COMPUTING WATERTIGHT VOLUMETRIC MODELS FROM BOUNDARY REPRESENTATIONS TO ENSURE CONSISTENT TOPOLOGICAL OPERATIONS

Markus Wilhelm Jahn and Patrick Erik Bradley

Karlsruhe Institute of Technology, Geodetic Institute (GIK), Karlsruhe, Germany
{markus.jahn, bradley}@kit.edu

Commission IVWG 7

KEY WORDS: Topology, Boundary Representation, Computational Geometry, Graph Database, City Model, CityGML

ABSTRACT:

To simulate environmental processes, noise, flooding in cities as well as the behaviour of buildings and infrastructure, 'watertight' volumetric models are a measuring prerequisite. They ensure topologically consistent 3D models and allow the definition of proper topological operations. However, in many existing city or other geo-information models, topologically unchecked boundary representations are used to store spatial entities. In order to obtain consistent topological models, including their 'fillings', in this paper, a triangulation combined with overlay and path-finding methods is presented by climbing up the dimension, beginning with the wireframe model. The algorithms developed for this task are presented, whereby using the philosophy of graph databases and the *Property Graph Model*. Examples to illustrate the algorithms are given, and experiments are performed on a data-set from Erfurt, Thuringia (Germany), providing complex geometries of buildings. The heavy influence of double precision arithmetic on the results, in particular the positional and angular precision, is discussed in the end.

1. INTRODUCTION

Wireframe models are one of the basic models to describe any kind of spatial entities used in city models or other geo-information models. One of the major issues is to triangulate those one-dimensional geometry types into higher-dimensional geometry types like triangle nets as surface models embedded into three-dimensional Euclidean space. Doing so will add value to the data-set. E.g. queries like "How large is the area of walls between neighbouring buildings?" can be answered thus. The same situation arises when dealing with boundary representation models (*B-Rep*) consisting of surfaces in order to model solids. A triangulation to true three-dimensional geometry types like tetrahedral nets provides more answerable queries than a *B-Rep* which only represents the boundary surface of a solid. An example is the query "How large is the overlapping volume within the city model?". Methods dealing with the construction of solids from wireframe models so far assume that the underlying model is topologically consistent, i.e. its building blocks form a partitioning of the space to be modelled, meaning that they do not overlap. If the model is topologically inconsistent, then such queries are not trivial to be answered using a *B-Rep*. The reason is that costly steps have to be taken to calculate the triangulations leading to higher-dimensional geometry types. If a city model for example consists of overlapping pieces of wall polygons, roof polygons, or even building parts are made up of overlapping planar polygon pieces, it gets much harder to retrieve those higher-dimensional geometry types they represent by being a *B-Rep* made of a collection of lower-dimensional *B-Rep*'s, where we consider not only connected planar surfaces as a *B-Rep* for solids, as in (LaCourse, 1995), but any n-dimensional representation of the boundary of an $n+1$-dimensional model, and call this also a *B-Rep*.

The main problem in dealing with topologically inconsistent models is that the inconsistency can affect all dimensions, meaning that there are overlaps between objects of any dimensions in the topologically inconsistent model. This means that the increment from dimension n to $n+1$ needs to be combined with a method that produces output which is guaranteed to be topologically consistent. In other words, many intersections need to be computed. This leads to numerical issues affecting the partitioning of overlapping pieces using double-precision arithmetic. If angles get to small, then intersection points between lines or intersection lines between planes become uncertain. If algorithms are chained in pipelines to achieve more complex results, then the uncertainty accumulates, too, or the pipelines break up, producing no results at all. This major problem of computational geometry is faced when using standard double-precision arithmetic. Nevertheless, calculating a topological model is sensitive and expensive with double-precision arithmetic (e.g. due to intersections, differences, overlay etc.) and has higher memory usage than only storing topologically inconsistent *B-Rep*s. But it provides reusable information on how the pieces are connected, thus yielding benefits and outlays which need to be examined.

This paper addresses the problem of producing watertight (i.e. topologically consistent) volumetric models from topologically inconsistent input data in the form of a polygonal *B-Rep* whose polygons are given as a collection of wireframes. In the example, the input data is an LoD2 CityGML dataset of Erfurt, Thuringia. For the handling of topologically inconsistent data, we provide a conceptual model based on the *Property Graph Model* and OGC's *General Feature Model* ISO 19109 and *Simple Feature Model* ISO 19107 design patterns which have already been partly introduced in (Jahn et al., 2017) or (Breunig et al., 2016).

Within geo-information there are formats like *CityGML* or *GML* which are XML-based, and hence semi-structured according to (Erl et al., 2015). One basic idea of geo-information is described in the *General Feature Model* ISO 19109 and *Simple Feature Model* ISO 19107. Everything may be seen as a *Fea-*

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W2-2021-21-2021

ture, a real world object with certain properties. The model insists on three different property types, the *spatial part*, the *temporal part* and the *thematic part*. The *Simple Feature Model* defines the spatial part and which geometry types should be provided. The main reason for using the *Property Graph Model* is that it is a basic model for graph databases to store topological information, in our case the topology of features. The connection between those two models and the introduction of basic relation types lead us to an algorithm which is able to tetrahedralise wireframe models by analysing the spatial parts and the topology. Even if the GML standards require topologically consistent data, in reality they do contain overlaps which are not explicitly modelled in the data (Giovanella et al., 2019).

In this article, the algorithm for transforming topologically inconsistent wireframe data into topologically consistent volume models is described, and a couple of results generated from a LOD 2 *CityGML* input data-set of Erfurt, Thuringia (Germany) are provided. The results have been produced for two companion papers on sensitivity analysis using double-precision arithmetic and the parallelisation of computing higher-dimensional models from lower-dimensional models in order to evaluate the conceptual model and the problems. This is not to be confused with the LOD paradigm, where the aim is to separate different LODs. Our aim here is to calculate higher-dimensional geometric models from topologically inconsistent lower-dimensional geometric models only. Therefore, the application is not restricted to city models, be they CityGML, IFC etc. It is also to say that any geometric model may be triangulated to higher-dimensional models in that sense. We also do not focus on any specific application (e.g. city models, sub-surface models, land use models etc.) in order to provide a versatile spatial or spatio-temporal model to manage a broad variety of geo-data which in turn is more easy to be analysed in a general way, i.e. the way of topology and graphs.

The following section will summarise the related work. The following Section 3 on methods introduces the data structure and the algorithms which have been implemented to solve the tetrahedralisation. This is followed in Section 4 by a discussion of the experimental results. The final Section 5 consists of the conclusions.

2. RELATED WORK

One workaround to answer the specific question of "How large can a city volume be?" was to use ray casting methods, like e.g. *The Voluminator* (Sindram et al., 2016), to produce raster data of a certain density. However, the input data are triangulated polygons which get shot by the ray casting method, which causes the same computational problems. Namely, the first problem is how to triangulate the polygons in order to form planar surfaces. The second problem is the uncertainty of the intersection point when the ray casting method is used on triangles with different normal vectors against the ray. There is also another problem just before those problems. Namely, how to produce the polygon as a planar closed ring in a way that a lot of planar closed rings actually form a computable closed hull as connected planar surfaces. Besides the arithmetic problems, there are memory issues also. As a matter of fact, a topologically well connected wireframe may save memory by not saving redundant polygon parts which are shared by neighbouring polygons, like two planar walls share the edge of a room.

Going beyond volume calculation, an important notion for having data suitable for simulations is that of topological consistency, as defined in (Giovanella et al., 2019), where also competing definitions from the literature can be found. In that article, it was found that CityGML data is usually not topologically consistent. Definition 2.1 below paraphrases this notion which is in compliance with the ISO 19107 standard. We observe that the notion from (Ledoux and Meijers, 2011) comes close to the definition used here.

Definition 2.1 *A configuration of geometrical objects in Euclidean space is* topologically consistent*, if two objects are either disjoint or identical.*

The main advantage of this definition is that it allows any topological query to be performed on the topological model itself without having to resort to geometry with its costly computations like intersection and others. Having such a notion of topological consistency, healing as e.g. in (Zhao et al., 2014) becomes conceptually more feasible. In this article, this becomes an important pre-processing step using computational geometry in order to achieve topologically consistent data. This gives rise to numerical issues with an accumulation of floating-point errors, as already seen e.g. in (Biljecki et al., 2014).

In the context of topologically consistent data, there are methods for computing solid models from wireframe. E.g. one using a shortest path method for calculating the B-Rep of solids from a wireframe is given in (Bagali and Waggenspack Jr., 1995). Other independent methods are (Vosniakos, 1997, Inoue et al., 2003). Topologically consistent 3D-models including tetrahedra are necessary for simulation models for environmental processes, infrastructure, noise or flooding (Ledoux, 2008, Breunig et al., 2017, Stoter et al., 2008, Zhang et al., 2006). Namely, it is topology that captures the relationships between entities, be they geometric or not. The use of topology in geo-informatics is widespread, leading to a vast literature beginning with the application of point-set topology in (Egenhofer and Franzosa, 1991). There, various relationships between regions are deemed topological. Already P. Alexandrov proved that any binary relation gives rise to a topology on a finite set of objects (Alexandrov, 1937), and this led to the notion of topological databases in (Bradley and Paul, 2010), fundamental for this work. The article (Breunig et al., 2020) reviews present and future directions for the application of topology in the management of geo-data. The question of how to produce higher-dimensional models from topologically inconsistent lower-dimensional models has not yet been addressed.

3. METHODS

3.1 Finite T_0-spaces

Here, we explain how the most general topology for geo-information arises from binary relations. It was Pavel Alexandrov who showed that any partial ordering on a finite set gives rise to a so-called T_0-topology and vice versa (Alexandrov, 1937). This very general kind of topology captures the situations where no directed circuit paths are allowed. Its flexibility can capture a lot of applications. A particular instance is the boundary relationship between geometric objects. Given a possibly topologically inconsistent boundary representation of a 3D model, i.e. a model not satisfying Definition 2.1, the overlay method below produces a topologically consistent finite T_0-space which captures the topology inherent in the geometric model. This model has a minimal representation as a directed acyclic graph, called

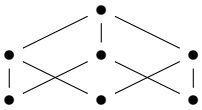


Figure 1. *Hasse diagram* of a triangle with its boundary relation.

Hasse diagram such that the reflexive and transitive closure of the edge relationship yields the topology in the sense of (Alexandrov, 1937). Fig. 1 illustrates this for the example of a triangle: The top node represents the area which is bounded by its three side edges, represented by the middle row of nodes, and each of these are bounded by two corners represented by the lower row of nodes. The edges are oriented downwards and represent the boundary relationship between objects.

In (Bradley and Paul, 2010) it was observed that this general structure can be implemented in a relational database with one table for the 'nodes' and another table for the direct 'relationships', and the reflexive and transitive closure operation can be used to make topological queries on this database. In applications, any entities considered as 'nodes' and any binary 'relationship' thus generates a topology, thus allowing topological methods to be applied to this setting, even if the entities do not carry geometric or topological information themselves. The most efficient way to store a T_0-topology in a graph database is to store the nodes and edges of its Hasse diagram, as this avoids the storing of redundant relationships. In this case, topological queries like neighbourhoods or connectivity become path queries in a graph. This is the approach followed in this article.

3.2 *Property Graph Model*

In order to store Hasse diagrams, we use the *Property Graph Model*. Every node of a graph within the *Property Graph Model* carries certain properties. The property types were chosen from the *Simple Feature Model* 19107 and the *General Feature Model* 19109, where every feature has a spatial part, a temporal part and a thematic part. We also defined six relationship types concerning aggregation, abstraction and incidence relations (Jahn et al., 2017). Furthermore, three different node types were implemented, cf. Fig. 3. One for non-spatial or non-spatio-temporal contents, one for spatial contents and one for spatio-temporal contents. Each maintains the basic graph node properties: name, description, identification, temporal and thematic. The relations to other nodes are organised by maps (key-value pairs). The key has two attributes, one for the type of relation and one for the node id.

Five basic building operations have been implemented within the spatial model which build an aggregation node, a number of sub-component nodes, overlay nodes, border nodes, or a $d+1$-dimensional node from a d-dimensional node. The operation to build an aggregate node collects the spatial properties of all *composite-of* neighbours into one spatial collection and replaces the node which called that operation by a new spatial node within the *Property Graph*. The new node is added to a spatial access method if the parameter was set. Its inverse operation creates all sub-nodes by analysing the spatial property. In case of a spatial collection as spatial property all non-empty d-dimensional spatial networks are linked with new sub-nodes as their spatial parts. Each new sub-node is then related to the main node in an aggregation relation. Fig. 2 illustrates all possible aggregation relations within the spatial and spatio-temporal model. The standard behaviour as described may be changed by manipulating the spatial predicate within this operation e.g. a node with a spatial collection as spatial property may create new sub-nodes which are linked to all contained spatial elements (simplices) directly. The operation which builds all overlay nodes calculates all intersections (ignoring border intersections) between the spatial part of a node and the spatial parts a set of given nodes. Each *node-to-node* intersection is linked to a new node and this new node is related to its parents in an aggregation relation in order to find topological inconsistencies. The operation for building all border nodes creates the border nodes of a node by calling the *getBorder()* operation of the spatial property and adds the nodes in an incidence relation. A spatial predicate is used to control the complexity of the sub-nodes.

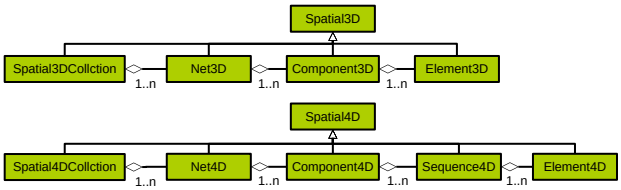


Figure 2. Spatial and spatio-temporal model

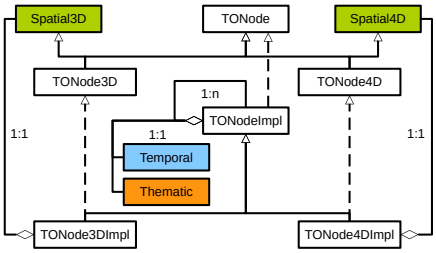


Figure 3. Implemented *Property Graph Model*

3.3 From d-dimensional to $d+1$-dimensional simplicial networks

The previous subsection introduced the implementation of the *Property Graph Model*. Here, we describe a method for producing a topologically consistent $d+1$-dimensional model from a topologically inconsistent d-dimensional model. It uses a combination of an overlay and a shortest-path method for finding cycles. In the following, we call any aggregation of simplicial complexes, which is not necessarily topologically consistent, a *simplicial network*. The operation which builds a $d+1$-dimensional node from a d-dimensional node is rather complex and contains the main algorithm to calculate d-dimensional simplicial networks into $d+1$-dimensional simplicial networks. Complexes in our model always maintain the manifold constraint and are made of equi-dimensional simplices only. The main idea behind the algorithm is to picture a d-dimensional simplicial complex as a patch and to cut and stitch a couple of patches together wherever there is a $d-1$-dimensional intersection between them to form a patchwork. This patchwork can be transformed into an incidence graph which can be analysed to find closed d-dimensional simplicial complexes, i.e. patchworks forming *B-Rep*'s. Those *B-Rep*'s can be triangulated to $(d+1)$-dimensional simplicial complexes. The disjoint union of $(d+1)$-dimensional simplicial complexes can be found by rejecting all triangulated $(d+1)$-dimensional simplicial complexes which non-trivially intersect any of the patches. The basic steps are shown in Alg. 1. Two examples are given in Fig. 4

and 5.

Algorithm 1: computing d-dimensional simplicial networks into $d+1$-dimensional simplicial networks

input : set of d-dimensional simplicial complexes
output: disjoint set of $(d+1)$-dimensional simplicial complexes

1 collect $(d-1)$-dimensional intersections;
2 create disjoint d-dimensional patches;
3 create disjoint $(d-1)$-dimensional seams;
4 stitch seams which share the same patches;
5 stitch patches to manifolds and clean up;
6 stitch manifolds to form *B-Rep*'s and triangulate them to $(d+1)$-dimensional simplicial complexes;
7 create difference;

Steps 1 to 5 of Alg. 1 are pre-processing steps to build the T_0-space to be used in step 6. Steps 1 to 3 create the T_0-space and steps 4 and 5 reduce the T_0-space. In step 6, a *Dijkstra* algorithm is used to search loops between each connected pair of patches in order to find all *B-Rep*'s. If the shortest path leads to a *B-Rep* which contains any patch, it is rejected and the *Dijkstra* algorithm retrieves the next path which is going to be checked. Special care has to be taken in case of curves in order to ensure the planarity of the triangulated surfaces. The algorithm in step 6 has been parallelized by giving each thread a different pair of patches, which reduced computation time significantly.

3.4 Constructing disjoint simplicial complexes

The algorithm of the previous subsection also uses standard algorithms of computational geometry. One of these is finding intersections. Our model uses a standard procedure shown in Algorithm 2. A wireframe generated within step 11 needs to be triangulated into a convex complex of a certain dimension. For this, we use a standard method which triangulates the point cloud of the wireframe from the centre to the most outer point by a sweep algorithm. Intersecting each simplex of a complex with any object will retrieve the overlaps.

Algorithm 2: simplex-to-spatial intersection

input : Simplex3D A and Spatial3D B to intersect with another and Spatial3D C to subtract from the intersection
output: Intersection of A and B without C

1 if A or B is instance of Point3D
  $\rightarrow$ return results of contains-point test;
2 if complexity(B) $>$ complexity(A)
  $\rightarrow$ return B intersection with A;
3 if MBB(A) does NOT intersect MBB(B)
  $\rightarrow$ return null;
4 if B contains-all-points of A
  $\rightarrow$ return A difference with C and vice versa;
5 create Wireframe3D W containing each point of A which is contained by B and vice versa;
6 if A is a SOLID3D
  $\rightarrow B' = B$;
7 if A is a SURFACE3D
  $\rightarrow B' = B$ intersection with plane of A;
8 if A is a CURVE3D
  $\rightarrow B' = B$ intersection with line of A ;
9 intersect border of B' with A and add results to W;
10 intersect border of A with B' and add results to W;
11 triangulate W into a Spatial3D with maximal dimension of minimum(dimension(A), dimension(B)) and return result;

A difference operation was added for enabling all operations needed in set theory. The difference between a segment and

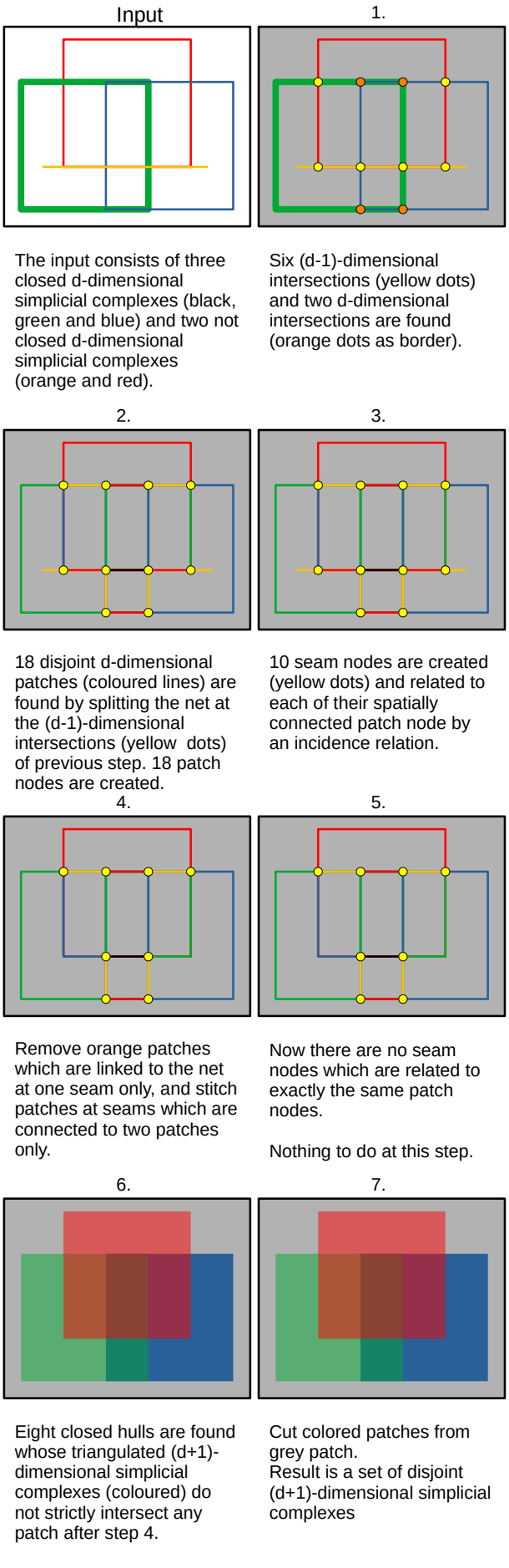
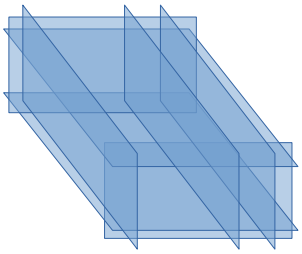


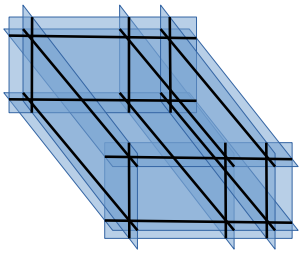
Figure 4. Five 1-dimensional simplicial complexes triangulated to two 2-dimensional simplicial complexes

Input



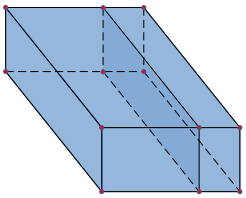
The input consists of seven d-dimensional simplicial complexes.

1.



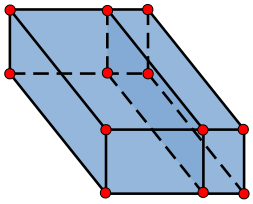
16 (d-1)-dimensional intersections (black) are found.

2.



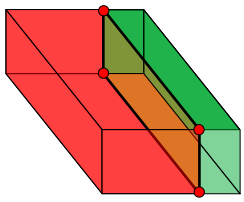
11 disjoint d-dimensional patches are found (blue) by applying the algorithm on the set of (d-1)-dimensional intersections of the previous step. 11 patch nodes are created.

3.



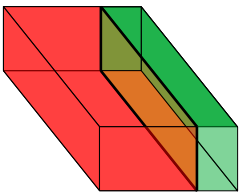
20 seam nodes (black with red dots as borders) are created and related to each of their spatially connected patch node by an incidence relation.

4.



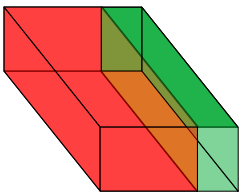
Three patch nodes (red, green, orange) remain after stitching patches at seams which were connected to two patches only. 4 seam nodes remain (black with red dots as border).

5.



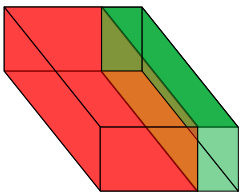
One seam node (black) remains after stitching the four seams which are related to exactly the same patch nodes.

6.



Two closed hulls are found whose triangulated (d+1)-dimensional simplicial complexes do not strictly intersect any patch after step 4.

7.



There does not exist any surrounding (d+1)-dimensional simplicial complex.

Nothing to do at this step.

Algorithm 3: simplex-to-spatial difference

input : Simplex3D A and Spatial3D B
output: A without B

if MBB(A) does NOT intersect MBB(B) → return A;
d = dimension(A);
I = intersection between A and B;
if dimension(I) != dimension(A)
  → return A;
if I is a NET3D
  → subtract each component of B from A and return difference;
A_b = border of A;
I_b = border of I;
$A'_b = A_b$ without I_b;
$I'_b = I_b$ without A_b;
A'_{bb} = border of A'_b;
D_b = intersection of A'_b and I'_b without A'_{bb};
if $D_b! = NULL$ →
  triangulate d-dimensional simplices around elements of A'_{bb}
  add them to result set
  add their united border to A'_b;
if $D_b == NULL$ and $A'_{bb} == NULL$ →
  triangulate d-dimensional simplices between A'_b and I'_b
  add them to result set
  add their united border to A'_b;
glue A'_b and I'_b and triangulate all *B-Rep*'s;
add the triangulated *B-Rep*'s to result set;
build a valid *Spatial3D* from result set and return it;

Algorithm 4: Small bite by sweep

input : set of segments which form polygon (*B-Rep*)
output: set of triangles which form triangle complex

geometrically sort the corner points of the input set (e.g. ascending XYZ order);
for *corner point in sorted corner point set* **do**
  take the two segments which share the corner point;
  create a triangle by adding the missing segment;
  if the triangle is not valid or intersects the rest of the polygon
    → try a different geometrical sort and go on;
  remove the two segments which share the corner point from the input set;
  add the newly created segment to the input set if it is not contained in the input set to close the input set;
  add the triangle to the result set;
end
return the set of triangles;

Algorithm 5: Big bite by sweep

input : set of triangles which form closed triangle complex (*B-Rep*)
output: set of tetrahedrons which form a tetrahedron complex

geometrically sort the corner points of the input set (e.g. ascending XYZ order);
for *corner point in sorted corner point set* **do**
  take all triangles which share the corner point (rice hat);
  triangulate its border polygon to create the second triangle complex (rice hat capping);
  create the tetrahedrons from the corner point to each triangle within the second triangle complex;
  try to bite each tetrahedron;
  if any not solvable problems occur →try a different geometrical sort and go on;
end
return set of tetrahedrons;

Figure 5. Six 2-dimensional simplicial complexes triangulated to

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W2-2021-21-2021 | © Author(s) 2021. CC BY 4.0 License.

any spatial object reduces to an interval difference algorithm. Algorithm 3 shows the basic steps to calculate a simplex-to-spatial difference for triangles and tetrahedrons. However, the algorithm includes a triangulation of *B-Rep*'s in step 14 which is described in Algorithm 4 and 5. Special care has to be taken, if holes are cut out of a simplex. This is done in steps 12 and 13 of Algorithm 3 which reduces the problem to a triangulation of disjoint *B-Rep*'s.

Algorithms 4 and 5 are sweep algorithms which triangulate concave polygons or closed hulls (closed triangle complexes). Both algorithms were designed with two constraints. First, no additional points should be added to the point set. Second, the input is a closed d-dimensional simplicial complex and the output is a $d+1$-dimensional simplicial complex. The basic idea of Algorithm 4 is to bite a triangle from a polygon. The algorithm does not necessarily depend on planar polygons. While the polygon itself is a collection of segments unambiguously connected through their bordering corner points, it is possible to sort those corner points and sweep over this list to generate $(d+1)$-dimensional simplices. The algorithm will fail if all sweep orders find invalid triangles (validity in accordance with the inaccuracy model, e.g. not too small, not too sharp) or valid triangles which intersect any of the remaining segments of the polygon only. It triangulates as long as there are neither invalid triangles nor bad intersections.

It is possible to use the *Delaunay* condition in order to reduce the number of sharp triangles while sweeping over the set of corner points. The *Delaunay* condition reduces the number of ways a surface is able to be built, but since the algorithm depends on the type and order of the geometrical sort, there still exists a number of ways surfaces are able to be built.

The basic idea of Algorithm 5 is to bite a couple of tetrahedra from a closed triangle complex (closed hulls). For each corner point, there exists a set of triangles forming a triangle complex (like a rice hat). The boundary of this triangle complex is a polygon which can be triangulated to a second triangle complex (like a rice hat capping) by Algorithm 4. Both triangle complexes together form a closed triangle complex if they do not overlap. Each triangle of the second triangle complex together with the corner point may form a tetrahedron. However, the failing issues of Algorithm 4 still remain (cf. last paragraph).

4. EXPERIMENTAL RESULTS

The algorithms presented in the previous section were tested on CityGML raw data. The data were produced by the *tridicon CityModeller* (Leica/Hexagon) and are manually revised with the *tridicon 3D Editor* (Leica/Hexagon). The produced *CityGML* data is stored in *3dcitydb*. The production steps were carried out by the Thuringian state office for land management and geo-information (TLBG). In order to not increase the point set, we focused on not adding additional points to the data-set to establish "beautiful" triangles or tetrahedra which are not too thin etc. We also did not want to operate on data which had been manipulated by some CityGML optimiser or validator. These may be optimised in application through the modelling steps or generation steps of the input set. CityGML is just a show case for a more general problem when dealing with *B-Rep*'s. The data was zeroed to the coordinates $lat = 642004.72$, $lon = 5649093.259$ $z = 229.074$.

The positional precision ϵ and angular precision ζ were chosen to be 10^{-6} for this presentation. Fig. 7 and 8 illustrate some difficult situations when dealing with planarity and double precision arithmetic. Planarity of triangle complexes is checked against the normal vector of an arbitrary triangle within the triangle complex by calculating the cosine between the reference normal vector and the other normal vectors. If the cosine is equal to one with a devation not larger than ζ, then both normal vectors are interpreted as collinear. The top pictures of Fig. 7 and 8 filter non-planar surfaces from the input set for Alg. 1. This is one of the main reasons why Alg. 1 sometimes fails to build volumes. The second pictures from the top show triangulated polygons (orange) which are not checked for planarity. Without discarding non-planar triangulated polygons, Alg. 1 would not recover the non-planar polygons while trying to calculate closed hulls.

On the other hand, the *Dijkstra* in step 6 finds other polygons which are not part of the input set (e.g. the floor of the attic in Fig. 8 top). But if the triangulated surfaces are not part of the input set, then they will be rejected. These two situations can be handled by splitting each triangle complex into a set of planar triangle complexes as a pre-processing step and ignoring that every surface of step 2 needs to be part of the input triangle net. The results are shown in Fig. 7 and Fig. 8, third picture from the top and the overall results from Erfurt city centre show more than 80% tetrahedralised buildings / building parts in Fig. 6 ($\epsilon = 10^{-2}$, $\zeta = 10^{-9}$). The results shown in Fig. 8 also ignored the cross checking of the triangulated surfaces against the input. The roof is one tetrahedron complex (red) and the main building is another disjoint tetrahedron complex (blue). This example shows a valid tetrahedralisation. If the building has concave structures like an open atrium they will be closed and the results are invalid.

Three different methods of tetrahedralising triangle nets were tried. The first method is to aggregate every triangle complex of each building part of every building into one triangle net, and to tetrahedralise this net. The second method aggregates every triangle complex of each building part into one triangle net per building and tetrahedralises those triangle nets separately. The third method tetrahedralises each building part separately. The resulting tetrahedron complexes are aggregated into one tetrahedron net for further analysis. Nevertheless, the first method leads to disjoint tetrahedron complexes. The second method leads to overlapping volumes between buildings, if they overlap. The third method leads to overlapping volumes within buildings and between buildings, if the different tetrahedron complexes overlap. Fig. 8 at the bottom illustrates that situation and shows building parts of the Erfurter Dom and the Severikirche which do overlap (coloured) and need to be subtracted from another to form distinct volumes. The analysis has been parallelised. Each Thread calculates one triangle net. Since the computation of one net itself was also parallelised (cf. end of paragraph on Alg. 1), the computation time was reduced significantly.

5. CONCLUSIONS

Calculating topology with double precision arithmetic remains a difficult task. The paper provided straight-forward methods to calculate higher dimensional geo-objects from topologically inconsistent lower-dimensional *B-Rep*'s. The algorithm-pipeline produces volumes, if the objects are simple enough, but illustrates the problems which may occur. The major control parameters are the positional precision ϵ and angular precision ζ. If ϵ is set large, then segments from the input set will be smoothed

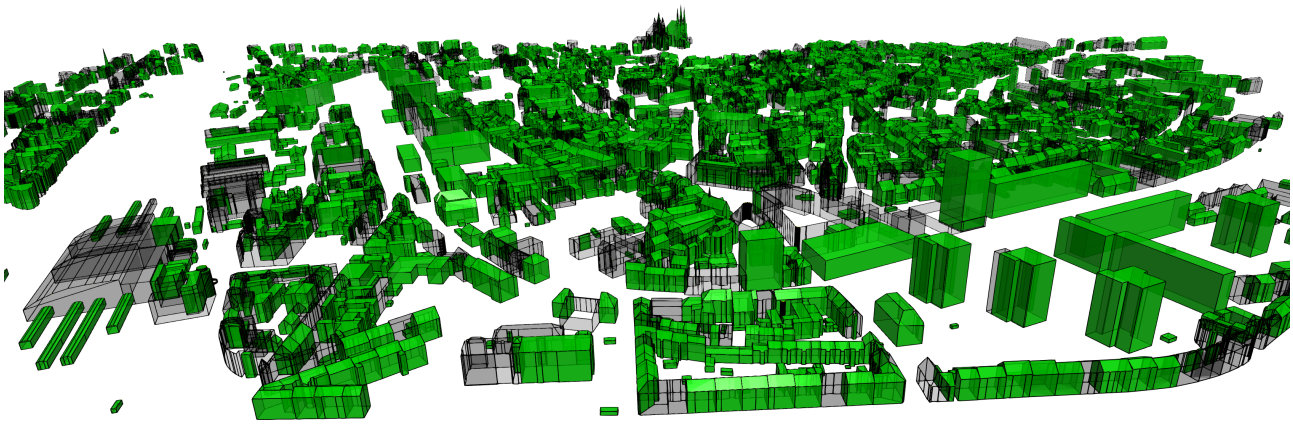


Figure 6. Erfurt city center, CityGML wireframe (black), triangle complexes (grey), tetrahedron complexes (green).

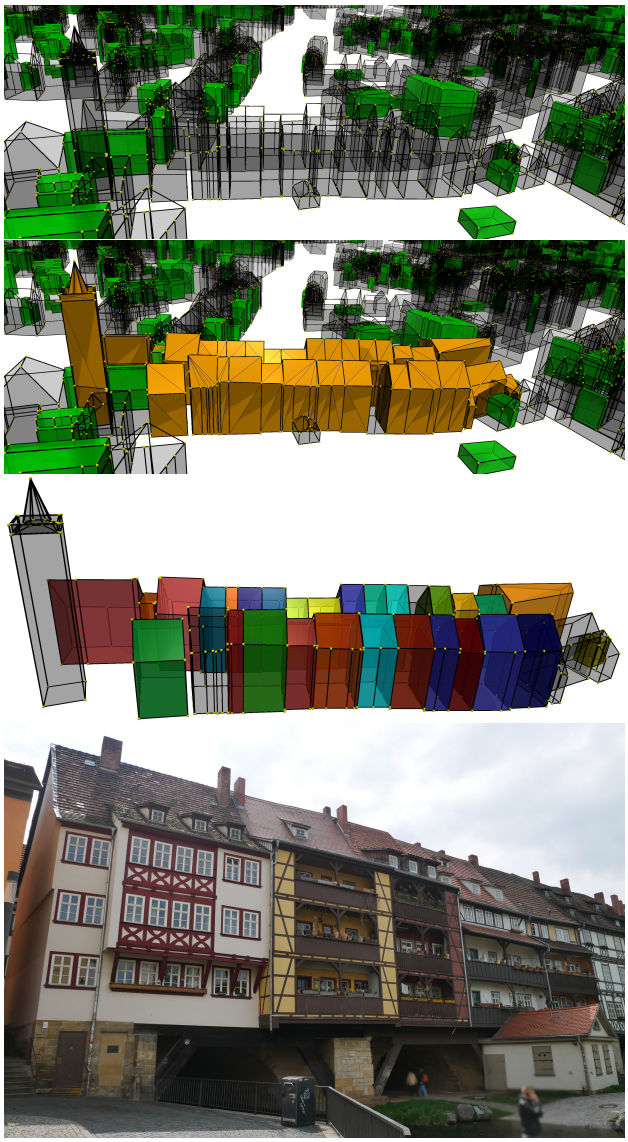


Figure 7. Krämerbrücke (Europe's longest bridge with fachwerk houses on both sides), CityGML wireframe (black), triangle complexes (grey) with ignored planar constraint while triangulation (orange), tetrahedron complexes (green) with splitting of non-planar triangle complexes as pre-processing step (coloured)

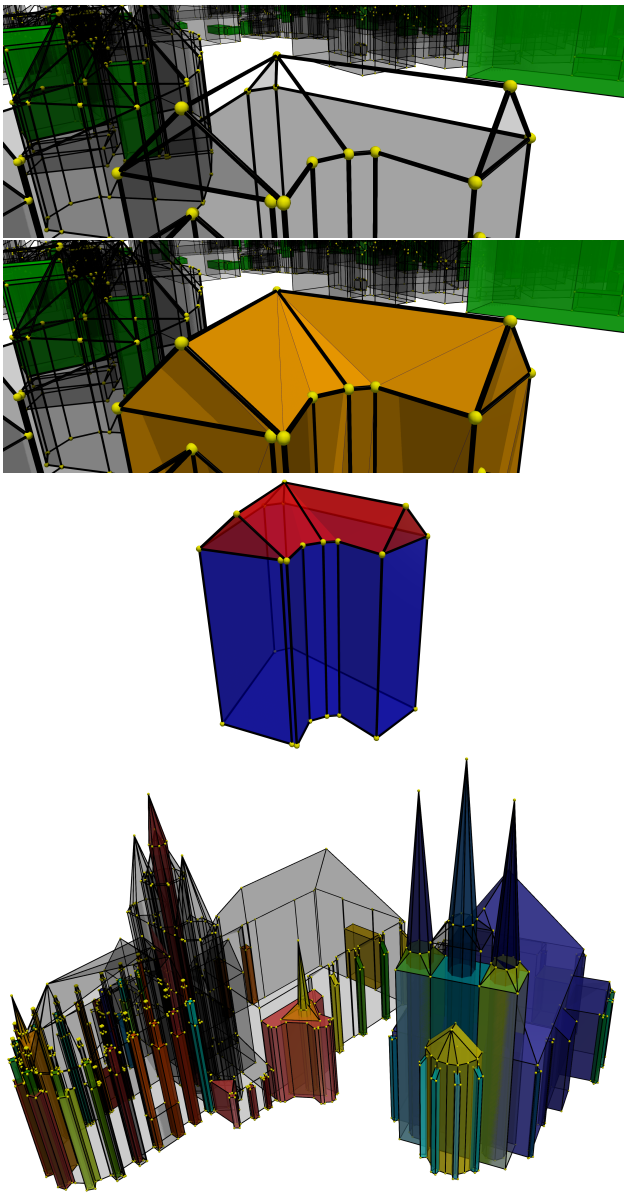


Figure 8. Juri-Gagarin-Ring 2 (top three), Erfurter Dom & Severikirche (bottom), CityGML wireframe (black), triangle complexes (grey) with ignored planar constraint while triangulation (orange), tetrahedron complexes (coloured)

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-VIII-4-W2-2021-21-2021 | © Author(s) 2021. CC BY 4.0 License.

out, but the planarity check of curve patches within the Dijkstra step 6 of Algorithm 1 will find more valid patches to stitch together to form closed polygons. If ϵ is small, then points will not be recognised on planes (e.g. planarity of wire-frame patches), intersections between walls and roofs may also not be recognised anymore, which leads to topological inconsistency. If ζ is large, then thin triangles get rejected easily and non-planar surfaces may be triangulated wrongly. If ζ is small, then triangulations get rejected, because they are found to be "non-planar", and closed hulls are not to be found because the wire-frame patches do not form planar polygons. These problems are not new and are highly sensitive to the quality of the input data. However, the presented conceptual model which combines the philosophies of graph databases with the OGC's General Feature Model helped to produce complex topological structures and added value to the data-set. The algorithm-pipeline may easily be extended and optimised. Parts of the algorithm-pipeline may be exchanged (e.g. the triangulation algorithms, planarity checks etc.) or reconfigured with different precision parameters to optimise processing efficiencies (better result or better run time through the scalability of graph databases). We are working on the optimisation of the algorithm-pipeline to produce scientifically reliable results which may be visualised nicely also, or used within simulations or other scientific analysis.

ACKNOWLEDGEMENTS

Martin Breunig and the anonymous referees are thanked for valuable remarks and suggestions for improving this paper. The Thuringian state office for land management and geo-information (TLBG) is thanked for their public geo-data repository. This work is partially funded by the Deutsche Forschungsgemeinschaft under grant numbers BR 2128/18-1 and BR 3513/12-1.

REFERENCES

Alexandrov, P., 1937. Diskrete Räume. *Matematicheskii Sbornik (N.S.)*, 2, 501–518.

Bagali, S., Waggenspack Jr., W., 1995. A shortest path approach to wireframe to solid model conversion. C. Hoffmann, J. Rossignac (eds), *Proceedings of the Third ACM Symposium on Solid Modeling and Applications*, ACM, Salt Lake City, UT, 339–350.

Biljecki, F., Ledoux, H., Stoter, J., 2014. Error propagation in the computation of volumes in 3D city models with the Monte Carlo method. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, II-2, 31–39.

Bradley, P., Paul, N., 2010. Using the Relational Model to Capture Topological Information of Spaces. *The Computer Journal*, 53(1), 69–89.

Breunig, M., Borrmann, A., Rank, E., Hinz, S., Kolbe, T., Schilcher, M., Mundani, R.-P., Jubierre, J. R., Flurl, M., Thomsen, A., Donaubauer, A., Ji, Y., Urban, S., Laun, S., Vilgertshofer, S., Willenborg, B., Menninghaus, M., Steuer, H., Wursthorn, S., Leitloff, J., Al-Doori, M., Mazroobsemnani, N., 2017. Collaborative Multi-Scale 3D City and Infrastructure Modeling and Simulation. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4/W4, 341–352.

Breunig, M., Bradley, P., Jahn, M., Kuper, P., Mazroob, N., Rösch, N., Al-Doori, M., Stefanakis, E., Jadidi, M., 2020. Geospatial Data Management Research: Progress and Future Directions. *ISPRS Int. J. Geo-Inf.*, 9(2), 95.

Breunig, M., Kuper, P., Butwilowski, E., Thomsen, A., Jahn, M., Dittrich, A., Al-Doori, M., Golovko, D., Menninghaus, M., 2016. The story of DB4GeO – A service-based geo-database architecture to support multi-dimensional data analysis and visualization. *ISPRS J. Photogramm. Remote Sens.*, 117, 187–205.

Egenhofer, M., Franzosa, R., 1991. Point-Set Topological Spatial Relations. *International Journal of Geographical Information Systems*, 5(2), 161–174.

Erl, T., Khattak, W., Buhler, P. (eds), 2015. *Big Data Fundamentals. Concepts, Drivers & Techniques*. The Prentice Hall Service Technology Series, Prentice Hall.

Giovanella, A., Bradley, P., Wursthorn, S., 2019. Evaluation of Topological Consistency in CityGML. *ISPRS Int. J Geo-Inf.*, 8(6), 278.

Inoue, K., Shimada, K., Chilaka, K., 2003. Solid Model Reconstruction of Wireframe CAD Models Based on Topological Embeddings of Planar Graphs. *J. Mech. Des.*, 125(3), 434–442.

Jahn, M., Bradley, P., Al-Doori, M., Breunig, M., 2017. Topologically consistent models for efficient big geo-spatio-temporal data distribution. *ISPRS Ann. Photogramm. Remote Sens.Spatial Inf. Sci.*, IV-4/W5, 65–72.

LaCourse, D. (ed.), 1995. *Handbook of Solid Modeling*. McGraw-Hill.

Ledoux, H., 2008. The kinetic 3D Voronoi diagram: A tool for simulating environmental processes. P. van Oosterom, S. Zlatanova, F. Penninga, E. Fendel (eds), *Advances in 3D Geoinformation Systems*, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, Heidelberg.

Ledoux, H., Meijers, M., 2011. Topologically consistent 3D city models obtained by extrusion. *International Journal of Geographical Information Science*, 25(4), 557–574.

Sindram, M., Machl, T., Steuer, H., Pültz, M., Kolbe, T., 2016. Voluminator 2.0 - Speeding up the approximation of the volume of defective 3D building models. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, III-2, 29–36.

Stoter, J., de Kluijver, H., Kurakula, V., 2008. 3D noise mapping in urban areas. *International Journal of Geographical Information Science*, 22(8), 907–924.

Vosniakos, G., 1997. Conversion of Wireframe to ACIS Solid Models for $2\frac{1}{2}$-D Engineering Components. *Int J Adv Manuf Technol*, 14, 199–209.

Zhang, K., Chen, S.-C., Singh, P., Saleem, K., Zhao, N., 2006. A 3D visualization system for hurricane storm-surge flooding. *IEEE Computer Graphics and Applications*, 26(1), 18–25.

Zhao, J., Stoter, J., Ledoux, H., 2014. A framework for the automatic geometric repair of CityGML models. B. D. Buchroithner M., Prechtel N. (ed.), *Cartography from Pole to Pole*, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, Heidelberg.