A star-based data structure to store efficiently 3D topography in a database

Hugo LEDOUX[*]
Delft University of Technology

Martijn MEIJERS
Delft University of Technology

December 24, 2013

For storing and modelling three-dimensional topographic objects (e.g. buildings, roads, dykes and the terrain), tetrahedralisations have been proposed as an alternative to boundary representations. While in theory they have several advantages, current implementations are either not space efficient or do not store topological relationships (which makes spatial analysis and updating slow, or require the use of an expensive 3D spatial index). We discuss in this paper an alternative data structure for storing tetrahedralisations in a DBMS. It is based on the idea of storing only the vertices and *stars* of edges; triangles and tetrahedra are represented implicitly. It has been used previously in main memory, but not in a DBMS. We describe how to modify it to obtain an efficient implementation in a DBMS, and we describe how it can be used for modelling 3D topography. As we demonstrate with different real-world examples, the structure is compacter than known alternatives, it permits us to store attributes for any primitives, and has the added benefit of being topological, which permits us to query it efficiently). The structure can be easily implemented in most DBMS (we describe our implementation in PostgreSQL) and we present some of the engineering choices we made for the implementation.

Keywords: 3D GIS, tetrahedralisation, DBMS, data structures.

1 Introduction

Several data models to represent three-dimensional (3D) topographic objects (e.g. buildings, roads, dykes or the ground) and to store them in a database manage-

[*]Corresponding author: `h.ledoux@tudelft.nl`

ment system (DBMS) have been proposed. Some of them store only the geometry of single objects while others permit us to explicitly store the topological relationships between objects (and also those between the lower-dimensionality primitives of the representation of an object). Geometry models usually store objects with a boundary representation, called *b-rep*, and one popular data structure used in practice is GML, the Geography Markup Language (OGC, 2007). It can be seen in the overview of Zlatanova et al. (2004) that many topological models are variations of the *formal data structure* (FDS) (Molenaar, 1990, 1998), which, unlike its name implies, is a conceptual model—with rules and constraints to preserve the validity—that can be implemented in a DBMS. The FDS is a boundary representation in which four primitives are kept (nodes, arc, edge and face; bodies are implicit), and where the topological relationships between them are stored.

An alternative to b-rep models is to use *tetrahedralisations*, where 3D objects are decomposed into tetrahedra and where empty space (e.g. between buildings) is also decomposed into tetrahedra (dubbed the 'air' tetrahedra) and integrated in the model[1]. Carlson (1987) and Pilouk (1996), among others, argue that tetrahedralisations have several advantages to represent 3D objects, in the same way that triangulations have advantages in 2D (Frank and Kuhn, 1986). The advantages the most often cited are: storage is simplified as only one convex primitive is needed (Penninga, 2005), spatial analysis operations can be performed more efficiently (Ledoux and Gold, 2008), and the overall implementation is simpler, thus more robust. However, tetrahedralisations also have theoretical drawbacks, Zlatanova et al. (2004) state in their comparison of the different 3D GIS models: "An additional disadvantage of TEN is its much larger database size compared with other representations." However, as Penninga and van Oosterom (2008) state, the storage penalty is true only if all the primitives of the tetrahedra are explicitly represented (nodes, edges and faces, as with the FDS). To design a compact data structure, they propose two variations of a data structure where only vertices and tetrahedra are represented: edges and triangles are extracted on-the-fly when needed. The first approach can be seen as akin to *Simple Features* (OGC, 2006): a tetrahedron is formed by the concatenation of the 4 coordinates of the 4 vertices and only one table is stored. The second approach is a variation: vertices are stored in one table and have an ID, and one tetrahedron is formed by the concatenation of 4 vertex IDs into a series of bits. For both approaches they ensure that all tetrahedra are correctly *oriented* (i.e. the ordering of the vertices is the same for all tetrahedra), which speeds up spatial analysis and the incremental update of the model.

While Penninga and van Oosterom's data structures are compact, the topological relationships between the tetrahedra are not explicitly stored (neither adjacency between tetrahedra nor incidence from one tetrahedron to primitives in other tetrahedra). These can be extracted in *views*, but still require a global scan of the the tables. That means that spatial analysis operations will perform slowly on big datasets, and so will incremental updates. Penninga and van Oosterom (2008) advocate using an auxiliary spatial index for each tetrahedron, such as an R-tree (Guttman, 1984), although that has not been implemented nor tested. The main problems with an auxiliary index are: (i) storage space is expensive, the bounding box of a tetrahedron (required by the R-tree) has only one point less than the tetrahedron itself; (ii) a R-tree is a rather complex structure and updating incrementally a large

[1]Notice that this model is often called "TEN", which stands for either TEtrahedral Network, TEtrahedral irregular Network, TEtrahedronised irregular Network or TEtrahedron Network, depending on the authors. For the purpose of this paper, they are all equivalent and a TEN is a tetrahedralisation, as defined in Section 2.

tree can be slow. Triangulations in 2D often are not indexed at the triangle level for the same reasons, see Finnegan and Smith (2010).

We discuss in this paper an alternative data structure for storing tetrahedralisations in a DBMS. Instead of storing explicitly tetrahedra, we store only two lower-dimensionality primitives (vertices and edges), and the *stars* of edges. It has been used previously in 3D in main memory (Blandford et al., 2005), but to our knowledge no attempt has been made to implement it in a DBMS. We define in Section 2 the concept of stars, which is related to that of a tetrahedralisation. We present in Section 3 the data structure and we detail how attributes and constraints can be stored. One important advantage of our star-based structure is that it permits us to avoid the use of an auxiliary 3D spatial index, the topological relationships between the tetrahedra are instead exploited; only a standard binary-tree (B-tree) is needed to index the tetrahedra. As explained in Section 4, the structure is very compact, can be implemented easily in a DBMS, and can be efficiently queried. We report in Section 5 on the storage space needed in PostgreSQL/PostGIS for different real-world 3D city models. It can be seen that the size of the DBMS tables required for our structure (including the indexing) is smaller than simply the R-tree needed to index the tetrahedra when stored with Penninga and van Oosterom (2008)'s structure. We finally discuss in Section 6 some of the engineering choices we made while implementing the structure, and our future challenges.

2 Constrained tetrahedralisation and stars

2.1 Constrained tetrahedralisation

Given a set S of points in $\mathbb{R}^d$ (the Euclidean space of dimension d), a triangulation decomposes the convex hull of S into non-overlapping *simplices*. A simplex is the simplest element in a given dimension. Specifically, a k-dimensional simplex is called a k-simplex and it is the convex hull of a set of $(k+1)$ linearly independent points in $\mathbb{R}^k$; a 0-simplex is a node, a 1-simplex an edge, a 2-simplex a triangle, a 3-simplex a tetrahedron, and so on. To simplify the notation, a k-simplex σ formed by the vertices a, b and c, is simply denoted abc. The *facets* of σ are the $(k-1)$-simplices forming it.

If the set S contains both points and *constraints*, then a constrained triangulation (CT) decomposes the convex hull of S into simplices that are non-overlapping, and every input constraint appears in the triangulation of CT(S). As shown in Figure 1a, the constraints in $\mathbb{R}^2$ are (straight-line) segments in the plane; if several segments form a loop (which defines a polygon), a CT permits us to triangulate the interior of this polygon. In $\mathbb{R}^3$, the constraints are planar surfaces, as Figure 1b shows. In our case these constraints are the walls of buildings for instance; as is the case in 2D, if several adjacent surfaces form a closed polyhedron, a CT will permit us to tetrahedralise its interior.

It is known that the CT of a set of points and segments in $\mathbb{R}^2$ is always possible (Shewchuk, 1997). However, in 3D the problem is more complex. This is because some arbitrary polyhedra cannot be tetrahedralised without the addition of extra vertices, the so-called *Steiner* points. Figure 2 shows an example, as it was first illustrated by Schönhardt (1928). This polyhedron is formed by twisting the top face of a triangular prism to form a 6-vertex polyhedron having eight triangular faces (each one of the three quadrilateral faces adjacent to the top face will fold into two triangles). It is impossible to select four vertices of the polyhedron

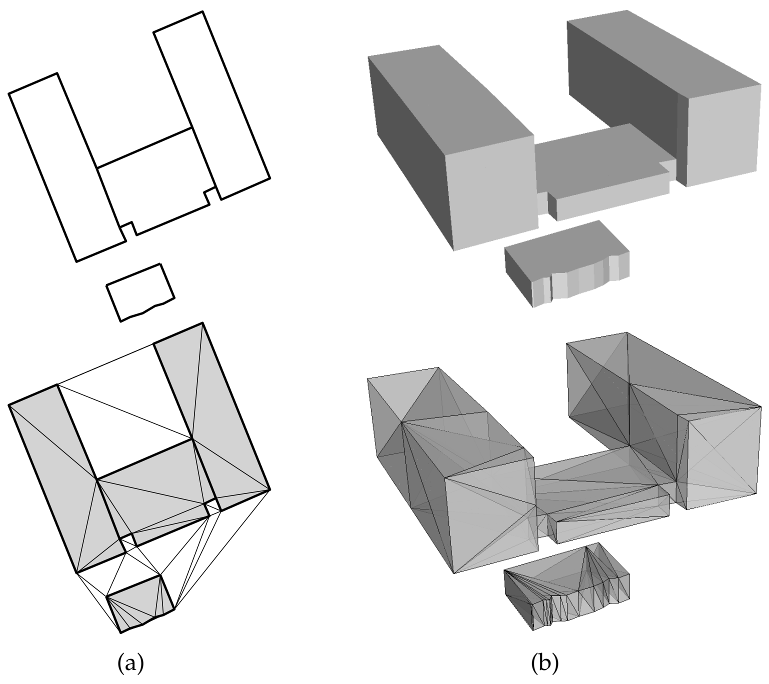

Figure 1: (a) Two-dimensional polygons representing buildings footprints, and their constrained Delaunay triangulation. (b) Three-dimensional representation of the same buildings (polyhedra in this case, obtained by extruding the footprints in (a)) and their constrained Delaunay tetrahedralisation (for clarity only the tetrahedra inside the polyhedra are shown here, but the whole convex hull in 3D is partitioned into tetrahedra).

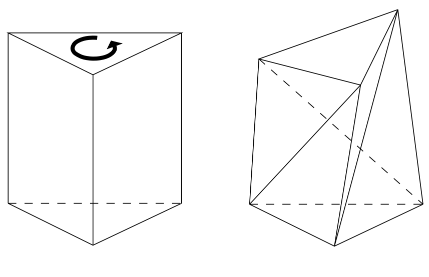

Figure 2: The Schönhardt polyhedron is impossible to tetrahedralise without adding extra vertices..

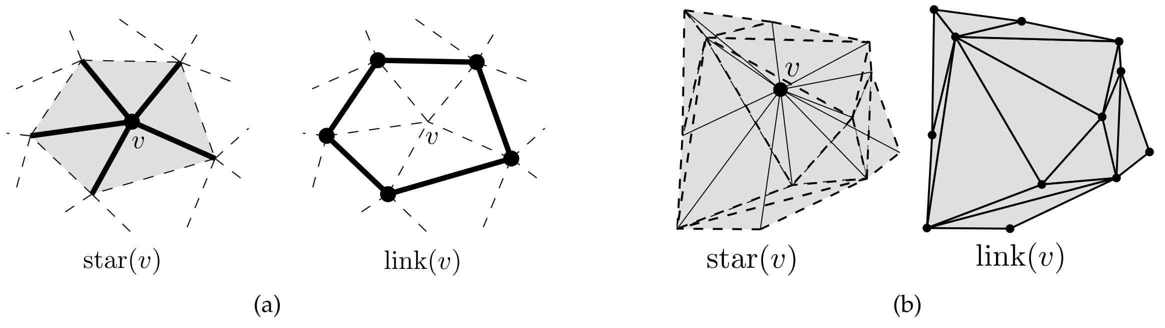

Figure 3: The star and the link of a vertex v in (a) 2D and (b) 3D.

such that a tetrahedron is totally contained inside the polyhedron, as none of the vertices of the bottom face can directly 'see' the three vertices of the top triangular face. Adding one Steiner in the centre of the prism allows the CT to be constructed.

To construct a CT in 3D, there are three approaches. The first one is the *conforming Delaunay tetrahedralisation*, where additional vertices are inserted and where each tetrahedron respects the Delaunay criteria, i.e. its circumsphere is empty of any other vertex. While in certain disciplines having well-shaped tetrahedra is important (e.g. in finite-element analysis), in the case of 3D topography it is most likely less as the partitioning of space is used as a data structure, and the shape of the element is not used directly. Moreover, conforming Delaunay tetrahedralisations often insert *several* new vertices, although there are techniques to reduce them in practice (Cohen-Steiner et al., 2004). The second solution is the *constrained* Delaunay tetrahedralisation (CDT) (Shewchuk, 2002; Si and Gärtner, 2011), where the tetrahedra are not fully Delaunay but where the number of Steiner points is minimised. A third approach consists of creating tetrahedra without any guarantee on their shapes, this requires the use of Steiner points but these can be minimised. Robust and efficient implementations of the last two approaches exists, we use for the experiments in Section 4 the software TetGen (Si, 2004).

2.2 Stars and links

Let v be a vertex in a d-dimensional triangulation. Referring to Figure 3, the star of v, denoted star(v), consists of all the simplices that contain v; it forms a star-shaped polytope P. For example, in 2D, all the triangles and edges incident to v form star(v). Notice however that the edges and vertices disjoint from v—but still part of the triangles incident to v—are not contained in star(v). From a point-set topology point-of-view, star(v) is the interior of P. Also, observe that the vertex v itself is part of star(v), and that a simplex can be part of a star(v), but not some of its facets.

The boundary of P is defined as the *link* of v. It is formed by the set of simplices incident to the d-simplices forming star(v), but 'left out' by star(v). For example, if v is a vertex in a 2D triangulation, link(v) is formed by the vertices and edges that are incident to the triangles incident to v. In 2D it is a polyline, and in a tetrahedralisation, link(v) is a two-dimensional triangulation formed by the vertices, edges and triangular faces that are contained by the tetrahedra of star(v), but are disjoint from v. In a d-dimensional triangulation, the link of a given vertex is a $(d-1)$ triangulation.

The closely related concepts of star and link also apply to *edges* in 3D: in Figure 4

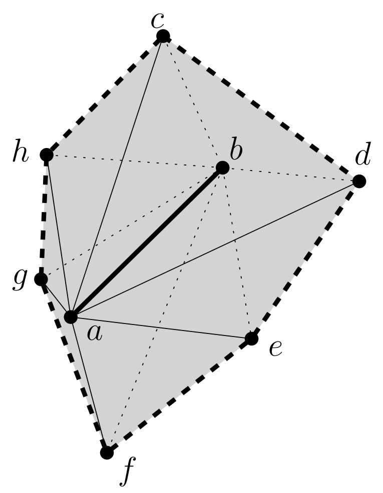

Figure 4: The link of the edge *ab* in 3D is formed by the bold dashed polyline *cdefghc*.

star(*ab*) is formed by all the incident simplices (6 in this case), and link(*ab*) is a 1-dimensional triangulation.

3 A star-based data structure

To our knowledge, Cline and Renka (1984) are the first to design a data structure where stars of vertices are used to represent a triangulation, albeit in 2D. Their structure is compact, but does not allow incremental updates (which is arguably important for a GIS data model). Shewchuk (2005) uses a star-based data structure to manipulate 3D triangulations, but his structure is very verbose since the star of every vertex is stored as a 2D triangulation, itself stored with stars. Blandford et al. (2005) fix both issues with their structure, which is valid for 2D and 3D triangulations. Their representation indeed uses about a factor 3 less memory than traditional representations in 3D and at the same time can be queried and dynamically modified. To achieve this compression, they designed a pointer-less structure where each vertex is assigned a label (an integer) and they compress based on labels: if possible they store the integers using 4 bits (they use differences between the labels to achieve that) and use different optimisations to keep the memory footprint low.

While it would theoretically be possible to implement the compression in a DBMS, we are interested in their basic idea of using labels for vertices and storing the stars of edges in 3D. In a nutshell, a star-based data structure for a tetrahedralisation in a DBMS is as follows. First, each vertex is given a label. For an edge *ab* of the tetrahedralisation, we store its link as an ordered list of labels. The orientation of all the edges must be the same; we use in this paper the *right-hand rule*: if the thumb points from *a* to *b*, the vertices of link(*ab*) are ordered in the direction of the curled fingers of the right hand. In Figure 4, link(*ab*) is the ordered list $[c, d, e, f, g, h, c]$; notice that this is a circular list and that the starting vertex could be any vertex. The link list is of variable length: its minimum is 2 (one tetrahedron) and its theoretical maximum is the number of vertices in the tetrahedralisation minus 2. A tetrahedron is formed by *ab* and 2 consecutive vertices in the list; *abcd* and *abhc* are two examples of tetrahedra implicitly represented in link(*ab*). Notice that the length of the list gives the number of incident tetrahedra to *ab*. Also, lower-dimensionality simplices (triangles and edges) are implicitly represented in links: for instance, referring to Figure 4, the triangle *abc* is present in the links of its 3 edges. Since the link is an ordered list, the simplices implicitly represented are also ordered.

The key idea behind the structure is that if we represent the link of each edge then we obtain a data structure where relationships such as incidence and adjacency between tetrahedra and their lower-dimensionality simplices are present. It is the overlap between the (ordered) links that permits us to represent explicitly that information. Observe that each tetrahedron is represented in the link of its 6 edges, and that since these are ordered, we can easily navigate from tetrahedron to tetrahedron. We demonstrate in Section 4.3 queries that can be efficiently answered with such a data structure.

3.1 Representative edges

Storing the link for each edge of a tetrahedralisation yields a powerful and topological data structure, but also one that is not space efficient. Indeed, if the CT of a set S of n points contains t tetrahedra, then the number e of edges is significantly higher: Blandford et al. (2005) estimate it at $(7/6)t$ for a CT where the points are uniformly distributed in space. The real-world datasets of buildings used for the experiments in Section 5 corroborate this. Since, as explained in the Introduction, one of the disadvantages of using tetrahedralisations to model topography is the larger storage requirements, this is clearly not optimal.

To reduce the number of edges whose star is stored, we store only the *representative edges* (RE), as Blandford et al. (2005) suggest. If the label given to each vertex is an integer, a RE is one where its 2 vertex labels are either odd or even. If we randomly label the vertices, that should reduce by a factor of about 2 the number of edges to be stored and still permits us to represent at least once each triangle and each tetrahedron (which is fundamental to ensure that all topological relationships are present). Indeed, it ensures that each triangle has at least one RE: a triangle has either 3 REs (3 odd or 3 even labels) or one RE (1 odd / 2 even; 2 odd / 1 even).

Figure 5 shows the same 6 tetrahedra as in Figure 4, and the REs are in bold. It can be seen that out of the 19 edges, 6 are representative, and that each triangle, and each tetrahedron, contains at least one RE.

3.2 Storage space

Evaluating the theoretical storage space for a given dataset is difficult since the number of tetrahedra in a CT depends on the locations of the points and the constraints (Shewchuk, 1997). However, to obtain an order of magnitude, we can state that we need on average 3 labels per tetrahedron. Indeed, each tetrahedron has 4 triangles, which are shared by 2 tetrahedra (if we ignore those on the convex hull); thus 2 triangles per tetrahedron. A triangle has 3 labels, but since it appears in 3 stars and that only half of the edges are represented, we obtain $1\frac{1}{2}$. Thus: $2 \times 1\frac{1}{2} = 3$ labels per tetrahedron. Our experiments with real-world datasets corroborate that, see Section 5.

3.3 Attributes and constraints

Attaching attributes to the tetrahedra is possible, although one must be careful since tetrahedra are present in multiple stars and only implicitly. As Blandford et al. (2005) suggest, we exploit the fact that each vertex has a unique label (which

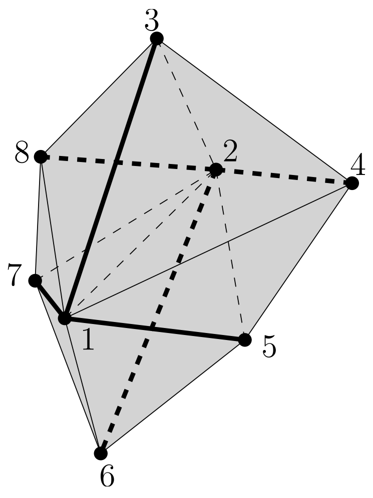

Figure 5: A set of 8 vertices yields a tetrahedralisation with 6 tetrahedra. Out of the total 19 edges the 6 representative edges (REs) are stored (and shown in bold): $\langle 1,3 \rangle$, $\langle 1,5 \rangle$, $\langle 1,7 \rangle$, $\langle 2,4 \rangle$, $\langle 2,6 \rangle$, $\langle 2,8 \rangle$.

can be ordered) and attach the attributes to the star of the REs whose origin is the lowest; in case there are more than one, the one having the lowest destination is chosen. For example, an attribute for the tetrahedron *abcd* is stored in the star of the RE *ab*. Given a tetrahedron *abcd*, we can find out in constant time which RE stores its attributes. Each attribute is stored in its own list having the same length as the list of the star. Table 1 shows one example for a specific dataset, and a concrete example is given in Section 4.3.

In the context of modelling 3D city models, an example of an attribute would be the ID (or name) of the building inside which a tetrahedron lies. If we store this ID for each tetrahedron (and assign also an ID for the 'air' tetrahedra), then we can also explicitly represent the original constraints in the dataset: they are formed by the triangles whose incident tetrahedra have different IDs.

3.4 Spatial indexing: using the tetrahedralisation itself

An advantage of a star-based structure—or of any structure in which adjacency and incidence relationships are stored—is that a spatial index, such as an R-tree (Guttman, 1984), is not necessary to access efficiently the tetrahedra. Instead, the tetrahedralisation itself can be used to determine which tetrahedron contains a query point q: the adjacency relationships between the tetrahedra are used to navigate in the tetrahedralisation. The latter can be implemented with the *walking* algorithm as described in Mücke et al. (1999). It is a sub-optimal algorithm that is favoured by practitioners since it does not require an auxiliary data structure and yields fast practical performances (Mücke et al., 1999; Devillers et al., 2002).

The idea is as follows: starting from a given tetrahedron σ, we move to one of the neighbours of σ (we choose one neighbour such that the query point q and σ are on each side of the triangular face shared by σ and its neighbour) until there is no such neighbour, then the tetrahedron containing q is σ. In Figure 6, only the grey triangles are visited during the walk.

To minimise the number of tetrahedra visited, the starting tetrahedron should be close to q. Mücke et al. (1999) investigated a 'bucketing' approach where a certain number of triangles are randomly selected, and each walk starts from the closest

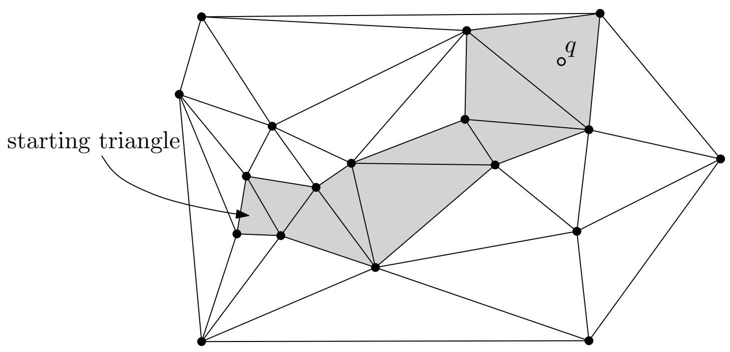

Figure 6: Walking in a 2D triangulation, starting from a given triangle to the query point q. In 3D the principle is the same: the walk is performed from tetrahedron to tetrahedron.

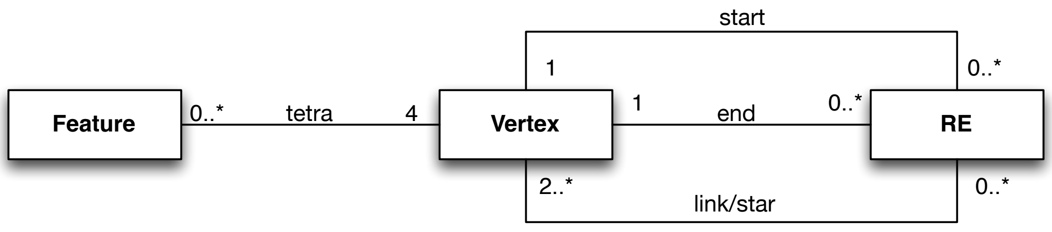

Figure 7: The UML diagram of the data model for our star-based data structure.

one (selected by a simple Euclidean distance test); it is called the *jump-and-walk* method. The result of a query can be either a tetrahedron, or one representative edge in that tetrahedron. Modifying this algorithm to start from a representative edge is trivial.

4 Implementation in a DBMS

This section describes a prototype implementation of the star-based data structure in a specific object-relational DBMS (PostgreSQL), but since the data structure is based solely on lists of labels, implementing it in another DBMS is straightforward. Several engineering decisions were taken while implementing the structure in PostgreSQL, and we report here on the main ones.

4.1 Data model

Figure 7 shows the UML class diagram for the data structure.

The tetrahedralisation itself is represented with two classes: one for the vertices, and one for the REs. The third class is used to represent features in a dataset, e.g. solids such as buildings.

Each vertex has a location with 3D coordinates. A RE has a start and an end vertex, and a link (which is an array of vertices). The relationship 'tetra' between the classes 'Feature' and 'RE' represents *one* tetrahedron (four vertices) located inside a feature: it permits us to efficiently locate one RE for a tetrahedron inside the feature without performing a global search.

(a) Vertex table

ID	x	y	z
1	0.0	0.0	0.0
2	1.0	0.0	0.0
3	1.0	1.0	0.0
4	0.0	1.0	0.0
5	0.0	0.0	1.0
6	1.0	0.0	1.0
7	1.0	1.0	1.0
8	0.0	1.0	1.0

(b) RE table

start	end	link[]	attribute[]
1	3	[-1, 4, 8, 7, 2]	[-1, 1, 1, 2, -1]
1	5	[-1, 6, 7, 8]	[-1, 2, 1, -1]
1	7	[6, 2, 3, 8, 5]	[2, -1, -1, -1, -1]
2	6	[-1, 7, 1]	[]
3	7	[-1, 8, 1, 2]	[]
4	8	[-1, 1, 3]	[]
5	7	[-1, 6, 1, 8]	[]

(c) Feature table

ID	owner	tetra[]
1	'John Smith'	[1, 3, 4, 8]
2	'Bob Brown'	[1, 5, 6, 7]

Table 1: The 3 tables for the dataset shown in Figure 8.

4.2 PostgreSQL tables

These three classes are straightforward to implement in a DBMS supporting an array type of variable length such as PostgreSQL. An example of the tables filled with data, for the two features shown in Figure 8, is shown in Table 1. In our implementation, we define an ID for each Vertex and it is the primary key (which creates a B-tree index on the column). This ensures efficient access and enforces uniqueness. The primary key of the table RE is defined as the concatenation of the columns 'start' and 'end'. It should be noticed that these two columns could be merged into one column, but that would require using a data type twice as large (64-bit integers for instance) and would not permit us to compress the storage.

Table 1 also shows how attributes are stored in the 'RE' table.

Also, to be able to represent triangles and tetrahedra two custom types have been defined. Both types are a sequence of vertex IDs, where triangles are represented by 3 vertex IDs and tetrahedra by 4. Based on these custom types a view can be defined that 'glues' the geometry of the vertices and edges together to triangles and tetrahedra (performed by a database join).

4.3 Examples of topological queries

We describe in the following how a few typical queries are answered with a star-based structure. All the queries refer to the example in Figure 8, and the resulting tables in PostgreSQL (Table 1).

Observe that out of the 19 edges in the tetrahedralisation, 7 are REs. In Table 1, in the link column of the RE table the ID '-1' is used for links that do not form a cycle, i.e. the edge is on the boundary of the convex hull of the dataset.

Observe also that the array of the column attribute is filled only for some tetrahedra: as explained in Section 3.3, for a given tetrahedron, we attach an attribute to it in the RE whose origin is the lowest. If a given RE is one such RE for a tetrahedron, then an array of the same length as that of the link column is used and '-1' is used

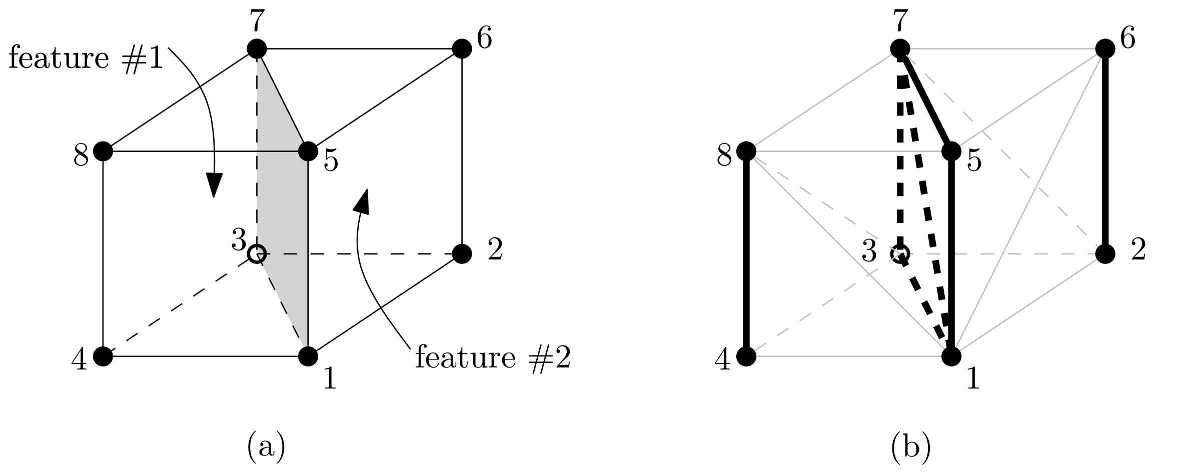

Figure 8: **(a)** Two adjacent 3D features (both triangular prisms) sharing a surface (in grey). **(b)** The two features tetrahedralised, with the 7 REs in bold black.

when the tetrahedron's attribute is stored in another RE. If a RE is not used to store the attributes of any tetrahedra (i.e. the attributes of the incident tetrahedra to the RE are stored in other REs), then the array can be empty to save storage space. The last four rows in the Table 1(b) have an empty array.

Since triangles and tetrahedra can be present multiple times in the structure, querying the structure has to be done with care. For example, tetrahedron $[1,5,6,7]$ is present in the edge table in the links of all its representative edges: $[1,5]$, $[1,7]$ and $[5,7]$.

Is the tetrahedron $[1,2,6,7]$ **present?** First, one RE has to be found: $[1,7]$ is one of them. Observe that a RE is found in constant time only by finding locally 2 odd or even IDs in the tetrahedron. Second, the IDs 2 and 6 have to appear consecutively (not necessarily in that order) in the link of that RE. It is the case, thus the tetrahedron is present.

What attribute has tetrahedron $[1,2,6,7]$**?** First, make sure the tetrahedron exists, as above. In the attribute[] column of its RE ($[1,7]$) find the first instance of either 6 or 2: first position in the link. The attribute (value is 2) is at the same position in the attribute list.

Which tetrahedra are adjacent to $[6,7,1,5]$**?** First, find one RE, as above ($[1,5]$) and find the position of vertices 6 and 7 in the link. The vertices before and after the tuple give 2 adjacent tetrahedra: $[1,5,0,6]$ and $[1,5,7,8]$; the former does not exist since 0 is in the link. Second, find another RE ($[1,7]$) and repeat the same operations in its link: tetrahedra $[1,7,8,5]$ and $[1,7,6,2]$ are found, the former having been previously identified. Since we know that each triangle is represented in at least one RE, this operation will always return the 4 tetrahedra, unless the tetrahedron is on the boundary of the convex hull. Notice that this is also a mechanism to identify which triangles are on the boundary of the convex hull (triangles $[5,6,7]$ and $[6,5,1]$ in this case).

Total number of tetrahedra? Here we apply the same criteria as for storing attributes in Section 3.3: the lowest concatenation of the IDs is the one representing the tetrahedron. For instance, tetrahedron $[7,8,1,5]$ is conceptually stored in edge $[1,5]$ and not $[1,7]$. Thus, it suffices to scan the RE table and take a local decision to extract tetrahedra: 6 tetrahedra are present in Figure 8.

Volume of a given feature? From the Feature table we obtain a tetrahedron inside the feature; let us find the volume of the feature 1 in Table 1. First find a RE for the tetrahedron $[1,3,4,8]$ ($[1,3]$) and identify the adjacent tetrahedra. If these tetrahedra also have the same attribute (same as above), then add them to the list of tetrahedra. Repeat this step for all the found tetrahedra, and sum the volume of the tetrahedra.

Are features #1 and #2 adjacent? As in the case of finding the volume of a feature, we need to visit all the tetrahedra inside the feature 1 and test if one of their adjacent tetrahedra has the attribute 2. For the feature 1 in Table 1, we would first find the RE $[1,3]$ and the tetrahedron $[1,3,8,7]$. From this tetrahedron, we repeat the same operation and find $[1,3,7,2]$, which has the attribute 2 in the list. Features 1 and 2 are thus adjacent.

5 Experiments with real-world data

To test the star-based data structure in PostgreSQL/PostGIS, we have made experiments with three real-world datasets containing 3D buildings. These were all obtained by *extruding* the 2D footprints to their height (which was obtained by averaging the height of the LiDAR samples inside a footprint). To be able to tetrahedralise the resulting 3D datasets, the surfaces of the buildings should be unique, i.e. if two buildings are adjacent, their shared wall should be present only once. We have created such a topologically consistent dataset with the methodology described in Ledoux and Meijers (2011). In a nutshell, it consists of ensuring that the 2D footprints are topologically consistent (i.e. no two footprints overlap and the graph of the footprint is a planar graph), and extrude each edge to a vertical wall.

We have constructed the constrained tetrahedralisation of the models with TetGen[2] (Si, 2008). For populating the DBMS, we have created a program that takes as input the result of the tetrahedralisation (a list of vertices and tetrahedra) and outputs a list of REs and their links.

5.1 Datasets used

The three datasets, shown in Figure 9, are areas in the Netherlands: 'campus' is the campus of the Delft University of Technology; 'kvz' is an area of Rotterdam; 'engelen' an area of the municipality of Den Bosch. Table 2 gives the details of the 3D extruded datasets, and the results of the construction of the tetrahedralisation. It should be noticed here that we tried to minimise the number of Steiner points when tetrahedralising the datasets; as it can be seen this was very satisfactory as only one point needed to be inserted for the dataset 'campus', and none for the other two. The cuboid shape of the buildings explains this.

5.2 Comparisons with alternatives

For each dataset, we have created 3 databases in PostgreSQL 9.2.2/PostGIS 2.0.2:

b-rep: each building is stored as a 3D primitive, the type POLYHEDRALSURFACE was used, which is a boundary representation. The original surfaces of a building were used, no triangulation was performed. A 3D R-tree index was used to index each building.

[2] `http://www.tetgen.org`

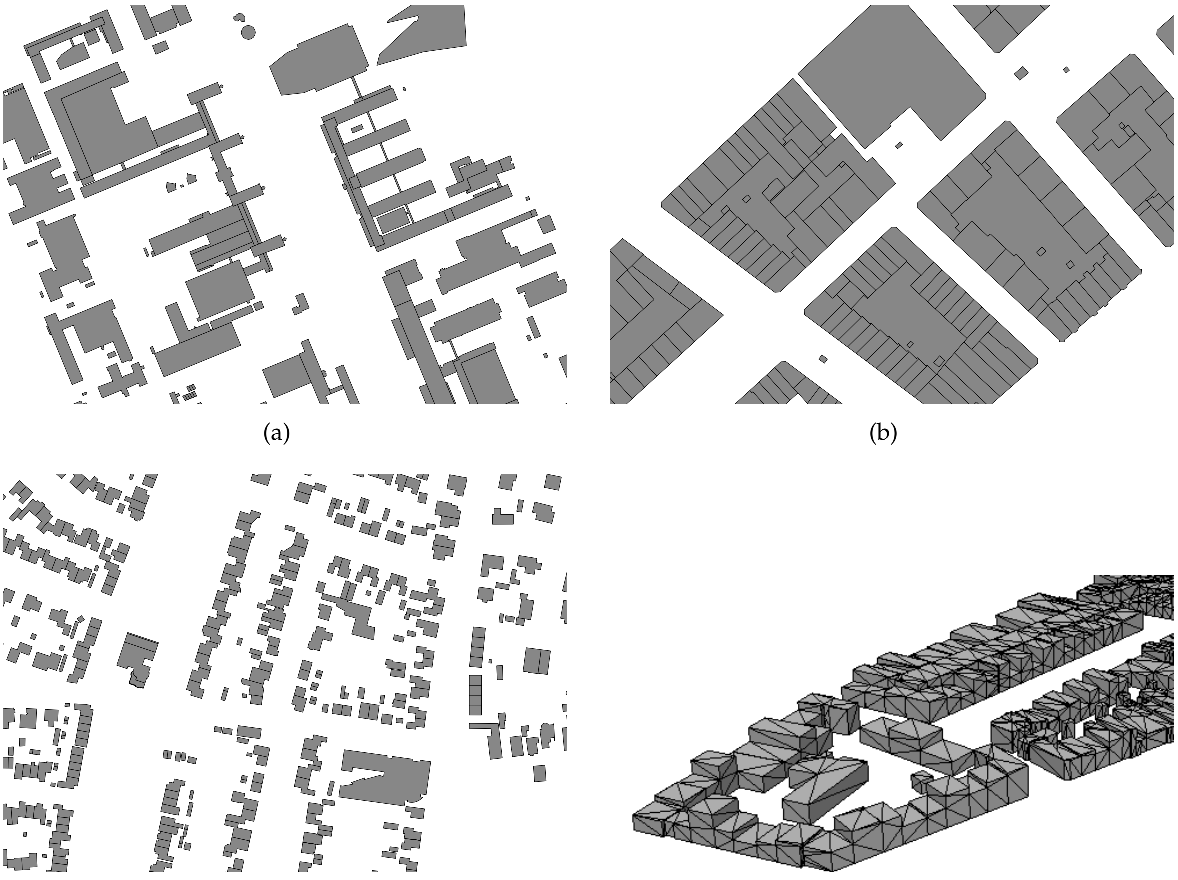
(a) (b) (c) (d)

Figure 9: Part of the footprints of the datasets (a) campus, (b) kvz, and (c) engelen. In (d) a view on the kvz dataset tetrahedralised; notice that the 'air' tetrahedra are not shown.

Table 2: Details concerning the datasets used for the experiments.

	3D model		input		CDT		
	solids	faces	vertices	constraints	vertices	edges	tets
campus	370	4 298	5 970	3 976	5 971	36 790	28 024
kvz	637	6 549	8 951	13 571	8 951	59 332	50 376
engelen	1 629	15 870	23 732	15 868	23 732	145 296	110 232

Table 3: Details of the stars.

		link				
	REs	avg	max	min	sum	ID/tet
campus	16 004	4.79	27	3	76 693	2.73
kvz	28 763	5.04	35	3	145 158	2.88
engelen	71 841	4.98	23	3	357 984	3.25

Table 4: Size of the PostgreSQL/PostGIS tables.

	b-rep		tetrahedra		star-based			
	table	index	table	index	vertex	index	RE	index
campus	424kB	320kB	15MB	3 440kB	408kB	152kB	1 408kB	368kB
kvz	768kB	224kB	26MB	5 072kB	600kB	216kB	2 560kB	648kB
engelen	1 936kB	304kB	57MB	12 000kB	1 584kB	536 kB	6 360kB	1 592kB

tetrahedra: each tetrahedron was stored as a POLYHEDRALSURFACE, which is conceptually similar to what Penninga and van Oosterom (2008) propose. They originally propose to concatenate the 4 xyz coordinates of the 4 vertices. This is not possible without creating a totally new data type. We have instead opted to use a POLYHEDRALSURFACE for each tetrahedron—more storage is used since this is a generic type and the ordering of the vertices cannot be inferred. We estimate that the storage penalty is around a factor 3 since each vertex is stored in 3 surfaces. As is the case for the buildings, a 3D index was used to index each tetrahedron. Notice that we could not test the other approach of Penninga and van Oosterom (2008) (where a tetrahedron is formed by the concatenation of 4 IDs, and another table with vertices is used for the coordinates) since it is not possible to spatially index the table of tetrahedra. A join between the 2 tables should first be performed, and then this table can be indexed; that would increase significantly the size and duplicate the information.

our star-based structure: as was described in Section 4.

It should be noticed that for the experiments we do not consider the features, we focus only on the tetrahedra.

The first observation is that for a given dataset with n solids, the CT of the dataset contains several tetrahedra: in our experiments, this number is around 75 times more. If every tetrahedron is stored individually, a direct consequence is that the table will have several more rows, and more importantly, the R-tree will be significantly bigger and more complex to update.

Table 3 presents the details of our star-based data structure for the three datasets. Notice that around half of the edges of the CT are stored as RE, as theory states. In our experiments, the number of REs is around 60% of that of the number of tetrahedra. This is inline with what theory states for a tetrahedralisation in which the points are uniformly distributed in space: $(7/6)t$ (where t is the number of tetrahedra), and only half of the edges are REs. This is therefore expected for most datasets. It should be noticed here that this translates into less rows in a table to store the tetrahedra, but also into a leaner index to build and maintain.

Table 3 also shows the details of the links of the REs. The average number of IDs stored in a link is about 5, its maximum is 35 for one dataset, and its minimum is 3 (an edge on the boundary of the convex hull). The column 'sum' is the sum of all the IDs in all the links. As described in Section 3.2, the number of IDs used for one tetrahedron is in theory 3—we obtain similar results with real-world datasets.

This number is however higher in practice in a DBMS since we must also assign IDs to the REs ('start' and 'end' attributes), but is nonetheless of the same magnitude as Penninga and van Oosterom (2008). As an example, for the kvz dataset we need to have (28 763 $\times$ 2) IDs for the REs, plus 145 158 IDs in the links; the total is thus 202 684 IDs. If the second approach of Penninga and van Oosterom (2008) was used to store the tetrahedra (where IDs of vertices are used) then each tetrahedron would need exactly 4 IDs: 50 376 $\times$ 4 $=$ 201 504 IDs.

Table 4 shows the size of the tables and the indices in PostgreSQL/PostGIS for the three datasets; for the star-based structure both tables are shown, and the index used is a B-tree for both. The total size of the tables for the star-based structure is around 4 times that of the b-rep structure, but around 8 times less than with the tetrahedra structure. The former was expected since far less objects/solids are stored with the b-rep, and the connections between non-adjacent buildings (the 'air' tetrahedra) are not stored. The latter is explained by the fact that Penninga and van Oosterom (2008) cannot be stored with generic types; even if the storage was reduced by a factor 3 the size of the tetrahedra structure would still be larger. Furthermore, the size of the indices for the star-based structure is around 6 times less than that of the R-tree for the tetrahedra structure. In fact, both tables and both indices for the star-based structure take around 33% less storage space than the R-tree index alone for the tetrahedra structure.

The size of the tables of the b-rep option is obviously smaller than that of the star-based structure, but it should be noticed that we are comparing apples to oranges here, especially since, as the number of solids grow in a dataset, the number of tetrahedra will grow even more (depending on the complexity and distribution of the points). A star-based structure permits us to represent more than simply the features: their adjacency and even their connection is encoded in the tetrahedralisation, which is useful for several applications. Topological queries are much faster answered as no comparison of coordinates need to be performed, and it is for instance possible to query for features that are neighbours, but do not touch.

6 Discussion and future work

We have shown that a star-based data structure implemented in a DBMS can be *both* compact and topological at the same time, two criteria that are usually contradictory. Our structure is in theory as compact as the most compact structure implemented in a DBMS so far (that of Penninga and van Oosterom (2008)), but if we consider that spatial indexing is needed then our structure is several times compacter. While storage space in a DBMS is not the most important factor, having smaller tables (in terms of rows) with leaner indices means faster updating and maintenance when a dataset is modified. As three-dimensional datasets become more available and grow in size, this is becoming more important.

Another strong point of the star-based structure is that it can be easily implemented in any DBMS supporting variable length arrays with two simple tables, and that no complex spatial index is needed. While the structure seems more cumbersome to maintain when the data are updated, the users need not be aware that this is the structure used. We plan to add functionalities to the DBMS so that views can be created over the edges and stars so that only features (e.g. buildings) are shown to the user, which is what van Oosterom et al. (2002) advocate for storing 3D GIS datasets in a DBMS. We have plans to make the data structure fully dynamic, i.e. supporting insertions and deletes of tetrahedra and of features, updating the already stored tetrahedra in the database.

Object relational DBMS systems keep evolving and new powerful features have been added to mainstream systems. One example is Common Table Expressions (SQL-99) which paves the way to deal with more complex data structures like trees and graph storage inside such systems natively. We intend to see how far these features are useful during implementation of the query part of our data structure and how much custom procedural functionality still needs to be built.

Apart from 3D topography, 3D models can also be useful for modelling space and map scale in one integrated 3D data structure, e.g. van Oosterom and Meijers (2011). Hence, one operation we want to investigate in more detail is to create a cross section through the stored 3D model: selecting intersecting tetrahedra, performing intersection and then create a topologically clean output of the intersected elements to see whether this tetrahedra based model can be a useful underlying technology for producing *vario-scale* data.

Acknowledgements

This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs (project codes: 11300 and 11185).

References

Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24, 2005.

E. Carlson. Three-dimensional conceptual modeling of subsurfaces structures. In *Proceedings 8th International Symposium on Computer-Assisted Cartography (Auto-Carto 8)*, pages 336–345, Falls Church, VA, USA, 1987.

A. K. Cline and R. J. Renka. A storage-efficient method for construction of a Thiessen triangulation. *The Rocky Mountain Journal of Mathematics*, 14:119–139, 1984.

David Cohen-Steiner, Eric Colin de Verdière, and Mariette Yvinec. Conforming Delaunay triangulations in 3D. *Computational Geometry—Theory and Applications*, 28:217–233, 2004.

Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13(2):181–199, 2002.

David C. Finnegan and Michael Smith. Managing LiDAR topography using Oracle and open source geospatial software. In *Proceedings GeoWeb 2010*, Vancouver, Canada, 2010.

A. Frank and W. Kuhn. Cell graphs: A provable correct method for the storage of geometry. In *Proceedings 2nd International Symposium on Spatial Data Handling*, Seattle, USA, 1986.

Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM Press, 1984.

Hugo Ledoux and Christopher M. Gold. Modelling three-dimensional geoscientific fields with the Voronoi diagram and its dual. *International Journal of Geographical Information Science*, 22(5):547–574, 2008.

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

Martien Molenaar. A Formal Data Structure for three dimensional vector maps. In *Proceedings 4th International Symposium on Spatial Data Handling*, pages 830–843, Zurich, Switzerland, 1990.

Martien Molenaar. *An introduction to the theory of spatial object modelling for GIS*. Taylor & Francis, 1998.

Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Computational Geometry—Theory and Applications*, 12:63–83, 1999.

OGC. OpenGIS implementation specification for geographic information—simple feature access. Open Geospatial Consortium inc., 2006. Document 06-103r3.

OGC. Geography markup language (GML) encoding standard. Open Geospatial Consortium inc., 2007. Document 07-036, version 3.2.1.

Friso Penninga. 3D topographic data modelling: Why rigidity is preferable to pragmatism. In A. G. Cohn and David M. Mark, editors, *COSIT—Proceedings International Conference on Spatial Information Theory*, volume 3693 of *Lecture Notes in Computer Science*, pages 409–425. Springer, 2005.

Friso Penninga and Peter van Oosterom. A simplicial complex-based DBMS approach to 3D topographic data modelling. *International Journal of Geographical Information Science*, 22(7):751–779, 2008.

Morakot Pilouk. *Integrated modelling for 3D GIS*. PhD thesis, ITC, The Netherlands, 1996.

E. Schönhardt. Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 98:309–312, 1928.

Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA, 1997.

Jonathan Richard Shewchuk. Constrained Delaunay tetrahedralization and provably good boundary recovery. In *Proceedings 11th International Meshing Roundtable*, pages 193–204, Ithaca, New York, USA, 2002.

Jonathan Richard Shewchuk. Star splaying: An algorithm for repairing Delaunay triangulations and convex hulls. In *Proceedings 21st Annual Symposium on Computational Geometry*, pages 237–246, Pisa, Italy, 2005. ACM Press.

H. Si and K. Gärtner. 3D boundary recovery by constrained Delaunay tetrahedralization. *International Journal for Numerical Methods in Engineering*, 85(11):1341–1364, 2011.

Hang Si. Tetgen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. User's manual v1.3 9, WIAS, Berlin, Germany, 2004.

Hang Si. *Three Dimensional Boundary Conforming Delaunay Mesh Generation*. PhD thesis, Berlin Institute of Technology, Berlin, Germany, 2008.

Peter van Oosterom and Martijn Meijers. Towards a true vario-scale structure supporting smooth-zoom. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19, Paris, 2011.

Peter van Oosterom, Jantien Stoter, Wilko Quak, and Siyka Zlatanova. The balance between geometry and topology. In Dianne Richardson and Peter van Oosterom, editors, *Advances in Spatial Data Handling—10th International Symposium on Spatial Data Handling*, pages 209–224. Springer, 2002.

Sisi Zlatanova, Alias Abdul Rahman, and Wenzhong Shi. Topological models and frameworks for 3D spatial objects. *Computers & Geosciences*, 30(4):419–428, 2004.