “cubes meet at edge”, “cubes meet on a face” and for testing on *equal* for two identical cubes. All timings are in milliseconds.

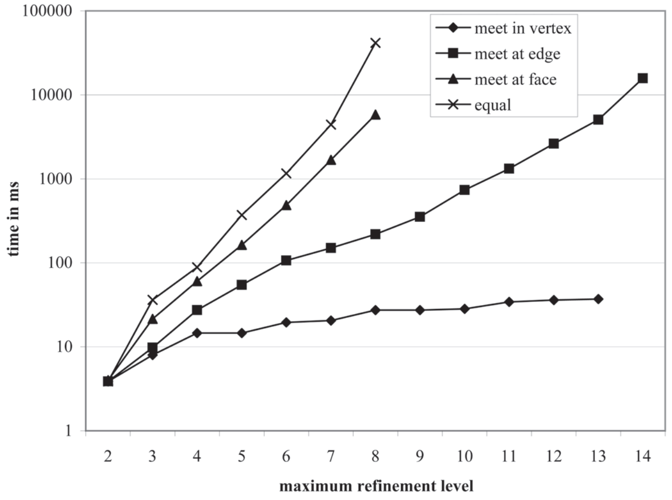


Fig. 21. Measurement results plotted logarithmically. In best-case scenarios the algorithm shows linear behavior, in worst-case scenarios it shows exponential behavior.

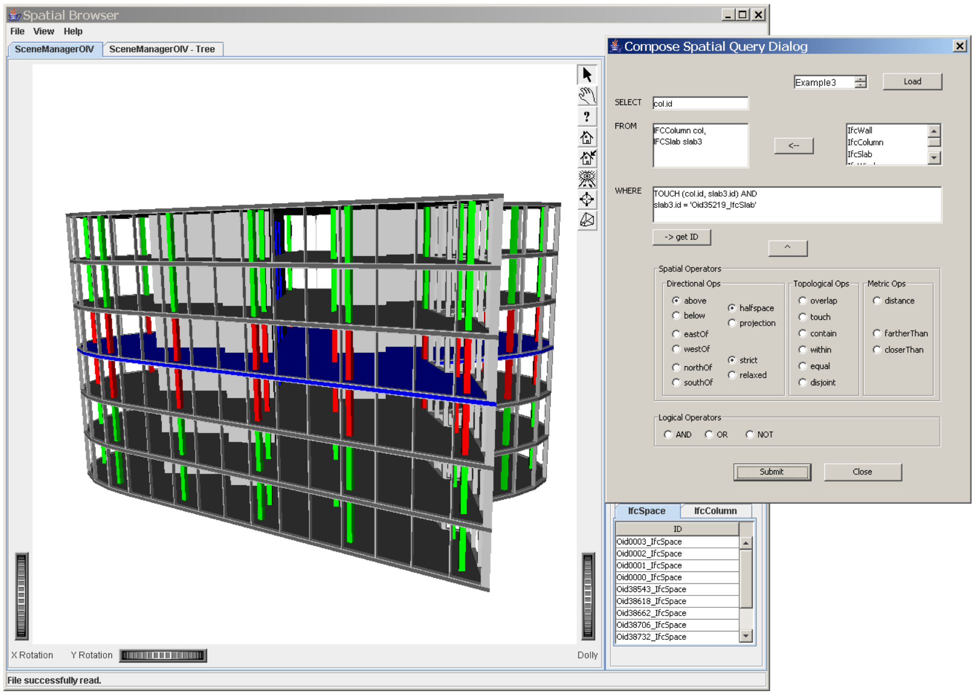
The rationale behind these choices is that they represent extreme cases, because the algorithm refines the octrees only at the significant places. For both *touch* and *equal* this is where the boundaries of the two objects meet, i.e. in case 1 at one vertex, in case 2 at an entire edge, in case 3 on an entire face, and in case 4 on the entire surface of the cubes. The number of examined octants per level is consequently constant in case 1, twice as high between one level and the next level in case 2 and four times as high in cases 3 and 4.

The runtime measurements (Fig.s 20 and 21) reflect this behavior. For case 1 we observer a linear behavior ($\mathcal{O}(n)$) w.r.p. to the maximum refinement level, and for cases 2, 3 and 4 we observe an exponential behavior ($\mathcal{O}(n^k)$) with variable degrees of steepness.

The tested algorithms are implemented in Java. All performance tests were run using JDK 1.6.0_02 on an Intel Pentium M 2.13 GHz machine with 1.5 GB installed RAM.

7.2. *Real-world performance tests*

Since the scope of our work is the development of a spatial query language for building information models we also



Example3

SELECT col.id

FROM IFCColumn col,
IFCSlab slab3

IfcWall
IfcColumn
IfcSlab

WHERE TOUCH (col.id, slab3.id) AND
slab3.id = 'Oid35219_IfcSlab'

Spatial Operators

Directional Ops: ● above ○ below ○ eastOf ○ westOf ○ northOf ○ southOf ● halfspace ○ projection ● strict ○ relaxed

Topological Ops: ○ overlap ○ touch ○ contain ○ within ○ equal ○ disjoint

Metric Ops: ○ distance ○ fartherThan ○ closerThan

Logical Operators: ○ AND ○ OR ○ NOT

ID
Oid0003_IfcSpace
Oid0002_IfcSpace
Oid0001_IfcSpace
Oid0000_IfcSpace
Oid38543_IfcSpace
Oid38618_IfcSpace
Oid38662_IfcSpace
Oid38706_IfcSpace
Oid38732_IfcSpace

File successfully read.

Fig. 18. Screenshot of the prototype application showing the dialog for composing spatial SQL queries and the 3D viewer highlighting the result set.

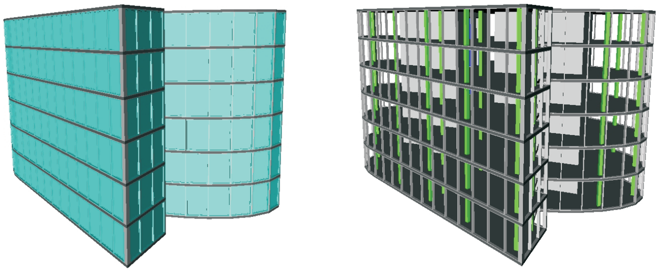


Fig. 22. Specimen model of a building for the real-world test. It contains 1180 entities.

tested the performance of our algorithms on a typical building model applying typical spatial queries. The building model we used is shown in Fig. 22. It consists of 1180 entities including 426 walls, 108 columns, 7 slabs, 24 doors, 288 windows and 13 spaces.

The queries performed on this building model are shown in Alg. 4. Query 1 selects all columns that touch a certain slab, whereas query 2 selects the walls that touch a certain space. Query 3 is used to find all columns within the same space and query 4 finds all overlapping building components. Query 5 finds all elements that have the same geometry and position but different IDs, i.e. it can be assumed that they have been doubled by mistake. In query 6 we combine a topological predicate with a directional one [15] to find all columns directly on top of a certain slab.

The maximum extension of the building model is 32.8 m, so a maximum refinement level of 8 corresponds to an accuracy of 7.28 cm and a maximum level of 9 corresponds to 0.11 cm. Accordingly, level 9 should normally suit the users’ precision requirements.

Query 1

```
SELECT col.id
FROM IFCColumn col
WHERE TOUCH(col.id, ’Oid35219_IfcSlab’)
```

Query 2

```
SELECT w.id
FROM IFCWallStandardCase w
WHERE TOUCH(w.id, ’Oid0006_IfcSpace’)
```

Query 3

```
SELECT col.id
FROM IFCColumn col
WHERE CONTAIN( ’Oid0006_IfcSpace’, col.id)
```

Query 4

```
SELECT elem1.id, elem2.id
FROM IFCElement elem1, IFCElement elem2
WHERE elem1.id<>elem2.id AND OVERLAP(elem1.id, elem2.id)
```

Query 5

```
SELECT elem1.id, elem2.id
FROM IFCElement elem1, IFCElement elem2
WHERE elem1.id<>elem2.id AND EQUAL(elem1.id, elem2.id)
```

Query 6

```
SELECT col.id
FROM IFCColumn col, IFCSlab slab3
WHERE ABOVE_HS_RELAXED(col.id, slab3.id)
AND TOUCH (col.id, slab3.id)
AND slab3.id = ’Oid35219_IfcSlab
```

Algorithm 4: The queries used for real-world performance tests.

max. level	query 1	query 2	query 3	query 4	query 5	query 6
8	162	603	3076	52407	3919	44
9	182	1059	5914	120729	5825	49
10	237	2126	-	-	13266	73

Fig. 23. Performance measurement results for the real-world query tests.

Figure 23 shows the results of the real-world performance tests. Evaluating the results it can be stated that in most cases we achieve an acceptable runtime behaviour up to level 9.

8. Summary

This paper has discussed the topological operators which have been made available in a 3D Spatial Query Language for Building Information Models. It has presented formal definitions of topological predicates using the 9-Intersection method. An octree-based implementation technique of topological operators has been introduced including a detailed discussion of its main advantage – the fuzzy handling of topological relationships. Moreover, a possible realization of the spatial query functionalities on the basis of relational and object-relational database management systems has been described. Finally, we presented detailed runtime measurements for theoretical worst-case / best-case scenarios and a real-world building model example. Although the exponential behavior of the algorithm is not optimal, the real-world example tests have shown acceptable performance.

9. Future research work

Our current efforts focus on developing an alternative implementation of the topological operators, which will be based directly on the boundary representation of both the reference and the target object and will involve traditional computational geometry approaches, such as ray-triangle intersection tests. We hope to use this approach to overcome the exponential behavior of octree-based algorithms.

At the current phase of our project we only store IDs of the objects in the relations of the database. In the future, we intend to store the full set of information available in BIMs in database relations including attributes and relationships which will make it possible to employ such information as conditions in the WHERE part of a SQL query.

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

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

Acknowledgments

The authors gratefully acknowledge the support for the ongoing project provided by the German Research Foundation (DFG) under grant Ra 624/17-1. In addition, the authors would like to thank Stefanie Schraufstetter for her contribution to the implementation of the algorithms prsented here and the team under guidance of Karl-Heinz Häfele at Karlsruhe Institute of Technology for their support regarding IFC-VRML issues.

References

[1] B.-C. Björk, Basic structure of a proposed building product model, Computer-Aided Design 21 (2) (1989) 71–78.

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

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

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

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

[6] 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.

[7] T. Froese, K. Yu, K. Liston, M. Fischer, System architectures for aec interoperability, in: Proc. of the 22nd Conference on Information Technology in Construction, 2000.

[8] A. Kiviniemi, M. Fischer, V. Bazjanac, Integration of multiple product models: Ifc model servers as a potential solution, in: 22nd CIB-W78 Conference on Information Technology in Construction, 2005.

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

[10] C. Schultz, R. Amor, B. Lobb, H. Guesgen, Qualitative design support for engineering and architecture, Advanced Engineering Informatics In Press, Corrected Proof (2008).

[11] 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.

[12] A. Borrmann, Extended formal specifications of 3D spatial data types, Tech. rep., Technische Universität München (2006).

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

[14] A. Borrmann, E. Rank, 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).

[15] A. Borrmann, E. Rank, Specification and implementation of directional operators in a 3D spatial query language for building information models, Advanced Engineering Informatics.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] S. Shekhar, S. Chawla, Spatial Databases: A Tour, Pearson Education, 2003.

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

[25] 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.

[26] 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.

[27] 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.

[28] E. Clementini, P. Di Felice, A comparison of methods for representing topological relationships, Information Sciences - Applications 3 (3) (1995) 149–178.

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

[30] M. Egenhofer, R. Franzosa, Point-set topological spatial relations, Int. Journal of Geographical Information Systems 5 (2) (1991) 161–174.

[31] D. Pullar, Data definition and operators on a spatial data model, in: Proc. of the ACSM-ASPRS Annual Convention, 1988.

[32] M. Egenhofer, J. Herring, Categorizing binary topological relations between regions, lines, and points in geographic databases, Tech. rep., Department of Surveying Engineering, University of Maine (1992).
URL http://www.spatial.maine.edu/ max/9intReport.pdf

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

[34] 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.

[35] M. Schneider, B. Weinrich, An abstract model of three-dimensional spatial data types, in: Proc. of the 12th annual ACM Int. Workshop on Geographic Information Systems (GIS'04), 2004.

[36] A. Borrmann, S. Schraufstetter, C. van Treeck, E. Rank, An octree-based implementation of directional operators in a 3D spatial query language for building information models, in: Proc. of the 24th CIB-W78 Conf. on IT in Construction, 2007.

[37] M. Breunig, A. Cremers, W. Müller, 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.

[38] 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.

[39] 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.

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

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

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

[43] 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.

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

[45] M. Schneider, T. Behr, Topological relationships between complex spatial objects, ACM Transactions on Database Systems 31 (1) (2006) 39–81.

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

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

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

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

[50] 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.

[51] E. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387.

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

[53] A. Eisenberg, J. Melton, SQL:1999, formerly known as SQL3, SIGMOD Rec. 28 (1) (1999) 131–138.

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

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

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

[57] H2 database engine.
URL http://www.h2database.com

[58] IFCStoreyView website.
URL http://www.iai.fzk.de/www-extern/index.php?id=1139

[59] 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/

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

[61] 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.