Topologically consistent 3D city models obtained by extrusion

Hugo Ledoux
h.ledoux@tudelft.nl

Martijn Meijers
b.m.meijers@tudelft.nl

One of the simplest methods to construct a 3D city model is to extrude building footprints to obtain "block-shaped" polyhedra representing buildings. While the method is well-known and easy to implement, if the 2D topological relationships between the footprints are not taken into account, the resulting 3D city models will not necessarily be *topologically consistent* (i.e. primitives shared by 3D buildings will be duplicated and/or intersect each others). As a result, the model will be of little use for most applications, besides visualisation that is. In this paper, we present a new extrusion procedure to construct topologically correct 3D city models. It is based on the use of a constrained triangulation, is conceptually simple, and offers great flexibility to create city models in different formats (e.g. CityGML or a surface-based representation). We have implemented the procedure, tested it with real-world datasets, and validated it.

1 Introduction

Technologies such as airborne laser scanning (or LiDAR—Light Detection And Ranging) permits us to rapidly and easily collect height information for a given area. With the information collected it is possible to construct automatically three-dimensional urban models (Alexander et al., 2009; Zhou et al., 2004; Rottensteiner, 2003), and many cities around the world are currently acquiring their own model. The simplest way to automatically construct a 3D city model is arguably with *extrusion*. That is, given a set of footprints representing the buildings (polygons in the plane),

assign them a height (by averaging LiDAR data for instance) and then push them upwards to create polyhedra. This volumetric representation will be "block-shaped": roofs will be horizontal planes without additional structures, and walls will be vertical planes; Figure 1 depicts the general idea. This representation is also referred to as the LOD1 (level of detail) in CityGML, the international standard for representing and storing 3D city models (Kolbe, 2008; OGC, 2012). There are five LODs for a city model: from LOD0 where only the terrain is stored to LOD4 where buildings have detailed roofs structures, windows, rooms and even pieces of furniture. Extruded models obviously limit the representation of a city (many real-world cases cannot be modelled) but it is nevertheless widely used since the required datasets are usually readily available and it is an automatic procedure (by contrast LOD3 and LOD4 often require manual operations).

Extrusion is widely considered to be an easy operation, and as a consequence little attention has been paid to it. We argue in this paper that it is indeed an easy operation if the only thing you want to do with your 3D city model is to *look* at it! As explained in Section 2, many commercial GIS packages can extrude polygons to obtain buildings, but the resulting models cannot be stored in a topological data structure and therefore cannot be used for further analysis. The reason is that the created models are not *topologically consistent*. In a nutshell, that means that they contain for example duplicate points, overlapping faces, faces intersecting where there are no points, etc. Individually each polyhedron is valid, but the set of polyhedra contains such problems; we define formally topological consistency in Section 3. Figure 1c shows a simple example of the problems encountered: the "front face" of the extruded building *A* should be modelled with two separate faces since the footprint of building *B* is adjacent to *A* in 2D.

It should be stressed here that topological consistency of a dataset has several advantages. These have been recognised for years for 2D datasets (Molenaar, 1998; Theobald, 2001; van Oosterom et al., 2002), and the same advantages are present in 3D. The benefits the most often mentioned are: it is easier to consistently maintain a dataset when changes occur over time; spatial operations on the objects are possible and their outcome are guaranteed to be valid; the objects can easily be stored with a topological data structure; etc. It is true that at this moment the use of 3D city models is mostly restricted to their *visualisation*, but we believe that in the foreseeable future these models will also be used to *analyse* the properties of a city and will also help in the decision-making process for urban planners. There are already several examples of such applications: noise modelling in 3D (CityGML has an application extension for noise mapping), flood modelling (Schulte and Coors,

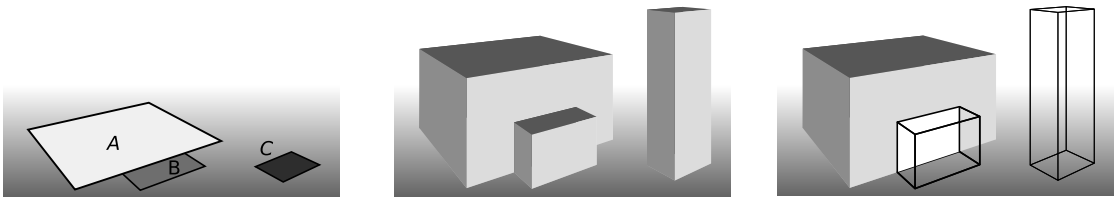


Figure 1: **(a)** Three polygons in the plane. **(b)** Three polyhedra obtained by extrusion of the three polygons. **(c)** To be topologically consistent the front polygon of the polyhedron obtained by the extrusion of *A* should be modelled with two polygons.

2008), 3D navigation (Lee and Zlatanova, 2008), disaster management (Kolbe et al., 2008) and urban planning (Köninger and Bartel, 1998).

In this paper, we present a new procedure to construct topologically consistent 3D city models by extrusion of building footprints. The procedure, which is described in Section 4, is conceptually simple and straightforward to implement. It takes as input a topologically consistent dataset in 2D, and can output different formats, including for instance CityGML. The fact that we maintain a topological data structure in 3D during the extrusion process gives us flexibility and we can easily add other output formats, if needed, for specific applications. We have put strict requirements on the input because cleaning a 2D dataset is a well-known task for which different tools exist. In 3D, the same task is titanic, for, to the best of our knowledge, there are simply no tools available. At the end of Section 4 we demonstrate how our results can be validated. Furthermore, we report in Section 5 on our implementation of the algorithm to create the 3D city model of a test area. In this section, we also describe the tools we used to obtain topological consistency out of a 2D *shapefile* input, and show examples of the results obtained. It should be noticed that this paper builds on our early results for extrusion (Ledoux and Meijers, 2009), and contains several significant improvements: (i) the data structure used to handle the 2D footprints is a constrained Delaunay triangulation (CDT), which as explained in Section 4.1 has several advantages; (ii) we easily handle holes/islands in polygons, so that inner courts of buildings can be modelled; (iii) the output of the prototype we developed can now have several forms, including a "2.75D" surface (see Section 5); (iv) the extrusion process takes into consideration that the ground is not necessarily a horizontal plane, and a digital terrain model (DTM) can be used.

2 Related work

As mentioned in the Introduction, the "simple" extrusion of footprints, that is without considering other footprints, is a straightforward task and has been implemented in many commercial products, for instance Oracle Spatial 11g[1], ArcGIS[2], and Google Earth[3]. Each resulting polyhedron is valid (i.e. it is a simple and watertight polyhedron), but there are no guarantees that a set of footprints will yield a topologically consistent city models. It should be made clear here that most commercial products consider extrusion simply as a visualisation task, i.e. footprints are extruded to polyhedra only in the visualisation window (e.g. the OpenGL window) and no persistent models are created (although some offer this possibility, but each polyhedron is saved individually).

Tse et al. (2005) extrude footprints and output topologically consistent models, but take a totally different approach than ours. Instead of using a volumetric approach, they model a city and its buildings with a single triangulated surface (a 2-manifold embedded in 3D space). Unlike 2.5D models, the surface is allowed to have vertical walls (and also overhangs, tunnels and bridges), so their approach is referred to as "2.75D". They start by constructing the constrained Delaunay triangulation of the footprints, and then extrude the buildings by always keeping a single triangulated surface for the whole area. The main advantage of using a surface-based representation is that you "get 3D for the price of 2D" (Gröger and Plümer, 2005), i.e. 3D modelling is performed but the data

[1] http://www.oracle.com/technology/products/spatial/index.html
[2] http://www.esri.com/arcgis
[3] http://earth.google.com

structure used for the storage is two-dimensional, thus simpler. The main drawback is that, unlike the approach we propose in this paper, it is impossible to represent the vertical walls between two connected buildings (e.g. the 2 buildings *A* and *B* in Figure 1 are stored without the internal walls) as the whole dataset must be represented by a single surface. Notice that the algorithm we present in Section 4 permits us to also model a city with such surface, if required.

Another way to obtain topologically consistent 3D city models is to use constructive solid geometry (CSG). With this method, polyhedra are represented as Boolean combinations (union, intersection and difference) of simpler objects such as cylinders, cubes, spheres or pyramids. CSG objects are by definition topologically consistent, and it is possible to convert them to a boundary representation (see Requicha (1982) for more details). Haala and Brenner (1999) create extruded 3D city models by first decomposing footprints into rectangles, and from these rectangles create CSG polyhedra, which are then combined to reconstruct the building. The main problem of that approach is that not every building footprint can be decomposed into rectangles; as can be seen in Section 5.5, some of the footprints of our test area have for instance circular shapes (discretised into polylines). Another potential problem of CSG solutions—or of any CAD data structures—is that the data structures used are rather "heavyweight", and consequently scaling to datasets with hundreds of thousands of objects is rather problematic.

Cell decomposition is another approach that can be used to obtain topological consistency (Haala et al., 2007). Each edge of a building becomes a half-space plane (an infinite vertical plane defined mathematically), and intersections of planes permits us to reconstruct the building. It is not clear how this method would scale up to big datasets since it appears that every half-space plane has to be tested with every other one in the dataset.

3 Topological concepts related to extrusion

This section introduces the key concepts of graph theory and spatial modelling related to extrusion. It should be noticed that in the literature related to these fields different names are often used to refer to the same concepts, and our aim in this section is not to give a thorough overview of spatial modelling (see Molenaar (1998), among others, for that), but rather to introduce our terminology and avoid ambiguity in the rest of the paper.

3.1 Two-dimensional spatial modelling

Geographical phenomena (e.g. cities, roads and houses) are often modelled as embedded in $\mathbb{R}^2$, the two-dimensional Euclidean space, with 3 classes of geometric primitives: (i) points (0-dimensional objects); (ii) line segments (2-dimensional objects); and (iii) polygons (2-dimensional objects). A sequence of line segments connected by their end-points is called a *polyline*; each point in a polyline is part of two lines, except the first and the last ones (we call them the *extremes* of the polyline).

A spatial object σ of dimensionality higher than 0 is formed by lower dimensionality primitives that define its boundary, denoted $\partial\sigma$. If σ is a line segment, then $\partial\sigma$ is formed by both end-points of σ. If σ is a polygon, then $\partial\sigma$ is formed by one or more polyline(s) forming a *loop*; more than one loops are required for polygons having *holes*, which occurs often in practice. A polygon can have several holes if: (i) these holes do not intersect each other or the outer boundary of σ; (ii) the interior of σ, denoted σ°, stays connected; (iii) holes are located inside σ. The validity of a

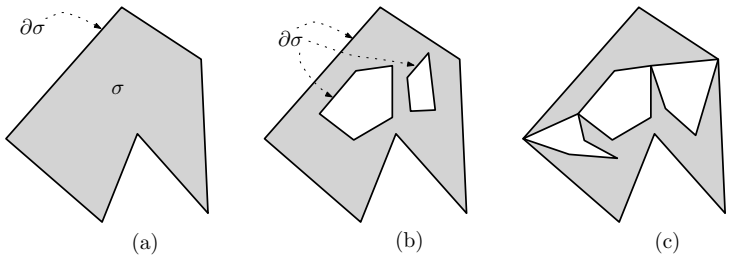


Figure 2: **(a)** A simple polygon σ with one polyline forming its boundary. **(b)** σ has 2 holes, and thus $\partial\sigma$ is formed by 3 polylines. **(c)** According to ISO (TC211), this polygon is not valid since its interior is not connected; two polygons would be needed to represent such a shape.

polygon is a complex process and more rules are necessary, readers can find more information in ISO (TC211) and van Oosterom et al. (2004). Observe that holes can also contain other polygons, and these are named *islands*. These concepts are shown in Figure 2.

Topological consistency. Let M be a set of spatial objects in $\mathbb{R}^2$, we say that M is *topologically consistent* if the following rules are valid:

1. every line segment in M is formed by two points also in M;
2. the intersection of two line segments σ_1 and σ_2, denoted $\sigma_1 \cap \sigma_2$, is either empty or is a point in M;
3. the intersection of the interior of a polygon σ_1 with another object in M, $\sigma_1^\circ \cap \sigma_2$, is empty.

Rule 3 implies that the intersection of two polygons σ_1 and σ_2, is either empty or a finite set of objects in M (these can be points and/or line segments).

Connectedness. Consider two geometric primitives σ_1 and σ_2. We say that σ_1 is *incident* to σ_2 if σ_2 is a primitive forming $\partial\sigma_1$ (the dimensionality of the two primitives is different), and we say that σ_1 and σ_2 are *adjacent* if they share a lower-dimensionality primitives (here σ_1 and σ_2 are of the same dimensionality). If the set of shared primitives contains at least one line segment, σ_1 and σ_2 are *strongly* connected, and if it contains only points, then they are *weakly* connected (in Figure 3a, polygon A and D are strongly connected, while A and B are weakly connected).

3.2 Three-dimensional spatial modelling

If geographical phenomena are modelled in $\mathbb{R}^3$, then one more class of geometric primitives has to be introduced: *polyhedra* (3-dimensional objects). As is the case in 2D, a polyhedron is formed by lower dimensionality primitives. The concept of topological consistency in $\mathbb{R}^3$ is a straightforward generalisation of the three rules previously defined. The only differences are that: (i) all the primitives are embedded in $\mathbb{R}^3$ (for our purposes, we assume that polygons embedded in $\mathbb{R}^3$ are

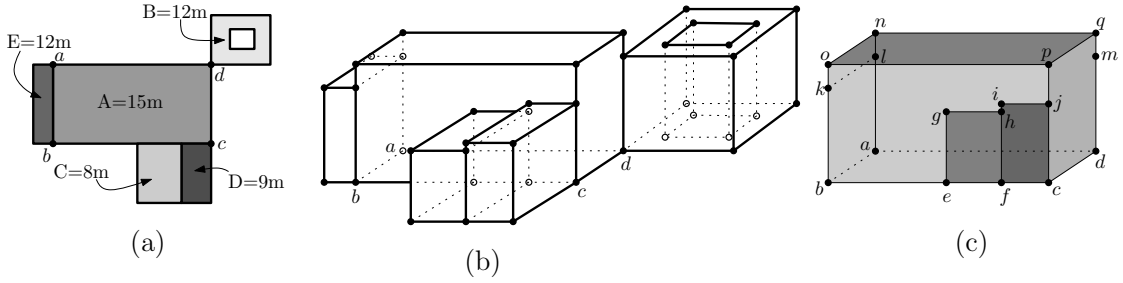


Figure 3: Overview of the extrusion of building A, by taking into account other buildings in the neighbourhood. **(a)** The footprint of A is the polygon $abcd$. **(b)** Perspective view of the result (with connected buildings). **(c)** A is represented with several polygons.

planar); (ii) another rule has to be added: the intersection of the interior of a polyhedron σ_1 with another primitive $\sigma_1^\circ \cap \sigma_2$, is empty.

Observe here that while general polyhedra are allowed to have holes, polyhedra obtained by extrusion will by definition never contain holes. If a polygon contains a hole, then its extruded polyhedron can still be represented as one connected volume. As shown in Figure 3, the footprint B results in a polyhedron having no hole (but a genus 1).

3.3 From 2D to 3D with extrusion

In the context of 3D city models, extrusion means that a building (a polyhedron) is constructed by "pushing upwards" its footprint (a polygon). For a footprint with n line segments, the resulting building is formed of $n + 2$ polygons:

- the polygon on the ground corresponds to the footprint;
- the roof polygon has the same shape as the floor but all points are at the extruded height;
- every line segment becomes a wall (a vertical polygon).

If the building footprints are considered independently, or if a footprint does not have any other footprint adjacent, then as discussed previously creating a building is an easy task. However, if there are connected footprints (weakly and/or strongly), the result will not be automatically topologically consistent. Indeed, consider the case in Figure 3a where five connected footprints are extruded to different heights, yielding the set of polyhedra shown in Figure 3b.

Let us focus on the building A whose footprint is polygon $abcd$. Figure 3c shows the resulting extruded building, with different shades of grey for every extruded polygon. First of all, notice that A has 17 points and 9 polygons; a "simple extrusion" would have created here 8 points and 6 polygons. Other observations:

- the floor polygon is not the polygon $abcd$, but $abefcd$, because C and D are strongly connected to A;
- the roof polygon is however formed only of the four points forming the footprint, for the buildings adjacent to it are not as high;

- line segment *ab* is extruded to two polygons *abkl* and *klno* since *A* and *E* do not have the same height. Observe also that polygon *abkl* will be a polygon of both *A* and *E*;
- buildings *A* and *B* are weakly connected and have different heights, which means that the line segments incident to *d* in 2D (*cd* and *ad*) are extruded to polygons containing a point at the elevation of building *B* (here polygons *cdmqpj* and *dmqnla*). Notice also here that point *l* is in the second polygons because *E* is adjacent to *A*;
- line segment *bc* in the footprint is the most complex one to extrude since three polygons must be extruded (*beghijpok*, *efhg* and *fcjih*).

As can be seen from these observations, two factors are to be considered when extruding building footprints: (i) the relative heights of two connected buildings, (ii) their type of connectedness. It should also be said that we cannot simply consider pair-wise the building footprints as more complex situations arise, for instance in Figure 3c the point *f* has 3 incident buildings (*A*, *C* and *D*) that are extruded to different heights. When creating the polygons containing *f*, all its adjacent footprints have to be taken into account. It is worth pointing out that many cases are also very simple to handle, for instance extruding a line whose end-points have only one other incident line, as is the case for two lines of building *B* (the two top-right lines). Here, no special cases arise and the simple extrusion is sufficient.

3.4 Representing and storing topological relationships

Storing spatial objects and storing explicitly the topological relationships between them requires the use of topological data structures. These data structures are based on concepts developed in graph theory (and also in combinatorial topology), where the primitives *nodes* and *edges* are used to represent respectively points and line segments. If the graph is planar (embedded in $\mathbb{R}^2$ and no 2 edges cross) then a loop of edges form a *face* (which represents a polygon).

A graph can also be embedded in $\mathbb{R}^3$. Nodes and edges organised in a suitable way will define faces (for this paper, we restrict to planar faces), and if a set of faces form a closed surface then a polyhedron is represented.

Two-dimensional data structures. If a set *M* of spatial objects modelled in the plane is topologically consistent then it is straightforward to store it with a topological structure such as the well-known *node-edge-face* data structure (NEF). However, if *M* is not topologically consistent, then the task is more complex as cleaning is required; we discuss one way to clean GIS datasets in Section 5.2. With a NEF structure, each edge is directed (it has a start and an end node), and the faces left and right of each edge are also stored (most GIS textbooks, such as Longley et al. (2001) and Worboys and Duckham (2004), give detailed description of the structure). This permits us to explicitly represent the concept of strong connectedness, but not weak connectedness. To represent the latter, another data structure where the ordering of the edges incident to nodes must be used; the DCEL structure (Muller and Preparata, 1978) or the *half-edge* (Mäntylä, 1988) are two examples. It should however be noticed that, if not explicitly stored, weak connectedness can always be derived, e.g. by brute force computations, or from the NEF structure.

Three-dimensional data structures. If the spatial objects are in $\mathbb{R}^3$ (e.g. buildings in a city) then different alternatives are possible. Polyhedral objects are usually represented by their exterior boundaries (called *b-rep* models), and thus the use of 2D topological data structures can be extended into the 3D domain. Zlatanova et al. (2004) give an excellent summary of the data structures that have been proposed for GIS-related applications, and also discuss how topological relationships between objects are derived when they are not explicitly stored. Two examples of data structures for 3D spatial objects are the *3D Formal Data Structures* of Molenaar (1990), and the TEN structure (TEtrahedralized irregular Network, the 3D equivalent of a TIN—triangular irregular network) discussed and used by several (Egenhofer et al., 1989; Pilouk, 1996; Penninga, 2008).

4 Extrusion with the CDT and the node column approach

This section presents our new approach to extrude 2D footprints and construct a set of topologically consistent polyhedra. This approach to extrusion has three novel parts. First, it is based on the idea that the input of the extrusion algorithm is a set of topologically consistent polygons (in 2D), as defined previously. As far as we know, there are no tools available to clean a 3D dataset (e.g. remove slivers and duplicate faces, split faces that intersect, etc), thus we put strict requirements on the 2D input so that our goal can be achieved. Second, we model the topological relationships between polygons and polyhedra with the concept of a *node column*. Third, to construct and maintain the topological relationships between the footprints we use a triangulation, which guarantees a perfect fit between our extruded buildings and the terrain.

The extrusion algorithm we present takes as input a topologically consistent dataset of polygons (to which an extrusion elevation is assigned), and outputs a list of topologically consistent buildings and a list of faces embedded in 3D. A building is modelled by its bounding faces, and thus in the algorithm it is defined as a container for these faces. Faces shared by adjacent polyhedra are not duplicated, and a building simply has a reference to a set of faces. The algorithm, called EXTRUDE and described in Figure 4, works as follows. First, the constrained Delaunay triangulation (CDT) of the input dataset is created, and then the ground and roof faces of a given building are created by identifying the triangles in the CDT forming its footprints. The vertical walls are then created by visiting the constrained edges in the CDT and extruding them, taking into account the configuration of incident footprints. The important steps of the algorithm are detailed below.

4.1 Representing topology with a constrained triangulation

In our previous work (Ledoux and Meijers, 2009), we used a NEF data structure to store the input polygons. While that permitted us to extrude the polygons, two problems were present: (i) extra mechanisms would have had to be added to the NEF structure to handle holes, which would have resulted in a rather slow and cumbersome implementation; (ii) the NEF did not store explicitly the weakly-connected polygons, and these had to be extracted on-the-fly.

Instead of the NEF we use for the present work a constrained triangulation. Given a set E of points and straight-line segments in $\mathbb{R}^2$, the constrained triangulation of E is a triangulation where each segment in E is also present in the triangulation. If stored with a data structure such as the DCEL (Muller and Preparata, 1978) or the *half-edge* (Mäntylä, 1988), then both strongly- and weakly-connected components are explicitly stored. As can be seen in Figure 5, each polygon is

Input: a topologically consistent dataset D with 2D footprints (with extrusion height for every footprint)
Output: a topologically consistent list B of polyhedra (the 3D buildings)

$\tau \leftarrow$ create the CDT of D
flag every triangle in τ with ID of footprint
for footprint in D **do**
  $b \leftarrow$ create a new building
  $F \leftarrow$ fetch triangles in τ having ID of footprint
  $b \leftarrow$ create floor face from F {with ground elevation}
  $b \leftarrow$ create roof face from F {with extrusion elevation}
  $B \leftarrow$ add b
end for
for constrained edges in D as e **do**
  build the node column for e_{start} and e_{end}
  if only one footprint is incident to e **then**
    $b \leftarrow$ extrude edge to 2 triangular faces {b is the building whose footprint is incident to e}
  else
    $b_{left}, b_{right} \leftarrow$ extrude edge to 2 triangular faces {b_{left} and b_{right} are the 2 buildings incident to e}
    $b_{highest} \leftarrow$ extrude upper face of e {$b_{highest}$ is the building with the highest extrusion height}
  end if
end for

Figure 4: The EXTRUDE algorithm.

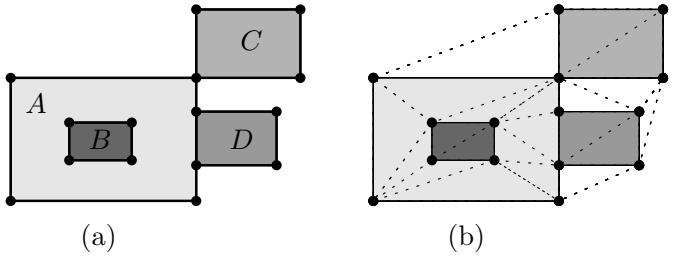


Figure 5: **(a)** Four polygons. **(b)** The CDT of the points and line segments of the 5 polygons; each triangle is coloured according to the polygon inside which it is located.

decomposed into triangles, and to add the knowledge of polygon objects in the triangulation we simply flag each triangle with the ID of the input polygon (the colour in this case). Each polygon can then be represented by a set of connected triangles. Observe that the structure also permits us to handle holes in polygons: a hole is always connected to the outer boundary of its polygon by an (unconstrained) edge, and therefore no auxiliary data structure has to be implemented and maintained. For instance in Figure 5 polygon *B* is an island inside *A* and its boundary is connected by several triangulation edges to the outer boundary of *A*.

If the triangles are used to represent the bounding faces of buildings, then the *constrained Delaunay triangulation* (CDT) is probably a better choice than an arbitrary constrained triangulation since it will avoid, as much as possible, long and skinny triangle. Although its name might imply that the triangles are Delaunay (i.e. their circumcircle is empty), this is not always the case. Indeed, a CDT is a triangulation where triangles are empty of points that are visible to them (constraints act as visibility blockers); see Shewchuk (1997) for more details.

4.2 Ground surface for extrusion and TIN integration

Embedding the input polygons in a triangulation is not only useful to represent the topological relationships between spatial primitives, it also has other benefits for the extrusion. The main advantage is that it permits us to integrate the buildings with a DTM; a TIN can be used directly, and if the DTM is raster-based then it has to be converted to a TIN. We can thus obtain a triangulated surface containing both the DTM (bare Earth surface) and the buildings, as in the work of Tse et al. (2005) on 2.75D surfaces.

The extrusion of a set of footprints can be performed from two different ground surfaces:

1. an horizontal plane at a given elevation;
2. a DTM/TIN.

The first case, although not being very realistic, is usually the definition given to extrusion. The main problem of the method is that once the buildings are extruded, then making them "fit" with a DTM is rather difficult since all the ground floor of buildings are flat faces. Fitting means that there is a correspondence between the TIN and the ground faces of the extruded buildings. One cannot simply modify the elevation value of the vertices so that they fit with a DTM for the face will most likely not be planar anymore. A decomposition of that face into planar faces is needed.

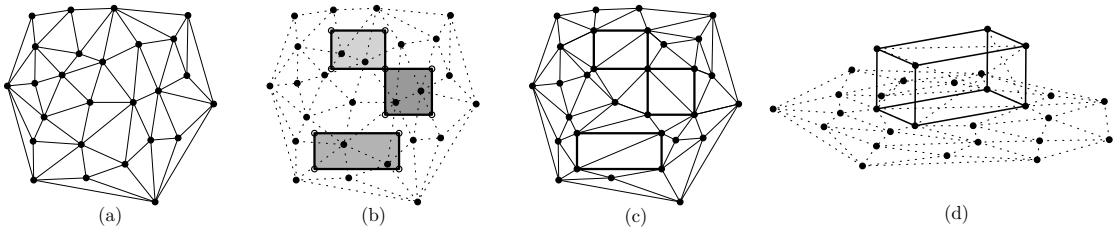


Figure 6: **(a)** The input TIN for the ground surface. **(b)** Three building footprints have the height of their points interpolated. **(c)** The input TIN is modified so that it fits with the building footprints. **(d)** Perspective view of an extruded building integrated in the TIN.

The second option solves that problem by considering a TIN while the extrusion is performed. With our approach, a TIN can be used as input for the ground surface, and the results of the extrusion process is a modified TIN (containing constraints) in which the triangles forming the footprints of the buildings are present. As Figure 6 illustrates, this is performed in different steps. First the input set of points is triangulated (a Delaunay triangulation), then the points of the input footprints have their elevation estimated by interpolation in the TIN (with linear interpolation in the triangles). To ensure that the footprints and the TIN will fit, the TIN is modified by removing all the points located inside a footprint, and the boundaries of the footprints are inserted as constraints in the TIN, with the elevation that were previously interpolated.

To obtain a 2.75D surface representation of all the objects (TIN + 3D buildings), it suffices to delete from the TIN and the polyhedra the triangles forming the ground surface of every object, and the triangles forming the inner walls (with our approach, these are automatically identified). As Penninga (2005) states, while the approach is perhaps less attractive than a volumetric representation, managing surfaces is easier and users might prefer this option.

4.3 Extrusion with the node column method

In Section 3 we have shown that finding the geometry of the (vertical) wall faces is complex due to the different heights and given connectedness of the buildings' footprints. Our approach tackles this problem elegantly by introducing the concept of *node columns*. At the location of each node in the triangulation, a node column is erected. Such a column consists of a sorted list of all the different heights of the polyhedra incident to this node, plus the ground height at this location. As mentioned, a triangulation of the input polygon permits us to obtain these relationships directly. Figure 7a illustrates a part of the node columns related to building *A* (as shown in Figure 3a).

We use this concept when extruding constrained edges of the CDT to walls. A function, let us call it ExtrudeSegment, performs the extrusion by taking into account the incident footprints and the differences in height, and output the correct number of wall faces for each building. Its input is a constrained edge, and the start and end heights of the face to be created. A wall face is formed as follows: the given start node column is ascended from the start height, until the end height is reached. Subsequently the end node column is descended from that height, until the start height is encountered.

The resulting geometry of the face is then added to the building(s), taking into account the ori-

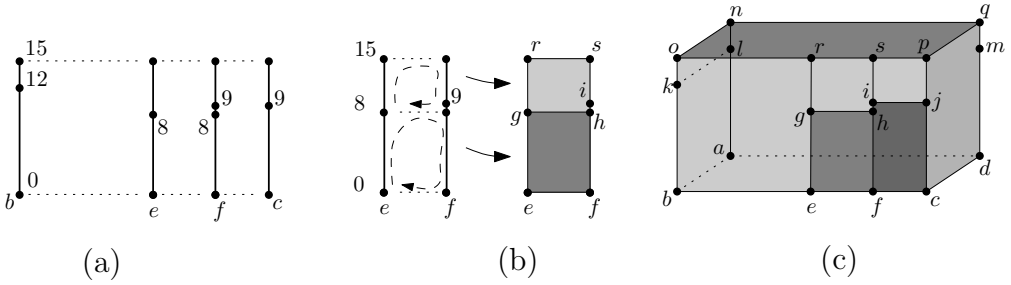


Figure 7: **(a)** Node columns generated for the front side of building *A* (same configuration as in Figure 3). **(b)** Extrusion of edge *ef* results in two faces **(c)** Extruded result for the footprint of building *A*.

entation of the face. The orientation of the face means here that the ordering of its nodes define a normal vector pointing outwards the building. If two faces are incident to the edge to be extruded, then the resulting face will be added to both buildings. Furthermore, if one building is higher than the other, the process will have to be repeated, this time the start height will be that of the lowest building. Observe also that if both buildings are of the same height, than only one face is constructed and assigned to both buildings.

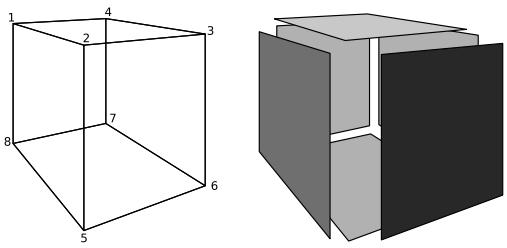
Figure 7b shows an illustration of how ExtrudeSegment uses the node columns: the node columns at location *e* and *f* are given to the ExtrudeSegment function when edge *ef* is processed. Edge *ef* is adjacent to building *A* and building *C*, for which building *A* is the tallest of the two. First, the ground height (let us assume 0m here; other values can be used, see Section 4.2) and the height of building *C* (8m) are given. This way a loop is formed over the two given node columns and results in face *eghf*. Second, the height of building *C* (8m) and the height of building *A* (15m) are given, together with the two node columns and face *grsih* is the result. Face *eghf* is added to the buildings *A* and *C*, as both are adjacent to edge *ef*. Face *grsih* is only added to building *A* (as this is the tallest of the two buildings). Notice here that since we extrude buildings by processing edges, the resulting face between building *A* and buildings *C* and *D* is modelled with five faces, and not three as in Figure 3c.

Also, since an edge is extruded to a rectangular face, we can simply arbitrarily decompose this face into two triangles. Coupled with the triangle-based representation for the ground and the roof faces, we obtain a boundary representation of a building where all the faces are triangulated.

4.4 Geometric validation of the results with the constrained Delaunay tetrahedralization

The geometric validation of the polyhedra obtained by extrusion can not be performed with the rules defined by the *International Organization for Standardization* for 3D objects (ISO, TC211; Kazar et al., 2008; Verbree and Si, 2008) since these consider independently each polyhedron, and here we want to validate the set of polyhedra together by taking into account their relationships. Such an operation is possible in 2D because many commercial packages, such as ArcGIS or Oracle Spatial, offer that possibility. This is based on the validation of geometrical and topological rules that the planar graph of the input must fulfill. We are not aware of any such tools for 3D datasets.

Since it is conceptually simple, we can intuitively see that the algorithm presented in this section



(a)

```
# The list of points
8 3 0 0
1 0.0 0.0 1.0 # index, x, y, z
2 1.0 0.0 1.0
3 1.0 1.0 1.0
...
# The list of polygons
6 0
# The top polygon
1 # 1 polygon
4 1 2 3 4 # 4 points, by IDs
# Other polygons
...
```

(b)

Figure 8: **(a)** A cube is represented with 6 polygons. **(b)** Example of a PLC file for the cube.

is valid. If the input footprints are topologically consistent, then so should be the output polyhedra. However, to validate our implementation of the algorithm, we have designed three tests for the output set of polyhedra:

1. no interior of 2 polyhedra intersect;
2. the set of faces is topologically consistent;
3. each polyhedron is formed by a closed bounding surface; we say it is *watertight*.

Rule 1 is relatively easy to verify. If the interior of the footprints do not overlap, and if only vertical walls are extruded, then the interior of the resulting polyhedra should not intersect. This can be verified by projecting the polyhedra to the $x - y$ plane, and ensuring that they stay within their own footprint.

To verify the two other rules, we use the properties of the *constrained Delaunay tetrahedralization*, which we will abbreviate with CDT3 to avoid confusion with its 2D counterpart. As explained in Shewchuk (2002), CDT3s have strong mathematical foundations and are used in the generation of meshes in engineering. The input of a CDT3 algorithm is a *piecewise linear complex* (PLC), as first introduced by Miller et al. (1996). A PLC, which is a general boundary description for 3D shapes, contains a set of points, together with a set of straight line segments and polygons (embedded in 3D space). Figure 8 shows one example for a cube.

Line segments and polygons in a PLC are allowed to intersect at a shared point, and two polygons may intersect only at a finite number of shared points or lines. PLCs are more general than polyhedra, i.e. every polyhedra as defined in Section 3 is a PLC, but not vice versa. Observe that these rules are the same as the ones for topological consistency as defined in Section 3. As a consequence, if the input of a CDT3 algorithm is not a valid PLC, then it is impossible to compute the CDT3.

To verify rule 2, we therefore need to convert our set of faces to a PLC format—a straightforward operation given the output of our approach. To compute the CDT3, we use the software TetGen (Si, 2004).

To validate rule 3, we exploit one property of a CDT3: the whole area covered by a dataset (its convex hull) is tetrahedralized. This can be seen in Figure 5 for the 2D case: both the interior of

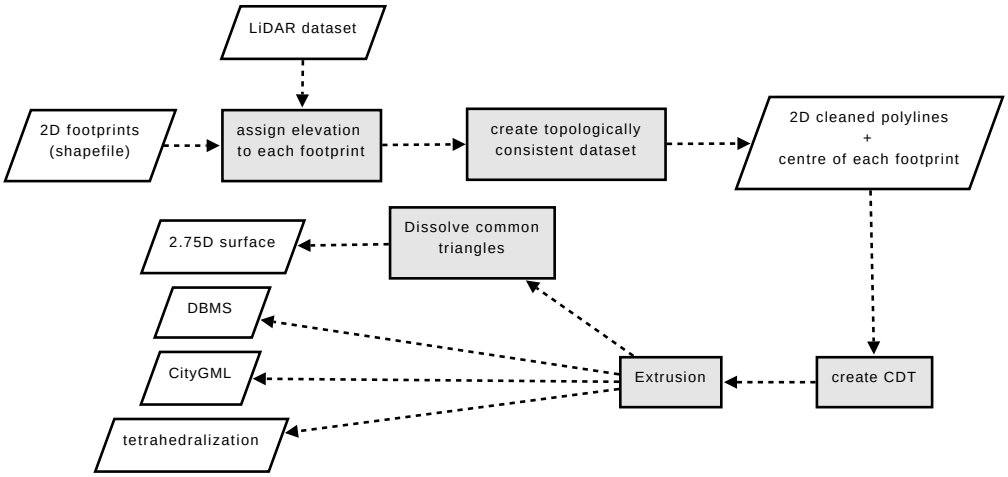


Figure 9: Our workflow to obtain a extruded 3D city model.

the polygons and the "air" between there are triangulated (the 3D case is a straightforward generalisation). To ensure that the extruded polyhedra are watertight, we generate one point inside each polyhedron and then we start "walking" in the CDT3 to visit the neighbouring tetrahedra, until we are blocked by a constraint face (as we go, we flag tetrahedra). Think of a depth-first search on the dual of the tetrahedralization. At the end of the process, we simply need to ensure that the tetrahedra flagged as "air" have not been overwritten: one non-watertight polyhedron would indeed make the walk visit all the "air" tetrahedra.

5 Implementation and experiment

We report in this section on our implementation of the algorithm described in the previous section, and on the experiment we carried out with real world data.

We implemented the EXTRUDE algorithm with the Python scripting language[4]. To extrude a dataset with footprints, we used the workflow shown in Figure 9. In the following we discuss key points of our implementation, including the output formats.

5.1 Input dataset

The input that can be used by our implementation is a standard GIS file in *shapefile* format (ESRI, 1998), where each polygon (a building footprint) has one value assigned for its extrusion. Other standard GIS formats, such as GML, could also be used. How the height values are calculated is out of scope for this paper, but can be done for instance by simply taking the median (or the average) of all LiDAR points located inside a footprint.

[4] `http://www.python.org`

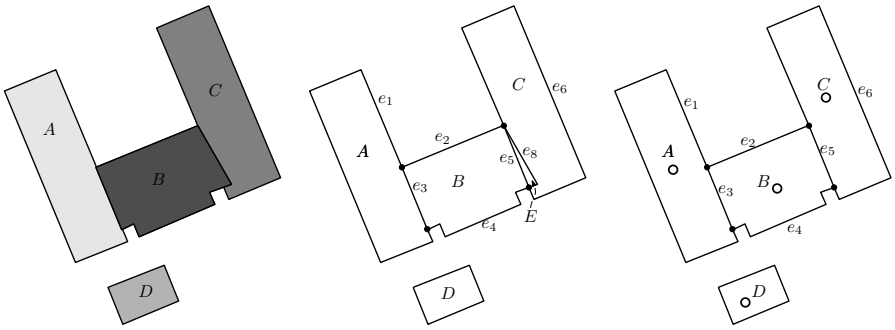


Figure 10: **Left**: 4 input polygons input obtained from GRASS. **Middle**: The planar graph of the 4 polygons. **Right**: the cleaned planar graph where overlapping areas are removed.

5.2 Cleaning the input with GRASS

The input of our procedure is a shapefile with a set of polygons, which means that each polygon is represented independently (the polyline between 2 adjacent polygons is thus duplicated). In practice, it is often the case that the polygons overlap a bit each other, because of the overshoot/undershoot problems when snapping lines. In order to obtain a topologically consistent 2D input dataset, we use the open-source—and free—GIS GRASS[5]. Note that GRASS is not the only GIS package capable of performing this task, others could also be used.

First, a planar map (a planar graph) of the input polygons is constructed, and stored in the GRASS data structure, which is basically a NEF plus mechanisms to handle holes (Neteler and Mitasova, 2008). Figure 10 shows the idea for 4 polygons. Observe that polygons *B* and *C* overlap a bit (and we need to fix that). During the construction of the topology, GRASS removes the duplicate line segments, and the points can be snapped to each others (based on a user-defined tolerance) so that small overlap between polygons are avoided. GRASS also offer operators to clean the result, for example `v.clean` is used to remove faces that are smaller than a given threshold; that permits us to ensure that there is no overlapping polygons in the resulting map. In Figure 10, the small face caused by the overlapping polygons was for instance removed. Observe also that faces in GRASS are identified by their "centroid" (a point located inside the face); the boundary of the face is reconstructed from the edges enclosing that centroid.

5.3 Creation of the constrained triangulation

The CDT is computed with the Python bindings of CGAL[6]. The input for the CDT computation is the output from GRASS: the set of edges representing the boundaries of the footprints (they will form the constraints). Since the CDT does not have the knowledge of the polygonal objects (but only of edges), we flag each triangle in the CDT so that they contain the ID of the building they are part of. We do that by starting at the centroid of each polygon (obtained from GRASS), and then we start walking in the triangulation to visit the neighbouring triangles, as explained in Section 4.4 for the 3D case. This way, as shown in Figure 11, each triangle inside a polygon is flagged with

[5] `http://grass.itc.it`

[6] Computational Geometry Algorithms Library. `http://www.cgal.org` and `http://cgal-python.gforge.inria.fr`

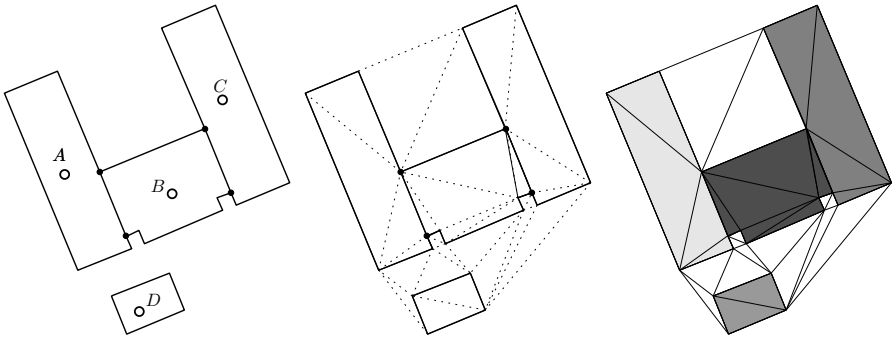


Figure 11: **Left**: The input obtained from GRASS. **Middle**: the CDT of the input. **Right**: the CDT where each triangle is flagged according to the input polygons. The white triangles are the "universe" triangles.

the ID of its footprint. The triangles forming the "universe" (the space between footprints) will be flagged as such (the white triangles in Figure 11).

5.4 Output of the program

Since our program creates a list of unique oriented faces, and that each building simply contains references to these faces (with the proper orientation), it is easy to generate different output formats. At this moment, we offer as output format:

- CityGML;
- a PLC file;
- a triangulated surface of the whole scene;
- and also the list of faces and buildings so that these can be used directly (and stored in a DBMS for instance).

In a CityGML file, each building is stored as a `gml:Solid`, which means that it is a volume with a watertight boundary. Notice that CityGML does not offer (yet) mechanisms to represent topological relationships between primitives, which means that nodes and edges shared by higher-dimensionality primitives are repeated. The only mechanism available at the time of writing is the possibility to store only once faces shared by two adjacent polyhedra *A* and *B*: the face is represented only once, e.g. *A* contains that face, and *B* has a pointer to the face (an `xlink` in XML language).

5.5 Experiment

To test our implementation, we have setup an experiment with real world data of the Delft University of Technology campus (TU Delft). For this area, covering 2.3 km^2 with 370 buildings, we obtained the Large Scale Base map of Delft (*Grootschalige Basiskaart* in Dutch), see Figure 12. This

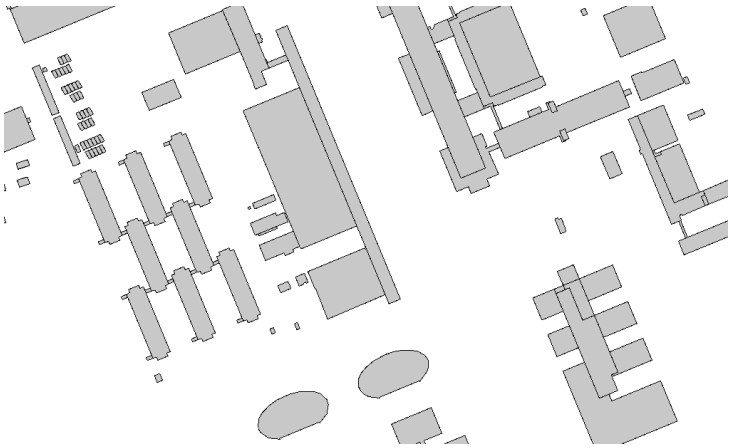


Figure 12: Part of the footprints used for the extrusion.

map was a line-oriented dataset, consequently the first step was to form footprint polygons out of the lines.

As the footprints appeared to be too coarse to get an accurate city model with respect to the height of the buildings, it was decided to split all building footprints into multiple, with respect to height, homogeneous parts. This was a manual work and was accomplished by using interpretation of additional aerial photographs of the area. While this was not necessary, the resulting model was more realistic, and was a better test case as more footprints interacted with each other. Observe that this operation could also in theory be done automatically, if the algorithm of Vosselman and Dijkman (2001) was used.

After subdividing the footprints, a height was assigned to each part. This value was obtained by using the median z value of all LiDAR points within a given footprint.

Having a complete footprint dataset, we followed the workflow described in Section 5. Cleaning the dataset with GRASS implied trying different parameters for the snapping of vertices and the removal of small faces (slivers), until the output contained the same amount of polygons as the input. Note that our input dataset was far from being topologically consistent, so different iterations were necessary. There are also known cases from the practice, e.g. the Large Scale Base map of the municipality of Amsterdam, for which a topological structure is used to maintain the dataset. If this had been the case, the cleaning part could have been skipped.

Figure 13 shows a part of the extruded TU Delft campus, once tetrahedralized. As can be seen, each polyhedron is decomposed into a set of tetrahedra. Notice also that while the tetrahedra representing the "air" are not shown, they are still present. For our set of 370 buildings' footprints, the PLC has 5841 points, and 8152 triangular faces (which act as constrained faces for the CDT3). The result of this operation is a set of 20486 tetrahedra. We have validated this output with the 3 rules defined in the previous section, and no problems were found.

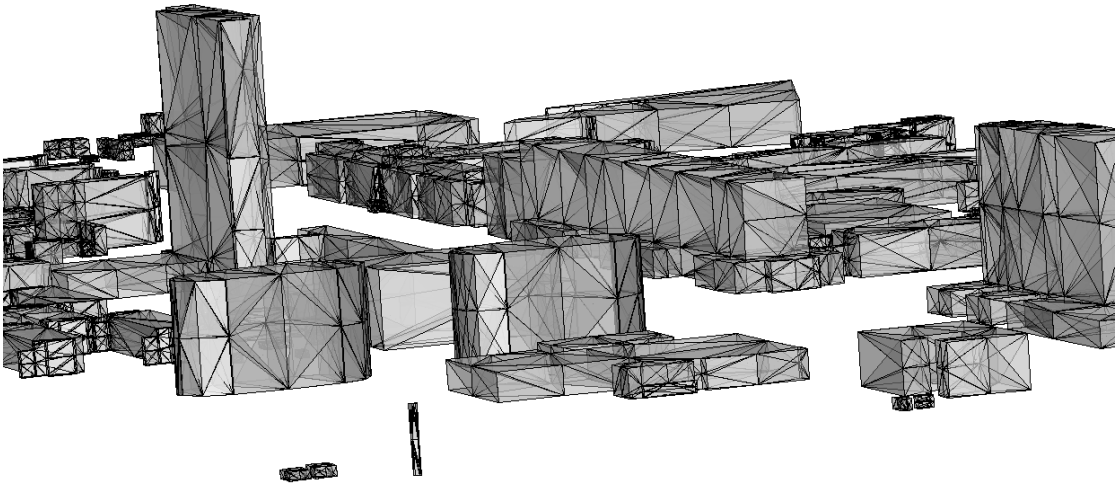


Figure 13: Part of the TU Delft campus tetrahedralized.

6 Conclusions

We have shown that a problem that appears to be rather simple at first glance—the extrusion of building footprints—turns out to be considerably more complex if we put requirements on the resulting datasets. We have solved the problem by detailing an algorithm that is conceptually simple, and easy to implement. A requirement of our approach is that we require the input of our algorithm to be a set of topologically consistent polygons in 2D, since "cleaning tools" are readily available. Our solution introduces the concept of a node column, which permits us to elegantly translate to 3D the topological relationships existing between polygons in the plane. This way, we can construct faces and assign them to the correct buildings (modelled as polyhedra), which yields a topologically consistent 3D dataset. We hope that this work has also shed some light on the topological relationships between 2D data and 3D data obtained by extrusion, and that this will be used in other contexts.

We have used a triangulation as the foundation for the extrusion, and that has according to us several benefits. First, all the topological relationships between footprints can be stored explicitly (including the presence of holes in polygons). Second, the faces created can all be triangulated directly, which gives us flexibility to produce output in different formats (full polyhedra or 2.75D surfaces). Third, triangulated surfaces can directly be used for visualisation since most graphical cards use triangles as their primitives. Fourth, our algorithm can easily scale to very big datasets since the operations to retrieve topological relationships are all local, and it is known that triangulation with several millions of points can be constructed rather quickly, see for instance Isenburg

et al. (2006) and Amenta et al. (2003).

For future work, we plan to automatically construct LOD2 models by "glueing" detailed roofs (obtained from photogrammetry or with LiDAR-based methods) to the block-shaped buildings we have obtained. It suffices indeed to replace the flat roof by another more detailed one, and ensure that its $x - y$ projection fits in the footprint of the building.

Acknowledgements

This research was financially supported by the project "3D Topografie" (RGI-011) in the Netherlands. We would like to thank some colleagues and former colleagues: Sisi Zlatanova and Friso Penninga for posing the problem and for fruitful discussions on the topic; Tarun Ghawana for helping with the datasets and testing the software; Jacynthe Pouliot and Edward Verbree for proofreading and making very useful suggestions.

References

Alexander C, Smith-Voysey S, Jarvis C, and Tansey K (2009). Integrating building footprints and LiDAR elevation data to classify roof structures and visualise buildings. *Computers, Environment and Urban Systems*, 33(4):285–292.

Amenta N, Choi S, and Rote G (2003). Incremental constructions con BRIO. In *Proceedings 19th Annual Symposium on Computational Geometry*, pages 211–219. ACM Press, San Diego, USA.

Egenhofer MJ, , Frank AU, and Jackson J (1989). A topological data model for spatial databases. In Buchmann A, Gunther O, Smith T, and Wang Y, editors, *Design and Implementation of Large Spatial Databases: First Symposium SSD'89*, pages 271–286. Santa Barbara, CA, USA.

ESRI (1998). Esri shapefile technical description: An esri white paper. Available at: `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`.

Gröger G and Plümer L (2005). How to get 3-D for the price of 2-D—topology and consistency of 3-D urban GIS. *GeoInformatica*, 9(2):139–158.

Haala N, Becker S, and Kada M (2007). Cell decomposition for building model generation at different scales. In *Proceedings Urban Remote Sensing Joint Event*, pages 1–6. Paris, France.

Haala N and Brenner C (1999). Virtual city models from laser altimeter and 2D map data. *Photogrammetric Engineering & Remote Sensing*, 65:787–795.

Isenburg M, Liu Y, Shewchuk JR, and Snoeyink J (2006). Streaming computation of Delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049–1056.

ISO(TC211) (2003). ISO 19107:2003: Geographic information—Spatial schema. International Organization for Standardization.

Kazar BM, Kothuri R, van Oosterom P, and Ravada S (2008). On valid and invalid three-dimensional geometries. In van Oosterom P, Zlatanova S, Penninga F, and Fendel E, editors, *Advances in 3D Geoinformation Systems*, Lectures Notes in Geoinformation and Cartography, chapter 2, pages 19–46. Springer Berlin Heidelberg.

Kolbe TH (2008). Representing and exchanging 3D city models with CityGML. In Zlatanova S and Lee J, editors, *3D Geo-Information Sciences*, chapter 2, pages 15–31. Springer.

Kolbe TH, Gröger G, and Plümer L (2008). CityGML—3D city models and their potential for emergency response. In Zlatanova S and Li J, editors, *Geospatial Information Technology for Emergency Response*, ISPRS Book Series, pages 257–274. Taylor & Francis, London.

Köninger A and Bartel S (1998). 3d-GIS for urban purposes. *Geoinformatica*, 2(1):79–103. ISSN 1384-6175.

Ledoux H and Meijers M (2009). Extruding building footprints to create topologically consistent 3D city models. In Krek A, Rumor M, Zlatanova S, and Fendel E, editors, *Urban and Regional Data Management, UDMS Annual 2009*, pages 39–48. CRC Press.

Lee J and Zlatanova S (2008). A 3D data model and topological analyses for emergency response in urban areas. In Zlatanova S and Li J, editors, *Geospatial Information Technology for Emergency Response*, ISPRS Book Series, pages 143–168. Taylor & Francis, London.

Longley PA, Goodchild MF, Maguire DJ, and Rhind DW (2001). *Geographic information systems and science*. Wiley, London.

Mäntylä M (1988). *An introduction to solid modeling*. Computer Science Press, New York, USA.

Miller GL, Talmor D, Teng SH, Walkington N, and Wang H (1996). Control volume meshes using sphere packing: Generation, refinement and coarsening. In *Proceedings 5th International Meshing Roundtable*, pages 47–61. Pittsburgh, USA.

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

Molenaar M (1998). *An introduction to the theory of spatial object modelling for GIS*. Taylor & Francis, London, UK.

Muller DE and Preparata FP (1978). Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217–236.

Neteler M and Mitasova H (2008). *Open Source GIS: A GRASS GIS Approach*. Springer.

OGC (2012). OGC city geography markup language (CityGML) encoding standard. Open Geospatial Consortium inc. Document 12-019, version 2.0.0.

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

Penninga F (2008). *3D Topography: A Simplicial Complex-based Solution in a Spatial DBMS*. Ph.D. thesis, Delft University of Technology, Delft, the Netherlands.

Pilouk M (1996). *Integrated modelling for 3D GIS*. Ph.D. thesis, ITC, The Netherlands.

Requicha AAG (1982). Representation of rigid solids—theory, methods and systems. *ACM Computing Surveys*, 12(4):437–464.

Rottensteiner F (2003). Automatic generation of high-quality building models from lidar data. *IEEE Computer Graphics and Applications*, 23(6):42–50.

Schulte C and Coors V (2008). Development of a CityGML ADE for dynamic 3D flood information. In *Proceedings Joint ISCRAM-CHINA and GI4DM Conference on Information Systems for Crisis Management*. Harbin, China.

Shewchuk JR (1997). *Delaunay Refinement Mesh Generation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA.

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

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

Theobald DM (2001). Topology revisited: Representing spatial relations. *International Journal of Geographical Information Science*, 15(8):689–705.

Tse ROC, Dakowicz M, Gold CM, and Kidner D (2005). Building reconstruction using LIDAR data. In *Proceedings 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS*, pages 156–161. Pontypridd, Wales, UK.

van Oosterom P, Quak W, and Tijssen T (2004). About invalid, valid and clean polygons. In Fisher PF, editor, *Developments in Spatial Data Handling—11th International Symposium on Spatial Data Handling*, pages 1–16. Springer.

van Oosterom P, Stoter J, Quak W, and Zlatanova S (2002). The balance between geometry and topology. In Richardson D and van Oosterom P, editors, *Advances in Spatial Data Handling—10th International Symposium on Spatial Data Handling*, pages 209–224. Springer.

Verbree E and Si H (2008). Validation and storage of polyhedra through constrained Delaunay tetrahedralization. In Cova TJ, Miller HJ, Beard K, Frank AU, and Goodchild MF, editors, *Proceedings 5th international conference on Geographic Information Science*, volume 5266 of *Lecture Notes in Computer Science*, pages 354–369. Springer.

Vosselman G and Dijkman S (2001). 3D building model reconstruction from point clouds and ground plans. In *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, volume XXXIV–3/W4, pages 37–43. Annapolis, USA.

Worboys MF and Duckham M (2004). *GIS: A computing perspective*. CRC Press, second edition edition.

Zhou G, Song C, Simmers J, and Cheng P (2004). Urban 3D GIS from LiDAR and digital aerial images. *Computers & Geosciences*, 30(4):345–353.

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