Relations between spatial objects

Simon Fromme

edited on November 13, 2014

Contents

1 Motivation

In recent years 3D geo data becomes increasingly available and more and more cities decided to create digital 3D models of their building stock. This development was aided by many public agencies commissioning laser-scanning (LiDAR) and photogrammetry of their territories with the so deduced data oftentimes serving as a basis for 3D city models. Besides buildings city models often also include entities such as infrastructure objects (e.g. tunnels, bridges), city furniture (e.g. monuments, fountains) and vegetation thus providing a basis for a variety of potential applications based on 3D spatial data.

Conceivable projects based on the data provided by 3D city models include among others:

- visibility analysis, e.g. predicting whether the construction of a high-rise building would block the view to a historic church from certain apartments
- shadow analysis, e.g predicting the location and intensity of cast shadows in the planning phase of a new skyscraper
- visualization of new buildings in a realistic environment prior to construction
- illumination simulations (e.g. for the illumination of historic churches or castles)
- calculation of the solar energy potential for entire areas
- noise and pollutant predictions (e.g. the implications of a new urban freeway on the noise level in surrounding neighborhoods may be computed using the data of a city model).

To facilitate some of these tasks it would oftentimes be of great help to have a spatial query language that allowed for standardized spatial requests to carry out common tasks such as

- retrieve all spatial entities that are in a 300 meter radius of a certain building,
- retrieve all buildings that are directly connected to a certain building or
- retrieve all residential buildings west of an exhaust emitting factory.

However, at the moment of this writing no established query language is known to the author that allows for complex queries based on the geometry of spatial entities (in particular building models) in 3D. Most query languages only take into account relations explicitly modeled in the language the feature was modeled in but don't take into account the information implicitly given by the geometry.

2 Purpose

The purpose of this work is to investigate spatial relations based explicitly on the geometry of geographic features. This means that all geometries are referenced in a geographic coordinate system thus have a definite position and orientation in space.

In a first step (section 3) general aspects and previous work on spatial relations are examined. This includes both the formal definition and the underlying models needed for an actual implementation of a program that can decide whether the defined relation holds *true* or *false*.

In this section and in the whole course of this work three kinds of spatial relations are distinguished:

1. *Topological* (section 3.1),
2. *Metric* (section 3.1.3) and
3. *Directional* relations (section 3.2).

In section 4 the programmatic implementation of a number of spatial relations between geographical features is presented. For this purpose a Ruby program was written that is capable of

- building up an index of geographical features from a CityGML-file by parsing and subsequently converting individual buildings into a custom data model,
- computing all features from the spatial index for which a specific spatial relation holds true relative to a specified feature,
- providing access to the spatial index by means of a REST API. Return formats include XML and KMZ (Google Earth) and
- providing an interface by means of a REST API to allow for the retrieval of those features for which a specific spatial relation holds true, provided a reference feature for the relation is specified.

The choice and the actual implementation of the supported spatial relations are discussed in the sections 4.6, 4.7 and 4.8. Since a requirement for the program was to return the resulting dataset as a KMZ-file at request to allow for rendering of the feature geometries in Google Earth, a number of difficulties has to be addressed also described in section 4.

The program and Web API were tested using a subset of the 3D city model of the city of Karlsruhe in CityGML as an input. This model was kindly provided by the real estate office ("Liegenschaftsamt") of the city of Karlsruhe in LoD[1] 2. As the city model contains 1685 individual buildings the runtime of parsing, the computation of spatial relations and the generation of output files became an issue. Since the implementation did not focus on performance a short analysis of the status quo and possible improvements regarding runtime are discussed in section 5.

Eventually a conclusion and an outlook are given in section 6.

[1] LoD = "Level of Detail", see section 4.1 for more detailed information

3 Spatial relations

A spatial relation as used here is a binary relation between two real world objects with the focus on objects having dimension $d = 3$. Since we are dealing with real world objects such as houses we will only consider relation R with $R \subseteq \mathbb{R}^3 \times \mathbb{R}^3$ assuming a real world object can be considered subsets of $\mathbb{R}^3$.

Commonly three types of spatial relations are distinguished, that are

1. *Topological* (section 3.1),
2. *Metric* (section 3.1.3) and
3. *Directional* relations (section 3.2).

Each relation category will be reviewed taking into consideration a formal definition, underlying models for an implementation and possible applications.

Spatial relations that do not fall in either of the three categories such as *"close to"*, *"about 10 km southwest of"* (approximate relations) or *"into"* (relation depending on the motion of an object) will not be considered here.

For real world use one often only wants to deal with only a certain set of (theoretically infinite) possible relations. Also one wants to assign names to relations that are understandable by humans but still needs to make sure these relations are defined unambiguously on a formal level which is not always the case for solely linguistic. E.g. its is not clear what is meant exactly meant when saying an object is located *"west"* of some other object.

This poses the problems of

1. the identification of a set of spatial relations that is small enough to be reasonably used in an application and yet is big enough to not lose expressiveness,n
2. finding a formal definition for a relation that complies with the name of the relation in natural language and
3. determining a model for the actual implementation of the specific relation depending on the data structure in which the spatial objects are given.

These will be addressed in the following sections.

3.1 Topological relations

The characteristic that distinguish topological relations from other kinds of relations is that they are preserved under topological transformations (continuous maps) of the involved objects. These transformations include e.g.

- *translation*,
- *rotation* and

- *scaling.*

Hence topological relations do not depend on the size or orientation of the involved objects in space but merely on what would casually be called their connection. For example two objects would be considered neighbors if they share a common boundary. The relation "A is neighbor of B" would not depend on the size nor the orientation of this boundary. Examples of topological relations would be *insideOf*, *touching* and *intersecting*.

In an abstract sense a continuous map $f : X \to Y$ between two topological spaces X and Y, requires the preimage of any open set in Y to be an open set in X. Given a binary relation $T \subseteq X \times X$ between sets in X the relation would be unchanged if these set would undergo a transformation f first.

Since we will be dealing with real world objects rather than subsets in abstract topological spaces we will from now on consider only topological relations $T \subseteq \mathbb{R}^n \times \mathbb{R}^n$ with n being 1,2 or 3 depending on what dimension the involved objects have.

Possible applications that would make use of querying based on topological relations in a building context would be:

- querying for all rooms *"inside of"* a building (if those are modeled as separate entities), e.g. to quickly calculate the amount of color needed to paint them.
- querying for *intersecting* building geometries in two different building data that partially overlap to identify duplicates.
- identify all buildings of a building block by querying for making use of subsequent *"is touching"* queries.

3.1.1 Prerequisites

As a prerequisite for the following sections a number of definitions are given that can also be found in most elementary text books on topology and multidimensional analysis.

Let A be a subset of $\mathbb{R}^3$.

Interior
The interior of a set A is denoted by A°. It is the union of all open subsets of A and hence is a subset of A itself. It is formally defined as

$$A^o := \{x \in \mathbb{R}^3 : \exists \epsilon > 0 : U_\epsilon(x) \subseteq A\},$$

where $U_\epsilon(x_0) = \{x \in \mathbb{R}^n : \|x - x_0\|_2 < \epsilon\}$ is the unit sphere around x_0. That means that for every point x in A^o there exists some arbitrary small sphere around x that is still entirely inside of A.

Boundary
The boundary of a set A is denoted by ∂A and is formally defined as

$$\partial A := \{x \in \mathbb{R}^3 : \forall \epsilon > 0 : U_\epsilon(x) \cap A \neq \varnothing \wedge U_\epsilon(x) \cap (\mathbb{R}^3 \setminus A) \neq \varnothing\}.$$

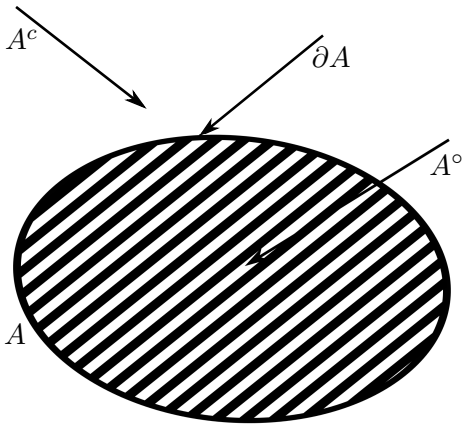

Figure 3.1: the set $A \subset \mathbb{R}^2$ together with its interior A°, boundary ∂A and exterior A^c

The idea of this definition is that an arbitrary small sphere around a point in ∂A intersects with both the set A and its exterior $(\mathbb{R}^3 \setminus A)$.

When an object is represented in BRep the boundary ∂A is thus equal to the given $n-1$ dimensional shell.

Exterior
The exterior of a set A is equivalent to the complement of A denoted by A^c. Thus it holds

$$A^c := \{x \in \mathbb{R}^3 : x \notin A\}$$

3.1.2 The 4-Intersection Model

To formalize topological relations between spatial objects different approaches were taken, one being the 4-Intersection Model which was later extended to a 9-Intersection Model. Since the latter will be used here it will be discussed in the following section.

The 4-Intersection Model (4-IM) was described by M. J. Egenhofer and R. Franzosa in 1991 [4] and makes use of point set theory. The idea is to treat objects as sets in a topological space (here $X = \mathbb{R}^3$) and to examine a 2×2 matrix composed of the mutual intersections of the objects interiors and boundaries that is defined as

$$I := \begin{pmatrix} A^o \bigcap B^o & A^o \bigcap \partial B \\ \partial A \bigcap B^o & \partial A \bigcap \partial B \end{pmatrix}.$$

As each matrix element can either be empty ($\varnothing$) or non-empty ($\neg\varnothing$), $2^4 = 16$ different configuration of the matrix are possible. Eight of those configurations are identified as predicates that occur in real situations. For objects in 2D Egenhofer identifies eight possible relations depicted in Figure 3.2.

with the corresponding 4IM matrices:

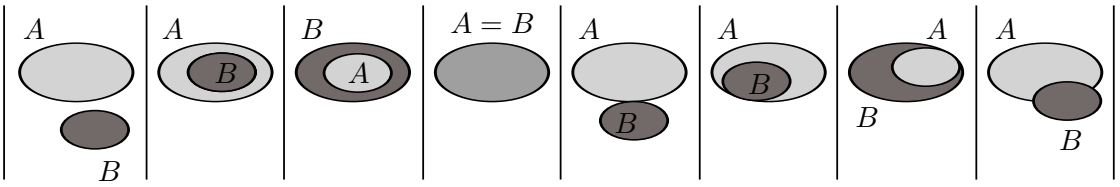

Figure 3.2: topological relations in 2D as distinguished by the 4-Intersection Model

$$\begin{pmatrix} \varnothing & \varnothing \\ \varnothing & \varnothing \end{pmatrix}, \begin{pmatrix} \neg\varnothing & \neg\varnothing \\ \varnothing & \varnothing \end{pmatrix}, \begin{pmatrix} \neg\varnothing & \varnothing \\ \neg\varnothing & \varnothing \end{pmatrix}, \begin{pmatrix} \neg\varnothing & \varnothing \\ \varnothing & \neg\varnothing \end{pmatrix}, \begin{pmatrix} \varnothing & \varnothing \\ \varnothing & \neg\varnothing \end{pmatrix}, \begin{pmatrix} \neg\varnothing & \neg\varnothing \\ \varnothing & \neg\varnothing \end{pmatrix}, \begin{pmatrix} \neg\varnothing & \varnothing \\ \neg\varnothing & \neg\varnothing \end{pmatrix}, \begin{pmatrix} \neg\varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing \end{pmatrix}.$$

3.1.3 Metric Relations

3.2 Directional Relations

Another kind of relation to distinguish are relations that take into considerations the relative directions between objects. Intuitive examples are predicates like *left*, *right*, *above*, *below*, *eastOf*, ...

In the context of city models one might be interested in all buildings west of an exhaust emitting factory, since that is the prefered wind direction in central europe. A geometry based query for bridges might be based on the *above* relationship relative to a specified street and one might want to verify the statement that riverbanks on the northern hemisphere erode more on the right handside relative to the direction of flow by querying for objects on this side of the river.

To decide whether a specific directional relation holds or to find all objects that are related to a given object in terms of a given relation with computational means, it is necessary to clearly define

1. a *set of relations* to be supported by the program and
2. *underlying models* for these relations that the computer then uses to decide whether a relation does hold or not in a given spatial context.

For both questions a number of approaches exist, some of them presented in this work in a first step.

3.2.1 formal definition

When speaking about directional relations it is important to specify a reference frame relative to which these relations are being considered. Following [9] Borrmann and Rank [2] distinguish three different kinds of reference frames, namely

- an *intrincic* reference frame that is the defined by the inner orientation of the object such as the front side of a house or the right side of a river relative to the flow,
- a *deitic* a reference frame that is dependent on the position of an external observer and
- an *extrinsic* reference frame that is defined by some external reference points.

Taking into account the different meaning of the predicates *left* and *right* in a reference frame relative to an observer or relative to the frontside of a building it becomes clear that a reference frame must be specified when using either of those predicates.

We thus define a directional relation as a tuple (D, RF) where

$$D \subseteq A \times B$$

with A and B being sets of objects and RF a reference frame.

Since directional relationships can be defined unambiguously only for point sized objects it is necessary to work out models to define those relations for spatially extended objects in higher dimensions.

Borrmann and Rank [2] list a number of previous approaches that were developed to address this problem for objects with at least one having dimension 1 or above.

3.2.2 previous work

The most simple yet common approach is to use an objects center of gravity (or some other distinguished point) and to define directional relations between spatially extended objects as 0D-0D-relationships between those points. This is easy and unambiguous, however, it sometimes yields counter intuitive results as seen in Figure 3.3 where the center of gravity of B is *above* the center of gravity of A although a person would probably classify the relationship the other way around. Thus using this simple approach might be easy to implement but sometimes lacks significance in complex scenarios.

In a two-dimensional space Borrmann and Rank [2] distinguish two different approaches to define directional relations between point size objects: the *cone-based* and the *direction-based* model.

In the cone-based model the space around a point gets dissected into a number of equally sized partitions (usually 4 or 8) and the direction of a target point from

In the direction-based model the space is divided into a northern and southern and an eastern and western halfspace relative to some reference point respectively. By projecting a target point on each pair of halfspaces one gets the four possible directions: *northEastOf*, *southEastOf*, *southWestOf* and *northWestOf*.

A different approach to define directional relations between 2D objects is to use their minimum bounding rectangle (MBR) representation.

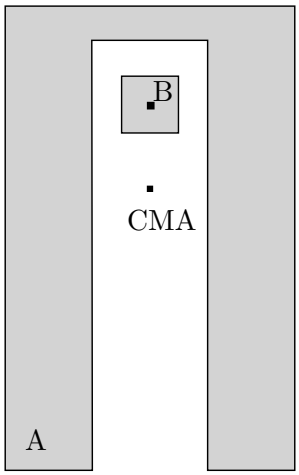

Figure 3.3: counter intuitive result when defining directional relations via the centers of gravity. The relation "B is *above* A" would hold in this case although that certainly violates common opinion.

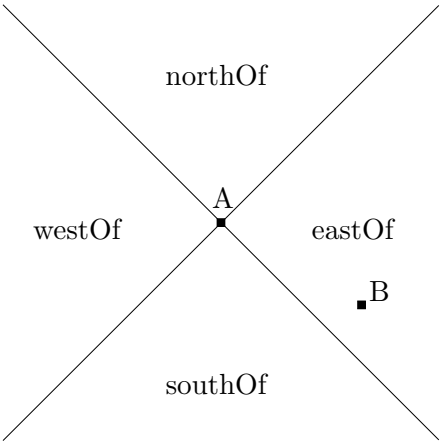

Figure 3.4: According to the cone-based model in 2D-space the point B is located *eastOf* the point A.

Definition 1. *A minimum bounding rectangle (MBR) is a rectangle, oriented to the x- and y-axes, that bounds a geographic feature or a geographic dataset. It is specified by two coordinate pairs:* $x_{\min}$, $y_{\min}$ *and* $x_{\max}$, $y_{\max}$.[2]

Given the MBR of two objects: $(x^{(1)}_{\min}, y^{(1)}_{\min}, x^{(1)}_{\max}, y^{(1)}_{\max})$ and $(x^{(2)}_{\min}, y^{(2)}_{\min}, x^{(2)}_{\max}, y^{(2)}_{\max})$ one examines the projections onto the x- and y-axis of each bounding boxe and uses the characterization of 1D interval relations between $[x^{(1)}_{\min}, x^{(1)}_{\max}]$, $[x^{(2)}_{\min}, x^{(2)}_{\max}]$ and $[y^{(1)}_{\min}, y^{(1)}_{\max}]$, $[y^{(2)}_{\min}, y^{(2)}_{\max}]$ respectively.

Having defined n relations between 1D intervals one obtains n^d interval relations in a d-dimensional space that can then be used to define directional relations between the two objects.

Allen [1] classifies 13 relations between (originally temporal) intervals in a 1D space, such as *Interval* I_1 *is before Interval* I_2 or I_1 *overlaps* I_2 that are used by other authors for classifications of 1D relationships. Out of those relations Guesgen [7] uses a set of eight (*left*, *attached*, *overlapping*, *inside* and the reversed versions) to describe to and formally reason about spatial relationships in more-dimensional models. This is done by means of d-dimensional relationship tuples, since each dimension is considered to be independent from the others.

An example of this method is pictured in Figure 3.5. If one sticks to the interval relations proposed by Guesgen one would describe the relations of A to B as the tuple (*above*, *overlapping*) where the first tuple entry refers to the x- and the second one to the y-direction. However, the *overlapping* predicate is considered to be rather a topological relation in this thesis.

Papadias et al. [8] use the whole set of Allen's 13 relations thus obtaining 169 relations in 2D but a distinction between topological and directional relations is not being made.

Borrmann also mentions a model by Goyal [6] that parts the space around a reference object on the basis of its MBR and introduces a matrix that records in which segments a target object falls. This is easily extendable to three dimensions, however, Goyal does not assign names to the different matrix configurations making it unsuitable for a spatial query language.

Coming to the conclusion that little work has been put into developing models on directions in 3D space Borrmann [2] proposes two new models: the *projection based* and the *half space model*.

the projection-based directional model Both models are suited for objects of any dimensions. Spatial relations are examined separately for each axis, thus no combined relations such as *northEastOf* is being computed. In three dimensions six distinct relations are part of the model:

The idea is to extrude the reference object in the direction of one axis (e.g. the x-axis

[2] definition according to `http://en.mimi.hu/gis/minimum_bounding_rectangle.html`

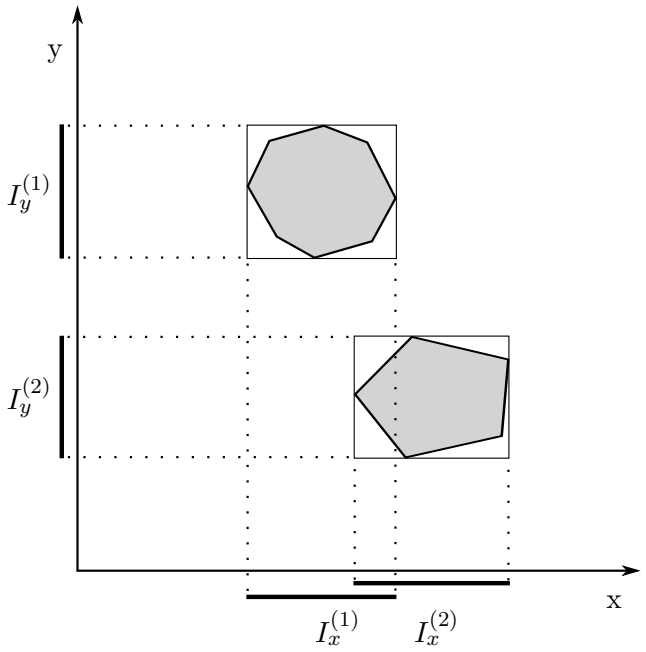

Figure 3.5: Using minimum bounding rectangles to define directional relations in 2D. The MBRs are projected on both the x- and y-axis. Subsequently one-dimensional relations between the projection intervals $I_x^{(1)}$, $I_x^{(2)}$ and $I_y^{(1)}$, $I_y^{(2)}$ are evaluated independently and are then combined in a tuple.

axis	**relations**
x	*northOf*, *southOf*
y	*eastOf*, *westOf*
z	*above*, *below*

for the *northOf* relation) and to check for intersections with the extrusion. Borrmann distinguishes a relaxed and a strict version of these relations. For the "relaxed" version to hold it is sufficient for a target object to intersect with the extrusion of the reference object in the respective direction. For the "strict" version it is necessary for the target object to lay fully inside of the extrusion on the respective side of the axis.

This can be formalized in the following way.

relaxed

$$\begin{aligned}
\text{eastOf_proj_relaxed}(A,B) &:\Leftrightarrow \exists a,b : a_y = b_y \wedge a_z = b_z \wedge a_x < b_x,\\
\text{westOf_proj_relaxed}(A,B) &:\Leftrightarrow \exists a,b : a_y = b_y \wedge a_z = b_z \wedge a_x > b_x,\\
\text{northOf_proj_relaxed}(A,B) &:\Leftrightarrow \exists a,b : a_x = b_x \wedge a_z = b_z \wedge a_y < b_y,\\
\text{southOf_proj_relaxed}(A,B) &:\Leftrightarrow \exists a,b : a_x = b_x \wedge a_z = b_z \wedge a_y > b_y,\\
\text{above_proj_relaxed}(A,B) &:\Leftrightarrow \exists a,b : a_x = b_x \wedge a_y = b_y \wedge a_z < b_z,\\
\text{below_proj_relaxed}(A,B) &:\Leftrightarrow \exists a,b : a_x = b_x \wedge a_y = b_y \wedge a_z > b_z.
\end{aligned}$$

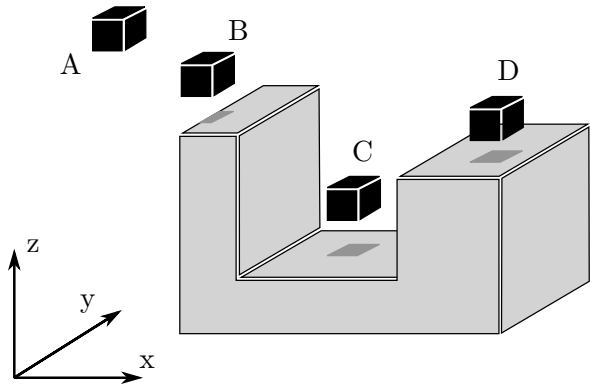

Figure 3.6: direction-based model: According to the relaxed version of the *above* relation the cubes B, C and D would be considered to be *above* the gray object. According to the strict version only the cubes B and C would be.

strict[3]

$$\begin{aligned}
\text{eastOf_proj_strict}(A,B) :\Leftrightarrow \forall b :&(\exists a : a_y = b_y \wedge a_z = b_z \wedge a_x < b_x) \wedge \\
&(\nexists a : a_y = b_y \wedge a_z = b_z \wedge a_x \geq b_x), \\
\text{westOf_proj_strict}(A,B) :\Leftrightarrow \forall b :&(\exists a : a_y = b_y \wedge a_z = b_z \wedge a_x > b_x) \wedge \\
&(\nexists a : a_y = b_y \wedge a_z = b_z \wedge a_x \leq b_x), \\
\text{northOf_proj_strict}(A,B) :\Leftrightarrow \forall b :&(\exists a : a_x = b_x \wedge a_z = b_z \wedge a_y < b_y) \wedge \\
&(\nexists a : a_x = b_x \wedge a_z = b_z \wedge a_y \geq b_y), \\
\text{southOf_proj_strict}(A,B) :\Leftrightarrow \forall b :&(\exists a : a_x = b_x \wedge a_z = b_z \wedge a_y > b_y) \wedge \\
&(\nexists a : a_x = b_x \wedge a_z = b_z \wedge a_y \leq b_y), \\
\text{above_proj_strict}(A,B) :\Leftrightarrow \forall b :&(\exists a : a_x = b_x \wedge a_y = b_y \wedge a_z < b_z) \wedge \\
&(\nexists a : a_x = b_x \wedge a_y = b_y \wedge a_z \geq b_z), \\
\text{above_proj_strict}(A,B) :\Leftrightarrow \forall b :&(\exists a : a_x = b_x \wedge a_y = b_y \wedge a_z > b_z) \wedge \\
&(\nexists a : a_x = b_x \wedge a_y = b_y \wedge a_z \leq b_z).
\end{aligned}$$

the half-space based model In the half-space based model one does not consider the extrusion of an object but the entire half space in one direction when checking for intersection of the target with the reference object. If the target object lays completely inside the half-space (e.g. above an object) one considers the "strict" version of the relation to hold.

In the "relaxed" version one does also allow for intersections of the target with the

[3]It has to be noted that Borrmann made a mistake in his original publication [2] where at some points he mistakenly exchanged a for b in the definitions of the "strict" relations for the projection-based directional model. These rather apparent mistakes were corrected here.

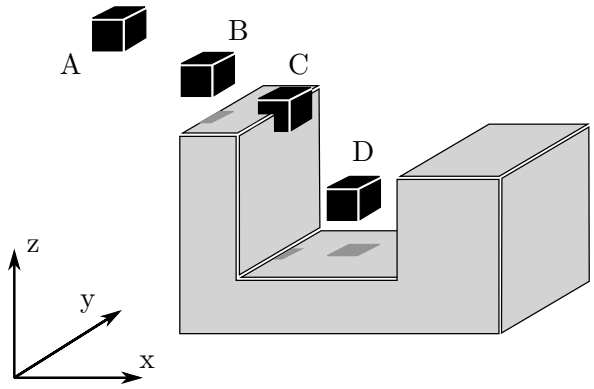

Figure 3.7: half-space based model: According to the relaxed version of the *above* relation the cubes A, B and C would be considered to be *above* the gray object. According to the strict version only the cubes A and B would be.
In contradiction to the direction based model C would not be considered to be *above* (relaxed/strict) the gray object

reference object requiring only part of the object to lay further in the direction of the specific relation.

***Example**: For the relation "B above_hs_relaxed A" to hold it is necessary that for every point of A there is at least one point of B with a greater z-component.*

Borrmann consecutively arrives at the following formal definitions:

relaxed-version

$$\begin{aligned}
\text{eastOf_hs_relaxed}(A,B) &:\Leftrightarrow \exists b : \forall a : a_x < b_x,\\
\text{westOf_hs_relaxed}(A,B) &:\Leftrightarrow \exists b : \forall a : a_x > b_x,\\
\text{northOf_hs_relaxed}(A,B) &:\Leftrightarrow \exists b : \forall a : a_y < b_y,\\
\text{southOf_hs_relaxed}(A,B) &:\Leftrightarrow \exists b : \forall a : a_y > b_y,\\
\text{above_hs_relaxed}(A,B) &:\Leftrightarrow \exists b : \forall a : a_z < b_z,\\
\text{below_hs_relaxed}(A,B) &:\Leftrightarrow \exists b : \forall a : a_z > b_z.
\end{aligned}$$

and **strict-version**

$$\begin{aligned}
\text{eastOf_hs_strict}(A,B) &:\Leftrightarrow \forall a,b : a_x < b_x,\\
\text{westOf_hs_strict}(A,B) &:\Leftrightarrow \forall a,b : a_x > b_x,\\
\text{northOf_hs_strict}(A,B) &:\Leftrightarrow \forall a,b : a_y < b_y,\\
\text{southOf_hs_strict}(A,B) &:\Leftrightarrow \forall a,b : a_y > b_y,\\
\text{above_hs_strict}(A,B) &:\Leftrightarrow \forall a,b : a_z < b_z,\\
\text{below_hs_strict}(A,B) &:\Leftrightarrow \forall a,b : a_z > b_z.
\end{aligned}$$

4 Implementation

4.1 input datasets

As the objective of this thesis is to implement geometry-based relations for spatial, georeferenced objects the first task was to identify suitable datasets as a basis for the following computation. The most important requirements for this were as follow.

1. The input data should be of low complexity to reduce computational costs and to minimize efforts for parsing and mapping the input data to the chosen internal data-structure. Especially the use of raw sensor data (e.g. laser-scanning and photo-grammetry data) was considered to be disproportional for the purpose of this work.
2. The input data should be widely-used in a geospatial context to allow for the reuse of parts of the program.
3. The data format of the input data should allow for parsing with publicly available tools that are available under a suitable license.

An appropriate candidate that was used here is an extract of the 3D city model of Karlsruhe, Germany. The data set consists of 1625 buildings in the city center spanning an area of about $2\,\mathrm{km}^2$. Each building consists of a number of building parts (mostly roof and base body) and is represented in BRep[4] with each boundary surface being a plane, convex[5] polygon.

The datasets file format is CityGML 2.0, which is an XML-based encoding for the representation, storage and exchange of virtual 3D city and landscape models.[6] CityGML is an application schema of for the Geography Markup Language (GML) which is an international standard[7] for the exchange of spatial data. It allows for the description of a wide range of city objects including vegetation, bridges and water bodies. CityGML is open, can be used free of charge and is used globally in various projects and software solutions. The CityGML website mentions infrastructure programs e.g. in the Netherlands, Germany, France, Malaysia and Abu Dhabi that rely on CityGML.

CityGML allows for the representation of 3D objects in five different levels of detail (LoD) ranging from a rather imprecise representation of a region/landscape (LoD 0) to architectural models with a modeled interior (LoD 4). It is common and possible to represent a given set of objects in multiple LoDs simultaneously allowing it applications to use the most suitable representation for a given purpose. The extract of the Karl-

[4] boundary representation: A geometry in BRep is represented by its boundary, e.g. a cube by the six squares bounding it.

[5] This characteristic allows for trivial triangulation of a polygon. If this can not be assumed, triangulation can be carried out e.g. as described by Chazelle [3] who proposed a linear time algorithm for the common case of simple polygons.

[6] `http://www.citygml.org/index.php?id=1523`

[7] issued by the Open Geospatial Consortium, OGC (`http://portal.opengeospatial.org/files/?artifact_id=4700`) and the ISO TC211 (`http://www.isotc211.org/`).

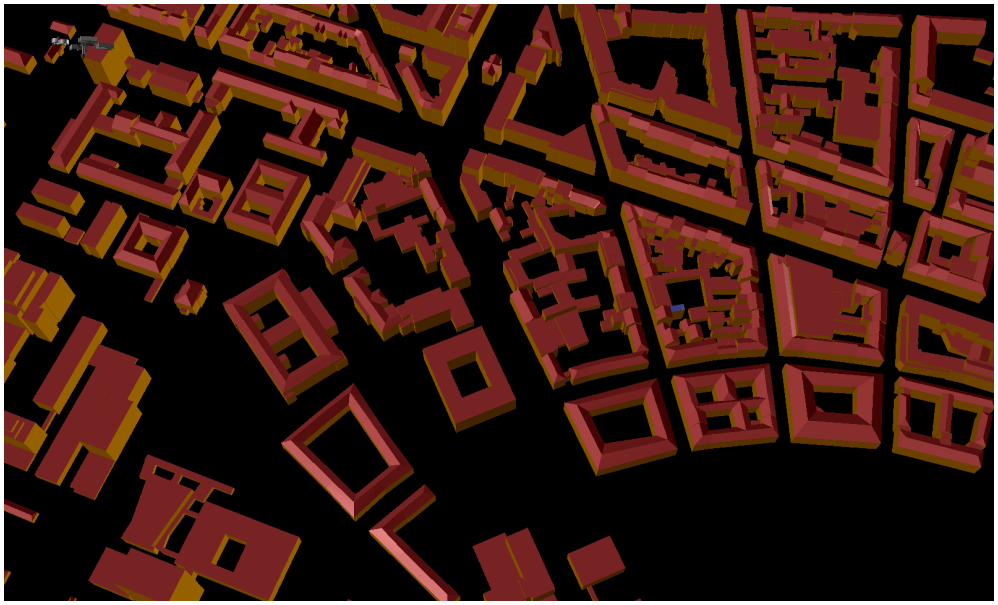

Figure 4.1: Part of the used dataset viewed in "Aristoteles3D GML Viewer"

sruhe model was made available in "LoD 2" only for the purpose of this work by the "Liegenschaftsamt Karlsruhe". Individual points were represented in the Gauss-Kruger reference system ("Gauss-Kruger, zone 3") in the particular dataset.

To also include objects of a lower dimension a number of 0D points of interest in the area of the 3D city model were obtained using the SPARQL endpoint of the German DBpedia (`http://de.dbpedia.org/sparql`) using the following query.

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX georss: <http://www.georss.org/georss>

SELECT ?res (bif:st_distance(?geo, ?bb_geo))
WHERE {
  <http://de.dbpedia.org/resource/Karlsruher_Pyramide> georss:point ?bb_geo .
  ?res  georss:point ?geo .
  FILTER (bif:st_intersects(?geo, ?bb_geo, 1000))
} LIMIT 100
```

Out of the returned dataset lat/long coordinates in WGS84, name and description were extracted.

4.2 internal data model

Before the actual computation of spatial relations can take place the input datasets are parsed and translated in an internal data model in a first step. As no suitable CityGML

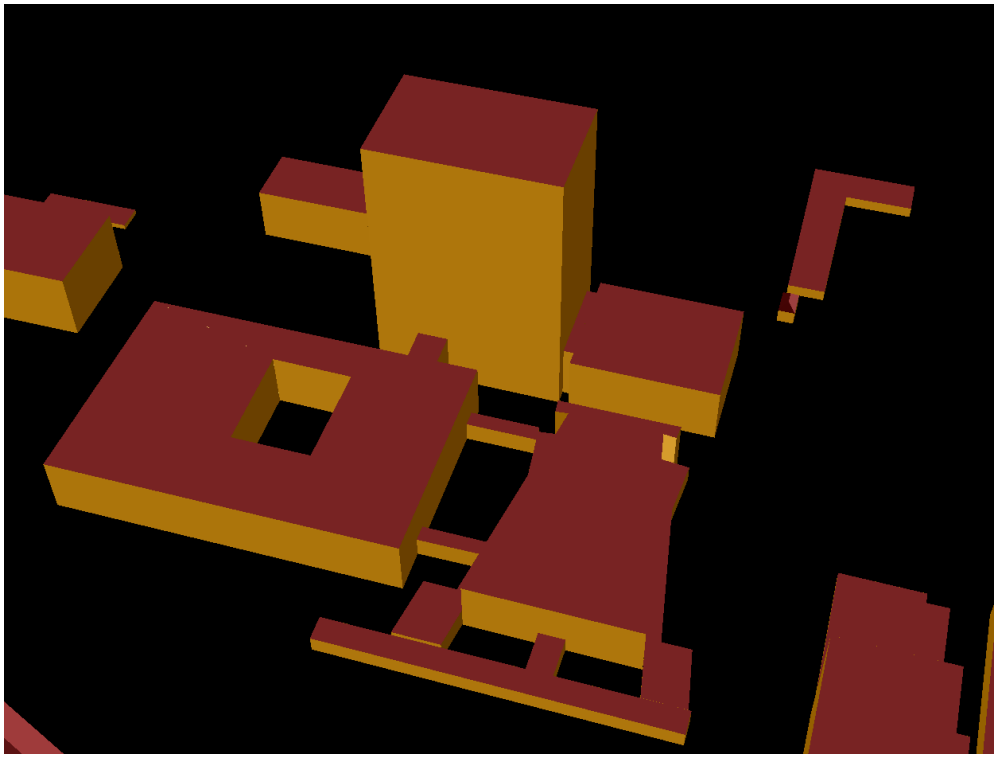

Figure 4.2: complex building in the dataset viewed in "Aristoteles3D GML Viewer"

parser with Ruby bindings was available at the time of this writing, the data was extracted with the Ruby library "Nokogiri"[8], an HTML, XML, SAX, and reader and parser based on the C-libraries "libxml2" and "libxslt", with the help of XPath expressions and CSS3 selectors. To translate WGS84 into Gauss-Kruger coordinates the bash-program "cs2cs"[9] that "performs transformation between the source and destination cartographic coordinate system on a set of input points", is invoked by the application.

To internally represent the individual features and building geometries a simple data model was developed that allows to represent a building geometry in different dimensions (0D, 1D, 2D and 3D).

A 0D-geometry is described by a single Point (east, north, height) in "Gauss-Kruger, zone 3" coordinate system which also is the fundamental entity in this model. 1D-geometries are described by a number of (connected) lines which each are represented by an ordered pair of points. 2D-geometries are modeled as a set of (simple[10] and convex) polygons

[8] `http://www.nokogiri.org/`

[9] used proj4 strings for coordinate transformation:
`+proj=tmerc +lat_0=0 +lon_0=9 +k=1.000000 +x_0=3500000 +y_0=0 +ellps=bessel +datum=potsdam +units=m +no_defs` (Gauss-Kruger, zone 3),
`+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs` (WGS84)

[10] flat and consisting of non-intersecting lines only

each represented by an ordered list of points with the first point one being equal to the last one. Finally a 3D-geometry is modeled as a list of 3D meshed which each mesh represented by its boundary as a set of (connected) polygons. Figure 4.3 shows an UML diagram of the chosen data model.

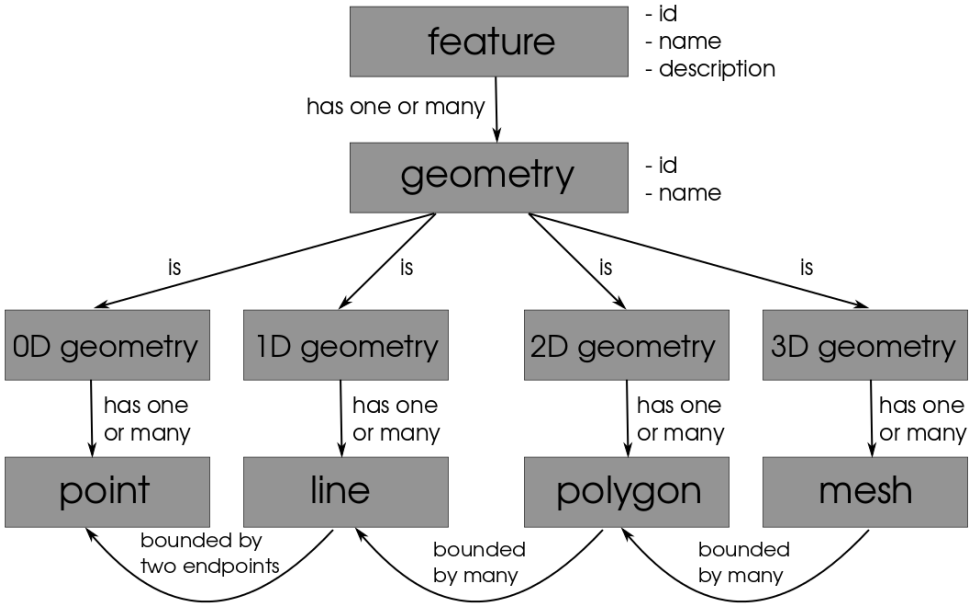

Figure 4.3: UML-diagram of the used data model to represent a feature

4.3 KML/Collada Export

As a main goal of this work was to visualize the computed relations within Google Earth it was necessary to export the 3D building models into Collada individually in a first step and to reference these Collada files in a KML file in a second step. This KML file can then be read by Google Earth to display the geometries on a world map.

The main requirement was to display a building from which a certain relation is computed in a certain color, and color all other building according to whether the relation holds between the two buildings or not. As KML does not support the assignment of colors or materials to 3D geometries within the file this has to be done in each Collada file individually.

A further obstacle is the fact that KML does only support a subset of the Collada specification and thus only allows for triangles as primitive geometry type[11]. This leads to the further requirement of transforming the polygon BRep geometries into sets of

[11] `https://developers.google.com/kml/documentation/kmlreference#model`

triangles which can then be represented in a Collada file. Assuming (as indicated before) for the polygons to be convex this was done linear time by choosing an arbitrary point of the polygon and connecting it to all other points but itself or is neighbors. Having a polygon with n vertices one so obtains a set of $n-2$ triangles. It though as to be remarked that this only does work for simple and convex polygons. If one drops the assumption of the polygons to be convex (but still simple) the triangulation can be done in linear time as proposed by Chazelle [3]. The implementation of this algorithm is fairly complex though.

In the actual implementation each `Feature` class contains a method `to_dae` that creates a Collada representation of the features, given a name, id and filepath. If the feature does not have a 3D geometry an Exception is raised.

4.4 API-Design

In order to provide an interface for the creation and retrieval of geographical features and the computation of spatial relations a REST-API was designed and subsequently implemented using the Ruby micro-framework "Grape"[12]. Grape provides a Ruby DSL for the development of REST-like APIS and supports a wide functionality, like e.g. multiple formats, versioning and content negotiation. Grape supports XML, JSON, BINARY and TXT content types. At the time of this writing its current stable release is 0.9.0. Grape is published under the MIT-License.

For matters of simplicity an API with only a very basic functionality was designed to conform to the following requirements:

1. building up a data basis of spatial features
2. retrieve information about this data basis (e.g full feature list, information about individual features)
3. retrieve a subset of the features data basis so that each feature in the data basis holds a specific relation (e.g. eastOf_hs_relaxed) relative to a specified feature. The possible serializations of this subset should include the serialization in KMZ to enable rendering of the feature geometries in Google-Earth.

The API provides programmatic access to features and relations by means of the following requests. The status codes of the server responses were chosen in accordance with RFC 7231 (Hypertext Transfer Protocol, HTTP/1.1) [5]. All requests are based on a standard domain with FQDN `example.org`.

Since a unique GET-access to a single feature is provided by "GET features/show/:id" the URI `http://example.org/features/show/:id` serves as a unique identifier to a feature.

GET features/lookup

[12] `https://github.com/intridea/grape`

Returns all features as a list. It no features are found, “404 Not Found” is returned.

Example Request

```
GET

http://api.example.org/features/lookup
```

Example Response:

```
<feature_list>
  <feature id="sample-feature-id-1" name="sample-feature-name-1">
    <uri>http://api.example.org/features/show/sample-feature-id-1</uri>
  </feature>
  <feature id="sample-feature-id-2" name="sample-feature-name-2">
    <uri>http://api.example.org/features/show/sample-feature-id-2</uri>
  </feature>
</feature_list>
```

GET features/show/:id

Returns a single feature, specified by the id parameter. The output format is the custom format CFF (as described in section 4.2), serialized in XML. It no feature with :id is found, “404 Not Found” is returned.

Example Request:

```
GET

http://api.example.org/features/show/sample-feature-id
```

Example Response:

```
<cff>
  <feature id="sample-feature-id" name="sample-feature-name">
    <description>sample description</description>
    <geometry>
      <geometry_3d id="sample-geometry-id" name="sample-geometry-name">
        <mesh>
          <polygon>
            <point>5430149.143 3456533.163 136.410</point>
            <point>5430150.332 3456528.155 136.410</point>
            <point>5430150.332 3456528.155 124.948</point>
            <point>5430149.143 3456533.163 124.948</point>
            <point>5430149.143 3456533.163 136.410</point>
          </polygon>
      ...
  </feature>
</cff>
```

POST features/create_from/gml

creates a number of features from a CityGML-file that is included in the POST requests body.

Parameters:

- **gml_file (required)**
  CityGML containing a number of building geometries

Example Request

```
POST

http://api.example.org/features/create_from/gml
```

Sample Curl-Request:

```
curl -F gml_file=@sample.gml http://api.example.org/features/create_from/gml
```

GET features/by_relation/:relation/reference_feature/:id

Returns a list of those features for which the relation :relation relative to a single reference feature with id :id holds true.

Parameters:

- **gml_file (required)**
  CityGML containing a number of building geometries

4.5 Web Interface

4.6 Topological Relations

4.7 Metric Relations

4.8 Directional Relations

4.8.1 Prerequisites

To determine a set of directional relations to query the specific city model in this work and to decide about which underlying models to use for the implementation of each relation some of the specific characteristics of the city model Karlsruhe and its potential use, that do influence that choice, are being recapped.

1. The city model used in this work is available in a fairly low level of detail (LoD2). This resolution does not allow for the modeling of separate floors or basements which makes it unnecessary to implement directional relations for the z-axis. However, if floors and basement are modeled as separate entities (e.g. in city models

of higher resolution) such a query function would be desirable.

Another consequence of this low level of detail is that considering the extrusion of an object when defining a particular directional relation is not very meaningful. This would have applications when querying for columns and walls *above* a certain ceiling or for all building parts *above* the floor plate.

Since a house is modeled in one or two parts only in the given city model the only reasonable queries the author can think of that would use this kind of relations would be queries for the roof of a building (*above_proj_strict/relaxed* house_base_body) or the neighboring buildings in a specific direction (e.g. *eastOf_proj_relaxed* building). These queries seem not important enough to implement them in the API.

2. As a city model will mostly be used in a way that is independent of the way one observes it (in contradiction to e.g. a virtual reality world for the purpose of direct user interaction), it seems appropriate not to include relations that require a deitic reference frame as defined in section 3.2. Thus relations of the kind *leftOf* and *rightOf* that depend upon the position and the view angle of the observer will be omitted.

Hence in the actual implementation the set of the following relations

- *eastOf_hs_relaxed*,
- *westOf_hs_relaxed*,
- *northOf_hs_relaxed*,
- *southOf_hs_relaxed*,

as previously described in section 3.2.2 are being used. In a more detailed model it would be very useful to extend this set but due to the above reasons this does not seem adequate for the special case.

Given the input data in a BRep representation (Borrmann [2] discusses the case for spatial objects in Octree representation) the implementation is fairly easy. If V_{O_1} and V_{O_2} denote the set the set of vertices in 3D space of the spatial objects O_1 and O_2 the following holds.

$$\text{eastOf\_hs\_relaxed}(O_1, O_2) = \begin{cases} true, & \max\{v_x : v \in V_{O_2}\} > \max\{v_x : v \in V_{O_1}\} \\ false, & \text{otherwise} \end{cases}$$

$$\text{westOf\_hs\_relaxed}(O_1, O_2) = \begin{cases} true, & \min\{v_x : v \in V_{O_2}\} < \min\{v_x : v \in V_{O_1}\} \\ false, & \text{otherwise} \end{cases}$$

$$\text{northOf\_hs\_relaxed}(O_1, O_2) = \begin{cases} true, & \max\{v_y : v \in V_{O_2}\} > \max\{v_y : v \in V_{O_1}\} \\ false, & \text{otherwise} \end{cases}$$

$$\text{southOf\_hs\_relaxed}(O_1, O_2) = \begin{cases} true, & \min\{v_y : v \in V_{O_2}\} < \min\{v_y : v \in V_{O_1}\} \\ false, & \text{otherwise} \end{cases}$$

For optimization purposes the minimum and maximum of the x-, y- and z-component of the vertices of a geometry are saved after parsing to avoid repeated sorting for multiple directional queries of a kind.

5 performance

6 outlook

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.

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

[3] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.

[4] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *International Journal of Geographic Information Systems*, (5(2)):161–174, 1991.

[5] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231 (Proposed Standard), June 2014.

[6] Roop Goyal, Roop K Goyal, and K Goyal. Similarity assessment for cardinal directions between extended spatial objects. 2000.

[7] Hans Werner Guesgen. *Spatial reasoning based on Allen's temporal logic*. International Computer Science Institute Berkeley, CA, 1989.

[8] Dimitris Papadias, Timos Sellis, Yannis Theodoridis, and Max J Egenhofer. *Topological relations in the world of minimum bounding rectangles: a study with R-trees*, volume 24. ACM, 1995.

[9] Gudula Retz-Schmidt. Various views on spatial prepositions. *AI Mag.*, 9(2):95–105, July 1988.