Query support for BIMs using semantic and spatial conditions

André Borrmann, Ernst Rank
Technische Universität München, Germany

ABSTRACT

A query language for Building Information Models allows users and third-party application programmers to not only analyze the digital building under specific criteria but also to extract partial models from a full building model. This functionality is of crucial importance, since the full BIM is meant to comprise the information of all domains involved in the planning process, but an individual user or programmer is normally interested in only a small subset of it. To specify this subset, a formal language is required which makes it possible to formulate conditions the resulting data set has to fulfill. This concept is also known as providing a certain view of the data available.

This chapter gives an overview of the currently available query technologies for BIMs and compares the different options in terms to expressive power and ease of use. The emphasis of the chapter, however, lies in the introduction of spatial query technology for BIMs that has been developed by the authors. Spatial operators extend the analysis and submodel specification capabilities of a query language substantially by providing an intermediate level of abstraction that is close to the human understanding of the geometric-toplological properties of building components and the relationships between them.

1 INTRODUCTION

The computer-based modelling of buildings has been an important topic of the construction informatics research community for more than 15 years now. An object-oriented Building Information Model (BIM) that not only captures the 3D geometry of the building elements but also their semantics and the relationships between them promises to enable a seamless integration of design software and downstream applications, and hence to serve as a solid basis for the highly collaborative work in AEC projects.

Sophisticated digital building models can facilitate the collaboration between the various participants involved in the design and engineering process, including architects, structural engineers, HVAC engineers and interior designers, to only name a few. Though in modern AEC processes these participants work in parallel on the same building, they only use specific subsets of the entire building information model tailored to the needs of their particular domain and/or their specific task. To store these sub-models in separate files for further processing, they need to be extracted from the full building model.

As the resulting partial models represent a certain view of the shared BIM, they are of considerable value also for the aforementioned downstream applications, such as the various visualization, analysis and simulation tools that form an integral part of modern construction engineering. In most cases, these tools require only a subset of the full building model data to perform their specific task.

A well-established technology for retrieving parts of digital models is the use of a declarative query language, which is most familiar from the context of database management systems. In general, a declarative query language enables the user to define conditions that need to be satisfied by the required model subset, while simultaneously hiding the complex task of an efficient query processing.

Besides creating partial model, however, a query language for BIMs also enables the *analysis* of building models with respect to its components, their properties and the relationships between them. A sophisticated query language can accordingly be used to define rules or conditions that need to be fulfilled by the building model. In contrast to employing a programming language for this purpose, these rules are defined independently of the processing algorithms and thus provide an excellent basis for the future encoding of national and international building regulations.

This chapter gives an introduction to the query techniques currently available for Building Information Models and compares them with respect to their expressive power and ease of use. It meanwhile focuses on two different types of conditions: *Semantic conditions* that rely on the values of the attributes and relationships predefined in the building information model, and *spatial conditions* that concern the topological and geometrical properties of the building model and its entities.

Since spatial operators are not yet available in current commercial implementations of BIM query languages, this chapter gives a detailed account of the development of a spatial query language for BIMs including the formal definitions of the spatial operators and their technical implementation.

2 BUILDING INFORMATION MODELS AND PRODUCT MODEL SERVERS

2.1 Building Information Models and STEP

A Building Information Model (BIM) is a digital representation of a building that is either at the planning stage or has already been built. It describes the structure of this building by means of an object-oriented model, capturing the 3D geometry of the building elements, their semantics and the relationships between them.

In order to achieve interoperability between different software applications used in the design and construction process, it is necessary to use a standardized data model (schema) representing a blueprint for the digital models of actual buildings. The most mature data model standards in the AEC domain are the Industry Foundation Classes (IFC) (International Organization for Standardization, 2005) and the CIS/2-Standard (Eastman, Wang, You, & Yang, 2005).

For historical reasons, both data models have been developed using STEP, the ISO standard for the exchange of product model data (International Organization for Standardization, 1995; Fowler, 1995). On this account, Building Information Models are also referred to as Building *Product Models*. Part 11 of the STEP framework defines the object-oriented data modelling language EXPRESS which has also been employed to describe the IFC and CIS/2 data models.

EXPRESS provides a rich set of object-oriented modelling features including the definition of *entity types* (the equivalent of a *class*) which encapsulate *attributes* representing the common properties of the objects belonging to the same entity type. Attributes are either of basic type, or are used to model associations with other entity types. In addition, EXPRESS provides pre-defined *collection types* (array, bag, set, list) that are able to hold a number of associated objects, and *select types* that basically model enumeration types. A particularity of EXPRESS is the existence of *inverse attributes* which provide means to identify an association also from the entity being referenced (an incoming association). EXPRESS can be used to define algorithms that are part of integrity constraints, but it does not support the definition of methods, i.e. the behaviour of objects cannot be described.

Of particular interest in the context of query functionality is the mapping language EXPRESS-X defined in Part 14 of the STEP standard and the mapping to the data modeling language XML-Schema defined in Part 28.

2.2 Product Model Data Structure: The IFC as a case study

In their capacity as the foremost AEC product model available today, the Industry Foundation Classes (IFC) make extensive use of object-oriented modelling techniques such as encapsulation, inheritance and relationships, leading to a fine-grained and particularly complex data model (Amor, Jiang, & Chen, 2007).

This is illustrated by Figure 1 which depicts the relationship between a wall and a window contained in this wall. Note that the IFC make use of *objectified relationships*[1], and employ the EXPRESS concept of *inverse attributes* that can be used to navigate an association in the opposite direction.

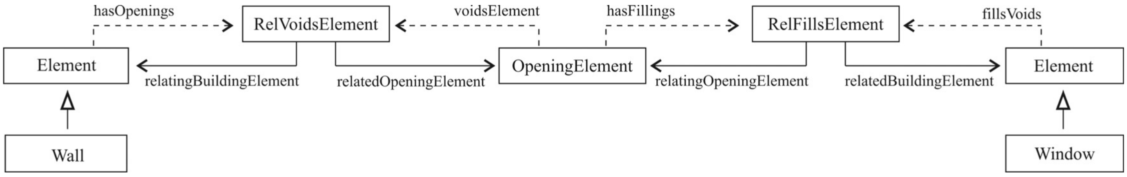


Figure 1: Fraction of the IFC model that represents the relationship between a wall and the windows contained in it. Note that the inheritance graph has been radically reduced to show the essential parts only.

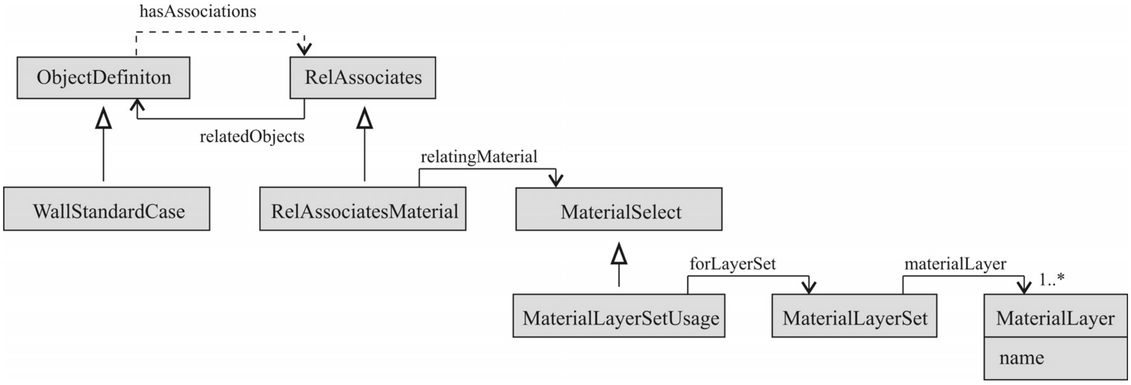


Figure 2: Fraction of the IFC model that represents the relationship between a wall and the material it is composed of. Note that the inheritance graph has been radically reduced to show the essential parts only.

[1] The respective entities in the IFC model have the prefix *IfcRel*.

The complex structure of the IFC data model makes the retrieval of relevant attributes very difficult. As shown in Figure 2, identifying the material of a wall involves 4 intermediate steps for navigating along the intermediary entities. This particularly affects the formulation of queries that use the values of such "faraway" attributes as selection predicate.

Throughout this chapter, queries that use attribute values are denoted as *semantic* queries in order to distinguish them from *spatial* queries that rely on purely geometric properties and relationships. As a specimen of semantic queries we will use the query "Find all walls that are made of concrete" and express it by means of the query languages that are introduced in the next Section.

2.3 Product model servers and their query functionalities

For the future, product model servers that manage the product model centrally and maintain its consistency are seen as the most promising solution to today's data management problems in the AEC industry (Kiviniemi, Fischer, & Bazjanac, 2005). Product model servers that are specialized in handling IFC data and are already on the market include the Jotne EDMServer and the EuroStep ModelServer (EMS). Another IFC model server was developed within the IMsvr research project realized by VTT and Secom Ltd. (Adachi, 2002). All available product model servers provide query languages that allow the users to extract parts of the full building model.

The individual realization of the query language concept depends largely on the technological basis of the product model server itself: On the one hand, there are model servers, that are natively based on EXPRESS and accordingly offer direct support for all EXPRESS constructs including access to aggregates, inverse attributes, SELECT data types and so on. EDMServer by Jotne is one example of a native EXPRESS server. For the purpose of model query functionality it provides proprietary languages that are based on, but not part of the official EXPRESS standard, including EDMexpressX^{TM} and EDMQuerySchema^{TM}.

Model servers that are implemented on the basis of XML, such as the IFC model server, the EuroStep ModelServer and the SABLE server, on the other hand, provide special XML schemata for describing the contents of partial models. In the case of EMS the query schema is called *Product Model Query Language* and in case of the IMsvr it is called *Partial Model Query Language*. We will have a closer look on these schemata in Section 3.2.2.

3 REALIZING SEMANTIC QUERIES

3.1 Native STEP query functionality

The STEP standard does not provide "real" query functionality. Although there is a QUERY operator available in EXPRESS, it can not be applied individually, but only within the WHERE part of an entity definition for the specification of integrity rules.

However, native EXPRESS query functionality can be realized by means of EXPRESS-X. The language was originally designed for defining mappings from entities in one EXPRESS schema to entities in another schema (Bailey, Hardwick, Laud, & Spooner, 1996). This mapping functionality of EXPRESS-X was applied in the CIS2IFC project[2], for example. However, since the result of a query can be interpreted as a view on the underlying model, EXPRESS-X can also be employed for the formulation of queries.

[2] http://www.coa.gatech.edu/~aisc/cisifc

The latest version of EXPRESS-X has a clearly declarative character and is reminiscent of the well-known query language SQL (Denno & Sanderson, 2000). An EXPRESS-X query consists of a FROM, a WHERE, and a RETURN part. The FROM part defines the entity types (extents) over which the query shall iterate and binds local variables to these types. The WHERE part specifies the constraints that have to be fulfilled by the selected entities, and the RETURN part defines how the result is returned.

The formulation of our specimen query "Find all walls that are made of concrete" is depicted in Figure 3. As it illustrates, the main strengths of EXPRESS-X lie in its expressive power with respect to the definition of constraints. This includes the

- navigation along an association chain by repeatedly applying the dot-operator,
- access to inverse attributes (navigating along inverse associations),
- native support for EXPRESS aggregations, e.g. built-in functions for determining the size of a collection

This results in a comparatively short and simple definition of the query without any syntactical overhead.

```
VIEW concrete_walls;
 FROM w : IFCWallStandardCase;
 WHERE w.hasAssociations[1].relatingMaterial.forLayerSet.
         materialLayers[1].material.name = 'Concrete';
 RETURN w;
END VIEW;
```

Figure 3: The specimen query defined in EXPRESS-X

Unfortunately, at the time writing there are not many EXPRESS-X engines available on the market. Two commercial versions are provided within the STEP toolkits *ST-Developer* by StepTools[3] and *ECCO toolkit* by PDTec[4], respectively. The only engine available under public license is the *Express Engine*[5] that was originally called *Expresso*, but is not under development any more since 2002.

3.2 Schema-based approaches

A frequently employed approach for realizing query functionality on product models is the definition of a schema whose instances can be used to describe the content of a sub-model, or a view of the original model, respectively. In contrast to declarative query languages such as EXPRESS-X, XQuery or SQL, the subset is not defined by using a constraint statement, but by an instance of an object-oriented schema. Two variants of query schemata can be distinguished: Those that are defined using EXPRESS and those that are based on XML technology.

3.2.1 Express-based schema approach

An example for an EXPRESS-based approach is the Generalized Model Subset Definition Schema (GMSD) which has been introduced in (Weise, Katranuschkov, & Scherer, 2003).

[3] http://www.steptools.com

[4] http://www.pdtec.de

[5] http://exp-engine.sourceforge.net

GMSD is an EXPRESS-based schema that defines entities which can be used to specify the contents of a partial model by identifying objects and attributes that need to be included.

The schema accordingly consists of two parts. The first part is applied to select individual object instances using set theory as baseline. The second part is intended for post-processing of the selected data in accordance with a specific partial model view. Such views can be completely predefined to a large extent to support standard domain perspectives.

To give a better insight on the GMSD schema we have a closer look on the entity *SelectInstances*. It contains the attributes *AttributeName*, *Value* and *Tolerance*. In a concrete instance of GMSD (a "query"), the user assigns values to these attributes, thereby specifying which instances of the full model should be included in the resulting partial model. Figures 4, 5 and 6 show the individual parts of the GMSD schema.

Note that GMSD allows only the definition of a view as a reduction of the original schema with respect to the number of attributes, it does not support mapping operations where the target schema is structurally different. Accordingly it is only of limited use for query scenarios that aim at analyzing the building model.

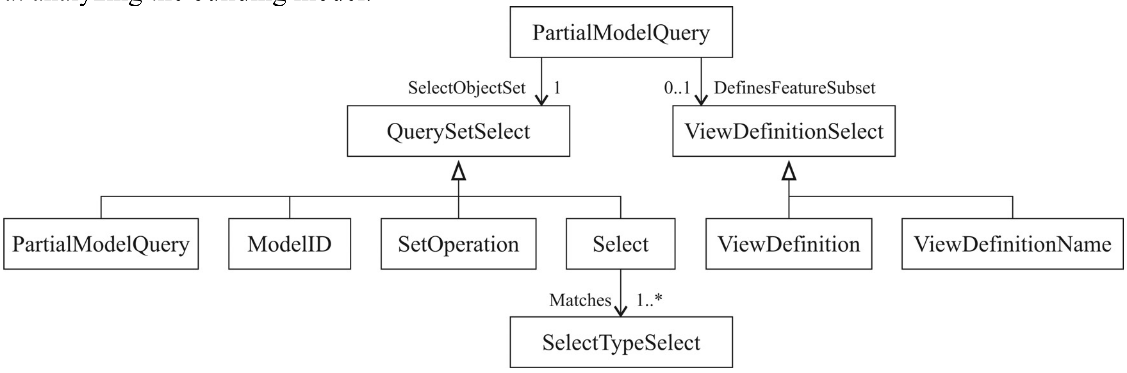


Figure 4: The GMSD schema consists of two parts: The left part selects individual object instances, the right part is used to define which attributes shall be included in the resulting view. Simplified UML representation.

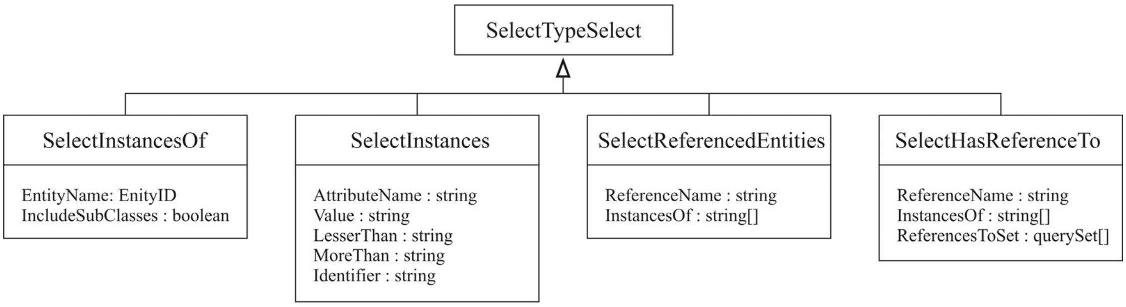


Figure 5: Part of GMSD schema used to include individual instances in the resulting subset. Possible criteria for inclusion are: the name of the corresponding schema, the value of specific attributes of the instance, or the existence of a relationship to another entity. Simplified UML representation.

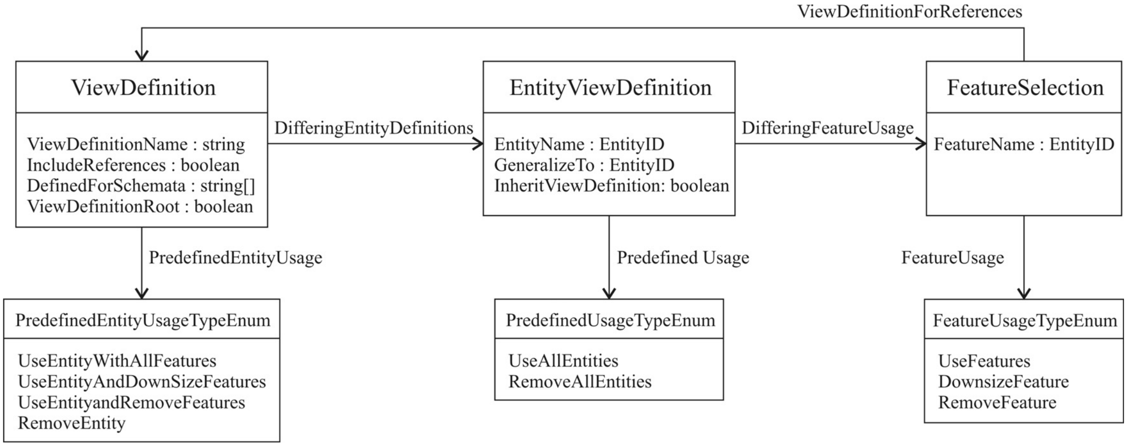


Figure 6: Part of GMSD schema used to define the attributes that are included in the resulting view. Note that in GMSD, attributes are referred to as features. Simplified UML representation.

3.2.2 XML-based schema approach

For the EuroSTEP Model Server (EMS), the IFC Model Server by VTT/Secom (IMsvr) and the SABLE server, XML schemata have been defined for specifying the contents of extracted submodels. The schemata are denoted as query languages although they are not languages in the narrower sense of the word.

Product Model Query Language. The schema of the Product Model Query Language (ProMQL) of the EuroSTEP Model Server[6] provides the XML elements SELECT and WHERE which are used in a manner similar to the corresponding parts in an SQL query: SELECT elements define the entities to be included in the results, and WHERE elements define constraints. The constraints are expressed by an XML tree consisting of attribute names, comparison operators and values. Values are either atomic, or sub-trees where the corresponding attribute represents an association to another entity.

The specimen query represented by a Product Model Query Language XML document is presented in Figure 7.

[6] http://ems.eurostep.fi/PMQL/Doc/index.htm

```
<webstep ql>
  <select>
    <entity name="IfcWallStandardCase">
      <where>
        <attribute name ="hasAssociations">
          <eq>
            <entity name = "IfcRelAssociatesMaterial">
              <where>
                <attribute name ="relatingMaterial">
                  <eq>
                    <entity name = "IfcMaterialLayerSetUsage">
                      <where>
                        <attribute name ="forLayerSet">
                          <eq>
                            <entity name = "IfcMaterialLayerSet>
                              <where>
                                <attribute name ="materialLayers">
                                  <eq>
                                    <entity name = "IfcMaterialLayer>
                                      <where>
                                        <attribute name ="material">
                                          <eq>
                                            <entity name = "IfcMaterial">
                                              <where>
                                                <attribute name ="name">
                                                  <eq>
                                                     <string value="Concrete"/>
                                                  </eq>
                                                </attribute>
                                              </where>
                                            </entity>
                                          </eq>
                                        </attribute>
                                      </where>
                                    </entity>
                                  </eq>
                                </attribute>
                              <where>
                            </entity>
                          </eq>
                        </attribute>
                      </where>
                    </entity>
                  </eq>
                </attribute>
              </where>
            </entity>
          </eq>
        </attribute>
      </where>
    </entity>
  </select>
</webstep_ql>
```

Figure 7: The specimen query as XML instance of the Product Model Query Language schema.

Partial Model Query Language. The partial model query language (PartMQL) has been developed for the IFC Model Server of Secom Ltd. and VTT (Adachi, 2002). It follows the same basic concept as the Product Model Query Language, as its users define the resulting set by means of hierarchical constraints (Figure 8). The two components of a appropriate query language, the selection part and the recombination part, that are clearly separated in EXPRESS-X (WHERE part and RETURN part) and in the GMSD schema (selection part and view creation part), are mixed up in PartMQL: The *cascade* element that allows to navigate along associations serves both purposes, on the one hand using the values of “faraway” attributes as selection criteria, and on the other hand in- or excluding the respective attributes in the resulting view.

```
<pmql>
  <select type="entity" match="IfcWall" action="get">
    <cascades>
      <select type="inverse" match="HasAssociations" action="nop">
        <cascades>
          <select type="attribute" match="RelatingMaterial" action="nop"/>
            <cascades>
              <select type="attribute" match="ForLayerSet" action="nop"/>
                <cascades>
                  <select type="attribute" match="MaterialLayers" action="nop"/>
                    <cascades>
                      <select type="attribute" match="Material" action="nop"/>
                        <where>
                          <expr value="name='Concrete'"/>
                        </where>
                      </select>
                    </cascades>
                  </select>
                </cascades>
              </select>
            </cascades>
          </select>
        </cascades>
      </select>
    </cascades>
  </select>
</pmql>
```

Figure 8: The specimen query expressed as an instance of the Partial Model Query Language schema.

SABLE Model Query Language. The SABLE model query language[7] results from the merging of ProMQL and PartMQL and a further advancement. It introduces additional schema elements such as *include* and *except* for an advanced specification of the resulting object set by introducing the above mentioned separation between object selection and view definition. Since the language does not operate on the IFC product model, but on a simplified version of the model, we refrain from further examination.

3.2.3 Assessment of schema-based approaches

Schema-based approaches avoid the error-prone usage of a query language and the parsing of the query statement by the processing engine, and allow employing user-friendly input-masks for weaving an object-network that specifies the desired result. They can thus be seen as a suitable solution for the specification of sub-models by members of a planning team.

However, the scope of the schema-based approaches is limited to "pure" sub-model creation, since none of the presented schemata allows for a recombination of attributes from different entities, i.e. a schema mapping is not possible. Accordingly, the schema-based approaches have to be classified as less appropriate for analysis purposes.

At the same time, schema-based approaches are less suitable for power users, such as programmers, because the instantiation of the query objects and the assignment of values are much more tedious and time-consuming than the use of a query language. This biggest problem, however, is the proprietary character of the presented solutions: In each of the presented cases, an engine for processing instances of the particular schema has been implemented only by the creators of the schema.

[7] http://www.blis-project.org/~sable/

3.4 SQL-based approach

An obvious technical approach for realizing queries on product models is the utilization of relational database management systems. They provide the well-known query language SQL that is based on the sound foundation of relational theory (Codd, 1970). Besides its primary function as query language, SQL serves also as data definition language (DDL) and data manipulation language (DML).

Two problems hamper the utilization of relational databases for storing and querying product models: The first problem is that there is no standard that defines a mapping from EXPRESS to SQL-DDL. The second problem is that the SQL standard itself is only loosely implemented by the database providers, with the result that there is now a wide variety of SQL dialects, for each of which a different mapping would have to be provided. However, some researchers and tool implementers, have proposed mappings from EXPRESS to specific SQL dialects.

Before having a closer look on them, we indicate that there are two distinct versions of the SQL standard in use today: The purely relational version known as SQL-92, and the object-relational version known as SQL:1999 (International Organization for Standardization, 1999; Eisenberg & Melton, 1999). The latter enables an extension of the database type system in an object-oriented way, especially by providing abstract data types (ADTs) which even may possess member functions (methods) (Melton, 2003; Türker, 2003; Türker & Saake, 2006).

3.4.1 Relational SQL

Purely relational databases as defined by SQL-92 do not offer object-oriented features and have therefore a much more limited expressive power than EXPRESS[8]. Hence, the main challenge is to emulate EXPRESS features such as inheritance, associations, aggregations and SELECT types by means of simple relations.

The technical possibilities for these emulations, including the advantages and drawbacks of the individual solutions, are discussed in detail in (You, Yang, & Eastman, 2004). The mapping of the CIS/2 data model on a relational database is presented by way of an example. However, the fundamental concept of an EXPRESS-to-SQL compiler had been discussed some years before in (Bicarregui & Matthews, 1998). Another initiative that employs a "hand-made mapping" from EXPRESS to SQL-DDL is the German OKSTRA[9] effort that aims at developing a comprehensive product model for roadways and provides separate SQL-DDL files for all schemata that are developed.

A more general approach is being pursued by the STEPset group at the Institute for System Programming of the Russian Academy of Sciences[10]. The group has developed the generic tool *exp2ddl* that translates an EXPRESS schema into various SQL-DDL dialects. A second tool, *p21tosql*, is used to convert STEP-P21 instance documents into SQL-DML statements. Again one of the supported SQL dialects has to be chosen.

For each non-abstract entity of the passed schema *exp2ddl* creates a corresponding table that contains a separate column for each attribute of the entity (including the inherited ones). Inverse attributes are not mapped to SQL. An additional column contains the IDs of individual entities taken from the entity number of the STEP-P21 file (e.g. #24). Attribute columns that represent associations to other entities contain the entity number of the referenced instances.

[8] This is widely known as "impedance mismatch" between the object-oriented and the relational world.

[9] Objektkatalog für das Straßen- und Verkehrswesen, http://www.okstra.de

[10] http://www.ispras.ru/~step/

In case the association is an aggregation of multiple instances and thus modelled using a container (an array, list, set, or bag), the references are stored as entity numbers separated by commas. This results in high syntactical overhead when a SQL query is formulated that involves the aggregation along such a multi-value association, since the reference string must be manually decomposed. To avoid this, we recommend the introduction of a helper function (AGGR_CONTAINS) that checks whether a certain ID is contained within the list representing an aggregation.

In Figure 9 the specimen query defined in SQL operating on a database created by employing a EXPRRESS-to-SQL mapping is depicted. The navigation along the various involved entities to reach the decisive attribute representing the material's name is realized by means of subsequent joins of the respective relations. This results in a longer but still manageable query statement.

```
SELECT w
FROM  IfcWallStandardCase wall, IfcRelAssociatesMaterial assoc,
      IfcMaterialLayerSetUsage usage, IfcMaterialLayerSet set,
      IfcMaterialLayer layer, IfcMaterial mat
WHERE AGGR_CONTAINS(assoc.relatedObjects, wall.id)
  AND assoc.relatingMaterial = usage.id
  AND usage.forLayerSet = set.id
  AND AGGR_CONTAINS(set.materialLayers,layer.id)
  AND layer.material = mat.id
  AND mat.name ='Concrete'
```

Figure 9: The specimen query expressed in SQL-92 using the helper function AGGR_CONTAINS.

3.4.2 Object-Relational SQL

The SQL:1999 standard describes the access to databases providing *object-relational features* (International Organization for Standardization, 1999; Eisenberg & Melton, 1999). The main feature of object-relational databases is the possibility to extend the database type system in an object-oriented way enabling the user to define abstract data types (ADTs) which may posses attributes and relationships to other such ADTs (Melton, 2003; Türker, 2003; Türker & Saake, 2006). Using object-relational databases, most of the object-oriented features of EXPRESS can be directly mapped to the corresponding SQL:1999 elements (Urban, Tjahjadi, & Shah, 2000). Accordingly, the impedance mismatch is considerably reduced, which facilitates also the formulation of queries.

The formulation of the specimen query in SQL:1999 is depicted in Figure 10. The query is very similar to that formulated in EXPRESS-X (Figure 3), since SQL:1999 also allows the navigation along (multiple) references by applying the dot operator, which avoids applying additional joins as necessary when using SQL-92. Unfortunately, at the time of writing there is no official mapping from EXPRESS to SQL:1999 available. Also, there are no publicly available tools known to the authors that provide an automated conversion of EXPRESS schemata into object-relational performing this mapping.

```
SELECT *
FROM  IfcWallStandardCase wall;
WHERE wall.hasAssociations[1].relatingMaterial.forLayerSet.
      materialLayers[1].material.name = 'Concrete';
```

Figure 10: SQL:1999 version of the specimen query

3.5 XML-based approaches

The Extensible Markup Language (XML) was initially defined in 1998 and is consequently a young data modelling standard compared to STEP. It was originally designed to facilitate the exchange of semantic content in the World Wide Web, by clearly separating the content of websites from their visual representation (Abiteboul, Buneman, & Suciu, 1999). In the meantime, however, the general approach of XML, its clear design, the extensive documentation, and the availability of a large set of tools for creating, validating and processing XML documents has resulted in XML being a well-established data modelling language and data exchange format in almost all IT domains.

In response to the growing demand for interoperability between XML and STEP, the STEP standardization workgroup defined a mapping from EXPRESS to XML-DTD in 2003, and a mapping from EXPRESS to XML-Schema in 2007, published as Part28, Editions 1 and 2, respectively (International Organization for Standardization, 2007).

The IFC model support group applied the mapping to the IFC data model resulting in the corresponding ifcXML model (Nisbet & Liebich, 2005). Based on this mapping, the Helmholtz research center Karlsruhe has developed a tool for converting STEP-P21 instance files into ifcXML instance files. The existence of the ifcXML standard schema and the conversion tool creates extensive opportunities especially for the construction informatics research community, since it allows the application of a wide range of software tools available for XML processing which, in contrast to EXPRESS tools, are largely distributed as open source software.

However, since XML-Schema does not provide the full range of data modelling features of EXPRESS, the conversion results in a certain loss of model semantics. This particularly applies to *inverse attributes* that enable EXPRESS users to navigate along associations in the opposite direction (see Section 2.1), but are not mapped to XML.

There were several candidate languages for querying XML, of which XQuery has prevailed (Boag et al., 2007). The basic syntax of XQuery is depicted in Figure 11; it consists of a combination of XML, XSLT and SQL language elements.

```
FOR $forvar1 at $posvar1 in <Expr>, $forvar2 at $posvar2
    in <Expr> ...
LET $letvar1 := <Expr>, $letvar2 := <Expr> ...
WHERE <BoolExpr>
ORDER by <Expr> ascending/descending
RETURN <Expr>
```

Figure 11: General structure of an XQuery statement

The general structure of XQuery is denoted as FLOWR, which is derived from the first letters of its FOR, LET, WHERE, ORDER BY and RETURN parts. The FOR part is the equivalent of the FROM part in SQL, here the element types are specified on which the search algorithm iterates, and to which variables are bound. In the LET part, additional variables can be bound, but their values are fixed, i.e. they do not provoke iteration loops. The WHERE part contains the conditional statement, the ORDER BY part provides means of ordering the resulting set, and the RETURN part describes its data structure. Figure 12 shows the specimen query expressed in XQuery.

```
let $uos := $this/ex:iso 10303 28/ifc:uos
for $wall     in $uos/ifc:IfcWallStandardCase,
    $assoc    in $uos/ifc:IfcRelAssociatesMaterial,
    $usage    in $uos/ifc:IfcMaterialLayerSetUsage,
    $set      in $uos/ifc:IfcMaterialLayerSet,
    $layer    in $uos/ifc:IfcMaterialLayer,
    $material in $uos/ifc:IfcMaterial
where $wall[@id=$assoc/ifc:RelatedObjects/ifc:IfcWallStandardCase/@ref]
  and $assoc/ifc:RelatingMaterial/ifc:IfcMaterialLayerSetUsage[@ref=$usage/@id]
  and $usage/ifc:ForLayerSet/ifc:IfcMaterialLayerSet[@ref=$set/@id]
  and $set/ifc:MaterialLayers/ifc:IfcMaterialLayer[@ref=$layer/@id]
  and $layer/ifc:Material/ifc:IfcMaterial[@ref=$material/@id]
return <QueryResult>{$wall}</QueryResult>
```

Figure 12: The specimen query expressed in XQuery

Although STEP P28 allows the mapping of inverse attributes to XML, the current configuration of the official mapping from IFC to ifcXML does not include them. In addition, the current standards of XQuery and XPath do not support the dereferencing of references. Due to these facts, the “navigation” to the decisive attributes must be realized using joins on the involved entities, rendering the query much more complicated than the EXPRESS-X statement shown in Figure 3.

However, XQuery provides the full range of query functionalities, including filtering and recombination of attributes for the resulting set. Thanks to the broad application area, there is a large number of XQuery engines available, among which many are provided as public domain software. Hence XQuery can be seen as a very suitable option for providing query support for building information models, even if its syntax is not as concise as that of EXPRESS-X.

What currently poses a serious problem with respect to the creation of partial models and their subsequent processing is the absence of any standardized back-mapping that converts a valid ifcXML file into a STEP-P21 representation. The respective software application that further processes the retrieved sub-model must therefore be able to read-in the ifcXML format.

3.6 Conclusion on semantic query functionalities

For querying EXPRESS-based Building Information Models using semantic criteria, the application of EXPRESS-X is in general the best choice, since it fully supports all of the EXPRESS language constructs leading to concise query statements. Since however, the number of available EXPRESS-X engines is very limited, alternative approaches have to be seriously considered.

Among them, SQL:1999 and XQuery form the most suitable option. The query language SQL:1999 that is used in object-relational databases provides a rich set of object-oriented language constructs, thus a mapping from EXPRESS to it can be realized with a limited “impedance mismatch”, resulting again in a concise query statement. Though there is a large variety of commercial and non-commercial object-relational databases available on the market, so far there is no standardized mapping from EXPRESS to SQL:1999.

By contrast, there exists an official mapping from EXPRESS to XML, and, even more specifically, an official ifcXML representation of IFC building models, which makes the application of XQuery an interesting option. XQuery provides most of the required functionality of a query language, but so far it does not support the navigation along references. Hence, querying faraway attributes requires the application of joins, which results in a much more complicated formulation of theses queries. XQuery processing engines are available in a large extent under both commercial and public licenses, however execution performance is a serious issue.

4 REALIZATION OF SPATIAL QUERIES

4.1 Motivation

Humans view buildings primarily as an aggregation of physical objects with well-defined geometry and specific spatial relations. In most cases, the architectural and/or structural function of a particular building component is closely related to its shape and its position in relation to other building components. For architects and engineers involved in designing buildings, geometric properties and spatial relations between building components accordingly play a major role in finding solutions for most of the design and engineering tasks.

Some of the spatial relationships which are possibly of interest for the user are directly represented in the building product model. In the case of the IFC, some examples of these predefined relationships are *IfcRelFillsElement, IfcRelVoidsElement* and *IfcRelContainedInSpatialStructure*. Unfortunately, many product modelling tools do not fill the entire set of spatial relations with appropriate data when exporting a building model into the IFC format. In a recently conducted test, we used the commercial BIM design tool Autodesk Revit 2008 to model a high-rise building completely equipped with interior fittings. The analysis of the exported IFC file showed that, while the *IfcRelFillsElement* and *IfcRelVoidsElement* relationships between walls and windows were set correctly, no *IfcRelContainedInSpatialStructure* relationships had been set. Accordingly it was not possible to query the product model for the furniture contained in a certain room or office, for example.

Other spatial relationships, such as directional relations (e.g. one object above another) are completely ignored by the IFC product model. From the point of view of product modelling this makes absolute sense, because storing all possible spatial relations would (1) result in huge models with many object relations not needed by the majority of applications and (2) introduce even more redundancy, since directional relationships are already implicitly defined by the shape and position of the respective objects. For this reason, we follow an analytic approach here, where spatial relations are not required to be set by the modelling tool in use, but are derived from the objects’ shapes and positions, i.e. the explicit geometry of the building’s components, instead.

Since product model servers do not know the geometric implications of semantic attributes, they are not able to interpret and process spatial information. This has to be seen as a major deficiency, since spatial relations between building components play a significant role in most of the design and engineering tasks of the AEC domain. To fill this technological gap we have developed concepts and techniques for a 3D Spatial Query Language for Building Information Models. It makes it possible to select specific building components fulfilling spatial constraints defined by the user.

4.2 Related work

The overall concept of providing a Spatial Query Language for analyzing Building Information Models is closely related to concepts and technologies developed in the area of Geographic Information Systems (GIS). Such systems maintain geographical data, such as the position and shape of cities, streets, rivers etc. and provide functionalities for the spatial analysis of this data. Due to the nature of this domain most GI systems only support spatial objects in two-dimensional space.

The first implementations of spatial query languages on the basis of SQL were realized in the GIS context. In the late 80's, a multitude of different dialects was developed, including PSQL (Roussopoulos, Faloutsos, & Sellis, 1988), Spatial SQL (Egenhofer, 1987), GEOQL (Ooi, Sacks-Davis, & McDonell, 1989), KGIS (Ingram & Phillips, 1987) and TIGRIS (Herring, Larsen, & Shivakumar, 1988). A good overview of the different dialects and the basic advantages of a SQL-based implementation is provided in (Egenhofer, 1992).

The GIS research community also coined the phrase *Spatial Database* to describe database management systems (DBMS) that provide spatial data types and spatial indexing techniques and thus allow for an easy and efficient access to spatial data (Rigaux, Scholl, & Voisard, 2002; Shekhar & Chawla, 2003). There is now a wide range of commercial 2D spatial database systems, the most widespread ones being *PostGIS*, *Oracle Spatial* and *Informix Geodetic Datablade*. The majority of spatial databases that are available comply with the standard set up by the OpenGIS consortium that defines a common interface for accessing 2D spatial data and accordingly enables the exchangeability of the database component in an overall GI system (OpenGIS Consortium, 1999).

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

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

In the 3D-GIS research community, the main interest lies in the modelling of the ground surface, buildings and infrastructure as well as the subsoil layers. The most important works in this area include (Breunig, Bode, & Cremers, 1994; Breunig, Cremers, Müller, & Siebeck, 2001; Balovnev et al., 2004) which report on the development of *GeoToolkit*, an object-oriented framework for efficiently storing and accessing 3D geographic and geologic data. The main disadvantage of using the framework for analyzing building models is the need to model all spatial entities according to the mathematical concept of *simplicial complexes*. The obligatory conversion of a boundary representation, as used in CAD tools, to a *simplicial complex* representation is expensive and, in some special cases, absolutely unfeasible. A more flexible, yet theoretic approach for applying algebraic topology on building models is presented in (Paul & Bradley, 2003; Paul, 2010).

In (Zlatanova, Rahman, & Shi, 2004; Zlatanova, 2006; Coors, 2003; Arens, Stoter, & Oosterom, 2005) concepts and data structures for storing 3D city models in spatial databases are presented and the suitability of different geometry models for querying topological relationships is discussed. In general, GIS research follows the approach of choosing geometry data structures that implicitly contain topological relationships. Accordingly many of the proposed data structures rely on a simplicial decomposition of the space (Egenhofer, Frank, & Jackson, 1989; Egenhofer & Herring, 1992; Shi, Yang, & Li, 2003).

In (Kriegel, Pfeifle, Pötke, Renz, & Seidl, 2003) a database system is introduced that allows for the spatial analysis of 3D CAD models. It provides simple volume, collision and distance queries, but supports neither topological nor directional predicates. The implementation of the system relies on a voxel approximation of the CAD parts stored in the database and a special index structure optimized for this representation. We follow a similar approach here but employ hierarchical space partitioning data structures which are created dynamically while processing a spatial query.

4.3 Spatial Algebra

4.3.1 Spatial Types

The proposed 3D Spatial Query Language relies on a spatial algebra that is formally defined by means of point-set theory and point-set topology (Borrmann, van Treeck, & Rank, 2006; Borrmann, 2006, 2007). Besides the fully three-dimensional objects of type *Body*, the algebra also provides abstractions for spatial objects with reduced dimensionality, namely by the types *Point*, *Line* and *Surface*.

This is necessary because building models often comprise dimensionally reduced entities, such as load points, power lines, plates, slabs etc. All types of spatial objects are subsumed by the super-type *SpatialObject*.

For the specification of topological predicates it was necessary to define the *interior*, the *boundary* and the *exterior* for each of the 4 spatial types defined within the Spatial Query Language. These definitions have been published in (Borrmann, 2006; Borrmann, Schraufstetter, van Treeck, & Rank, 2007) and are not repeated here. In summary, the purpose of these definitions is to transfer the specification of *interior* and *exterior* for each of the spatial types from the world of algebraic topology to the world of point-set topology. For example, we intend to define the endpoints of a *Line* element in 3D as its *boundary*, and all other points as belonging to its *interior*. Applying the neighbourhood concept from point-set topology in 3D space would result in all points of a Line belonging to its boundary. We accordingly define *Line* objects as mappings from 1D to 3D space, specify the boundary an the interior points in 1D, and assign "boundary" to the mappings of boundary points and "interior" to the mapping of interior points (Figure 13). As result we are able to distinguish interior from boundary points also for "flat" objects in 3D space.

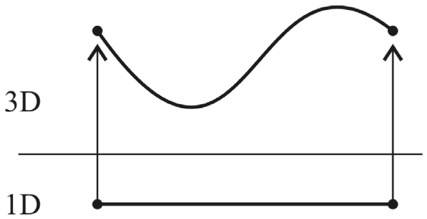


Figure 13: If 3D Line *objects are defined as mappings from 1D intervals, the notion of* boundary *can be preserved for the endpoints.*

4.3.2 Spatial Operators

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

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

operators.

Colloquial language is often vague and ambiguous when used to describe spatial relationships. Because an unequivocal definition is essential for using spatial relationships as conditions in a spatial query language, it is necessary to formally specify their semantics. These formal specifications are presented in the following Sections 4.3.3, 4.3.4 and 4.3.5.

4.3.3 Metric operators

All metric operators of the spatial query language rely on the Euclidean metric defined in 3D space. Let $p(x_p,y_p,z_p)$ and $q(x_q,y_q,z_q) \in \mathbb{R}^3$, then the Euclidean distance between p and q is defined as:

$$d(p,q) := \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2 + (z_p - z_q)^2}$$

The operator *distance* returns the minimal distance between two spatial objects as a real number. Let A and B be objects of type *Spatial* and $a \in A,\ b \in B$. Then *distance* is formally defined as follows:

$$distance(A,B) := \min_{a,b}(d(a,b))$$

The operator *distance* returns 0, if the operands touch or penetrate each other.

The operator *maxdist* can be used to determine the maximum distance between two spatial objects. It also returns a real value and is defined as follows:

$$maxdist(A,B) := \max_{a,b}(d(a,b))$$

The operators *isCloser* and *isFarther* are based on the minimal distance. Operands of both operators are two spatial objects A and B as well as a positive real value c. The operators return a Boolean value and are formally defined as follows:

$$isCloser(A,B,c) \Leftrightarrow \min_{a,b}(d(a,b)) < c$$

$$isFarther(A,B,c) \Leftrightarrow \min_{a,b}(d(a,b)) > c$$

These operators can be used to select objects that are inside or outside a buffer zone around the reference object.

The operator *diameter* returns the maximum distance between two points of one individual object. The operand is a spatial object *A*, the return value a real number. Let *a, b* ∈ *A*. Then the diameter is defined as:

$$diameter(A) = \max_{a,b}(d(a,b))$$

Figure 14 illustrates the semantics of the definitions by means of an example.

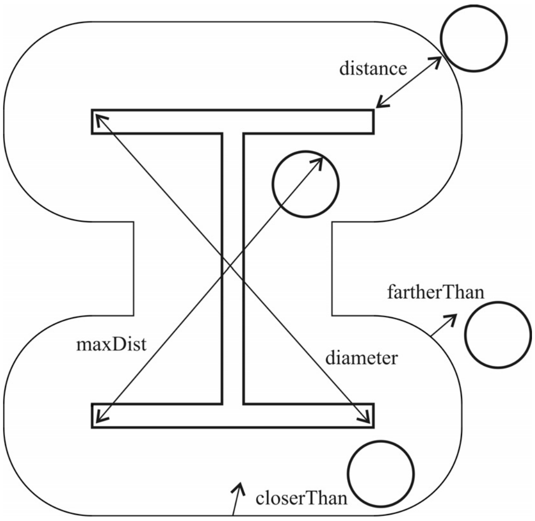


Figure 14: The semantics of the metric operators provided within the spatial query language.

4.3.4 Directional operators

Direction is a binary relation of an ordered pair of objects *A* and *B*, where *A* is the reference object and *B* is the target object. The third part of a directional relation is formed by the reference frame, which assigns names or symbols to space partitions.

According to (Retz-Schmidt, 1988), three types of reference frames can be distinguished: an *intrinsic* reference frame relies on the inner orientation of the spatial objects, such as that defined by the front of a building, for example. A *deictic* reference frame is aligned to the position and orientation of the observer. By contrast, an *extrinsic* reference frame is defined by external reference points. In geographical applications, for example, these external reference points are the earth's north and south pole.

In a geographical context, we usually distinguish between four *(north, east, south, west)* or eight space partitions *(north, north-east, east, southeast, south, south-west, west, north-west)*. In 3D context, normally the additional directional predicates *above* and *below* are used (Fuhr, Socher, Scheering, & Sagerer, 1998), which may also be employed in conjunction with the aforementioned 2D sub-direction, resulting in *north-east-above, east-above*, etc.

To meet the requirements of different application scenarios, we developed two new models for representing directional relationships between 3D objects: the *projection-based model* and the *halfspace-based model*. Both models use an intrinsic reference frame that is determined by the orientation of the coordinate system chosen by the user.

The proposed directional models are appropriate for arbitrary combinations of spatial types and are based on a separate examination of directional relationships with respect to the three coordinate axes. For each axis, there are precisely two possible relations: *eastOf* and *westOf* in the case of the *x*-axis, *northOf* and *southOf* for the *y*-axis and *above* and *below* for the *z*-axis. We haven chosen the names of geographical cardinal directions instead of *left, right, in front of, behind* to clearly label our models as observer-independent.

As opposed to the directional models used in (Guesgen, 1989; Papadias, Sellis, Theodoridis, & Egenhofer, 1995) and (Goyal, 2000), the directional relationships of the relevant axis are not superimposed. Accordingly, the relationship between two spatial objects is not *north-east*, for example, but *northOf* **and** *eastOf*.

Both models differentiate between two "flavours" of directional operators. Whereas the *strict* directional operators only return *true* if the entire target object falls into the respective directional partition, the *relaxed* operators also return *true* if only parts of it do so.

The projection-based directional model. In the projection-based model, the reference object is extruded along the coordinate axis corresponding to the directional operator. The target object is tested for intersection with this extrusion. Let reference object *A* and target object *B* be spatial objects of type *SpatialObject* and $a \in A$, $b \in B$. Then the formal definitions of the relaxed projection-based operators read:

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

The relaxed operators return *true* if there is an intersection between the extrusion body and the target object, otherwise *false*. By contrast, the *strict* projection-based operators only return *true* if the target object is completely within the extrusion body. Accordingly, the formal definitions of the strict operators are:

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

Figure 15 illustrates the consequences of these definitions. In colloquial language, the semantics of the operator *above_proj_strict*, for example, could be described as "directly above" or "exceptionally above". In Figure 16 the diverging semantics of the different directional operators are illustrated by a practical example.

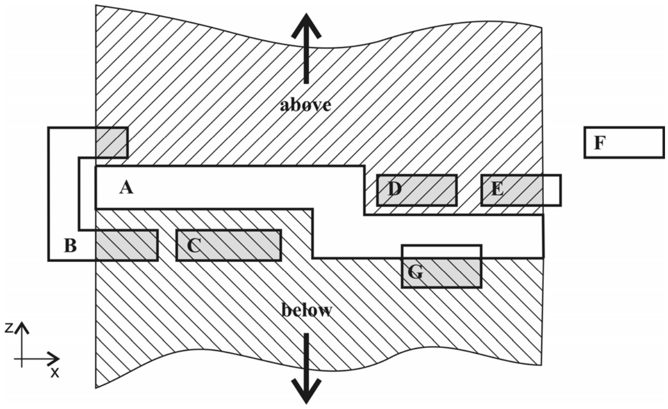


*Figure 15: The projection-based directional model relies on the extrusion of the reference object (*A*) along the respective coordinate axis. In the illustrated example, the relaxed operator* above_proj_relaxed *returns* true *for the target objects* B, D, E *and* G, *but* false *for any other target object. By contrast, the strict operator* above_proj_strict *also returns* false *for* B, G *and* E.

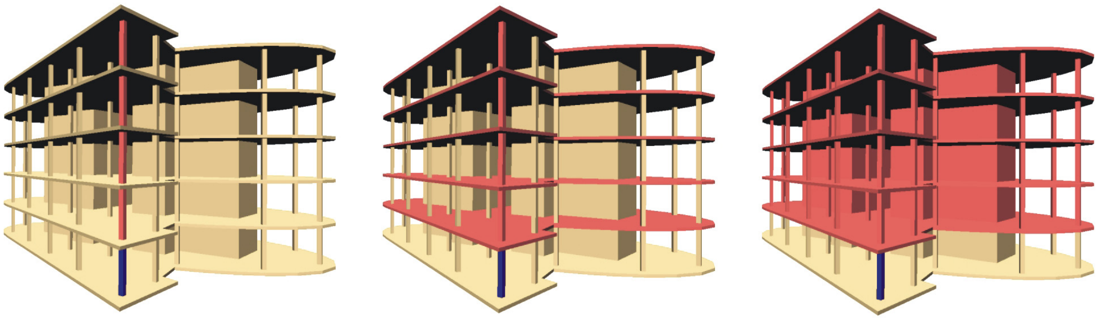


Figure 16: Example illustrating the diverging semantics of the different directional operators. In each case, the reference object is depicted in blue, whereas the resulting set identified by the particular operator is depicted in red. Left: above_proj_strict*. Middle:* above_proj_relaxed*. Right:* above_hs_relaxed*. Please note that, in this example, the result of* above_hs_strict *is equal to that of* above_hs_relaxed*.*

The halfspace-based model. The second model is based on halfspaces that are described by the reference object's axis-aligned bounding box (AABB). In this model, the target object is tested for intersection with the halfspace corresponding to the directional predicate. In analogy to the projection-based model, we distinguish strict and relaxed operators. The formal definitions of the relaxed operators are:

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

For the relaxed operators to return *true* it is sufficient if parts of the target object are within the relevant halfspace. By contrast, the *strict* operators only return *true* if the target object is completely within that halfspace. The formal definitions of the strict operators accordingly read:

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

The examples in Figure 17 illustrate the consequences of these definitions.

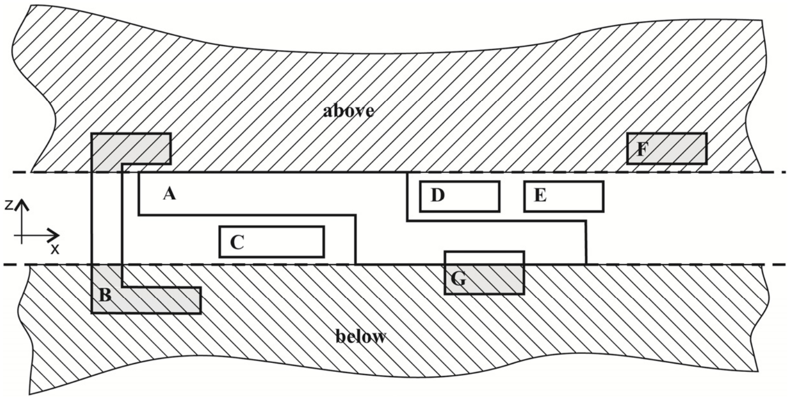


Figure 17: In the halfspace-based directional model the direction tiles are formed by halfspaces defined by the reference object's axis-aligned bounding box. In the given example, A *is the reference object. The* relaxed *operator* above_hs_relaxed *returns* true *for the target objects* B *and* F*, the* strict *operator* above_hs_strict *only returns* true *for* F.

4.3.5 Topological operators

Topological operators are used to query the topological relationship between two spatial entities. We distinguish *topological predicates* that represent a certain topological relationship and return a Boolean value indicating whether the operands show this relationship or not, and *selective topological operators* that return the predicate which is fulfilled by the operands.

Topological relationships can be formally described as follows (Clementini & Di Felice, 1995): Let X and Y be topological spaces. A mapping $f : X \rightarrow Y$ is continuous if for each open subset V of Y the set $f^{-1}(V)$ is an open subset of X. If the mapping f is a bijection and both f and f^{-1} are continuous, then f is called a *topological isomorphism*. Topological isomorphisms conserve neighbourhood relationships between points during the mapping. Typical isomorphisms include translation, rotation and scaling (*scaleFactor*≠0) as well as any combination of these transformations. Topological relationships are those relationships that are invariant under a topological isomorphism.

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

The first substantial step towards a formalization of topological relationships was the development of the *4-intersection model* by Egenhofer et al. (Egenhofer & Herring, 1990; Egenhofer & Franzosa, 1991). To formally specify the semantics of topological predicates the model determines the intersections between the interior and the boundary of the first object and the interior and the boundary of the second object as an empty or a non-empty set. In theory there are 16 possible configurations but, depending on the dimensionality of the geometric object in question, only a subset can be found in reality.

The intersection concept was first applied to intervals in one-dimensional space (Pullar, 1988) and later extended to cover also relations between simple regions in $\mathbb{R}^2$ (Egenhofer & Franzosa, 1991). In both cases, 8 different relations to which the natural language denominations *disjoint, touch, equals, inside, contains, covers, coveredBy* and *overlap* could be assigned, were distinguished.

To resolve topological relations between line elements in $\mathbb{R}^2$ more precisely, the 4-intersection model has been upgraded to the 9-intersection model (9-IM) by incorporating the exteriors of both operands (Egenhofer & Franzosa, 1991). The resulting nine intersections are recorded in a 3 × 3 matrix:

$$I_9(A,B) = \begin{bmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{bmatrix}.$$

A° denotes the *interior*, ∂A the *boundary* and A^- the *exterior* of the spatial object A (see Section 4.3.5). The 9-IM can also be applied for combinations of spatial objects with a different dimensionality (Egenhofer & Herring, 1992). Figure 18 shows the 9-IM matrices of the eight topological predicates defined by Egenhofer and depicts examples for 2D regions.

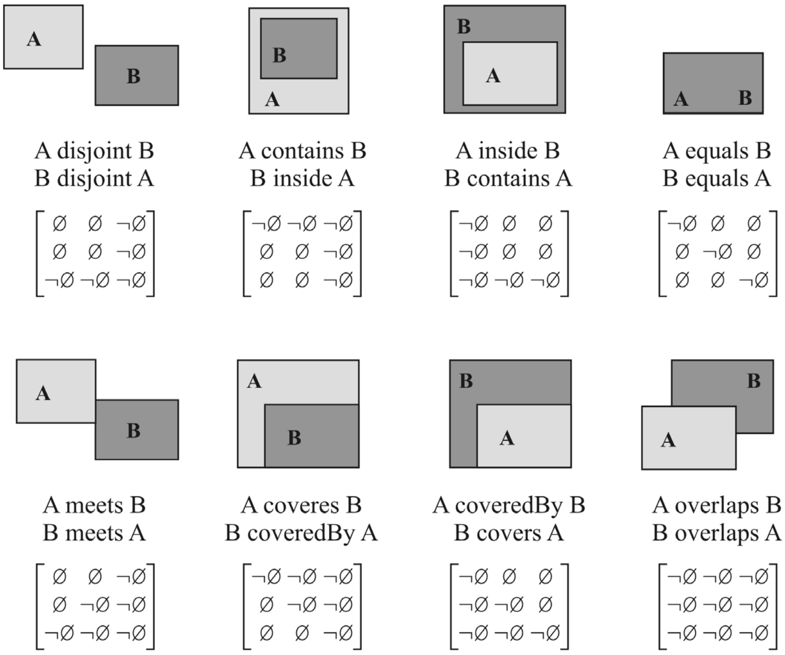

A
B
A disjoint B
B disjoint A
$\begin{bmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$
B
A
A contains B
B inside A
$\begin{bmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{bmatrix}$
B
A
A inside B
B contains A
$\begin{bmatrix} \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$
A B
A equals B
B equals A
$\begin{bmatrix} \neg\emptyset & \emptyset & \emptyset \\ \emptyset & \neg\emptyset & \emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{bmatrix}$
A
B
A meets B
B meets A
$\begin{bmatrix} \emptyset & \emptyset & \neg\emptyset \\ \emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$
A
B
A coveres B
B coveredBy A
$\begin{bmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \neg\emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \end{bmatrix}$
B
A
A coveredBy B
B covers A
$\begin{bmatrix} \neg\emptyset & \emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$
B
A
A overlaps B
B overlaps A
$\begin{bmatrix} \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{bmatrix}$

Figure 18: The 9-IM matrices originally defined by Egenhofer et al. for 2D space.

One drawback of the 9-IM is that some topological configurations that are intuitively different result in the same 9-IM matrix while others that are intuitively identical are treated as being different. The first problem is partially solved by the *Dimensionally Extended 9-Intersection Model* (DE-9IM) which additionally records the dimensionality of the intersection set (Clementini & Di Felice, 1995).

The DE-9IM forms the basis for the formal definitions of topological relationships in the OGC standard (OpenGIS Consortium, 1999). Here, *F* (false) is used in the matrices to denote an empty set, *T* (true) to denote a non-empty set, numbers may be used to define the dimensionality of the intersection set and, in addition, the wildcard (*) may be used at certain places in the matrix that are not relevant for the particular predicate, thereby solving the second of the aforementioned problems. Using this extended set of symbols, the OGC defines the predicates *contains, within, cross, disjoint, equals, intersect, touch* and *overlaps* for arbitrary combinations of (simple) *point*, *line* and *polygon* objects in 2D space.

An important pre-requisite for applying the 9-IM or its derivates is the formal specification of the *interior/boundary/exterior* of spatial objects. We can generally distinguish two different approaches to realize this: The first approach relies on algebraic topology using *cellular complexes* (Egenhofer & Herring, 1992) or *simplicial complexes* (Egenhofer et al., 1989; Breunig et al., 1994) to model spatial entities. This implies a complete partitioning of the entire space in a rigorously formal way and thus requires appropriate modelling tools, since conventional B-Rep modellers do not provide these capabilities.

The second approach relies on point-set topology (Egenhofer & Franzosa, 1991; Schneider & Weinrich, 2004; Borrmann et al., 2006): Here, *interior, boundary* and *exterior* are understood as point sets. The boundary point set is formed by points whose neighbourhood (a well-defined concept of point-set topology) contains both exterior and interior points. This concept can be easily applied to conventional B-Rep models (Borrmann et al., 2007). Special care has to be taken with dimensionally reduced entities in order to avoid all points becoming boundary points (see Section 4.3.1).

From the 3D GIS domain, there are a number of publications defining topological relationships by either applying the 9-IM (Oosterom, Vertegaal, Hekken, & Vijlbrief, 1994; Zlatanova, 2000) or the DE-9IM (Wei, Ping, & Jun, 1998). Unfortunately, these definitions are unsuitable for the application in the building model query language context, because they either rely on a cellular decomposition of space, or result in a very large number of topological predicates. For example, in (Zlatanova, 2000) 38 surface-surface relations are identified. This differentiation between topological constellations is much too fine-grained, since it is impossible to find equivalents in human language for each of the constellations, which is required to provide meaningful operators for the query language. For this reason, the authors have decided to set up own definitions.

For our definitions, we use the pure *9-Intersection Model* instead of the dimensionally extended version, because the *dimension* operator cannot be realized by means of the octree implementation technique presented in Section 4.4.1. In order to avoid an unmanageably large number of different topological predicates, we apply the clustering method, as proposed by (Schneider & Behr, 2006), which makes it possible to place wildcards (*) at those places in the 9-IM matrix that are not decisive for assigning a predicate to a certain constellation.

Besides showing the topological predicates provided within our Spatial Query Language, Figures 19 and 20 also present the corresponding 9-IM matrices and illustrate their semantics for different combinations of types in the form of pictograms. The given system of topological predicates fulfils the demands of *completeness* and *mutual exclusiveness*, i.e. we assign to any topological constellation exactly one of the predicates. This enables the introduction of an additional operator which returns the topological predicate for any given pair of spatial objects. This operator is called *whichTopoPredicate*.

There are several minor differences compared with the definitions given in (Schneider & Behr, 2006) with respect to the clustering of predicates: The predicates *coveredBy* and *cover* have not been adopted, because in the application domain considered here, it is normally irrelevant whether only the interiors of the operands overlap or whether their boundaries also overlap. Accordingly, these two constellations are subsumed under *within* and *contains*, respectively. In addition, the designation *touch* has been used instead of *meet* in order to gain a maximum compliance to the OGC standard.

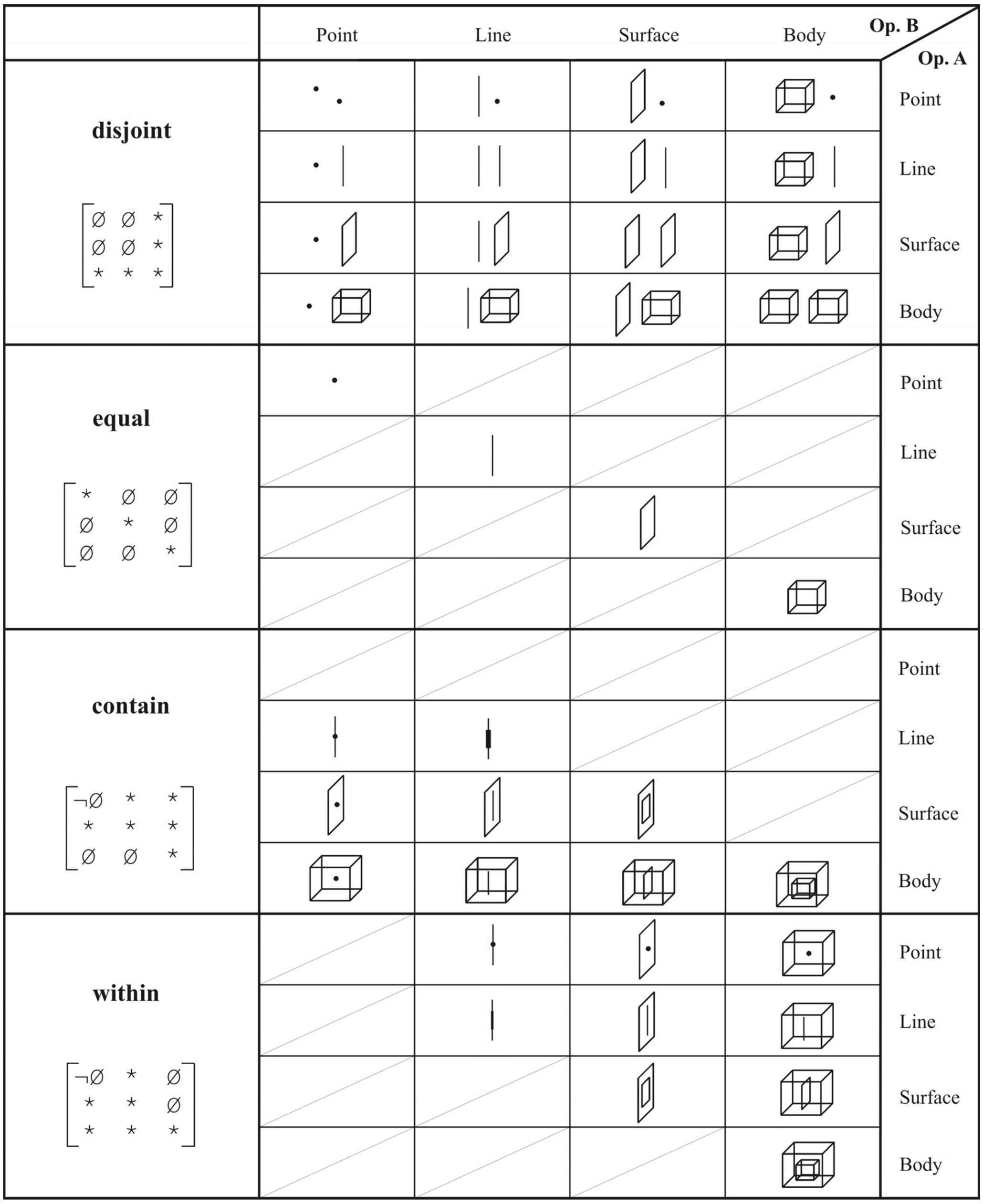


Figure 19: The topological predicates provided by the Spatial Query Language (part 1).

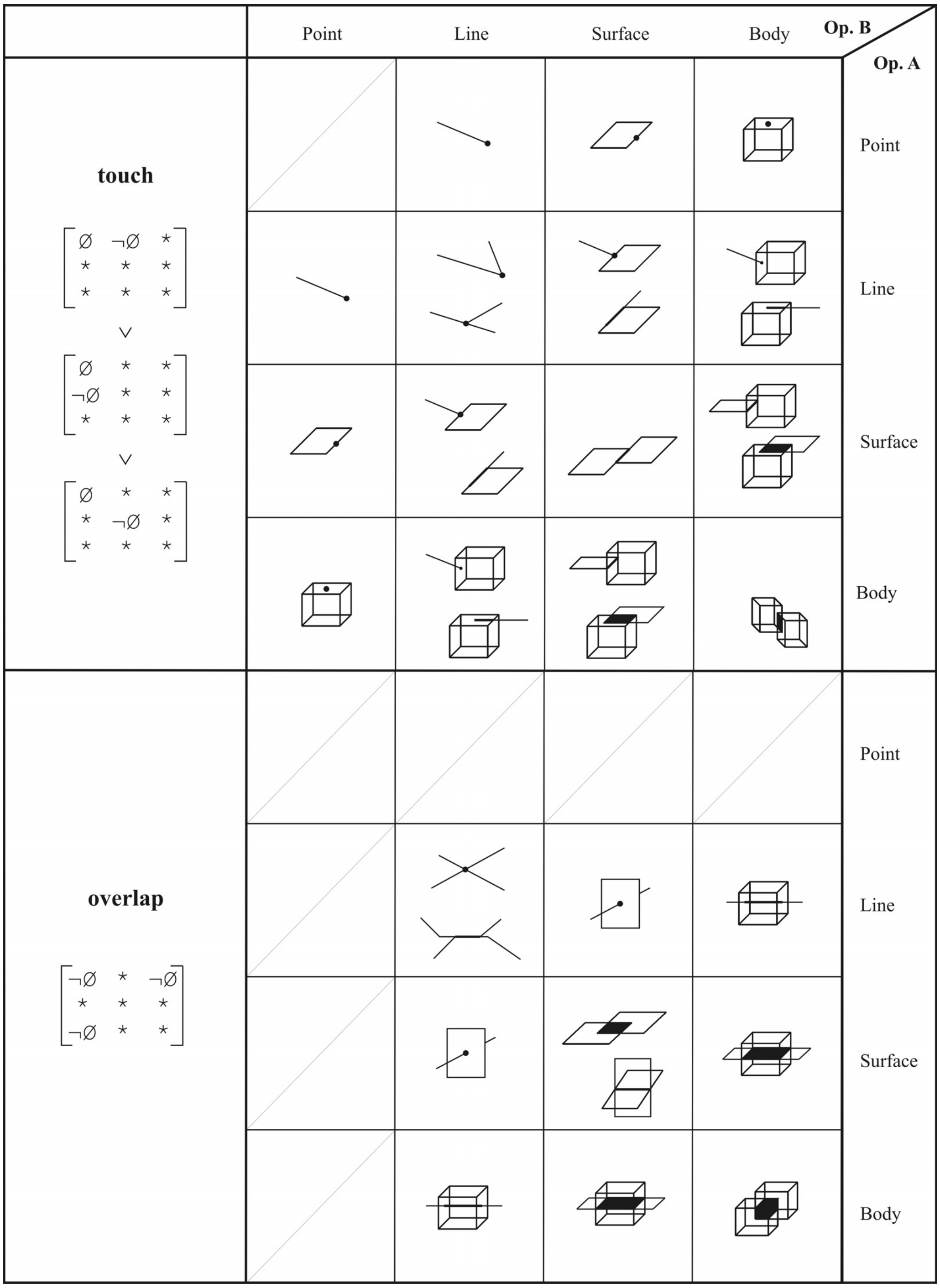

Op. B
Op. A
Point
Line
Surface
Body
Point
Line
Surface
Body
touch
$\begin{bmatrix} \emptyset & \neg\emptyset & * \\ * & * & * \\ * & * & * \end{bmatrix} \vee \begin{bmatrix} \emptyset & * & * \\ \neg\emptyset & * & * \\ * & * & * \end{bmatrix} \vee \begin{bmatrix} \emptyset & * & * \\ * & \neg\emptyset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

Figure 20: The topological predicates provided by the Spatial Query Language (part 2).

4.4 Implementation of spatial operators

For implementing the spatial operators the authors developed two different approaches. The first approach is based on a representation of the operands' geometry as an octree or an octree derivate. The second approach, on the other hand, uses the original boundary representation of the operands, applying more traditional algorithms from computational geometry.

Both approaches have advantages and disadvantages: Whereas the B-Rep-based implementation generally performs faster and returns a more precise result, the octree-based approach allows for a fuzzy handling of metric, directional and topological relationships. In some application areas such a fuzzy consideration is closer to users' requirements than an exact one.

4.4.1 Octree approach

The octree-based implementation of the spatial operators is discussed in detail in (Borrmann, Schraufstetter, & Rank, 2009), (Borrmann & Rank, 2009), and (Borrmann & Rank, 2009a). Here we will only give an overview.

The octree is a space-dividing, hierarchical tree data structure for the discretized representation of 3D volumetric geometry (Hunter, 1978; Jackins & Tanimoto, 1980; Meagher, 1982; Samet, 1985). Each node in the tree represents a cubic cell (an octant) and is either *black*, *white* or *gray*, symbolizing whether the octant lies completely *inside*, *outside* or on the *boundary* of the discretized object (Figure 21). Whereas black and white octants are branch nodes, and accordingly have no children, gray octants are interior nodes that always have eight children. The union of all child cells is equal to the volume of the parent cell, and the ratio of the child cell's edge length to that of its father is always 1:2. The equivalent of the octree in 2D is called quadtree.

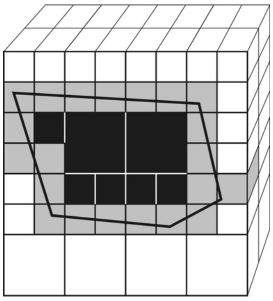


Figure 21: Cross-section through an octree. White *cells represent the* exterior*,* black *cells the* interior *and* gray *cells the* boundary *of the discretized object. Whereas* black *and* white *cells are branch nodes,* gray *cells always have eight children.*

In the implementation concept followed here, each spatial object is represented by an individual octree. There are several different approaches for generating an octree out of the object's boundary representation, most of which are based on a recursive algorithm that starts at the root octant and refines those cells that lie on the boundary of the original geometry, i.e. those which are coloured *gray*.

For our implementation we use the creation method developed by Mundani (Mundani, Bungartz, Rank, Romberg, & Niggl, 2003; Mundani, 2005) that is based on the halfspaces formed by the object's bounding faces. In Mundani's approach, the colour classification is based on a simple evaluation of the plane equation of each halfspace for the respective octant and a subsequent combination using Boolean expressions. Accordingly, the algorithm automatically marks inner cells as *black* without the need to perform a computationally expensive filling algorithm. As described in the next sections, the existence of *black* cells is an important prerequisite for the applicability of numerous rules in the algorithms implementing topological and directional relationships.

To cover dimensionally reduced entities with our algorithms, as well, we had to introduce the fourth colour *black/white*. Black/white cells represent areas where the exterior and the interior of the described object exist, but not its boundary (Figure 22).

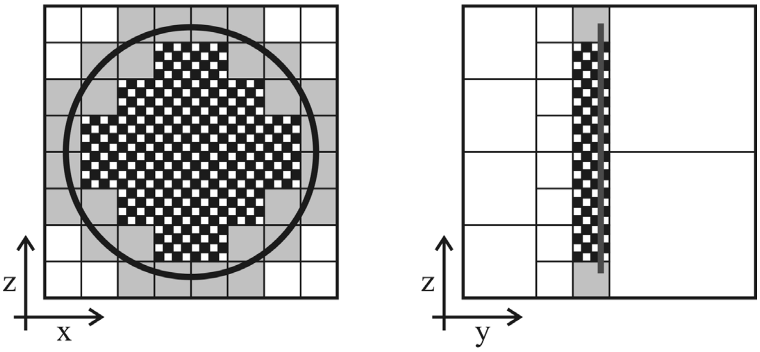


Figure 22: Dimensionally reduced objects like the disc shown here are discretized using the fourth colour black/white *that represents cells which contain interior and exterior points, but no boundary points.*

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

The basic element of a slot-tree is the *slot*. A slot of level *k* is formed by the extrusion of a level *k* cell along the examined axis (*x*, *y* or *z* according the definitions in Section 4.3.4). It contains all cells which intersect with this extrusion. If we take a look at the *z*-direction, for example, a slot contains all the cells that lie above one another (Figure 23). It accordingly possesses a list of octants in the order of their appearance. The octants may stem from different levels of the octree, and consequently may have different sizes (Figure 24). This also means that one octant might appear in the list of different slots. Introducing the slot data structure allows for the application of simple tests based on the colour and absolute position of the cells contained therein in order to decide whether the directional predicate under examination is fulfilled, or not.

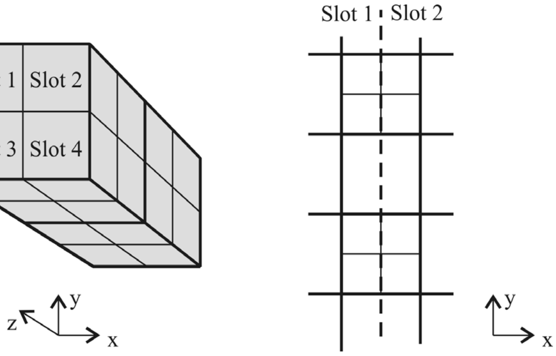


Figure 23: Slots in 3- and 2-dimensional space, respectively. A slot in z*-direction contains all the cells that lie above one another.*

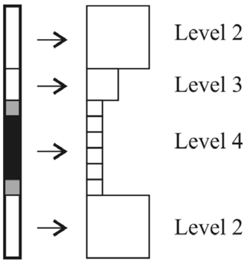


Figure 24: A slot in 2D that owns cells from different levels of the underlying quadtree (Slot 1212 in Figure 25).

In analogy to the octree, the slot-tree organizes the slots in a hierarchical manner. Each node in a 3D slot-tree has either 4 or no children, depending on whether the corresponding slot contains *gray* octants. A slot-tree may be directly derived from an existing octree representation, or generated on-the-fly while processing the algorithm of the directional operator. The procedure is illustrated in Figure 25. Traversing the octree from the top downwards in a breadth-first manner, we proceed to build up the slot-tree, generating child slots and inserting them into the slot-tree, as required. Such a refinement is necessary if at least one cell in the current slot is *gray*. By coupling the generation of octree and slot-tree with the processing of the directional operator, it is possible to avoid unnecessary refinements at places of no relevance for the operator's results.

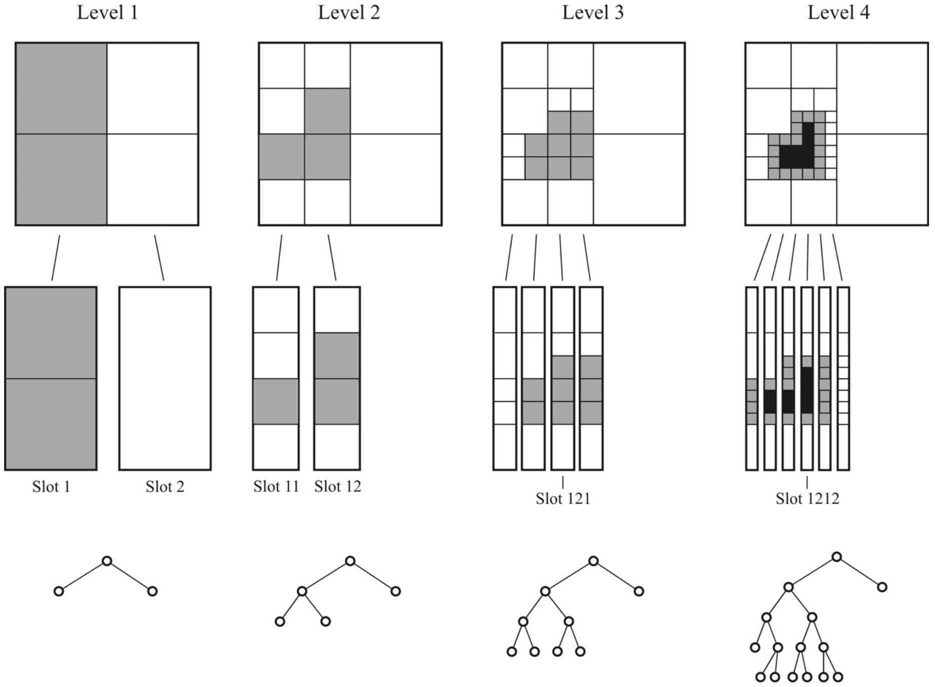


Figure 25: Generation of a 2D slot-tree up to level 4. A slot will only be refined if it possesses at least one gray *quadrant. A 2D slot tree can be derived directly from the quadtree presentation of the geometry of the objects, a 3D slot tree from an octree representation, respectively.*

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

4.4.2 General principle

All octree-based algorithms work according the same general principle. As mentioned above, the operands of the spatial operator being processed are encoded in separate octrees. In a first step, the root octants of both octrees are passed as input to the algorithm.

The algorithm consists in a simultaneous breadth-first traversal of both octrees. During the traversal it creates pairs of octants with one member from each octree. In the case of the algorithm implementing topological operators, both octants cover the same partition of the 3D space, whereas in the case of the metric operators the octant pair is among the candidates for the closest proximity.

The algorithm then applies certain operator-specific rules to the pairs of octants. Depending on the result of this test, the algorithm can either stop the recursion and return *true* or *false*, or it has to continue the recursive traversal by creating pairs of child cells, calling itself recursively and thus entering the next level. The user defines a maximum recursion level – if it is reached, the algorithm returns *true*, *false*, or a number representing the knowledge it has gained so far through the breadth-first traversal.

Though the algorithm implementing the directional operators works on slot-trees instead of octrees, it follows the same general principle. Here, the algorithm performs a breadth first-traversal of the slot-trees. During this traversal, pairs of slots are also created that represent the same partition of the 3D space, and rules are subsequently applied to these slot-pairs.

4.4.3 Metric operators

In the case of the metric operators, the core of the algorithm consists of calculating the upper and the lower bound of the distance of each cell pair (Borrmann et al., 2009). Since the exact position of the boundary of the objects is unknown when using an octree encoding (Figure 26), an upper and a lower bound of the distance value accordingly has to be calculated for each cell pair. These values represent the interval in which the real distance lies.

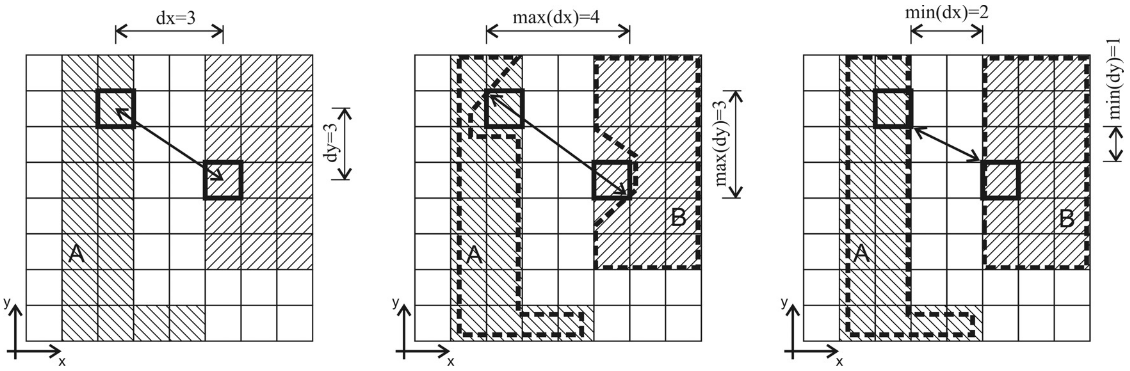


Figure 26: Since the exact position of the boundary of the objects is unknown when using octree encodings (left), an upper (middle) and lower bound (right) of the distance have to be determined for each cell pair based on the distance between the midpoints of the cells (left).

When processing the *distance* operator, the rule applied reflects the fact that all cell pairs whose distance is definitely higher than that of any other cell pair can be excluded from further refinement.

By computing the upper and lower bounds for all cell pairs, it is possible to identify the candidates for the closest cell pair. To this end, the lowest upper bound of all pairs of the current level is determined and all cell pairs whose lower bound is higher than this value are excluded.

All other pairs are candidates. For them, the algorithm is recursively repeated. They are refined, i.e. pairs of the relevant child cells are put together, and the filtering algorithm is applied to the resulting pairs of children, i.e. distance values are calculated, candidates are chosen, and so on. The recursion is aborted when the maximum refinement level is reached.

By descending both octrees in this way, the precision of the calculated distance is successively increased: the calculated distance can be expressed on each level by means of an interval, whose endpoints are calculated from the square root of the upper and lower distance values determined for the cell pairs on the level in question.

The interval in which the real distance lies is calculated after the recursion has finished. The final lower bound results from the square root of the lowest lower bound on the final octree level multiplied by the edge length of an octant on this level. The final upper bound is derived from the square root of the lowest upper bound on the final level, again multiplied by an octant's edge length. The return value of the algorithm is either a tuple of two real numbers representing the upper and lower bounds of the distance, or a single real number calculated as the arithmetic mean of upper and lower bound. The latter version can be integrated more easily into a spatial query language.

4.4.4 Directional operators

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

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

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

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

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

The rules for this exact examination depend on the direction and the version (strict/relaxed) of the operator that is being processed. They are not explained in detail here, but are shown in Figure 27 for the *strict* version of *above_proj* and in Figure 28 for the relaxed version of *above_proj*.

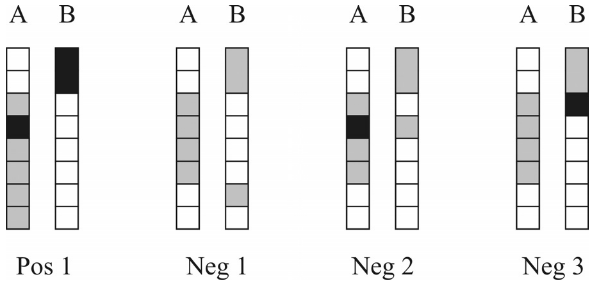


Figure 27: Examples of constellations where the rules Pos, Neg1, Neg2 *or* Neg3 *are applied during the processing of the algorithm* above_proj_strict(A,B). *The slots shown side-by-side actually occupy the same position in space.*

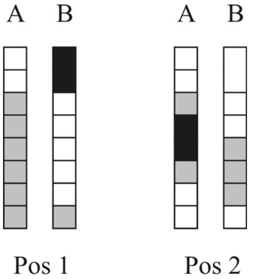


Figure 28: Examples of constellations where the rule Pos1 *and* Pos2 *are applied when processing the algorithm* above_proj_relaxed(A,B)*. The slots shown side-by-side actually occupy the same position in space.*

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

The creation of pairs of child slots is realized as follows: If both slots are *gray*, i.e. not leaf nodes of the corresponding slot tree, each of the four children of slot *A* is combined with a child of slot *B* at the same position, resulting in four pairs of child slots. If one of the slots is either *black* or *white*, i.e. a leaf node without children, it is combined with each child of the other slot, also resulting in four pairs of child slots. Consequently, there may be pairs of slots from different levels.

The algorithm calls itself recursively until the maximum refinement level is reached. If a decision is still not possible, rules are applied that take the most probable situation into account. This leads to fuzzy handling of directional relationships which is discussed in more detail in Section 4.4.6.

4.4.5 Topological operators

For implementing the topological operators, pairs of octants are created on each recursion level with one octant originating from object *A* and one octant from object *B*, both representing the same sector of the 3D space.

Each octant pair provides a colour combination to which specific rules can be applied. These rules may lead to filling a 9-IM *working matrix* that is maintained by the algorithm to keep track of the knowledge gained about the topological constellation. There are 12 positive and 9 negative rules altogether (Figure 29, 30 and 31). A positive rule can be applied when a certain colour combination occurs, and a negative rule if certain colour combinations do not occur over an entire level. Positive rules lead to empty set entries in the matrix, negative rules to non-empty set entries.

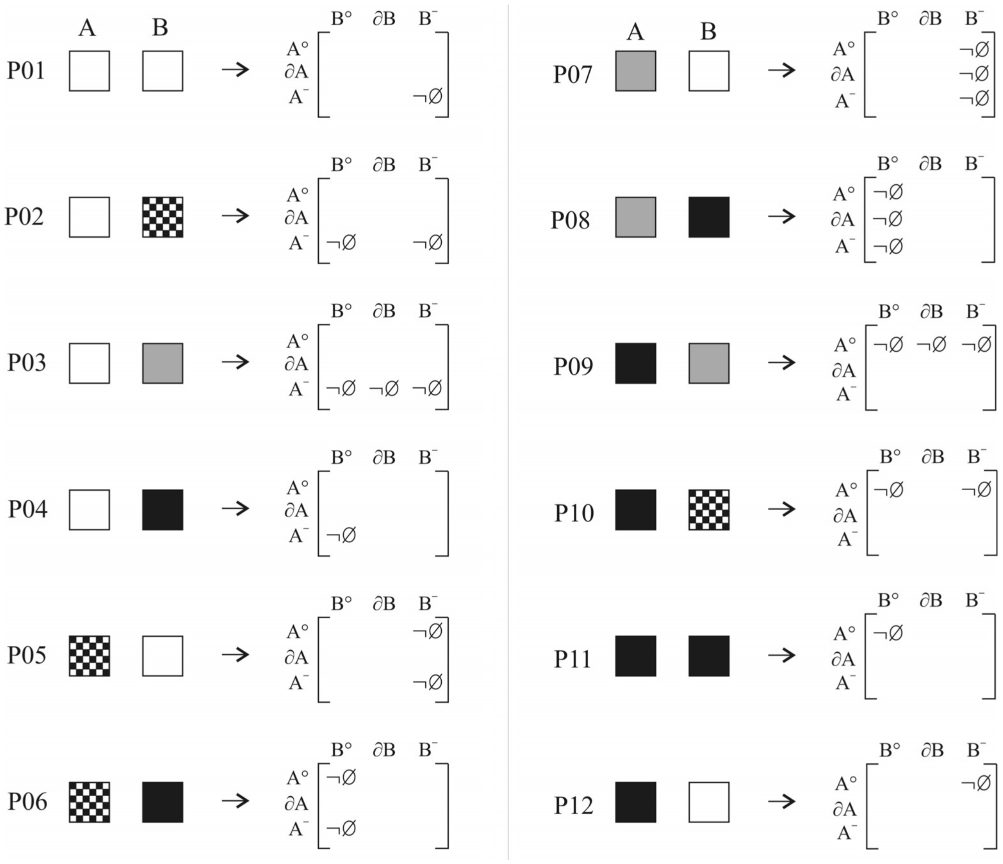


Figure 29: Positive Rules (Part 1). If the colour combination on the left-hand side is detected, the 9IM-Matrix can be filled according to the right-hand side. Combinations of mixed color cells (gray and/or black/white) never lead to filling the 9IM matrix, since they do not allow for a statement about the exact boundary position. For the same reason there is no color combination from which $\partial A \cap \partial B = \neg\emptyset$ *could be derived.*

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

The 9-IM working matrix is successively filled by applying these rules to all octant pairs. When processing the operator *whichTopoPredicate* the working matrix is compared with all predicate matrices of the formal definitions (Section 4.3.5). If the working matrix complies fully with one of them, the recursion is aborted and the algorithm returns the respective predicate. If there is any contradiction between the filled matrix and the matrix of a predicate, the respective predicate is precluded. If no unequivocal decision is possible for any of the predicates, a further refinement is necessary, i.e. octant pairs of the next level are created.

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

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

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

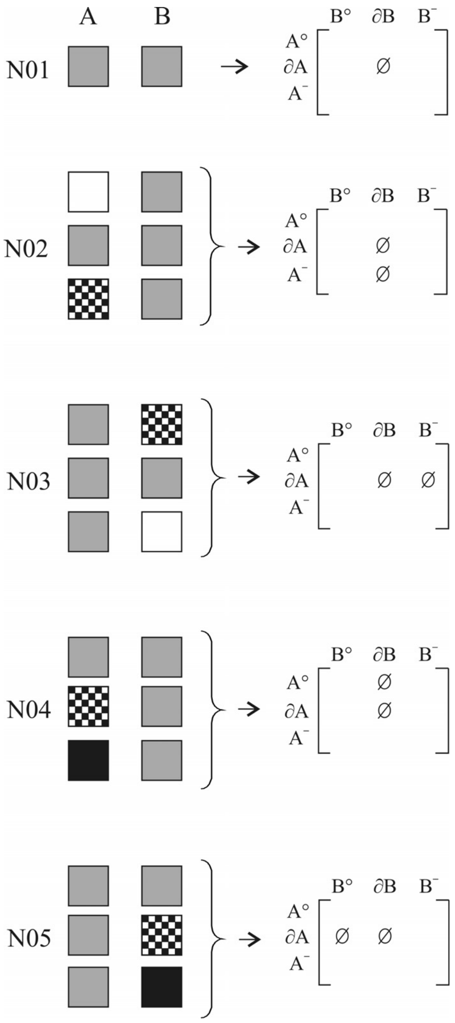


Figure 30: Negative Rules (1). If the colour combinations of the left-hand side do not occur across the entire domain, the 9-IM matrix can be filled according to the right hand side.

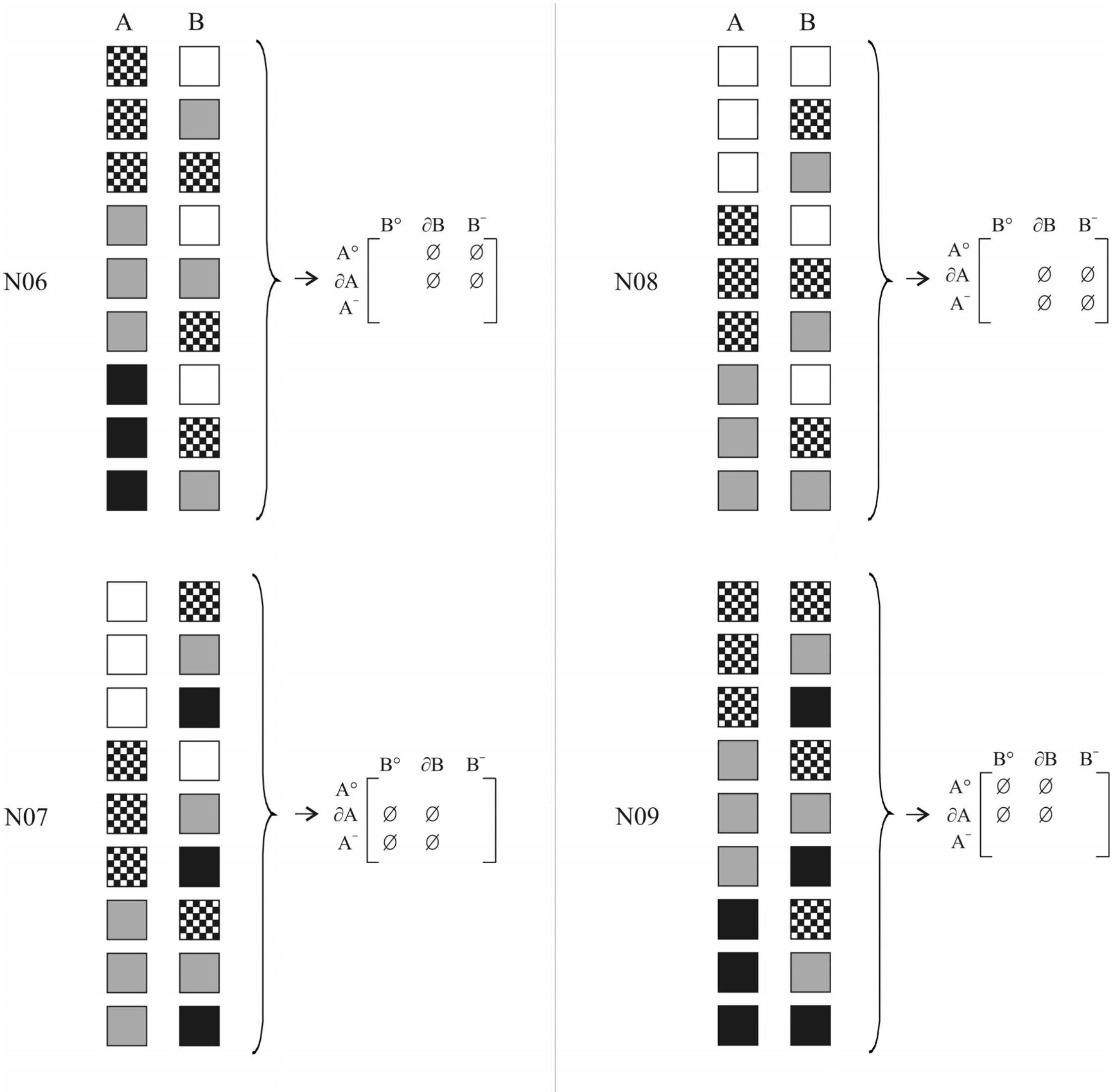


Figure 31: Negative Rules (2). If the colour combinations of the left-hand side do not occur in the entire domain, the 9-IM matrix can be filled according to the right hand side.

4.4.6 Fuzziness

The octree geometry representation shows a crucial peculiarity for the implementation of spatial operators: The boundary of an object encoded by an octree is not represented sharply, i.e. not as a set of points for each of which a neighbourhood exists that contains both interior and exterior points, but instead in the form of (grey) octants which define a boundary *layer*. The thickness of the layer shrinks with an increasing maximum refinement level (MRL): However, for finite values of the MRL it remains a layer.

This induces a certain fuzziness for all spatial operators. On the one hand such fuzziness results in inaccurate results if the MRL is not chosen high enough. On the other hand it enables the spatial operators to react more “mildly”, thus corresponding better to the way human handle qualitative spatial relationships.

A typical example is the relationship *touch*. Even if two building elements “slightly” overlap, in certain application scenarios the user, or some analysis program, might want to treat them as being in touch. The same applies if there is a slight gap between the elements.

In the following paragraphs, each instance of impreciseness involved in the octree approach is discussed individually for the metric, directional and topological operators.

Metric Operators. The metric operator *distance* returns a lower and upper bound between which the real distance lies. This can be seen as uncritical, because it does not involve any error. The same applies to *maxdist*.

By contrast, the predicate operator *closerThan* and *fartherThan* may incorrectly return *true* or *false*, respectively, if the chosen MRL is not high enough, as shown in Figure 32. Note that, due to the underlying logic of taking into account the cell pairs’ lower bound distances, *fartherThan* will never return *true* by mistake, not will *closerThan* return *false* by mistake.

In an alternative approach, one could consider to use a ternary instead of a Boolean value to be returned by *closerThan* and *fartherThan*, providing *unknown* as additional possible value. Unknown would be returned if the queried distance *c* lies between the lower and the upper bound calculated on the MRL. We will further examine this implementation option in future publications.

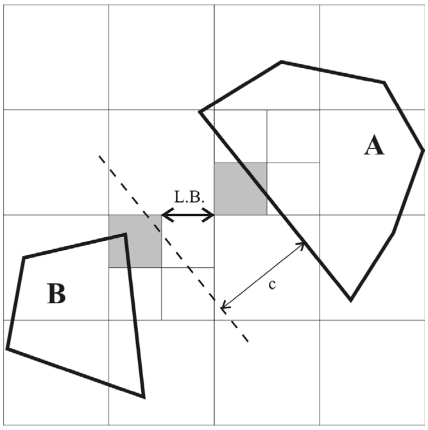


Figure 32: Situation where closerThan(A,B,c) *erroneously returns* true*, and* fartherThan (A,B,c) *erroneously returns* false *if the maximum refinement level (MRL) is too low. This is caused by the implementation of the operators which relies on the calculated lower bound (L.B.) of the distance value on the MRL. In an alternative approach, both operators would return* unknown *when c lies between the lower and the upper bound.*

Directional Operators. The interpretation of non-resolved slot pairs on the final level depends on whether the strict or the relaxed version of the directional predicates is being processed.

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

According to this interpretation, the strict operators may incorrectly return *true* when the definition is actually violated (Figure 33, left) while, on the other hand, the relaxed operators may return *false* although the definition is actually satisfied (Figure 33, right).

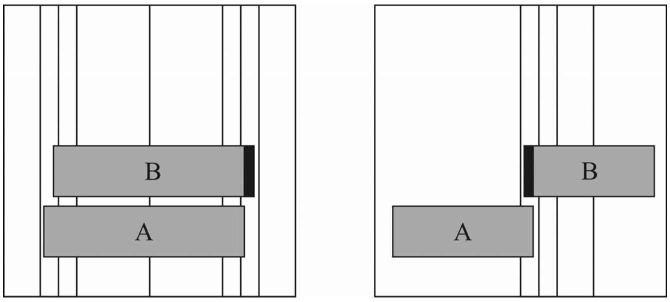


Figure 33: In the given examples, the critical parts (depicted in black) will not be detected by the slot-based algorithms if the maximum refinement level is not greater than 4. Left: *The operator* above_proj_strict *will incorrectly return* true. Right: *The operator* above_proj_relaxed *will incorrectly return* false.

Topological Operators. If, in the case of a predicate operator, the predicate under examination is neither proved nor disproved when reaching the MRL or, in the case of the *whichTopoPredicate* operator, none of the predicates is fully proved, the predicate hierarchy shown in Figure 34 is applied, i.e. the algorithm returns the highest non-disproved predicate of the hierarchy. The order of the hierarchy is chosen in such a way that, if the actual topological constellation complies with predicate *a*, all predicates above predicate *a* are disproved during successive refinement. On the other hand, the predicates below *a* are not necessarily disproved. In the sense of a "positivistic" approach it is assumed that the highest non-disproved predicate has been proven.

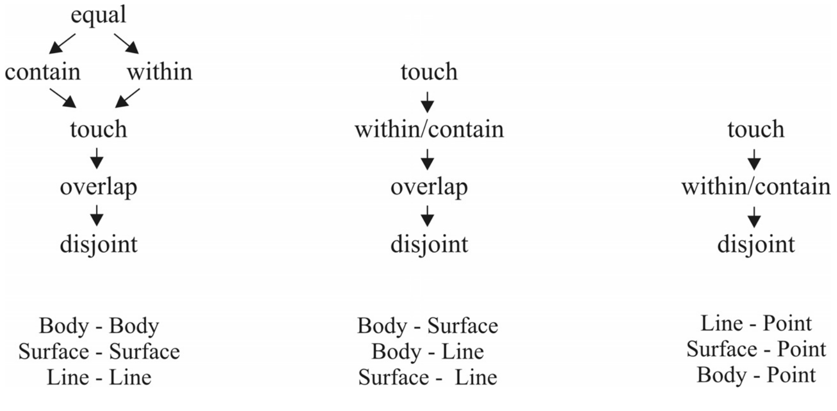


Figure 34: The hierarchy of the topological predicates for different type combinations. The algorithm returns the highest non-disproved predicate. The order of the hierarchy results from the observation that all predicates above a certain predicate x *are disproved during the ongoing refinement if the actual topological constellation complies with predicate* x*. This hierarchy permits a fuzzy handling of topological relationships.*

If both operands have the same dimensionality, *contain* and *within* are equivalent, i.e. for the validation of a “lower” predicate, both *contain* and *within* must be disproved. The equivalence of the predicates results from the fact that when disproving *equal*, either *contain* or *within* is disproved at the same time.

Applying the predicate hierarchy may result in the detection of an incorrect topological predicate if the MRL is too low. However, the hierarchy is chosen in such a way that these errors/misjudgements are acceptable, since they comply with the intuitive human understanding of qualitative spatial relationships. Figure 35 illustrates constellations where the application of the predicate hierarchy results in the detection of an incorrect topological predicate.

Using the “positivistic” approach, the requirements of *logical consistency*, *mutual exclusiveness* and *complete coverage* are met by the system of topological operators, since in any case precisely one topological predicate is detected for any topological constellation no matter if the user applies the predicate operators or *whichTopoPredicate*.

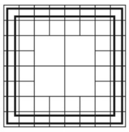
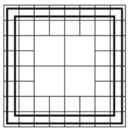
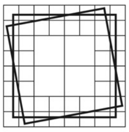
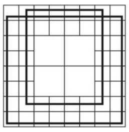
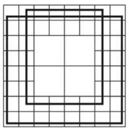
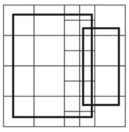
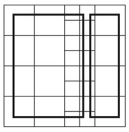
	equal situation	*contain* situation	*within* situation	*touch* situation	*overlap* situation	*disjoint* situation
erroneously detected as ***equal***	/			/		/
erroneously detected as ***contain***	/	/	/	/		/
erroneously detected as ***within***	/	/	/	/		/
erroneously detected as ***touch***	/	/	/	/		
erroneously detected as ***overlap***	/	/	/	/	/	/
erroneously detected as ***disjoint***	/	/	/	/	/	/

Figure 35: Misjudgements of the topological operators caused by inadequately refined resolution in the case of Body–Body *constellations. The predicate assigned by the algorithm (horizontal) to the found situation (vertical) results directly from the design of the predicate hierarchy. The*

maximum resolution chosen for this example is represented by the smallest visible cell resolution. Empty table cells indicate that the corresponding misdetection does not occur.

4.4.7 B-Rep approaches

The second general approach for implementing the spatial operators is based on the exact shapes of the geometric objects (the operands) given by their boundary representation (B-Rep). As opposed to the octree-based implementation, these algorithms work directly on the exact geometry description and accordingly return precise results.

Naive implementations on the basis of the B-Rep structure require a high computational effort or (e.g.) are restricted to convex bodies. In our approach we therefore use a hierarchical representation of the faceted objects. This makes it possible to exclude the irrelevant parts of the operands' geometry at an early stage, thus avoiding unnecessary computations. At the same time, parts that may have an impact on the result are examined in increasing detail. The principle employed here is also known as "divide-and-conquer" strategy.

In the presented approach, AABB trees are chosen for the hierarchical representation. AABB trees are binary trees that recursively divide the space, along one coordinate axis on each occasion. A box-shaped hull volume (axis-aligned bounding box, AABB) that encloses the hull volumes of both child nodes is assigned to each node in the tree (Figure 36). The leaf nodes of the tree contain one or more facets of the geometric object, as well as the bounding box containing these facets. It is important to note that an AABB tree, as opposed to the octree, is not a geometry representation, but a spatial indexing of the boundary representation.

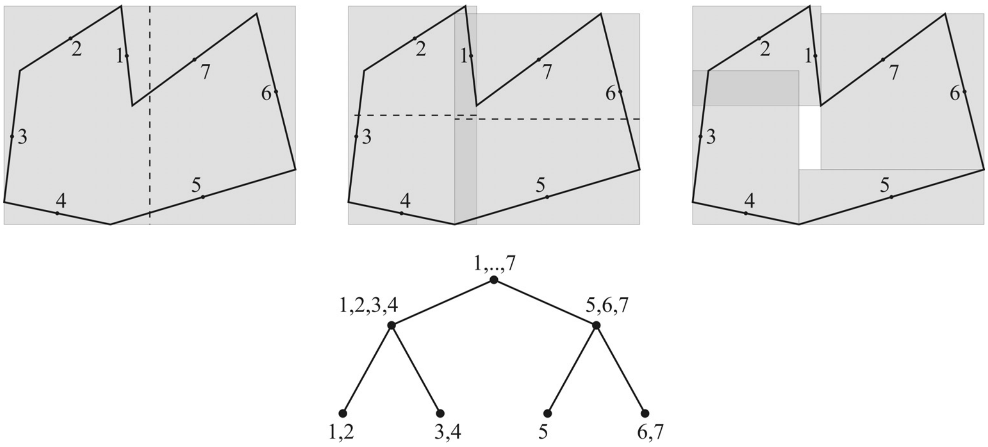


Figure 36: Example of the generation of an AABB tree for a polygon in 2D. On each level, the space is divided along the longest axis. Each edge of the polygon is then assigned to one of the subspaces by considering the location of its midpoint. Afterwards an axis-aligned bounding box is created containing all edges belonging to one of the subspaces. For this bounding box, the process is recursively repeated, entering the next level of the tree.

The choice of employing an AABB tree is motivated by its simple structure not only enabling a fast generation of the tree but also permitting simple and consequently computationally cheap tests on the AABBs, such as intersection tests, for example.

There are various strategies for the concrete partitioning of the facets (Bergen, 1997). A commonly used partition rule considers the projection of all the facets on the longest axis of the

AABB. If the projection of the midpoint of a facet is located to the left of the projection of the midpoint of the entire AABB, the facet is assigned to the left-hand subspace, in the other case it is assigned to the right-hand subspace. Once all the facets have been classified, the process is recursively repeated for both the left and the right-hand subspace. The recursion is aborted as soon as only one facet is left in an AABB, or the maximum tree level has been reached.

The basic structure for implementing spatial operators using AABB trees is as follows: the first step is to generate AABB trees for both operands *A* and *B* of the spatial operator. Then the AABB trees are traversed in breadth-first manner and pairs consisting of one AABB from *A* and one AABB from *B* are created. Depending on the operator being processed some of the AABB pairs of the current level can be excluded from further examination, because they obviously do not have any impact on the result. All the other AABB pairs are further refined.

If both members of the AABB pair are leaf nodes in the corresponding tree, no further refinement is possible. In this case, pairs of the facets contained in the AABBs are created. Depending on the operator being processed, precise tests, such as intersection tests, are now carried out on the facet pairs. This procedure makes it possible to reduce computationally expensive tests to a minimum.

Metric operators. For implementing the operator *distance*, the AABB trees are traversed in a similar fashion to the octree-based implementation, i.e. an upper and lower bounds are computed for each pair of AABBs and accordingly all irrelevant pairs and their children are pruned.

If an AABB pair is a potential candidate for the closest proximity and both AABBs of the pair are leaf nodes, we create the cross product of all primitives assigned to the two AABBs and, in a final calculation step, compute the exact minimal distance for all resulting pairs. The exact distance between two primitives under consideration is computed using the GJK algorithm for convex polyhedrons (Gilbert, Johnson, & Keerthi, 1988). Since an AABB generally encompasses a few triangles only, the computational effort is limited.

The computed distances between triangles may lead to a new minimum upper distance that can be used to exclude other AABB pairs in the same way as in the octree-based approach. It is possible that an AABB pair is a potential candidate for the closest proximity, but only one box of the pair is a leaf and, therefore, only the second bounding box has two children. In this case, the two children of the second AABB are paired with the first AABB, respectively. Finally, the global minimum distance turns out to be the shortest distance between one triangle and another, as computed using the GJK algorithm.

Topological operators. In Figure 37 the algorithm for implementing the topological operator *whichTopoPredicate* is depicted schematically. In the first stage, an intersection test is performed to ascertain whether the two operands overlap, or not. It works in the same fashion as the algorithm implementing the metric operators, except that all distance calculations are replaced by intersection tests. A further refinement in the AABB trees is realized each time two AABBs intersect. On reaching the leaf nodes, facet pairs are created and tested on intersection. If an intersection is detected the predicate *overlap* can be returned.

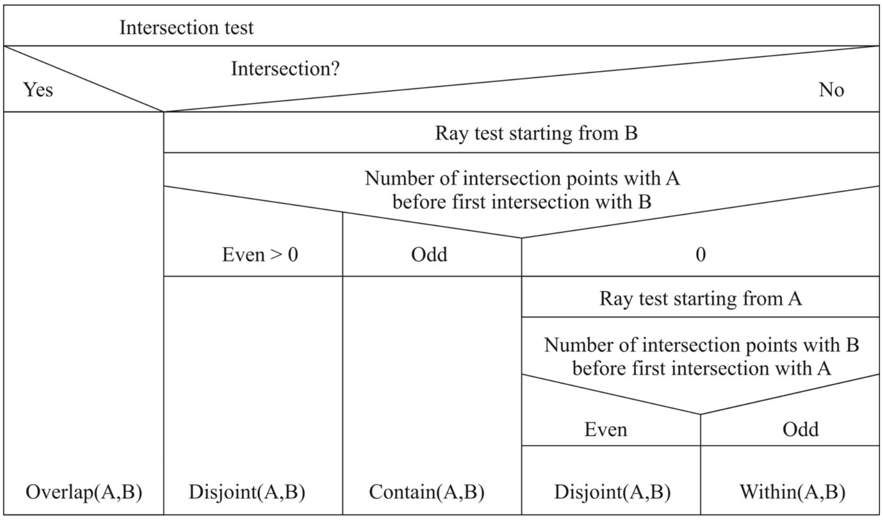


Figure 37: Schema of the BRep-based algorithm implementing topological operators.

If there is no intersection, the operands are either disjoint or one lies within the other. The exact topological relationship, i.e. which of the predicates *contain*, *within* or *disjoint* can be applied, is determined by using a search ray and counting the number of intersection points. The ray test can also be implemented by applying a “divide-and-conquer strategy”: Intersection tests for the ray and the AABBs (Williams, Barrus, Morley, & Shirley, 2005) are executed until the final level is reached – only then are intersection tests performed for the ray and individual facets (Moeller & Trumbore, 1997).

The actual algorithm can be described as follows: First we choose an arbitrary axis-aligned ray starting at the surface of *B*, determine the intersection points of this ray and the objects *A* and *B* and then sort them according to their occurrence in the direction of the ray. By choosing the surface of *B* as origin of the ray, the ray is guaranteed to hit at least one object and will not miss both of them. Now the number of intersection points of the ray and object *A* which occur before the first intersection of the ray and object *B* are taken into account (see Figure 38).

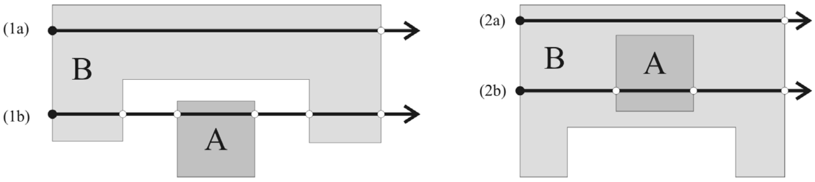


Figure 38: Ray tests for determining topological relationships. The origin of the ray is marked by a filled circle, the intersection points by an empty circle. Ray (1b) correctly detects the predicate disjoint(A,B) and ray (2b) the predicate within(A,B). No conclusions can be drawn, however, from the rays (1a), (1c) and (2a), since they do not hit operand A. *In this case, a second test is performed, this time using a ray that starts at the surface of* A.

If this number is odd, it can be stated that *B* is located within *A*, and the predicate *contain(A,B)* is returned. If the number of intersections is even, but greater than 0, the situation is vice-versa and the predicate *within(A,B)* is returned. If the number of intersection is 0, no topological predicate can be determined. Instead, a second ray test, this time starting at the surface of *A*, is required. An odd number of intersection leads to assigning the predicate *within*(*A,B*), whereas 0 or an even number of intersections lead to assigning the predicate *disjoint*(*A,B*).

Directional operators. BRep-based algorithms for the implementation of directional operators are still under development. However there are some promising approaches that again are based on the application of ray tests. Results will be presented in future publications.

4.5 Embedding spatial operators within a query language

The integration of the spatial types and operators defined in the previous section in one of the available query languages for Building Information Model discussed in Section 3 enables the user to apply both semantic and spatial conditions within a single query.

To find the most suitable of the options available, we experimented with three different query languages. For assessing the suitability of the query language basis we applied the following criteria:

1. **Extensibility.** The query language must allow for an extension of the type system or at least an integration of user-defined functions.
2. **Expressive power.** The query language must provide easy access to semantic data, including attribute values, collections, and navigation along references.
3. **Simplicity.** The query statement should be as short as possible, avoiding any syntactic overhead.
4. **Availability of processors.** There should be a number of processing engines available for the query language, preferably under public license, to facilitate their use in research context.

The first criterion is the most crucial requirement: If the query language in question has a fixed set of applicable operators that cannot be extended, it is unsuitable as a basis for spatial query functionality. All other criteria are “soft” in the sense that they do not form an absolute requirement.

As explained in detail in Section 3, EXPRESS-X is the most appropriate option thanks to its conceptual closeness to the original data modelling language EXPRESS, resulting in a maximum of achievable expressive power and coupled with short, simple query statements. However, it lacks the essential means to integrate user-defined operators and can thus not be considered as query language basis.

We accordingly decided not to use EXPRESS-X, but to experiment with alternative query languages, namely SQL and XQuery, instead.

4.5.1 Embedding spatial operators in SQL

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

SQL:1999. As discussed in Section 3.4.2, SQL:1999 provides the user the possibility to define abstract data types (ADTs), thus extending the database type system in an object-oriented way. These ADTs may not only possess attributes and references to other ADTS but also member functions (methods) that define the behaviour of the corresponding object instances. Accordingly, the spatial data types defined in Section 4.3.1 can be defined as ADTs providing the spatial operators as member functions.

The algorithms implementing the spatial operators that are presented in Section 4.4 require the explicit B-Rep geometry of the involved building components. Though building information models usually allow multiple ways to define geometry (extrusion, constructive solid geometry etc.), we have to assume the existence of a boundary representation here.

Taking the IFC data model as an example, a suitable place to introduce the spatial operators is the entity *IfcManifoldSolidBrep* which is used to model faceted BRep geometries that may contain voids. Using the capabilities of SQL:1999 the type can be easily extended by member functions representing the metric, directional and topological operators defined in Section 4.3. However, as presented in Figure 39, navigating from a building element object to its *IfcManifoldSolidBrep* representation involves 3 intermediate steps resulting in long and unhandy navigation expressions. The authors therefore propose the extension of the *IfcElement* class by the member function *shape()* that acts as short-cut and returns the corresponding *IfcManifoldSolidBrep* object.

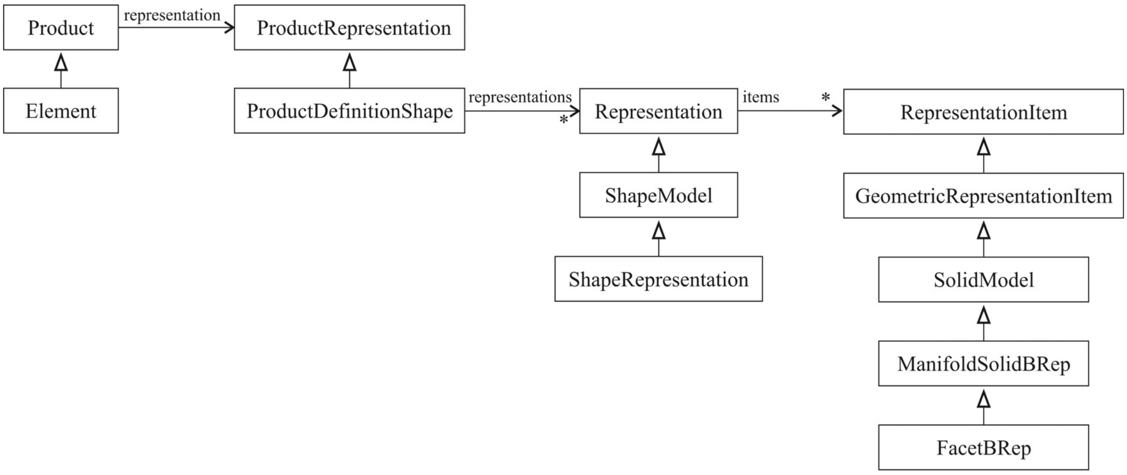


Figure 39: Due to the complex structure of the IFC, access to a building element's geometry representation has to be realized using 3 intermediate steps.

By realizing this, spatial query functionality can be made available to end-users and third-party programmers in an easily manageable manner. A specimen query that retrieves all columns that touch the slab whose ID is *Oid23089* then reads:

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

For a prototype implementation the authors used the commercially available ORDBMS Oracle 10g. For more detailed information on the integration of spatial operators in SQL using object-relational techniques, the reader is referred to (Borrmann & Rank, 2009; Borrmann et al., 2009).

The most important advantage of using an object-relational approach is the strong type safety provided by the declaration of user-defined types. The declaration of the *touch* member function, for example, forces the passed parameter to be of type *IfcElement* or one of its sub-types. Thus, type errors may already be detected by the query engine during the interpretation of the SQL statement and more specific error reports can be generated.

SQL-92. As for the desired purpose of a declarative spatial query language for BIMs, traditional database functionalities such as concurrency control, rights management and persistency are not of primary interest, the utilization of an in-memory database (IMDB) seems to be most appropriate. These systems, which are normally completely embedded in the final application, usually provide SQL query and data manipulation functionality while avoiding the high overhead of hard-disk access. Unfortunately, there are no in-memory databases available today that provide the full range of the SQL:1999 standard, especially with respect to the possibility of defining ADT's.

We therefore decided in a second approach to base the spatial query functionality on purely relational databases. Here, a semantically weaker way of defining the spatial operators has to be chosen. All spatial operators are defined as global functions whose parameters are *strings* representing the operand's IDs. The specimen query then reads:

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

4.5.2 Embedding spatial operators in XQuery

In XQuery, spatial operators are integrated in a similar manner as in SQL. It is an important prerequisite that the XQuery engine supports the calling of external functions. If this fulfilled, the external functions are first declared in the prolog of the query:

```
declare namespace spatial = 'java:de.tum.cie.SpatialOperators';
declare function spatial:touch($arg1 as xs:string, $arg2 as xs:string)
                 as xs:boolean external;
```

Afterwards they can be employed in the *where* part of the query:

```
let $uos := $this/ex:iso_10303_28/ifc:uos
for $column in $uos/ifc:IfcColumn,
    $slab in $uos/ifc:IfcSlab,
where spatial:touch( $column/id, $slab/id) and slab[@id=Oid23089]
return <QueryResult>{$column}</QueryResult>
```

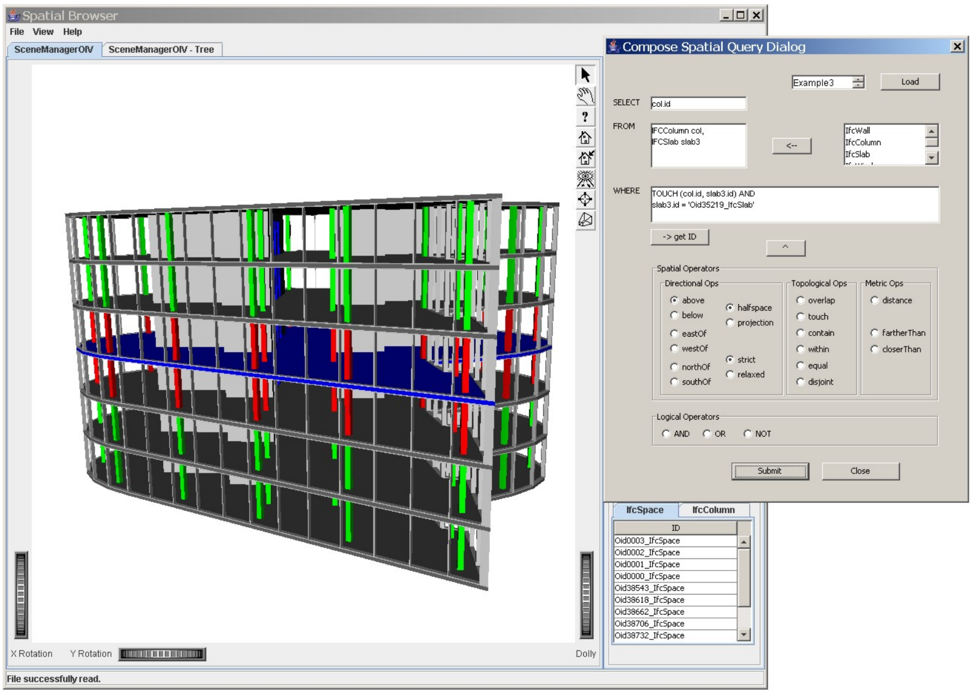
4.6 Software prototype

To prove the feasibility of the developed concepts we implemented a software prototype that offers spatial query functionality for building information models (Figure 40). It is capable to process IFC building models provided in either the STEP-P21 or the ifcXML file format. In order to avoid the laborious implementation of the geometry generation, we have chosen to take advantage of IFC-VRML files containing the explicit building geometry. Such files can be generated from an IFC model by using the *IFCStoreyView* program developed by the Karlsruhe Institute of Technology[11].

To realize the query support for STEP-P21 files the *exp2ddl* and *p21tosql* tools introduced in Section 3.4.1 have been employed. By the help of these tools, an in-memory database is created whose schema is capable to hold IFC data in a relational form. When reading the STEP-P21 file, the tables of the database are accordingly filled with data. At the same time the IFC-VRML file containing the corresponding explicit geometry representation is read into the spatial processing engine. The user can subsequently employ SQL-92 to query the semantic data of the IFC model and use the spatial operators available as global functions.

If the user provides a building model in the ifcXML format, the XML file is read into memory using the library *XMLBeans*[12]. Subsequently, the building model data is available for query processing employing the XQuery engine Saxon[13]. Again the geometry is read-in through a corresponding IFC-VRML file. All spatial operators are declared as external XQuery functions and will accordingly be called if they appear in a query.

After processing either the SQL query or the XQuery, the resulting set of building elements is highlighted in the 3D viewer.



Example3

SELECT col.id

FROM IFCColumn col,
IFCSlab slab3

IfcWall
IfcColumn
IfcSlab

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

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

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

Metric Ops: ○ distance ○ fartherThan ○ closerThan

Logical Operators: ○ AND ○ OR ○ NOT

ID
Oid0003_IfcSpace
Oid0002_IfcSpace
Oid0001_IfcSpace
Oid0000_IfcSpace
Oid38543_IfcSpace
Oid38618_IfcSpace
Oid38662_IfcSpace
Oid38706_IfcSpace
Oid38732_IfcSpace

File successfully read.

Figure 40: Screenshot of the prototype application showing the dialog for composing spatial SQL queries and the 3D viewer highlighting the result set.

[11] http://www.ifcwiki.org/index.php/IfcStoreyView_VRML_Export

[12] http://xmlbeans.apache.org

[13] http://www.saxonica.com

A serious condition for the correct functioning of the spatial query functionality is a high quality of the geometry provided in the IFC building model. If the model contains isolated or unconnected polygons or bodies with missing polygons, incorrect query results may be returned. For the future we therefore plan to develop a geometric pre-processor that is able to detect these kinds of imperfections beforehand and warn the user accordingly.

5 CONCLUSION

This chapter has discussed in detail the technical possibilities for querying Building Information Models applying both semantic and spatial constraints. The first section investigates the query technologies available for traditional, non-spatial conditions. The complex structure of the EXPRESS-based data models currently in use for building information modelling requires a language with a high expressive power that permits a simple, concise formulation of queries. Since decisive attributes are often linked to the desired entities by a number of intermediate steps, facilities for an easy navigation along references are particularly important. From a formal point of view, EXPRESS-X has been identified as the most appropriate query language option. Since there is only a very limited number of processing engines available, for one thing, and it lacks the required extensibility for integrating spatial query functionalities, for another, alternative options including SQL and XQuery are also presented here.

The second part focused on realizing the spatial query functionality. The proposed qualitative spatial operators form an intermediate level of abstraction between the technical representation of a building's geometry (using vertex coordinates etc.) and the way engineers and architects think about the geometrical and topological relationships between building components. By applying spatial operators the users can easily identify building components that fulfil certain metric, directional or topological conditions. The chapter provides detailed formal definitions of the semantics of the spatial operators and gives an overview on different implementation approaches.

For the future, declarative query support for building information models providing spatial and semantic conditions will be of increasing importance. On the one hand, the sound definition of the content of partial models is a crucial prerequisite for realizing BIM-based collaborative planning processes. On the other hand, it is expected that national and international regulation authorities will make extensive use of automated code checking procedures based on digital building models. For encoding the respective codes, rules and laws in form a readable by both humans and machines, a declarative query language providing both semantic and spatial constraints forms an excellent basis.

ACKNOWLEDGMENTS

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

REFERENCES

Abiteboul, S., Buneman, P., & Suciu, D. (1999). *Data on the Web: from relations to semistructured data and XML*. San Francisco: Morgan Kaufmann Publishers Inc.

Adachi, Y. (2002). Overview of IFC model server framework. In *Proc. of the 4th Europ. Conf. on product and process modeling.*

Amor, R., Jiang, Y., & Chen, X. (2007). BIM in 2007 - Are we there yet? In *Proc. of the 24th CIB-W78 Conference on Information Technology in Construction.*

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

Bailey, I., Hardwick, M., Laud, A., & Spooner, D. (1996). Overview of the EXPRESS-X language. In *Proc. of the 6th EXPRESS users group conference.*

Balovnev, O., Bode, T., Breunig, M., Cremers, A., Müller, W., Pogodaev, G., et al. (2004). The story of the GeoToolKit - an object-oriented geodatabase kernel system. *GeoInformatica*, *8*(1), 5–47.

Bergen, G. van den. (1997). Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, *2*(4), 1–13.

Bicarregui, J., & Matthews, B. (1998). The specification and proof of an express to SQL compiler. In J. Bicarregui (Ed.), *Proof in VDM: Case studies.* Berlin, Germany: Springer-Verlag.

Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J., Simeon, J., et al. (2007). *XQuery 1.0: An XML Query Language. W3C Recommendation.* Retrieved May 12, 2009, from http://www.w3.org/TR/xquery/

Borrmann, A. (2006). *Extended formal specifications of 3D spatial data types* (Tech. Rep.). Technische Universität München, Germany.

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

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

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

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

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

Borrmann, A., van Treeck, C., & Rank, E. (2006). Towards a 3D spatial query language for building information models. In *Proc. of the Joint Int. Conf. for Computing and Decision Making in Civil and Building Engineering.*

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

Breunig, M., Cremers, A., Müller, W., & Siebeck, J. (2001). New methods for topological clustering and spatial access in object-oriented 3D databases. In *Proc. of the 9th ACM Int. Symp. on Advances in Geographic Information Systems.*

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

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

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

Denno, P. O., & Sanderson, D. B. (2000). *Structural information mapping with EXPRESS-X.* NIST, Gaithersburg. Retrieved May 12, 2009, from http://www.mel.nist.gov/msidlibrary/doc/structx.pdf

Eastman, C. M., Wang, F., You, S.-J., & Yang, D. (2005). Deployment of an AEC industry sector product model. *Computer-Aided Design*, *37*(12), 1214-1228.

Egenhofer, M. (1987). An extended SQL syntax to treat spatial objects. In *Proc. of the 2nd Int. Seminar on Trends and Concerns of Spatial Sciences.*

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

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

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

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

Egenhofer, M., & Herring, J. (1992). *Categorizing binary topological relations between regions, lines, and points in geographic databases* (Tech. Rep.). Department of Surveying Engineering, University of Maine. Retrieved May 12, 2009, from http://www.spatial.maine.edu/~max/9intReport.pdf

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

Fowler, J. (1995). *STEP for data management, exchange and sharing*. Technology Appraisals.

Fuhr, T., Socher, G., Scheering, C., & Sagerer, G. (1998). A three-dimensional spatial model for the interpretation of image data. In P. Olivier & K. Gapp (Eds.), *Representation and processing of spatial expressions* (pp. 103–118). Mahwah, NJ: Lawrence Erlbaum Associates.

Gilbert, E. G., Johnson, D. W., & Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, *4*(2), 21-28.

Goyal, R. (2000). *Similarity assessment for cardinal directions between extended spatial objects*. Doctoral dissertation, University of Maine.

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

Guesgen, H. (1989). *Spatial reasoning based on Allen's temporal logic* (Tech. Rep.). Int. Computer Science Institute, Berkley, CA.

Herring, J., Larsen, R., & Shivakumar, J. (1988). Extensions to the SQL language to support spatial analysis in a topological data base. In *Proc. of GIS/LIS '88.*

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

Ingram, K., & Phillips, W. (1987). Geographic information processing using a SQL-based query language. In *Proc. of the 8th int. Symp. on Computer-assisted Cartography.*

International Organization for Standardization. (1995). *ISO 10303 - Standard for the exchange of product model data.*

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

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

International Organization for Standardization. (2007). *ISO 10303-28:2007 – Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas.*

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

Kiviniemi, A., Fischer, M., & Bazjanac, V. (2005). Integration of multiple product models: IFC model servers as a potential solution. In *22nd CIB-W78 Conference on Information Technology in Construction.*

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

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

Melton, J. (2003). *Advanced SQL:1999. Understanding object-relational and other advanced features.* San Francisco: Morgan Kaufmann.

Moeller, T., & Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of Graphical Tools*, *2*(1), 21-28.

Mundani, R.-P. (2005). *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*. Doctoral dissertation, Universität Stuttgart, Germany.

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

Nisbet, N., & Liebich, T. (2005). *IfcXML implementation guide* (Tech. Rep.). International Alliance for Interoperability.

Ooi, B., Sacks-Davis, R., & McDonell, K. (1989). Extending a DBMS for geographic applications. In *Proc. of the IEEE 5th Int. Conf. on Data Engineering.*

Oosterom, P. van, Vertegaal, W., Hekken, M. van, & Vijlbrief, T. (1994). Integrated 3D modelling within a GIS. In *Proc. of the Workshop on Advanced Geographic Data Modelling.*

OpenGIS Consortium – OGC (1999). *OGC Abstract Specification.* Retrieved May 12, 2009, from http://www.opengis.org/techno/specs.htm

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

Papadias, D., Sellis, T., Theodoridis, Y., & Egenhofer, M. (1995). Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data.*

Paul, N. (2010). Basic topological notions and their relation to BIM. In J. Underwood & U. Isikdag (Eds.), *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies.* Hershey, PA: IGI Global.

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

Pullar, D. (1988). Data definition and operators on a spatial data model. In *Proc. of the ACSM-ASPRS annual convention.*

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

Rigaux, P., Scholl, M., & Voisard, A. (2002). *Spatial databases with application to GIS*. San Francisco: Morgan Kaufmann.

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

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

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

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

Shekhar, S., & Chawla, S. (2003). *Spatial databases: A tour*. Upper Saddle River, NJ: Pearson Education.

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

Türker, C. (2003). *SQL:1999 & SQL2000. Objekt-relationales SQL, SQLJ & SQL/XML*. Heidelberg, Germany: dpunkt Verlag.

Türker, C., & Saake, G. (2006). *Objektrelationale Datenbanken*. Heidelberg, Germany: dpunkt Verlag.

Urban, S., Tjahjadi, M., & Shah, J. (2000). A case study in mapping conceptual designs to object-relational schemas. *Concurrency: Practice and Experience*, *12*(9), 863-907.

Wei, G., Ping, Z., & Jun, C. (1998). Topological data modelling for 3D GIS. In *Proc. of ISPRS Commission IV Symp. on GIS.*

Weise, M., Katranuschkov, P., & Scherer, R. J. (2003). Generalized model subset definition schema. In *Proc. of the 20th CIB-W78 Conference on Information Technology in Construction.*

Williams, A., Barrus, S., Morley, R. K., & Shirley, P. (2005). An efficient and robust ray-box intersection algorithm. *Journal of Graphics Tools*, *10*(1), 49–54.

You, S.-J.; Yang, D. & Eastman, C. (2004). Relational DB implementation of STEP based product model. In *Proc. of the 16th CIB World Building Congress.*

Zlatanova, S. (2000). On 3D topological relationships. In *Proc. of the 11th Int. Workshop on Database and Expert Systems Applications.*

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

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

KEY TERMS & DEFINITIONS

Building Information Model (BIM)
A computational representation of a planned or built building. The representation comprises the 3D geometry of the building elements and the spaces, as well as semantic (non-geometric) information, such as element types and material properties. Also there is a rich set of relationships between building elements stored in the building model. A BIM is modeled using object-oriented modeling techniques.

BIM query
A BIM query is used to retrieve a well-specified subset of the entire building model. The query is formalized using either a query language or a schema representation. BIM queries are used in human-machine as well as machine-machine communication.

Semantic query
A semantic query uses properties of BIM entities and/or relationships between them as selection criteria that are defined in the BIM.

Spatial Query
A spatial query uses properties and/or relationships that are of spatial nature and are not explicitly available in the BIM. To process a spatial query the 3D geometry model is analyzed.

Spatial Query Language
A formal language that allows formulating spatial queries by providing topological, directional and metric operators for specifying selection criteria.

Topological operator
Topological operators are used to query topological relationships between two entities of the building model. Topological relationships are invariant under affine transformations, such as rotation, translation and scaling (factor≠0). To use topological relationships as selection criteria in a spatial query language, the large set of possible topological constellations is clustered and a human language denomination *(touch, contain, within, ...)* is assigned to each of these clusters.

Directional operator
Directional operators are used to query directional relationships between two entities of the building model. Direction is a binary relation of an ordered pair of objects A and B, where A is the reference object and B is the target object. The third part of a directional relation is formed by the reference frame, which assigns names or symbols to space partitions. In the context of BIM queries, an *extrinsic* reference frame is applied, which is formed by the Cartesian coordinate system the BIM is placed in. The space partitions can be created applying different techniques (e.g. halfspaces or projections); in any case, the denomination of the cardinal directions *(northof, westof, above,...)* or combinations are assigned to them.

Metric operator

Metric operators are used to query distance relationships between two building entities. Examples for metric operators are *distance, closerThan* and *fartherThan*.