Boundary Representation-Based Implementation of Spatial BIM Queries

Simon Daum, André Borrmann
Technische Universität München, Germany
simon.daum@tum.de

Abstract. The computer-aided spatial analysis of building models is an important feature for efficient data management and productive downstream processes in Building Information Modelling. Users of building models have to deal with complex data sets. At the same time, it is also important to have a deep understanding of the functionality and design of a building. Topological and directional queries provide a means of gaining insights into the spatial structure of even a large-scale project. To handle the massive amount of data involved in these projects, the previously introduced Spatial Query Language for Building Information Models is extended by a new, more efficient approach for its topological and directional predicates. Instead of the Octree representation of components employed in the past, the new methods operate directly on the boundary representation, i.e. triangulated meshes. In addition, R-Trees are used for a two-stage spatial indexing of the scene. This makes it possible to achieve a substantial improvement in performance for topological and directional predicates of the Spatial Query Language.

1. Efficient Spatial Analysis in large-scale BIM Projects

The application of building information modeling (BIM) and its paradigms is taken for granted in the construction industry nowadays. In large-scale projects in particular, BIM methods lead to better management of the huge amount of data and the processes involved. It is possible to detect and rectify errors in the design and project scheduling at an early stage before the construction work commences. However, undetected mistakes in the planning of a building can lead to difficulties in the downstream process. High-quality results and efficient workflow in the construction phase can only be achieved if the data basis is accurate and functional. Formal spatial analysis can be employed to check the geometric validity of building information models. To this end, (Borrmann & Rank, 2009a, 2009b) devised a Spatial Query Language for Building Information Models. Algorithms based on Octree representations were proposed for implementing the topological and directional operators provided by this language. Due to the nature of the Octree representation and the associated algorithms, however, unsatisfactory processing times had to be taken into account when applied to full-scale building information models.

The aim of this paper is to introduce a new implementation approach designed to overcome these restrictions. Instead of the previously used strategies based on Octrees, the algorithm presented here operates directly on the boundary representation of the operands, thus considerably accelerating the execution of topological and directional predicates. This makes it possible to process complex spatial queries within reasonable runtimes from the viewpoint of an end user.

It is practically impossible to analyze large-scale models entirely manually. The spatial validation that is the subject of this paper is of crucial importance, especially in complex building models with a huge number of components and fine geometrical representations. In keeping with this requirement, the algorithms of the topological and directional predicates make use of a two-stage spatial indexing. At both levels, R-Trees (Guttman, 1984) are used.

This significantly contributes to the short execution times of the spatial predicates and prevents an exponential increase in runtime whenever advanced scenes are examined.

2. Topological Predicates evaluated from Boundary Representation

The Spatial Query Language provides eight predicates for analyzing topological relationships between objects in the three-dimensional modeling space. These predicates correlate two spatial entities and can be described by the 9-Intersection Model (9-IM) introduced in (Egenhofer, 1991). The 9-IM calculus is based on the mathematical theory of Point Set Topology (Gaal, 1964), which applies the notion of the neighborhood of a point to describe topological concepts such as the interior A°, the boundary δA and the exterior A^- of a point set A. The intersection of the interior, boundary and exterior of two entities results in a 3 x 3 matrix. The individual entries indicate whether there is a void or a non-void set for the intersection concerned. Figure 1 shows the 9-IM matrix for a simplified 2D constellation where object A is *inside* object B and object B *contains* object A, respectively.

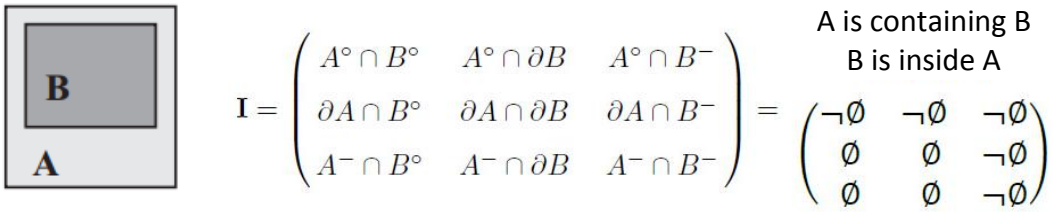


Figure 1: Containing/Inside-of relationship described by the 9-Intersection Model

The 9-IM matrix can be used to define the topological predicates *Disjoint*, *Touching*, *Equal*, *Inside-of*, *Containing*, *Covering*, *Covered-by*, and *Overlapping* in 3D space, as depicted in Figure 2. In contrast to the previously presented Octree algorithms, the B-Rep based algorithms introduced in this paper do not directly populate the 9-IM matrix. Instead, predicates are deduced indirectly by conducting tests on the geometry of the operands.

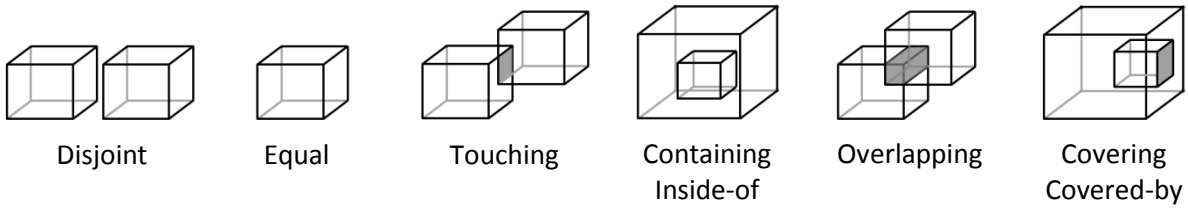


Figure 2: Available topological predicates in 3D (Borrmann & Rank, 2009b)

2.1 General approach to the B-Rep based implementation

The developed methods use triangle meshes of the model's components as input. Objects containing multiple shells and nested caves are not considered here. Nevertheless, non-convex objects, such as walls with openings can be used as input geometry. The following section describes the topological algorithms, which are based on the triangle intersection test, as presented in (Akenine-Möller, 1997). This test is extended in such a way that it distinguishes between intersecting and touching triangles. Moreover, the said algorithm also indicates whether triangles are co-planar.

2.2 Ray-based Inside/Outside Classification

Except for *Equal* and *Overlapping*, all topological predicates might require an inside/outside classification of the objects under investigation. This is realized by means of ray tests (Schraufstetter, 2007): an arbitrary triangle is chosen from the object's mesh and used as a starting-point for rays. Rays are emitted from the chosen triangle and the number of intersections with the mesh of the other operand mesh is counted. If an even number of intersections occurs, the triangle is located outside the mesh, whereas an odd number indicates that the triangle is inside the mesh. To simplify the pre-selection of potentially intersected triangles, the rays are always drawn parallel to axes of the coordinate system. To guarantee numerical stability wherever glancing intersections appear, several rays are used for one test in the practical implementation. In addition, the maximal component of the triangle's normal provides the most appropriate direction for the axis for testing purposes. Figure 3 depicts the principle for the classification of components based on the ray test.

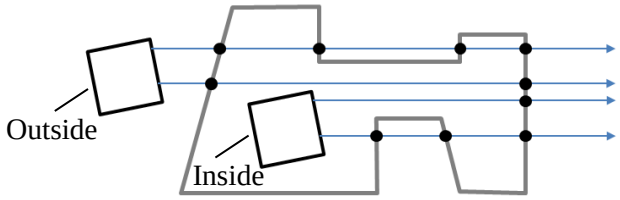
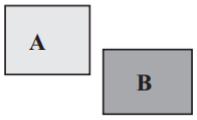


Figure 3: Principle for the classification of components based on ray tests

The following sections present the algorithms that have been developed to enable the processing of topological predicates using 3D boundary representations. They are based on the verification and falsification of entries in the 9-Intersection Model.

2.3 Disjoint



$$\begin{bmatrix} \varnothing & \varnothing & \neg\varnothing \\ \varnothing & \varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \end{bmatrix}$$

A is disjoint from B
B is disjoint from A

[Intersection between the interior of A and the interior of B disproves the disjoint relation]
Search for any intersecting triangle between A and B. If any occurs, return false.

[Intersection between the boundary of A and the boundary of B disproves the disjoint relation]
Search for any touching triangle between A and B. If any occurs, return false.

[A can be inside B and vice versa or disjoint]

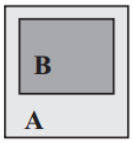
[A inside B disproves the disjoint relation]
Pick an arbitrary triangle of object A. Deduce the most suitable axis for ray testing by the triangle's normal. Draw a ray in the positive direction of the chosen axis of the coordinate systems. Use a location on the surface of the triangle as the starting-point.

[A is inside B]
If an odd number of intersections are counted, A is inside B. Return false.

[A outside B verifies the disjoint relation]
If an even number of intersections is identified, A is outside B. Return true.

[Switch objects over if zero intersections appear]
If this is the second execution, return true. Otherwise, switch the objects over so that A becomes B and vice versa. Repeat execution at [A inside B disproves the disjoint relation]

2.4 Containing



$$\begin{bmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{bmatrix}$$

A is containing B
B is inside A

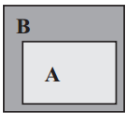
[Intersection between the interior of A and the interior of B disproves the containing relation]
Search for any intersecting triangle between A and B. If any occurs, return false.

[Intersection between the boundary of A and the boundary of B disproves the containing relation]
Search for any touching triangle between A and B. If any occurs, return false.

[B inside A verifies the containing relation]
Pick an arbitrary triangle of B. Draw a ray from the triangle, as described before.

[B is inside A]
If an odd number of intersections are counted, B is inside A. Return true, otherwise false.

2.5 Inside-of



$$\begin{bmatrix} \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$$

A is inside B
B is containing A

Use Containing algorithm with A and B switched over

2.6 Equal

$$\begin{bmatrix} \neg\emptyset & \emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{bmatrix}$$

A is equal B
B is equal A

[Equals can only be true if the surface areas of A and B are identical]

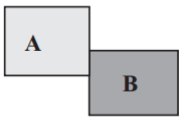
Add up all triangle surfaces of A. Do the same with B. If the values vary from one another, return false.

[Each triangle from A must be covered by triangles from B]
Iterate of all triangles in A. For each A triangle take only triangles from B which are located in its neighbourhood. This is realized by the R-Tree indexing of meshes described in X. If a triangle candidate is co-planar and overlay an A triangle, tag it.

[Each triangle from B must be covered by triangles from A]
Switch objects A and B over and repeat execution from [Each triangle from A must be covered by triangles from B]

[All triangles from A are covered by B triangles and vice versa]
If all triangles are tagged, return true. If untagged triangles exist, return false.

2.7 Touching



$$\begin{bmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$$

A is touching B
B is touching A

[Intersection between the interior of A and the interior of B disproves the touching relation]
Search for any intersecting triangle between A and B. If any occurs, return false.

[Intersection between the boundary of A and the boundary of B is mandatory to verify the touching relation]
Search for any touching triangles between A and B. If zero triangles touch, return false.

[A touches B or A is covering B or A is covered-by B]

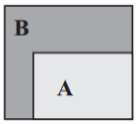
[A inside B disproves the touching relation]
Pick an arbitrary triangle of A but avoid triangles classified as touching. Draw a ray from the triangle, as described before.

[A is inside B]
If an odd number of intersections are counted, A is inside B. Return false.

[A outside B verifies the touching relation]
If an even number of intersections is identified, A is outside B. Return true.

[Interchange objects if zero intersections appear]
If this is the second execution, return true. Otherwise, switch the objects over so that A becomes B and vice versa. Repeat execution at [A inside B disproves the touching relation]

2.8 Covering



$$\begin{bmatrix} \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$$

A is covering B
B is covered-by A

[Intersection between the exterior of A and the exterior of B disproves the covering relation]
Search for any intersecting triangle between A and B. If any occurs, return false.

[Intersection between the boundary of A and the boundary of B is mandatory to verify the covering relation]
Search for any touching triangles between A and B. If zero triangles touch, return false.

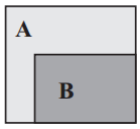
[A meets B or A covers B or A is covered-by B]

[If A covers B, parts of B can be classified as inside A]
Pick an arbitrary triangle of B but avoid triangles classified as touching. Draw a ray from the triangle, as described before.

[A inside B verifies the covering relation]
If an odd number of intersections are counted, A is inside B. Return true.

[B outside A disproves the covering relation]
If an even number of intersections is identified, B is outside A. Return false.

2.9 Covered-by

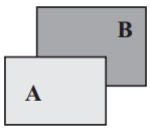


$$\begin{bmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{bmatrix}$$

A is covered-by B
B is covering A

Use Covering algorithm with A and B switched over

2.10 Overlapping



$$\begin{bmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$$

A is overlapping B
B is overlapping A

[Intersection between interior of A and the interior of B verifies the overlapping relation]
Search for any intersecting triangle between A and B. If any occurs, return true.

[A does not overlap B]
Return false.

3. Directional Predicates evaluated from Boundary Representation

The directional predicates use the definitions of the projection-based operators, presented in (Borrmann & Rank, 2009a). Predicates available are *eastOf, westOf, northOf, southOf, below* and as shown in Figure 4 *above*. There is a relaxed and a strict version of every directional predicate. In the relaxed version, the examined object only needs to partly intersect the projection volume. The strict version requires that the whole object is contained in the projection volume. So, the objects B and C in Figure 4 are *above* object A if the relaxed operator is used. By contrast, the strict operator only returns object B as *above* A.

As well as the topological predicates, the directional predicates considerably facilitate the spatial indexing of the established R-Trees. The integration of R-Trees in the directional and topological predicates is discussed in Section 4.

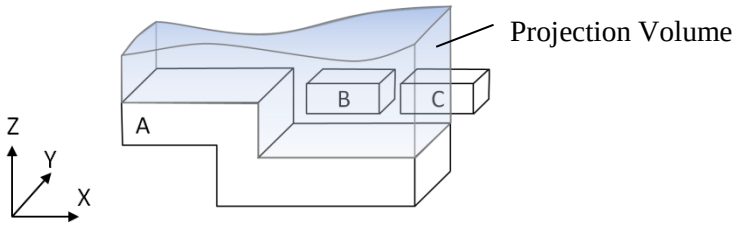


Figure 4: Projection-based principle of directional predicates *(above)*

At the level of boundary-based computation, it is necessary to verify whether two triangles are in a specific directional arrangement, such as one lying above the other. To test this, the triangles are projected on to the XY plan, as shown in Figure 5. If the projected triangles do not intersect, they are directionally disjoint and not suitable for classification. Otherwise, we can look at the plane defined by one of the triangles. An arbitrary point of the second triangle is used to measure its distance to the plane. The plane's normal and the computed distance can be used to deduce whether the second triangle lies above or below the plane.As the first test has already verified that the projected triangles do overlay, the above or below attribute also holds true for the two triangles.

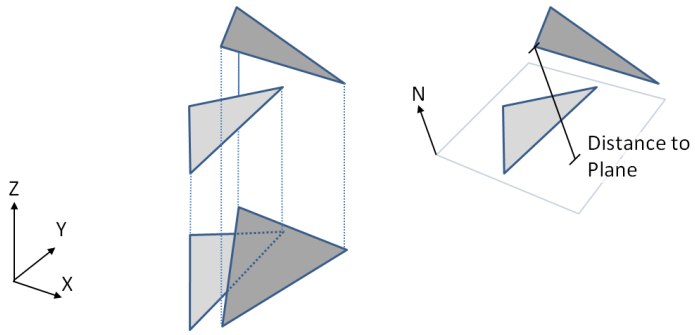


Figure 5: Method to deduce the directional arrangement of two triangles

The algorithms of the directional predicates using 3D boundary representation are given below by way of an example for the *above* case.

3.1 Above (Relaxed)

[Select all candidate objects above A]
Take the bounding box of A. Extend its maximal Z value to the maximal Z value of the scene-bounding box. Select all objects intersected by, or contained in this extended box (use global R-Tree).

Foreach object

[An intersection verifies the above relation]
If overlapping occurs, return true.

[Select all B triangles above A]
Take the bounding box of A. Extend its maximal Z value to the maximal Z value of the B bounding box. Select all B triangles intersected by, or contained in this extended box (use local R-Tree of B).

[For each triangle above B]
Take the bounding box of the triangle. Extend its minimal Z value to the minimal Z value of the A bounding box. Select all A triangles intersected by, or contained in this extended box (use local R-Tree of A).

[For each triangle of A]
Use the aforementioned triangle test to deduce the directional arrangement of two triangles.
If B triangle is verified as above, return true.

[B is not above A]
No triangle is found above. Return false.

End foreach

3.2 Above (Strict)

[Select all candidate objects above A]
Take the bounding box of A. Extend its maximal Z value to the maximal Z value of the scene-bounding box. Select all objects contained in this extended box (use global R-Tree).

Foreach object

[An intersection disproves the above relation]
If overlapping occurs, return false.

[For each triangle of B]
Take the bounding box of the triangle. Extend its minimal Z value to the minimal Z value of the A bounding box. Select all A triangles intersected by, or contained in this extended box (use local R-Tree of A).

[For each triangle of A]
Use the aaforementioned triangle test to deduce the directional arrangement of two triangles.
If a B triangle is verified as below, return false.

[B is above A]
All B triangles are verified as above. Return true.

End foreach

4. Integration of R-Trees in the Directional and Topological Predicates

The R-Tree (Guttman, 1984) is a hierarchical structure that makes it possible to index multidimensional data. The main objective of the R-Tree is to group items located in close vicinity. In the described context, the indexed items are 3D axis-aligned bounding boxes. The tree is constructed from nodes which can contain other nodes or the originally inserted bounding boxes. If a node contains boxes, it must be a leaf node. These boxes refer to the data objects from which they were computed, such as triangle meshes. All leafs are located at the same level of the tree. The tree can be fine-tuned by its two parameters m and M. They reflect the minimal and maximal number of children in each node except for the root, where $m \leq M/2$. Each node determines its own bounding box which encapsulates all children. If new data objects are inserted, the tree is traversed downwards. At each node, we select the child which needs the least enlargement for the new item, finally adding the object to a leaf node. If this node overflows, it is divided into two new nodes. The new nodes are inserted in the tree and, if overflows propagate to the root, the tree gains in depth. In this manner, we succeed in constructing a bounding box hierarchy. The tree can efficiently return data objects contained in, or intersected by query boxes. This is used extensively in the developed algorithms for the topological and directional predicates. For more details about the R-Tree data structure, see also (Samet, 1994).

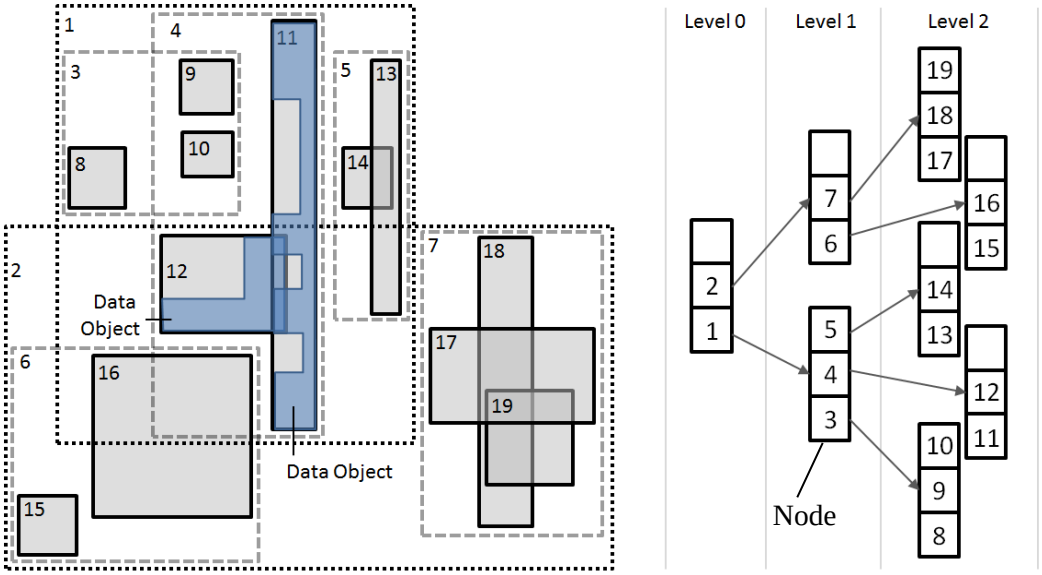


Figure 6: Example of the R-Tree structure calculated from 2D data, based on (Guttman, 1984)

In the developed concept, the global R-Tree is filled with the bounding boxes of the meshes in the scene. This enables the system to rapidly find candidates for the various spatial queries, thus excluding a large number of meshes from any further processing, while the system remains responsive even in complex scenarios. For example, rectangle R12 in Figure 6 is examined for overlaps. The tree is called on to return all objects intersecting with R12 in the form of a query box. Only R11 is returned and provides a data object - a polygon in this case - for further testing with the R12 polygon.

The local spatial indexing connects an R-Tree to every component's mesh in the scene. The mesh provides the tree with its triangles by way of data objects. A bounding box is calculated for every triangle as it is inserted into the tree, thus establishing a box hierarchy for the

triangle mesh. The local trees allow querying for triangle candidates in all six axis directions of the global coordinate system. The algorithms of the proposed spatial predicates make use of this functionality, which is realized by extending bounding boxes to minimal or maximal values of other boxes. These extended boxes are subsequently used to search for triangle candidates, as depicted in Figure 7.

In addition, second-stage indexing allows for an efficient identification of intersecting triangles between two meshes using a divide-and-conquer strategy (Gottschalk, 2000). Firstly, node candidates are identified by a parallel breadth-first search in both R-trees. At each search level, nodes from the first tree are tested against nodes in the second tree. If two nodes intersect, the second node is placed on the first node's candidate list. In the next level of execution, the first node is replaced by its children. Following that, all candidates are replaced by their children. The new nodes are now tested for intersection and new candidate lists are produced. If a node is a leaf, it is retained for the next level of node refinement until, ultimately, the main node and its candidates are all leaf nodes. At this step, the tree has performed the desired pre-selection of candidates and only a few, more intricate tests have to be computed on the explicit triangle geometry.

The topological predicates *Touching*, *Covering* and *Covered-by* require a classification of touching triangles. If we are to look for triangles of this description, it is necessary to slightly modify the algorithm: nodes in the tandem tree traversal which touch the main node are also saved in the candidate list.

For the identification of triangles for ray testing, the R-Tree is again facilitated. In the example depicted in Figure 7, positive X is used as a ray direction. The triangle's bounding box is consequently extended to the maximal X value of the bounding box owned by the second mesh. The extended box acts as query input for an intersection search in the second R-Tree. By this means, we are able to retrieve the triangle candidates for ray testing. Finally, the ray is drawn from the main triangle to obtain the number of intersections. Because of the odd value of intersection in Figure 7, the triangle can be classified as being inside the second object.

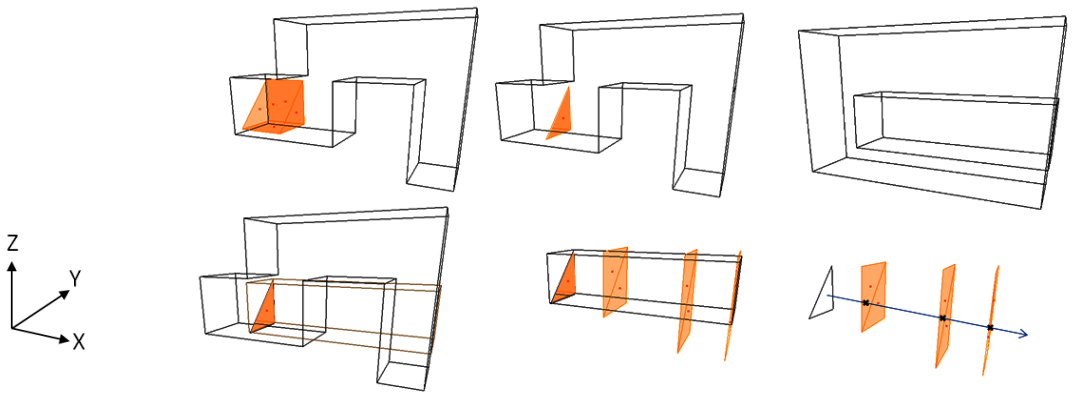


Figure 7: Inside/Outside test facilitating the second stage,
local R-Tree data structure of components

5. Conclusion

This contribution describes new boundary representation-based methods for the verification of topological and directional predicates as part of the Spatial Query Language for Building Information Models. It introduces a mapping between the 9-Intersection Model and the boundary-based examination for the topological predicates. In order to deal with the computational complexity arising in large-scale models efficiently, we combine two-stage spatial indexing with elaborated geometrical algorithms. Both the creation of R-Trees and the actual runtime of spatial predicates outperform the previously used Octree approach. The next step will be to setup a test environment in which the Octree and R-Tree algorithms are examined and compared by varying input data. Furthermore, the desired fuzziness of the topological predicates (see Borrmann and Rank, 2009b) will be incorporated into the concept.

References

Akenine-Möller, T. (1997). A Fast Triangle-Triangle Intersection Test. Journal of Graphical Tools, 2(2), pp. 25-30.

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

Egenhofer, M., Herring, J., (1989). A mathematical framework for the definition of topological relationships. In: Proc. of the 4th Int. Symp. on Spatial Data Handling.

Gaal, S. A. (1964). Point set topology (Vol. 16). Academic Press.

Gottschalk, S. (2000). Collision queries using oriented bounding boxes. Doctoral dissertation, The University of North Carolina, pp 30-33.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on Management of data (SIGMOD '84). ACM, New York, NY, USA.

Samet, H. (1990). The design and analysis of spatial data structures. Addison-Wesley.

Schraufstetter, S., Borrmann, A. (2007). AABB-Bäume als Grundlage effizienter Algorithmen für Operatoren einer räumlichen Anfragesprache. In: Proceedings of Forum Bauinformatik.