Advanced Engineering Informatics 28 (2014) 272–286

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Processing of Topological BIM Queries using Boundary Representation Based Methods ☆

Simon Daum *, André Borrmann

Chair of Computational Modeling and Simulation, Leonhard Obermeyer Center, Technische Universität München, Arcisstr. 21, 80333 München, Germany

ARTICLE INFO

Article history:
Received 22 October 2013
Received in revised form 12 March 2014
Accepted 10 June 2014
Available online 25 July 2014

Keywords:
Building information modeling
3D spatial query language
Topology
Boundary representation
QL4BIM

ABSTRACT

Building Information Models (BIM) are comprehensive digital representations of buildings, which provide a large set of information originating from the different disciplines involved in the design, construction and operation processes. Moreover, accessing the data needed for a specific downstream application scenario is a challenging task in large-scale BIM projects. Several researchers recently proposed using formal query languages for specifying the desired information in a concise, well-defined manner. One of the main limitations of the languages introduced so far, however, is the inadequate treatment of geometric information. This is a significant drawback, as buildings are inherently spatial objects and qualitative spatial relationships accordingly play an important role in the analysis and verification of building models. In addition, the filters needed in specific data exchange scenarios for selecting the information required can be built by spatial objects and their relations. The lack of spatial functionality in BIM query languages is filled by the Query Language for Building Information Models (QL4BIM) which provides metric, directional and topological operators for defining filter expressions with qualitative spatial semantics. This paper focuses on the topological operators provided by the language. In particular, it presents a new implementation method based on the boundary representation of the operands which outperforms the previously presented octree-based approaches. The paper discusses the developed algorithms in detail and presents extensive performance tests.

© 2014 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).

1. Introduction

Building Information Models are comprehensive digital representations of buildings. Their use for designing, engineering, constructing and operating buildings is increasingly being adopted by the AEC industry [1]. A BIM comprises not only the 3D shape of all building elements and the enclosed spaces, but also semantic information representing both the type of the elements and the relationships between them. Additionally, the model stores a wealth of non-geometric properties. By providing the appropriate data structures, a BIM serves as an interface and database for exchanging information across the disciplines involved throughout the design, construction and operation of the building. The use of BIM in the interdisciplinary design and engineering processes calls for well-defined points of information hand-over. The Information Delivery Manual (IDM) provides a framework for describing the processes and defining exchange requirements [2]. It precisely specifies the information needs of the receiving participant and/or application. The exchanged information has to satisfy high quality standards, as an efficient operation of downstream processes can only be achieved if the data basis is accurate.

Coupled with 3D modeling, the semantic comprehensiveness of building models provides a clearer insight into the consequences of planning decisions than the 2D plan-based approach. Nevertheless, the specific conditions in the construction industry and the unique characteristics of each building can spoil the quality of the resulting model:

1. Projects are developed during long periods of design and planning phases, often including significant changes to the building model(s).
2. There are a large number of contributors involved in the ongoing detailing of a model.
3. Each domain focuses on different parts of the model.

These conditions, coupled with the complexity of the modeled environment, lead to a massive amount of data and a high risk of undetected errors. In large-scale projects in particular, the amount and complexity of information stored in a BIM renders the manual

☆ Handled by W.O. O'Brien.

* Corresponding author. Tel.: +49 89 289 25065.
E-mail addresses: simon.daum@tum.de (S. Daum), andre.borrmann@tum.de (A. Borrmann).

http://dx.doi.org/10.1016/j.aei.2014.06.001
1474-0346/© 2014 The Authors. Published by Elsevier Ltd.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).

detection of spatial defects difficult and time-consuming. In addition, retrieving the data subset required for a specific exchange scenario is a far from trivial task. To overcome this hurdle, several researchers have proposed using formal query languages which make it possible to specify the desired information in a concise, well-defined manner. These approaches are presented in Section 2. One of the main limitations of the methods proposed so far, however, is the inadequate treatment of geometric information. None of the approaches support the use of qualitative spatial predicates for defining filter expressions, only allowing simple numeric comparisons between individual geometry coordinates. This is not suitable for expressing filters with higher spatial semantics.

This has to be seen as a major deficiency, as spatial relations between building elements play a significant role in most of the design and engineering tasks of the AEC domain. To fill this technological gap, the authors have developed concepts and techniques for a spatial query language for Building Information Models [3–5]. The devised technology makes it possible to select specific building elements by applying qualitative spatial relationships within the respective filter expressions. These relationships form an intermediate level of abstraction between the technical view on building geometry using numerical coordinates, and the way humans reason about buildings and the relations between their components.

Typical examples of queries concerned with spatial semantics are:

- Get all walls *within* the first storey.
- Does room 107 *contain* any heating equipment?
- Get all objects *within the distance* of 10 m from a certain point.
- Which columns *touch* slab No. 1?
- Are there any gas lines *below* the footing?

Possible applications for a spatial query language for BIM range from verifying construction rules to extracting partial models that satisfy spatial conditions. Such a partial model resulting from a spatial query can be used to fulfill the exchange requirements specified in the IDM process map, for example.

The proposed 3D spatial query language QL4BIM relies on a spatial algebra that is formally defined by means of the point-set theory and point-set topology [6]. The operators available for the spatial types are the most important part of the algebra.

They consist of

- metric operators, such as *Distance*, *CloserThan*, and *FartherThan* [4]
- directional operators, such as *Above*, *Below* and *NorthOf* [3] and
- topological operators, such as *Touch*, *Within* and *Contains* [7].

This paper focuses on the topological operators. In particular, it presents a novel implementation approach which is based on the direct use of the Boundary Representation (BRep) of the operands. As shown by the performed performance tests, this newly developed technique outperforms the octree-based approaches presented in [8].

The paper is structured as follows: Section 2 discusses related work in the field of query languages for BIM as well as spatial data processing in the domain of Geographic Information Systems (GIS). In addition, the standardized data model Industry Foundation Classes and its handling of BRep data is discussed. Section 3 presents the formal definitions underlying the topological operators provided by QL4BIM. After revising the octree-based approach for implementing the topological operators in Section 4, the developed BRep-based approach is presented in Section 5. Section 6 discusses the application of the R-Tree indexing method for achieving further performance gains during topological filtering. The results of test runs concerning runtime and time complexity are presented in Section 7, while Section 8 discusses the Live LINQ query system, which includes the proposed topological filters. The paper is concluded by a summary and discussion.

2. Related work

This section is about query functionality in the domain of Building Information Modeling and Geographic Information Systems. It also illustrates the handling of BRep geometry in the underlying data model used in the QL4BIM system.

2.1. Industry Foundation Classes

The query language QL4BIM operates on BIM models complying with the Industry Foundation Classes (IFC), a vendor-neutral, open and standardized data model for the representation of complex building models. The IFC are based on STEP, the ISO standard for the exchange of product model data [9]. STEP includes the specification of the data modeling language EXPRESS, which is employed for defining the schema of the IFC.

This treatise deals with the realization of efficient algorithms for processing topological predicates to filter IFC building models. The presented algorithms operate on the geometric representation of building elements and space objects. The IFC provide a large variety of different geometry representations, including Constructive Solid Geometry, Swept Area Representations and boundary representations. For the latter, different options exist of which the most important are:

- *IfcFacetedBRep:* A simple form of boundary representation model in which all faces are planar and all edges are straight lines. Edges and vertices are not represented explicitly in the model but are implicitly available.
- *IfcAdvancedBRep:* A boundary representation model in which all faces, edges and vertices are explicitly represented. It is a solid with explicit topology and elementary or free-form geometry.
- *IfcTriangulatedFaceSet:* A tessellated face set with all faces being bound by triangles.

The developed algorithms are based on the *IfcTriangulatedFaceSet* representation. The data structure is very similar to that underlying the indexed face set of the X3D standard [10].

In the prototypical implementation of the system we are employing the library IFC LOADER provided by the IFC Tools Project [11] to perform a pre-processing step which generates a triangulated surface representation for each individual object. This representation is stored using *IfcTriangulatedFaceSet* as an additional geometry representation of the respective semantic object.

2.2. BIM query languages

The handling of data sets is covered by databases and their query languages. The Structured Query Language (SQL) is the de facto standard in this domain. Relational algebra provide the theoretical foundation for this language [12]. SQL can be used to examine structured data represented by relations. This requires static and homogenous relationships between data items. By contrast, the IFC data model allows a very flexible build-up of interposing entity relationships. This makes a mapping in line with relational database technology difficult [13].

The process of filtering can be accomplished on the level of the meta-model (EXPRESS) or directly on the domain model (IFC). A domain specific language for BIM can provide a more direct access to the required data, as it is tailored to the domain model and hides

the details of the internal data representation [14]. The following section discusses currently available domain-specific languages for the filtering of building models.

The Partial Model Query Language (PMQL) is an XML-based language offering *Select*, *Update* and *Delete* operators [15]. It is integrated in the IFC Model Server and can be called up via SOAP-based web services. A user-defined filtering is established as a *Select* statement, which can comprise nested *Selects*. The final set of results can be restricted by *Where* statements, which check against object IDs, object types and generic SQL statements.

The Generalised Model Subset Definition Schema (GMSD) make it possible to define partial, domain-specific model views in a declarative manner as well as providing selection functionality at the instance level [16]. Weise provides a use case, where objects are firstly filtered at the instance level, followed by a second processing based on the defined view definition. An ad-hoc query in GMSD is stated by means of a *QuerySetSelect* entity, which can contain statements for filtering according to attribute and type values. Operations like *Union*, *Difference* and *Intersection* can also be performed on the result sets of sub queries.

The Building Information Model Query Language (BIMQL) is a domain-specific query language for building models [14]. It currently provides a *Read* and *Update* functionality, to which *Create* and *Delete* statements are to be added in the future. The language is inspired by PMQL and SPARQL. The latter is a graph-based query language in the context of the semantic web [17]. The language is especially suitable for the traversal of complex referential data models like the one provided by the IFC.

The strength of the presented languages lies in the semantic processing of the underlying data set. However, none of the languages take the geometric representation of IFC objects for the filtering of data into account. In QL4BIM, the IFC geometry is transformed into explicit BRep data, i.e. *IfcTriangulatedFaceSets*. This geometry representation is subsequently used by the provided spatial predicates.

2.3. Spatial query functionality provided by Geographic Information Systems

The overall concept of providing a spatial query language for analyzing BIMs is closely related to concepts and technologies developed in the area Geographic Information Systems (GIS). Such systems maintain geographical data of the environment, such as the positions and shapes of cities, streets, rivers, etc. and provide functionalities for the spatial analysis of this data. Due to the nature of this domain, most systems only support spatial objects in 2D space.

It was in the late 1980s when the first implementations of a spatial query language on the basis of SQL were realized. A multitude of different dialects were developed, including PSQL [18], Spatial SQL [19], GEOQL [20], KGIS [21] and TIGRIS [22]. Egenhofer [23] provides a good overview of the different dialects and the basic advantages of a SQL-based implementation. The Open Geospatial Consortium (OGC) publishes the Simple Feature Access standard, which includes a detailed description of spatial extensions for SQL [24].

The majority of Geographic Information Systems available support an extended spatial SQL, including ArcGIS [25] and PostGIS [26]. As these systems evolved historically from a 2D geometric representation approach, however, the functionality for processing 3D geometry is very restricted and the performance of the available tools is unsuitable for complex building data sets. The ArcGIS documentation [25], for instance, states: "Some 3D set operators are fairly process intensive and may take longer to execute. [...] As a result, care should be taken when deciding what feature classes to use as inputs to these tools, regarding size and complexity of data." In contrast, the QL4BIM system is design to handle queries against elaborated Building Information Models containing a high amount of elements and detailed geometry representations.

3. Formal definition of topological predicates

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 are invariant under translation, rotation and scaling as well as any combination of these transformations – except when a scaling factor of zero is used. They are qualitative in nature and provide a high level of abstraction which complies well with the way humans reason about buildings and the relationships between their elements. The authors consider the integration of topological operators in a BIM query language as an important functionality required for an enhanced, powerful analysis and verification.

Different mathematical systems have been presented for providing a formal definition for topological predicates. One main contribution in this field of research was the introduction of the 4-Intersection Model (4-IM, [27,28]). The 4-IM calculus is based on the mathematical theories of algebraic topology and set topology [29]. It applies the notion of the neighborhood of a point to describe topological concepts such as the interior A° and the boundary δA of a point set A. Topological predicates are defined by the intersection of the interior and the boundary of two operands. In this case, an intersection can yield an empty Ø or a non-empty set ¬Ø. These values represent the topological invariants of the topological relations, reflecting that they stay constant under transformations. The formal definitions of the topological predicates are depicted in Table 1.

By taking the exterior of an entity into account, the 4-IM is extended to become the 9-Intersection Model (9-IM). The $^-$ token indicates the exterior in the model. This extension makes the handling of line–line relations in R^2 possible. The 9-IM is notated as a 3×3 matrix, as depicted in Fig. 1.

The 9-IM can represent $2^9 = 512$ possible configurations, but only eight are encountered when two solids are examined. Borrmann and Rank [30] examined the use of the 9-IM as a formal basis for defining topological operators in an BIM environment, including cases of solid-to-solid relations. As solid objects are the major representation of spatial entities in BIM, this paper restricts itself to contemplate the topological configuration of solids only. The eight possible topological relations between two solids are *Disjoint*, *Inside*, *Equal*, *Touching*, *Containing*, *Overlapping*, *Covering*, and *CoveredBy*, shown in Fig. 2, which also depicts the 9-IM matrix corresponding to each topological configuration.

Table 1
The definitions of the topological predicates by the 4-IM.

∂A ∩ ∂B	A° ∩ B°	∂A ∩ B°	A° ∩ ∂B	
Ø	Ø	Ø	Ø	Disjoint
¬Ø	Ø	Ø	Ø	Touching
¬Ø	¬Ø	Ø	Ø	Equal
Ø	¬Ø	¬Ø	Ø	Inside
¬Ø	¬Ø	¬Ø	Ø	CoveredBy
Ø	¬Ø	Ø	¬Ø	Containing
¬Ø	¬Ø	Ø	¬Ø	Covering
¬Ø	¬Ø	¬Ø	¬Ø	Overlapping

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

Fig. 1. The matrix representation of the 9-Intersection Model.

The 9-IM does not provide the possibility to distinguish between the dimensionality of the intersection sets. Accordingly, the different touch constellations (vertex–vertex, edge–edge, face–face) cannot be distinguished. To overcome this issue, Clementini et al. developed the Dimensionally Extended 9 Intersection Method (DE-9IM), where the dimensionality of the intersection sets is recorded in the matrix [31]. Though the DE-9IM provides extended expressivity, the algorithms presented in this study are based on the 9-IM, as the covered set of topological operators is sufficient for most practical use cases.

4. Octree-based implementation approach

Borrmann and Rank [5] introduced an approach for implementing topological operators based on an octree representation. The octree is a hierarchical structure based on cell decomposition [32]. The geometry is decomposed with equally sized cubic cells, also called octants. Every octant has exactly one parent cell and either zero or eight child cells. Octants that are wholly inside or outside the represented object hull have no child cells. The ratio of the edge length of a child cell to its parental cell is always 1:2. The principle functioning of this tree data structure is illustrated by its 2D variant, the quadtree in Fig. 3.

In order to process a topological operator, the geometry of its operands (BIM elements) has to be converted into an octree representation. The level of refinement is controlled by a parameter steering the tree construction. As the geometry input is a solid represented by a *IfcTriangulatedFaceSet*, octant-triangle intersection tests can be used to create the tree in a top-down manner [33]. To classify interior and exterior cells, a flooding method is employed [8,34]. The result is a discretized representation of a building element made up of dedicated interior, boundary and exterior cells. It is important to distinguish this approach using the octree structure of a building element as an additional geometry representation from using an octree as a spatial indexing structure. To evaluate a topological predicate for a pair of geometric objects, the octree-based algorithm comprises the followings steps: (1) transform each of the boundary representations of the involved objects into a corresponding octree, (2) create a 9-IM matrix by superimposing the octrees and (3) determine the applicable topological predicate by comparing the created 9-IM matrix with the definition templates (Fig. 2).

Since the octree structure acts as a secondary geometry representation for BIM elements, their refinement level must be adjusted to reflect the desired resolution. This is problematic, as the creation and processing of the octree shows an exponential time complexity concerning this refinement level [5]. This approach is accordingly not applicable for the topological analysis of large data sets, as produced in BIM. The performance analysis in Section 7 depictures this. On the other hand, an efficient formal analysis technology including the topological exploration of building elements is advantageous, especially in elaborated building models. This extends to the efficient extraction of submodels. The next section presents the new approach for the calculation of topological predicates. It is suitable for use in building models with large number of elements and fine-grained geometrical representations.

5. BRep-based implementation method for topological operators

To overcome the limitations of the octree-based implementation method, particularly with respect to the runtime performance, a new implementation method based on the boundary representation of the operands has been developed. The method is generic and allows determining the topological predicate for any two geometric objects. These objects may be non-convex (such as a wall with openings) and may possess caves. However, to simplify the presentation in this treatise and without loss of generality, we do not discuss objects having multiple shells in the following explanations.

The presented algorithms are operating on a specific boundary representation, a triangulated surface mesh. As stated above, we make use of the IFC LOADER library of the IFC Tools Project [11]

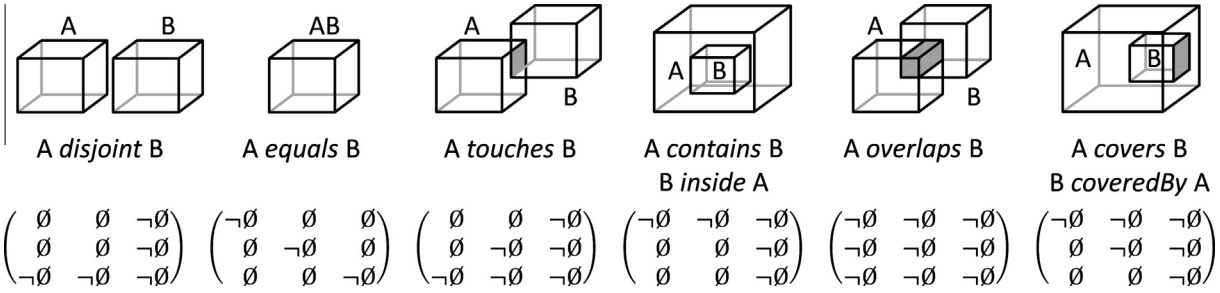


Fig. 2. The eight predicates defining the possible topological relationships between two solids in 3D.

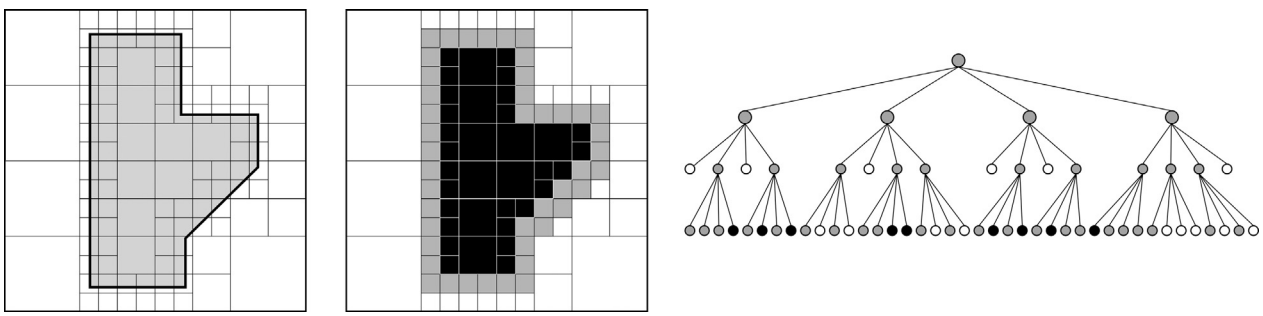


Fig. 3. A non-colored quadtree and a three-colored quadtree with its graph structure. White cells represent the exterior, black cells the interior and gray cells the boundary of the discretized object.

to generate an *IfcTriangulatedFaceSet* representation for each *IfcProduct* entity having a 3D shape. This including entities with curved surfaces. The developed algorithms require the operands to be manifold geometric objects. The original geometry representation (*IfcSweptAreaSolid*, *IfcAdvancedBrep* etc.) is preserved and may be used for downstream applications following the selection performed by means of the query language.

In contrast to the octree-based algorithms, the BRep-based algorithms do not directly populate a complete 9-IM matrix. This is due to the fact that the BRep data lacks the explicit representation of the interior and the exterior of a geometric object. By conducting further investigations into triangle-to-triangle constellations, however, it is possible to deduce the absent values in the 9-IM. Here, it is important to distinguish clearly between intersecting and meeting constellations. A meeting situation can occur if there are adjacent vertices or edges between pairs of triangles. As 3D geometry is used in the context in question, we consider a coplanar overlay of two triangles to be meeting rather than intersecting.

Main parts of the algorithms make use of a triangle-to-triangle intersection test and a ray-based inside/outside classification. We begin by describing these two tests before discussing the actual algorithms.

5.1. Triangle intersection test

For the triangle-to-triangle intersection test, we make use of the approach presented in [35]. In the description given here, we call the triangles T_1 and T_2 and their corresponding planes π_1 and π_2 (Fig. 4, left). First, we calculate the signed distances between all T_1's vertices to plane π_2. If all distances have the same sign, the triangle does not intersect the plane and the test immediately returns false. The test is repeated for the second triangle. If these two early rejection tests are passed, we proceed to calculate the line L of intersection between plane π_1 and π_2. The previous calculations guarantee that each triangle covers an interval on L. The triangles only intersect if these two intervals overlap. The original test is extended in a way that it distinguishes between touching and intersecting triangles (Fig. 4, right). Touching triangle pairs are necessary to verify the *Equal*, the *Touching* and the *Covering* relationship.

In the case of coplanar triangles, they are projected on to the most appropriate axis-aligned plane, where the areas of the triangles are maximized, before we proceed to a two-dimensional triangle-to-triangle overlap test [36].

5.2. Ray-based inside classification procedure

Due to the lack of an explicit representation of the interior and the exterior in the BRep data structure, we are obliged to use alternative methods to determine all intersection sets required by the 9-IM, such as the intersection of the exterior and the boundary.

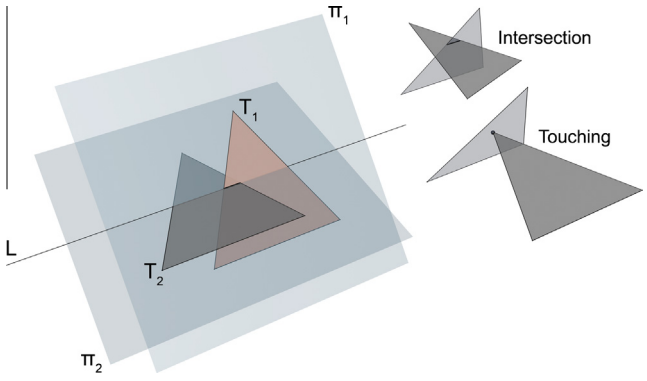


Fig. 4. Triangle intersection test (left), Intersection and Touching constellations (right).

The used inside test is realized by means of ray tests [4,37]. An arbitrary triangle is chosen from the entity's BRep and used as a starting-point for rays. The rays are emitted from the selected triangle and the number of intersections with the second operand's BRep is counted. If an even number of intersections occurs, the triangle is located outside, whereas an odd number indicates that the triangle is inside the *IfcTriangulatedFaceSet*. To determine the most appropriate axis direction for testing purpose, the maximal component of the triangle's normal vector is examined. Fig. 5 illustrates the described principle in 2D.

As the method can fail in special cases, additional measures have to be taken to ensure its reliability. In Fig. 5, entityA_1 is classified correctly, while we observe a glancing intersection for a ray emitted by A_2. This would lead to an erroneous examination of intersection counts and thus to a wrong result of the inside/outside test. To overcome this fragility, we use several rays in one test as shown for A_2' in Fig. 5 (direction $+X$). In consequence, the inside/outside test considers the ray with the highest number of intersections.

In case of ambiguities, another triangle can be chosen that exhibits a different direction for ray emission. This is shown for the object A_2', which emits rays in the $-Y$ direction instead of $+X$, as tried before. The application of these approaches can be fine-tuned by the end user by means of two parameters provided by the query system, which enable to balance the increment of reliability against an increase in runtime. The integer parameter *RayQuantity* defines the number of rays emitted in an inside test. The Boolean parameter *UseMultipleRayDirections* defines if a triangle with another emission direction is chosen when ambiguous intersection counts occur. In the established test bed, the usage of three rays per test case lead to 100% reliable results.

To simplify the pre-selection of potentially intersecting triangles, the rays are emitted parallel to the axes of the coordinate system. This strategy is well suited to be used in combination with the spatial indexing method presented in Section 6.2.

5.3. BRep-based evaluation of topological predicates

We begin by introducing the algorithms for evaluating the predicates *Disjoint* and *Touching* using 3D boundary representations. Like the octree-based algorithms, they are based on the verification and falsification of entries in the 9-Intersection Model.

The algorithm *Disjoint*, depicted in Fig. 6, first checks whether any triangle from object A meets or intersects any triangle from object B. If this is the case, the predicate can be rejected, because *Disjoint* only holds true if the intersection of the boundaries of A and B in the 9-IM is an empty set. The next step (line 3 et seq.) verifies that there is no containment relationship between A and B, thus determining whether the two objects are located inside each other. This is realized by the *InsideTest* algorithm. If no containment occurs, the *Disjoint* relationship is verified.

We will now proceed to present the subroutines *AnyTriangleTouchOrIntersect* and *InsideTest*.

The first routine iterates over all triangles from object A and tests every triangle from B for intersection or meeting using the triangle/triangle intersection test described above. If we take a look at the time complexity of *AnyTriangleTouchOrIntersect* (Fig. 7), the two nested *foreach* statements cause a quadratic increase, driven by the number of triangles n of the operands. The worst case time complexity is accordingly $O(n^2)$.

The *InsideTest* algorithm (Fig. 8) determines whether a BRep A is located inside a BRep B. A surface point on an arbitrary triangle from A is used as the starting point of a ray. The subfunction *MostAppropriateRayDirection* provides the designated coordinate axis

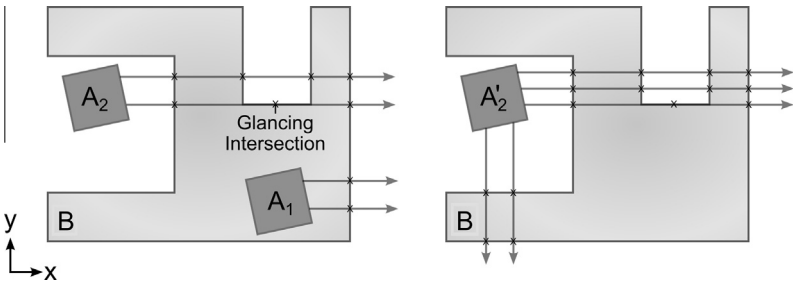
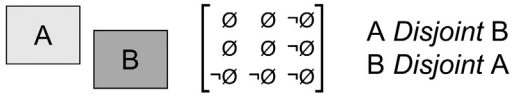


Fig. 5. Principle of the inside/outside classification of geometry objects based on ray tests.




```
Algorithm bool Disjoint(Brep A, Brep B)
1.   if AnyTriangleMeetOrIntersect(Brep A, Brep B)
2.         return false
3.   bool aIsInsideB = InsideTest (A, B)
4.   if aIsInsideB
5.         return false
6.   bool bIsInsideA = InsideTest (B, A)
7.   if bIsInsideA
8.         return false
9.   return true
```

Fig. 6. Algorithm Disjoint.

```
Algorithm bool AnyTriangleTouchOrIntersect (Brep A, Brep B)
1.   foreach Triangle triangleA in A
2.         foreach Triangle triangleB in B
3.               if TriangleTouchOrIntersect(triangleA, triangleB)
4.                     return true
5.   return false
```

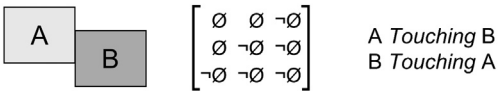
Fig. 7. Algorithm AnyTriangleTouchOrIntersect.

direction which shows the minimal deviation from the triangle's normal. Based on the starting point and the direction, we create a ray (line 3). In the subsequent *foreach* loop, every triangle from object B is tested for intersection with the ray. We consider the number of intersections to calculate the overall return value. An even number of intersections negates the inside classification. Because of previously rejected intersection tests, the returned classification not only holds true for the triangle but for the entire A object. As it is necessary to identify the exact number of intersections and no early abortion is possible, the time complexity of the *InsideTest* is always $O(n)$. Here, n is the number of triangles in the BRep of object B.

Fig. 9 depicts the algorithm *Touching* as pseudo code. The test uses two variants of the *AnyTriangleTouchOrIntersect* routine, which distinguish between touching and intersecting triangles. The worst case time complexity for both variants is again $O(n^2)$. If a pair of triangles from A and B intersect (line 1), this also indicates an intersection of the two interiors of the objects under examination. Since, however, the definition of the *Touching* predicate prohibits a non-empty set for the intersection of the interior of A with the interior of B, the *Touching* predicate can be rejected in the case of cutting triangles. On the other hand, there must be at least one pair of touching triangles between A and B to fulfill the predicate (line 3). Otherwise, the algorithm will stop at this point and return false.

```
Algorithm bool InsideTest (Brep A, Brep B)
1.   Triangle triangle = GetAbitraryTriangle(A)
2.   Direction direction = MostAppropriateRayDirection(triangle)
3.   Ray ray = CreateRay(triangle,  direction)
4.   Int intersectionCount = 0
5.   foreach Triangle triangle in B
6.         if TriangleIntersectRay(ray, triangle)
7.               intersectionCount++
8.   return IsEvenNumber(intersectionCount)
```

Fig. 8. Algorithm InsideTest.




```
Algorithm bool Touching (Brep A, Brep B)
1.   if AnyTriangleIntersect(Brep A, Brep B)
2.         return false
3.   if !AnyTriangleTouching(Brep A, Brep B)
4.         return false
5.   if(InsideTest (A, B) || InsideTest (B, A))
6.         return false
7.   return true
```

Fig. 9. Algorithm Touching.

If the execution has reached line 7, it is possible that A covers B or vice versa. A and B can nevertheless still touch each other. To disprove the *Covering* relationships, we carry out two inside tests. If these two tests return false, the objects cannot be nested and they must approximate each other from the exterior. This step verifies the *Touching* predicate.

As described in this section, it is possible to calculate topological predicates of 3D components by means of a triangle intersection and touching test coupled with inside tests. In contrast to the octree variant, this approach does not call for an explicit representation of the interior and the exterior of components. The algorithms developed for the remaining six topological predicates *Equal*, *Inside*, *Containing*, *Covering*, *CoveredBy*, and *Overlapping* follow the same principle: a hierarchy of intersection, touching and inside tests is established for every predicate. A detailed description of the algorithms implementing these predicates is available in [7].

6. Indexing methods for improving performance and complexity of spatial queries

6.1. Runtime performance considerations

Although the topological predicates presented here are functional, it is imperative to carefully consider their runtime behavior in complex building models with high object counts and more complex surfaces. This is particularly important as the value of an efficient spatial analysis of a model increases for complex scenes where a manual examination is practically impossible.

For the following performance considerations, the local processing to determine a topological predicate between two explicit objects has to be distinguished between a global inquiry. In the latter case, a whole set of objects is processed by a topological predicate. An example for a global querying is "Select all walls touching columns" in contrast to a local question like "Does wall 55 touch column 23?".

6.1.1. Local querying

As discussed above, the *InsideTest* increases linearly to the triangle count of the second operand. In addition, the subroutines for finding intersecting or touching triangles even suffer from a quadratic runtime increase in terms of the triangle count of the used *IfcTriangulatedFaceSets*. These algorithms are used for evaluating the local topological relationship between two elements and thereby operate on the level of the elements' triangles.

6.1.2. Global querying

To provide a comprehensive approach for the analysis of building models, the query system also offers global algorithms on the scene level. This is an extension to the local functionality on the level of two components. For example, it is possible to select from a set of all *IfcProducts* of a model those items which intersect other ones, as illustrated in the following code section. The algorithm

OverlappingIfcProducts takes a list of *IfcProducts* and checks each against all candidates (Fig. 10).

The processing of this type of global query, where a predicate is tested within the Cartesian Product of building elements has a time complexity of $O(n^2)$. Here n represents the number of elements in the model. When implemented without pre-filtering, the topological query functionality would accordingly suffer from an exponential runtime increase depending on the amount of objects in the model (global querying) and the number of triangles belonging to these objects (local querying).

6.2. R-trees

To overcome the exponential runtime complexity of global queries, and to make a system available that can deal with real-world building models containing tens of thousands of components, we have incorporated a spatial index structure in the system. We choose the R-tree data structure for this purpose, as it has proved to be highly effective for indexing multi-dimensional data [38].

The R-tree is a hierarchical data structure that makes it possible to index spatial data. Its basic functioning is illustrated in Fig. 11. The main objective of applying the R-tree data structure is to group items located in close vicinity. The established tree is height-balanced, so that the leaf nodes are all located on the same level. To provide an efficient tree construction and rapid spatial query functionality, the tree only operates on axis-aligned bounding boxes of indexed items. The tree is accordingly unable to calculate the final spatial predicates, but it can look for candidates rapidly.

In the context concerned, the building elements in a model are represented as three dimensional, closed shells, set up by means of BRep geometry. Because an R-tree is capable of indexing multi-dimensional items, it is suitable for dealing with this 3D geometry provided by the components used in BIM. The indexed items, denoted as data objects, are always connected to the leaf nodes of the tree. So a leaf of the tree contains the bounding box of an indexed geometry as well as a reference to the complex BRep and the corresponding, semantic BIM entity. This can be an *IfcWall* or an *IfcColumn*, for example.

The following section looks at the tree's internal structure. If we contemplate the tree from the point of view of a leaf, there is a node containing several leaves on the next level. This node also provides a bounding box, including all boxes from its referenced leaves. By repeating this approach of encasing boxes in a so-called directory box, we create a hierarchy where a node itself is characterized by its own directory box and pointers to child nodes or leaves which together determine the extent of the directory box. 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. The value of m must satisfy $m \leqslant M/2$.

As the R-tree is a dynamic index structure, items can be inserted, updated and deleted during runtime. If a new data object is inserted, the tree is traversed downwards to find the most appropriate leaf, picking the child that requires the least enlargement for the new item at each node that is visited and finally adding the object to a well-suited leaf node. In doing so, we try to keep the number of overlapping nodes in the tree to a minimum. If the selected node overflows, i.e. it now contains $M + 1$ children, it is divided into two new nodes to replace the overstaffed one. These nodes are inserted in the parent, which can cause it to overflow, too. The nodes are split recursively, maybe propagating to the root. If this happens, the tree gains in depth and a new root node is created. In this manner, the distinctive bounding box hierarchy of the R-tree is set up.

Similar to a B-tree [39], the R-tree uses this hierarchy to provide an efficient query functionality. Whilst the B-tree's hierarchy is built up by distinct elements, in the case of the R-tree, nodes on the same level can overlap. Thanks to its box hierarchy, the R-tree can rapidly return data objects completely contained in, or intersected by general query boxes. This feature is used extensively in the algorithms concerned for evaluating the topological predicates. Samet [40] described the R-tree data structure in more detail.

6.3. A two-tier spatial indexing

The proposed concept establishes a two-tier spatial indexing based on R-trees. This is necessary to accelerate the algorithms on both local triangle level and on global scene level. In the following section, the spatial indexing on scene level is termed global indexing because it treats the objects of the model as whole entities. The second level of spatial indexing deals with the triangles of building elements, so every object in the model with a three-dimensional representation is equipped with its own R-tree. This level of indexing is called local indexing in this treatise. Global indexing enables the system to rapidly find candidates for the various spatial queries, thus excluding a large number of elements from any further processing. Local indexing accelerates the evaluation of spatial predicates between two *IfcTriangulatedFaceSets* if the box pre-tests do not suffice.

6.3.1. R-tree query functionality

To illustrate the R-Tree indexing functionality, we first discuss the application of the R-tree to identify candidates for topological

```
Algorithm List OverlappingIfcProducts (List IfcProducts)
1.   List pairOfOverlappingProducts = new List()
2.   foreach IfcProduct productA in IfcProducts
3.         foreach IfcProduct productB in IfcProducts
4.               if productA.GlobalId == productB.GlobalId
5.                     continue
6.               if !BoundingBoxesOverlap(productA, productB)
7.                     continue
8.               if Overlap(productA.BRep, productB.BRep)
9.                     pair = new Pair(productA, productB)
10.                    overlappingProducts.Add(pair)
11.  return overlappingProducts
```

Fig. 10. Algorithm OverlappingIfcProducts.

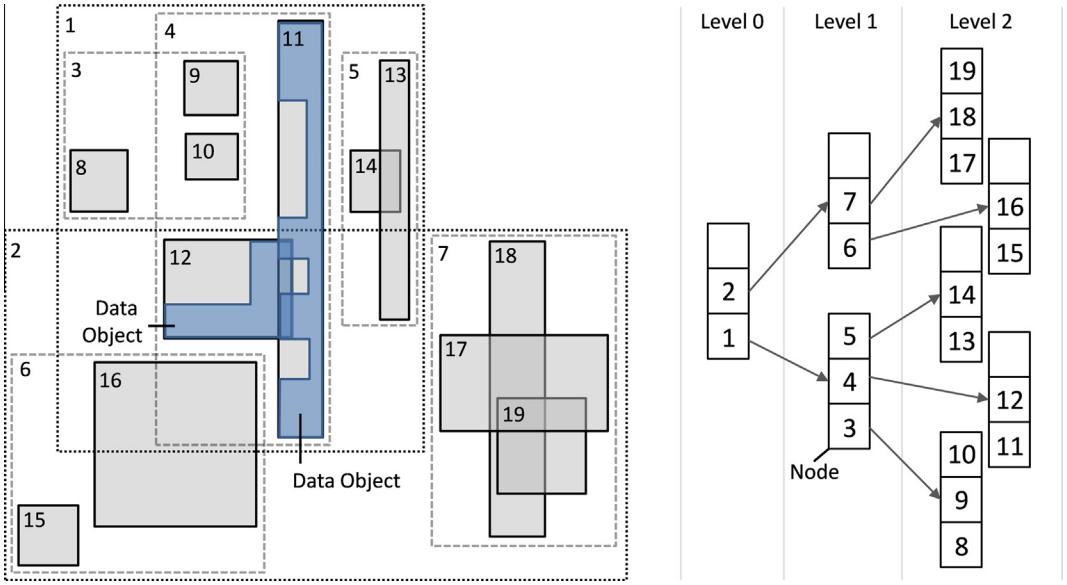


Fig. 11. Example of the R-tree structure calculated from 2D data, based on [38].

predicates by means of an example. For clarity, we examine a two-dimensional configuration, whereas the proposed system is designed for the three dimensional case. Rectangle R12 in the R-tree, shown in Fig. 11 is the bounding rectangle for data object 12. In this 2D example, the data object is a general polygon. The bounding shape of an indexed item always covers more or equal space than the referenced, complex form. Thus, if polygon 12 is overlapped by a second polygon, their bounding shapes must also intersect. This can be used to find candidates for the *Overlap* predicate, for example. The tree index structure is called on to return all objects intersecting with R12 in such a way that R12's bounding rectangle is designated as a query rectangle. Only R11 is returned and provides a data object – polygon 11 in this case – for further testing. All other polygons in the scene can be omitted from any further processing. Only the polygons 11 and 12 have to be examined in terms of their exact geometric representation, which are edges in this 2D case. In 3D, the triangles of the operands are employed for this further local testing.

To verify the *Overlap* predicate, it is necessary to find intersecting triangles. This is described in detail within the 3D algorithm *AnyTriangleTouchOrIntersect* (Fig. 7). In this 2D example of Fig. 11, we look for intersecting edges to verify the predicate: Polygon 12 is built with 6 edges and polygon 11 with 16 edges. If every edge from the first polygon is analyzed for intersection with every edge from the second polygon, $11 \times 16 = 172$ calculations have to be performed in the worst case. Switching from 2D to the 3D, the algorithm *AnyTriangleMeetOrIntersect* has to perform 250,000 triangle intersection tests if the used components A and B have 500 triangles each. To overcome the quadratic increase of calculations required, we utilize local indexing, which allows for an accurate identification of intersecting triangles between two meshes using a divide-and-conquer strategy [41]. We begin by identifying node candidates by means of a parallel breadth-first search. 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. In addition, all referenced candidates are replaced by their children. We now test the new nodes for intersection and produce new candidate lists. 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 described divide-and-conquer strategy is illustrated in Fig. 12.

The divide-and-conquer strategy can also be used on global scene level searching for overlapping components. We will now proceed to present the global version of this algorithm.

6.3.2. Candidate identification

The global version of the *OverlapCandidates* algorithm takes two R-trees as inputs (Fig. 13). The trees are constructed from the bounding boxes of BRep geometry, provided by the *IfcProducts* in a building model. In line 1, a list of pairs of *IfcProducts* is created and finally used as return value of the algorithm. This is followed by the instantiation of a stack (line 2) to which the root nodes of both trees are added. A *While* loops starts execution in line 4 until the stack is empty. The first stack item is popped in this loop, always containing one node from the first tree and a second node from the other tree. When the bounding boxes of these nodes do not intersect, the while loop continues its operations with a new iteration. In line 12, the current node pair is inserted in the return list, because the previous *if* statement proved that both nodes intersect and are leaves. An exact geometry test is therefore needed for the BRep geometry referred from this node pair. The Cartesian product of the children of both current nodes is built in the lines 13–19, bearing in mind that one node may already be a leaf. In this case, the leaf node itself is inserted in the corresponding *nodesFrom* list. Line 21 returns the candidates that have been detected.

The depicted *OverlapCandidates* algorithm is utilized to find overlapping *IfcProducts* on scene level. As with this global pre-selection of BIM components, it is possible to enhance the algorithms at triangle level. The algorithm is slightly modified to facilitate local indexing. To find intersecting or touching triangles from the two elements undergoing testing, their local R-trees are passed to a variant of the *OverlapCandidates* function. These R-trees have been established in advance on the basis of the bounding boxes of the individual triangles. Finally, a list of triangle pairs that possibly intersect or meet is returned. The more time-consuming triangle-to-triangle intersection tests have only to be performed on

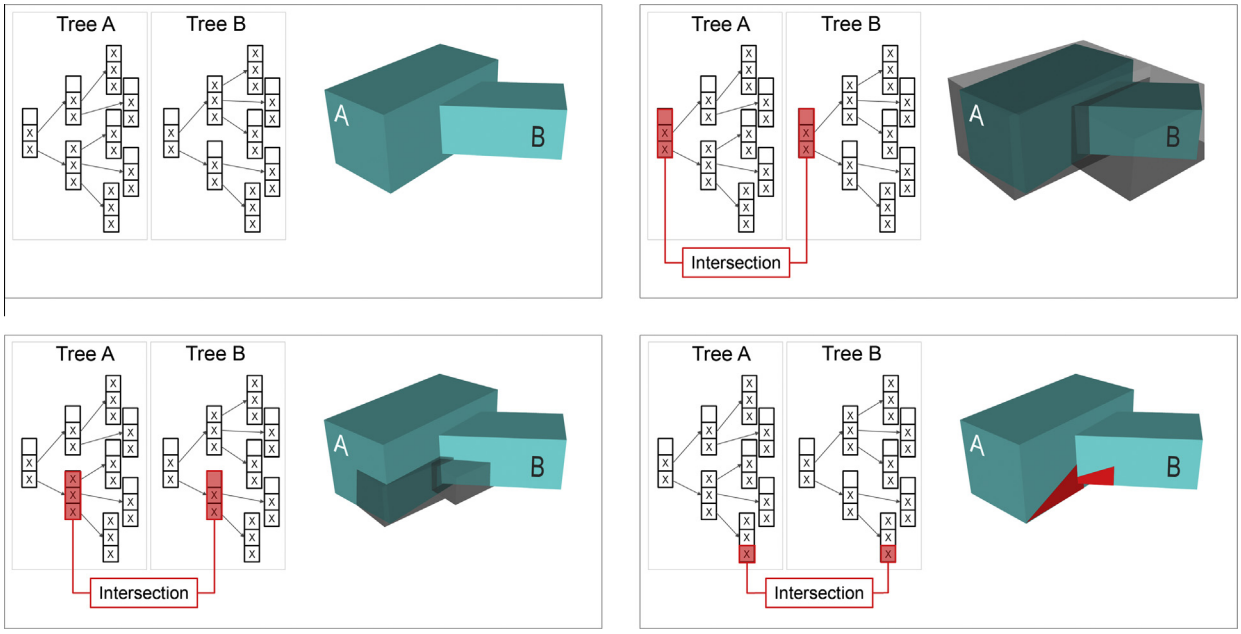


Fig. 12. Divide-and-conquer strategy for intersection search on triangle level.

```
Algorithm List OverlapCandidates (Rtree treeA, Rtree treeB)
1.   List candidatesPairs = new …
2.   Stack stackOfCandidatePairs = new …
3.   stackOfNodePairs.Push(treeA.Root, treeB.Root)
4.   while (stackOfCandidatePairs.Count > 0)
5.         var currentPair = stackOfCandidatePairs.Pop()
6.         var nodeA = currentPair.Item1
7.         var nodeB = currentPair.Item2
8.         if (!nodeA.Bounding.Intersects(nodeB.Bounding))
9.               continue
10.        if (nodeA.IsLeaf() && nodeB.IsLeaf())
11.              candidatesPairs.Add(currentPair);
12.        List nodesFromA = nodeA.IsLeaf()  ?
13.              new List{nodeA} : nodeA.Children()
14.        List nodesFromB = nodeB.IsLeaf()  ?
15.              new List{nodeB} : nodeB.Children()
16.        foreach (node1 in nodesFrom1)
17.              foreach (node2 in nodesFrom2)
18.                    stackOfNodePairs.Push(new Pair(node1, node2))
19.  return candidatesPairs
```

Fig. 13. Algorithm OverlapCandidates.

this reduced data set. The modified method that operates at triangle level is called *Overlap TriangleCandidates*.

Fig. 14 depicts the proposed high level method *OverlappingIfcProductsRTree* for the computation of the *Overlap* predicate. Exploiting the global and the local spatial indexing (algorithm *OverlapCandidates/OverlapTriangleCandidates*) leads to a considerable improve ment in the time complexity for *Overlap* verifications (see Section 7).

The evaluation of topological predicates using the BRep geometry of components also includes the inside/outside classification of triangles and meshes. The ray-based approach used for this classification is described in Section 5.3. The triangle-to-ray intersection test, which is performed against all triangles belonging to an *IfcTriangulatedFaceSet*, entails a time complexity of O(n), where n reflects the number of triangles.

During the inside test, it is possible to employ the R-tree to identify the minimal subset of triangles necessary for ray testing, thus increasing the runtime performance and reducing time complexity. The used approach is illustrated in Fig. 15. Here we examine the inside/outside configuration of two BRep objects A and B

```
Algorithm OverlappingIfcProductsRTree (List IfcProducts)
1.   List overlappingProducts = new …
2.   gloCandidates = OverlapCandidates (globalTree, globalTree)
3.   foreach productPair in gloCandidates
4.      locCs = OverlapTriangleCandidates(pairP.A.Rtree,pairP.B.Rtree)
6.         foreach trianglePair in locCs
7.            if TriangleOverlap(trianglePair.A, trianglePair.B)
8.               overlapingProducts.Add(productPair)
9.               break
10. return overlapingProducts
```

Fig. 14. Algorithm OverlappingIfcProductsRTree.

(15a). Firstly, an arbitrary triangle from the BRep A is chosen (b). BRep A has been previously tested for any triangle intersections. Since no intersection occurred, if one triangle is classified as being either inside or outside, the same attribute holds true for the entire BRep. The normal of the picked triangle let us chose +*X* as the direction in which to emit rays. The triangle's bounding box is consequently extended to the maximal *X* value of the bounding box of BRep B (c). The extended box acts as the query input for an intersection search in the second R-tree, established for the *IfcTriangulatedFaceSet* of B (see d). By this means, we retrieve the triangle candidates for ray testing (e). The ray is finally drawn from the A's chosen triangle to obtain the number of intersections. Because of the odd number of intersections in Fig. 15f), the triangle can be classified as being inside the object B. Accordingly, BRep A is verified of being located inside BRep B.

7. Performance tests

We set up a test bed to measure the absolute runtime of the topological predicates *Overlapping*, *Touching*, *Disjoint* and *Inside* for the purpose of verifying the practical usage of the proposed approach. At the same time, the time complexity in terms of the overall object count in a scene is indicated. Every object is made up of 192 triangles. The scene contains a cluster of four objects positioned closely to one another: two objects touch, three overlap and one object is located inside another one. We then duplicate the clusters to increase the object count in the scene (Fig. 16).

The test bed is implemented in C# in a single-threaded environment. The most demanding constellation comprises a scene made of 13,500 objects. Although the absolute, average runtimes obtained in the tests are within the range of a few seconds, which is promising for industrial applications, they can be further enhanced with a high performance, native implementation, using the programming language C or FORTRAN, for example. Table 2 shows the information of the used system and implementation detail.

The first performance test (Fig. 17) compares the runtime of the octree approach presented in [3] with that of BRep-based methods presented in this paper. Because of the required fine resolution of the involved octrees and the caused high amount of created octants, we see an immense performance improvement if BRep methods are used.

Fig. 18 contrasts the runtimes of the brute force implementation without any spatial indexing with the times achieved by the proposed two-tiered indexing for the *Overlapping*, *Touching*, *Disjoint* and *Inside* predicates. As the runtimes of the R-tree approach scale linearly and with a considerable low inclination, the system is able to filter real-world BIM data during appropriate execution times.

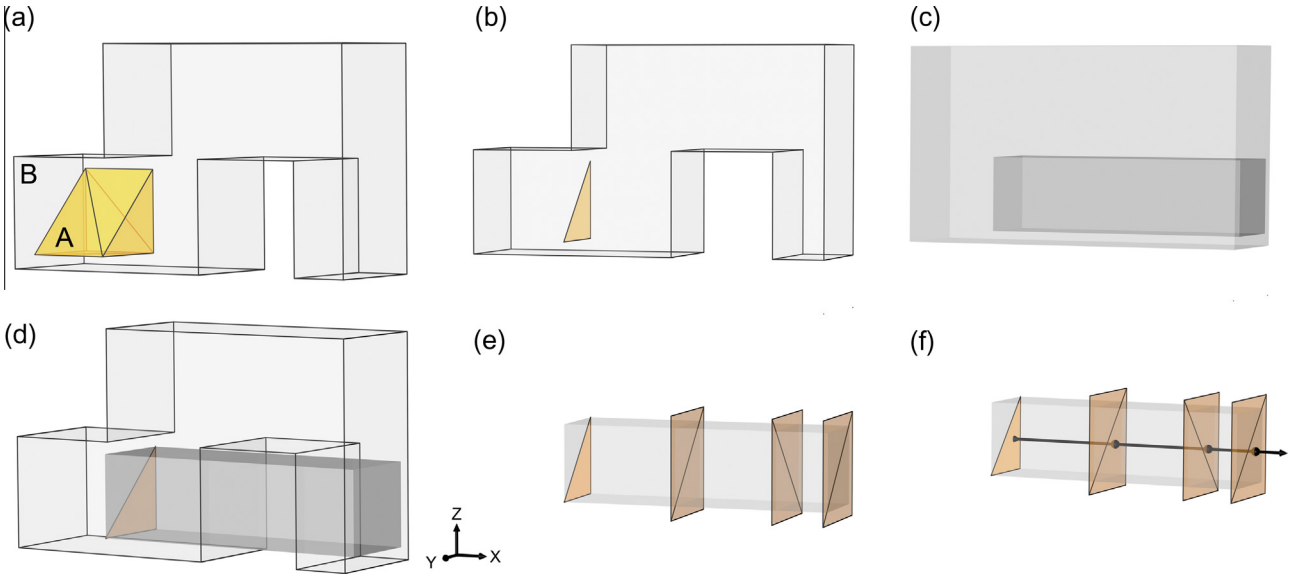


Fig. 15. Accelerating the inside test by applying the local R-tree indexing.

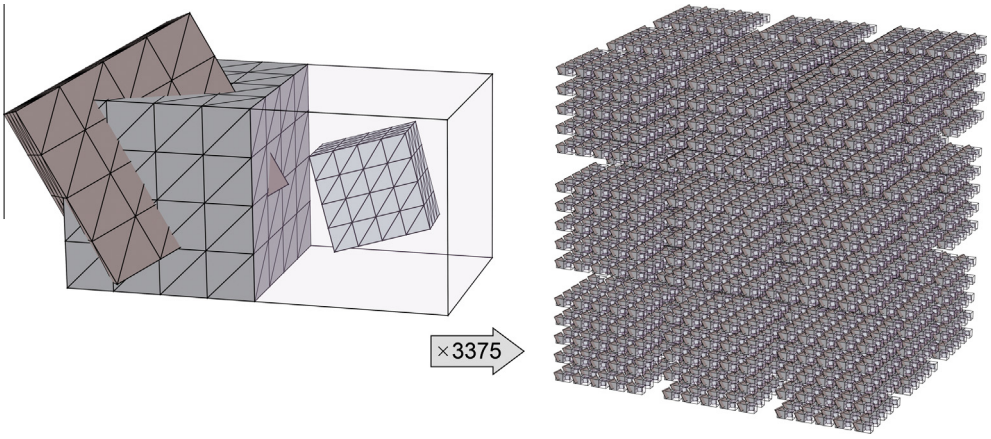


Fig. 16. The four geometric objects in a touch/overlap/inside configuration shown on the left side were repeated up to 3375 times to create the test bed with maximal 13,500 objects.

Table 2
Performance test configuration.

CPU frequency	3.40 GHz
Used threads	1
System main memory	16.00 GB
Maximal used memory octree	3.72 GB
Maximal used memory BRep	3.25 GB

8. Formulation of filter expressions and their evaluation

In the proposed concept, the user enters a query expression by way of source code. The expression defines a predicate, which is used to select objects from a given set. The returned objects satisfy the predicate in the expression. To give an example, a set of walls is filtered by the expression `wall.GlobalID.StartsWith(''lpJQ'')`. Here, wall is a formal parameter which is replaced by all walls in the set one after another.

8.1. LINQ

We use LINQ [42] for evaluating query expressions on set level, as it provides powerful query mechanisms for in-memory collections and object networks. LINQ is neatly integrated into the .NET framework and queries can be formulated using any .NET language. The queries are type safe and attributes and methods of involved objects can be used. For the definition of a query, an anonymous function, called Lambda expression is built up. An example of this expression type is `wall => wall.GlobalID.StartsWith(''lpJQ'')`. The proposed BIM query system uses C# in combination with LINQ for examining in-memory collections. This allows the filtering of building models to be executed directly in the fast main memory of the machine.

8.2. Live LINQ

Applied in a standard context, LINQ requires the definition of filter predicates during compilation time. While it is possible to pass parameters to a LINQ query during runtime, the system is restricted to pre-defined filters. To support flexible query statements, exclusively defined by the end-user at runtime, the expression he enters has to be converted into an executable filter object. This is achieved via an on-the-fly compilation of user input. We call this functionality Live LINQ. If the C# code that is entered is accurate, it is compiled and saved as a .NET assembly, shown as a DLL in the file system. The query application can then load this assembly at runtime and use the embedded filter object in the pending filter execution. The user gets an unfiltered set of *IfcProduct* entities by way of the original data source. Whenever a query is executed, the LINQ processor iterates over the *IfcProduct* collection. The returned set only contains entities which the dynamic filter evaluates as true.

In contrast to standard LINQ, the use of Live LINQ allows the user to define queries in a flexible manner. All object attributes can be used in a query and the system is not restricted to

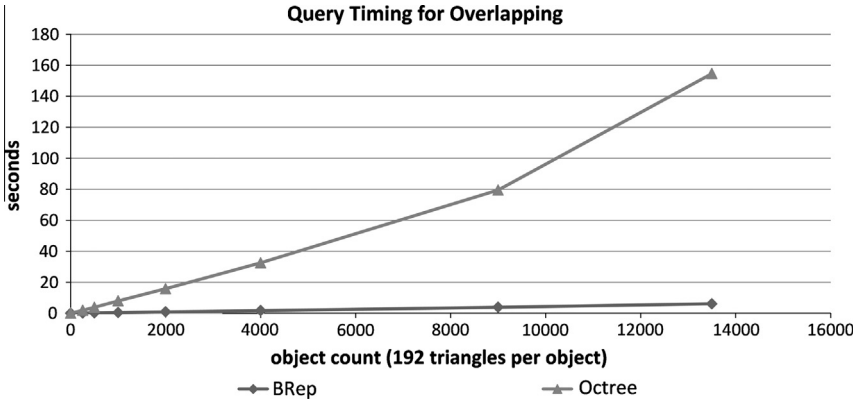


Fig. 17. Comparing execution timing for the Overlap predicate achieved with the developed BRep based methods versus the Octree approach. Octree refinement level = 6.

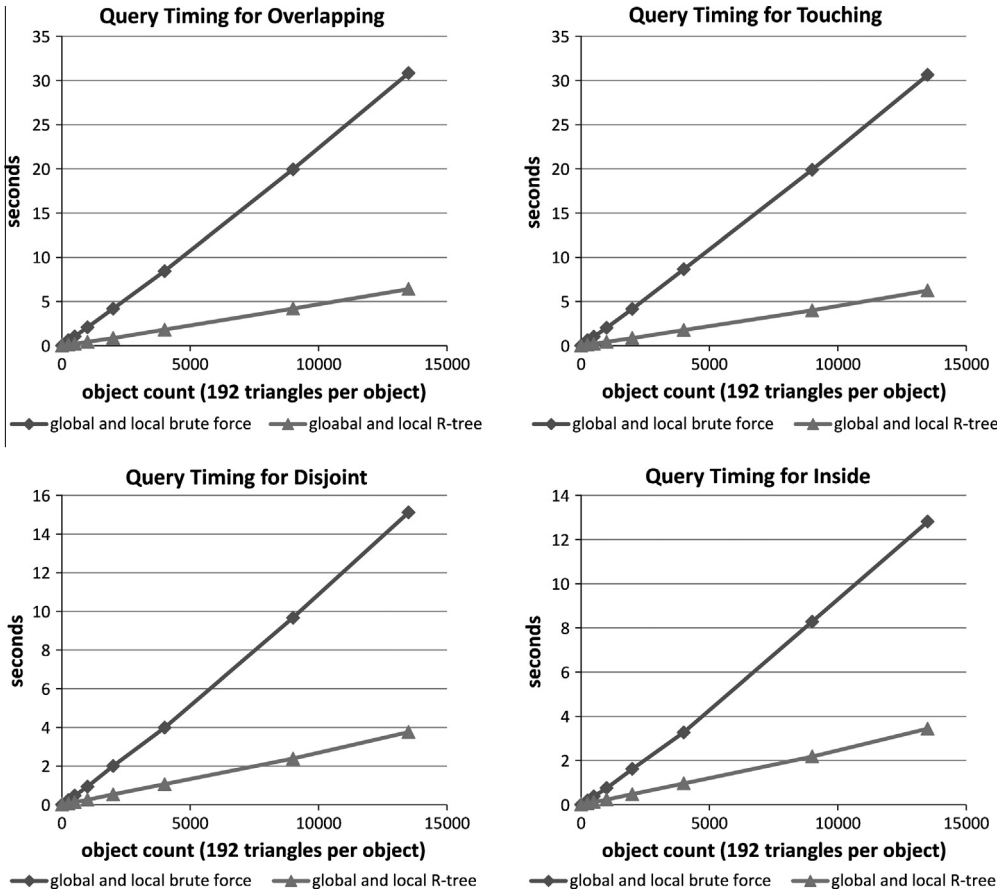


Fig. 18. Absolute runtime measurements and time complexity behavior of the algorithms implementing the Overlapping, Touching, Disjoint and Inside predicates.

predefined filters. In addition, as LINQ interacts directly on object level, methods published by objects are also accessible in queries that make the developed topological operators available. Finally, the filter generation at runtime promotes the compiler as a validation tool for user-defined query statements. If an inadequate code is submitted, the compiler returns clear error messages.

The Live LINQ system, including the proposed topological operators, has been successfully implemented in a BIM client, as depicted in Fig. 19.

8.3. Query examples

The following query examples illustrate the usage of the Live LINQ system and its incorporated spatial query functionality. The first query does not include a spatial predicate. It rather shows the principle query syntax. Besides filtering by IFC classes, Examples 2 and 3 make use of topological and directional predicates. Example 4 is a demonstration how to realize a nested spatial query.

Example 1. Select a wall with GlobalId 1pJQicIrH4UR_2BqvRP.

```
IfcProducts.Where(p =>
  p is IfcWallStandardCase && p.GlobalId ==
  ''1pJQicIrH4UR_2BqvRP'')
```

Example 2. Select walls which overlap with wall 1 (index 25)

```
IfcProducts.Where(p = > {
  var walll = IfcProducts [25];
  return p is IfcWallStandardCase &&
  p.Overlaps(walll);})
```

Example 3. Select walls which overlap with wall 1 (index 25) and which are located above slab 1 (index 12)

```
IfcProducts.Where(p => {
  var walll = IfcProducts [25];
  var slabl = IfcProducts [12];
  return p.Overlaps(walll) && p.Above(slabl);})
```

Example 4. Firstly, select walls below slab 1. Secondly, select all products which touch column 1 (index 34) but do not touch any of the fetched walls, located below slab 1.

```
var slabl = IfcProducts [12];
var wallsBelow = IfcProducts.Where(p => p is
  IfcWall && p.Below(slabl));
IfcProducts.Where(p = > {
  var columnl = IfcProducts [34];
  return p.Touch(columnl) && !wallsBelow.Any(w =>
  w.Touch(p));})
```

9. Conclusion and future work

BIM and its underlying data models, especially the IFC, have evolved over the past two decades. Today, they provide a well-grounded foundation for representing a wide range of buildings and the processes of their planning, construction and maintenance.

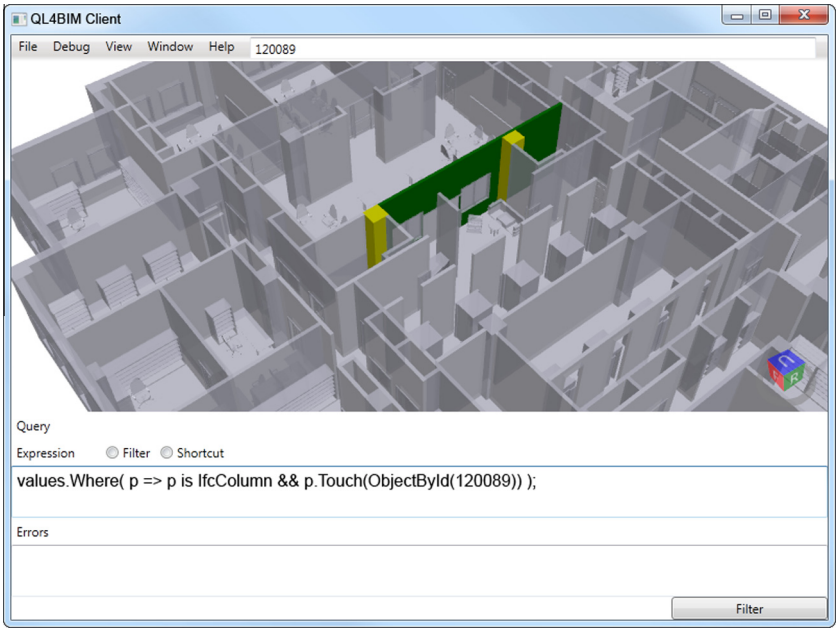


Fig. 19. Developed QL4BIM system with spatial query functionality and Live LINQ.

On the other hand, there are gaps in the analysis and handling functionality available for BIM, concerning the formal spatial exploration of models and the extraction of submodels.

This treatise describes new BRep-based methods for the verification of topological 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 of topological predicates.

In order to deal efficiently with the computational complexity arising in large-scale models, we combine two-tiered spatial indexing with efficient geometric algorithms. Both the actual runtime and the time complexity of the topological predicates outperform the previously used octree approach.

We devised a runtime parsing of ad-hoc queries, called Live LINQ, so that a user is not limited to predefined queries but is able to freely specify filters. The novel system of spatial processing using QL4BIM allows an efficient querying of complex Building Information Models. As proof of concept, we developed a BIM client which incorporates the proposed approaches for the topological predicates and the Live LINQ query system.

As one of the future aspects of our work, we will investigate whether and how the expressivity of the topological operators of QL4BIM can be enhanced by making use of the Dimensionally Extended 9 Intersection Method as a formal basis. Additionally, a user defined thickness of the boundary of building elements will be incorporated in the developed system. Doing so, the system is able to classify elements as touching, even if they are few millimetres apart, for example.

References

[1] C.M. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors, second ed., Wiley, Hoboken, NJ, 2011.
[2] ISO, Building Information Modelling – Information Delivery Manual – Part 1: Methodology and Format (ISO 29481-1). <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45501>, 2011.
[3] A. Borrmann, E. Rank, Specification and implementation of directional operators in a 3D spatial query language for building information models, Adv. Eng. Inform. 23 (1) (2009) 32–44, http://dx.doi.org/10.1016/j.aei.2008.06.005.
[4] A. Borrmann, S. Schraufstetter, E. Rank, Implementing metric operators of a spatial query language for 3D building models: octree and B-Rep approaches, J. Comput. Civ. Eng. 23 (1) (2009) 34–46, http://dx.doi.org/10.1061/(ASCE)0887-3801(2009) 23:1(34).
[5] A. Borrmann, E. Rank, Topological analysis of 3D building models using a spatial query language, Adv. Eng. Inform. 23 (4) (2009) 370–385, http://dx.doi.org/10.1016/j.aei.2009.06.00.
[6] A. Borrmann, Extended Formal Specifications of 3D Spatial Data Types, 2006.
[7] S, Daum, A. Borrmann, Boundary representation-based implementation of spatial BIM queries, in: Proc. of the EG-ICE Workshop on Intelligent Computing in Engineering, Vienna, Austria, 2013.
[8] S. Daum, A. Borrmann, Efficient and robust octree generation for implementing topological queries for building information models, in: Proc. of the EG-ICE Workshop on Intelligent Computing in Civil Engineering, 2012.
[9] ISO 16739:2013, Industry Foundation Classes (IFC) for Data Sharing in the Construction and Facility Management Industries (ISO 16739:2013).
[10] D. Brutzman, L. Daly, X3D: Extensible 3D Graphics for Web Authors, Morgan Kaufmann/Elsevier, San Francisco, CA, 2007.
[11] E. Tauscher, IFC TOOLS Project: Visualize Your Digital Knowledge. <http://www.ifctoolsproject.com>. [February 21, 2014].
[12] E.F. Codd, The Relational Model for Database Management: Version 2, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1990.
[13] A. Borrmann, E. Rank, Query support for BIMs using semantic and spatial conditions: handbook of research on building information modeling and construction informatics, in: J. Underwood, U. Iskidag (Ed.), Information Science Pub, 2009.
[14] W. Mazairac, J. Beetz, BIMQL – An open query language for building information models, Adv. Eng. Inform. 27 (4) (2013) 444–456, http://dx.doi.org/10.1016/j.aei.2013.06.001.
[15] Y. Adachi, Overview of partial model query language, in: Proc. of the 10th Int. Conf. on Concurrent Engineering, 2003.
[16] M. Weise, P. Katranuschkov, R.J. Scherer, Generalized model subset definition schema, in: Proc. of the 20th CIB-W78 Conference on Information Technology in Construction, 2003.
[17] B. Quilitz, U. Leser, Querying Distributed RDF Data Sources with SPARQL, The Semantic Web, Research and Applications, 2008.
[18] N. Roussopoulos, C. Faloutsos, T. Sellis, An efficient pictorial database system for PSQL, IEEE Trans. Software Eng. 14 (5) (1988) 639–650, http://dx.doi.org/10.1109/32.6141.
[19] 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.
[20] 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.
[21] 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.
[22] 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, 1988.
[23] M. Egenhofer, Why not SQL!, J Geogr. Inform. Syst. 6 (2) (1992) 71–85.
[24] OGC, OpenGIS Implementation Standard for Geographic Information – Simple Feature Access – Part 1: Common Architecture, first ed., 2011.
[25] ESRI, Working With 3D Set Operators: ESRI. <http://resources.arcgis.com/en/help/main/10.2/index.html#//00q80000009v000000> [September 30, 2013].
[26] PostGIS, PostGIS: Spatial and Geographic objects for PostgreSQL. <http://postgis.net/> [October 01, 2013].

[27] M. Egenhofer, J. Herring, A Mathematical framework for the definition of topological relationships, in: Proc. of the 4th Int. Symp. on Spatial Data Handling.
[28] M. Egenhofer, R. Franzosa, Point-set topological spatial relations, Int. J. Geogr. Inform. Syst. 5 (2) (1991) 161–174.
[29] S. Gaal, Point Set Topology, Academic Press, New York, 1964.
[30] A. Borrmann, E. Rank, Topological operators in a 3D spatial query language for building information models, in: Proc. of the 12th Int. Conf. on Computing in Civil and Building Engineering (ICCCBE), 2008.
[31] E. Clementini, P. Felice, P. Oosterom, A small set of formal topological relationships suitable for end-user interaction, in: G. Goos, J. Hartmanis, D. Abel, B.Chin Ooi (Eds.), Advances in Spatial Databases, Springer Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 277–295.
[32] H. Samet, Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS, Addison-Wesley, 1989.
[33] T. Akenine-Möller, Fast 3D triangle-box overlap testing, J. Graph. Tools 6 (1) (2001) 29–33.
[34] S. Daum, Octree-Generierung: visual debugging mit blender, in: 25. Forum Bauinformatik, Bochum, 2012.
[35] T. Akenine-Möller, A fast triangle-triangle intersection test, J. Graph. Tools (1997) 25–30.
[36] D. Kirk, F. Antonio, Graphics Gems III: Faster Line Segment Intersection, AP Professional, Boston, 1992.
[37] S. Schraufstetter, A. Borrmann, AABB-Bäume als Grundlage effizienter Algorithmen für Operatoren einer räumlichen Anfragesprache, in: Tagungsband des 19. Forum Bauinformatik, 2007.
[38] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1984.
[39] T.H. Cormen, Introduction to Algorithms, second ed., MIT Press, Cambridge, Mass, 2001.
[40] H. Samet, Spatial data structures, in: W. Kim (Ed.), Modern Database Systems: The Object Model, Interoperability, and Beyond, ACM Press/Addison-Wesley Publishing Co., 1995, pp. 361–385.
[41] S. Gottschalk, M.C. Lin, D. Manocha, OBBTree: a hierarchical structure for rapid interference detection, in: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, 1996.
[42] E. Meijer, B. Beckman, G. Bierman, LINQ: reconciling object, relations and XML in the .NET framework, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, ACM, Chicago, IL, USA, 2006. pp. 706–706.