Watertight Scenes from Urban LiDAR and Planar Surfaces

Marc van Kreveld

Thijs van Lankveld

Remco Veltkamp

Technical Report UU-CS-2013-007
May 2013

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Watertight Scenes from Urban LiDAR and Planar Surfaces

Marc van Kreveld, Thijs van Lankveld, and Remco C. Veltkamp

Abstract

The demand for large geometric models is increasing, especially of urban environments. This has resulted in production of massive point cloud data from images or LiDAR. Visualization and further processing generally require a detailed, yet concise representation of the scene's surfaces. Related work generally either approximates the data with the risk of over-smoothing, or interpolates the data with excessive detail. Many surfaces in urban scenes can be modeled more concisely by planar approximations. In earlier work, we presented a method for efficiently computing planar polygons approximating a point set. Here, we present a method that combines these polygons into a watertight model. In regions where no polygons were detected, the shape is closed with free-form meshes based on visibility information. To achieve this, we divide 3-space into inside and outside volumes by combining a constrained Delaunay tetrahedralization with a graph-cut. We compare our method with related work on several large urban LiDAR data sets. We construct similar shapes with a third fewer triangles to model the scenes. Additionally, our results are more visually pleasing and closer to a human modeler's description of urban scenes using simple boxes.

Keywords: Geometric Algorithm; Computer Vision; Surface Representation; Visible Line/surface Algorithm; Watertight Geometry; Delaunay Tetrahedralization

1 Introduction

Public interest in virtual cityscapes has been increasing for some years. Groups like municipal government, architecture firms, and emergency response units can use these models for urban planning, visualization, and training. As a result, image-based point reconstruction and LiDAR sensing have received a wide backing, yielding massive data sets of high detail point clouds. However, these point data are not sufficient for most applications. Consequently, reconstructing surfaces from point data is an active field of research.

Many applications prefer the geometry to be non-self-intersecting, while applications like disaster management simulations may require the surfaces to be enclosing a volume as well, i.e. watertight. Watertight geometry divides 3-space into two distinct volumes, inside and outside the model. The surfaces of the model are exactly where these two volumes meet. In practice, we ignore the surfaces needed to close the volume from below, because we assume the scene is firmly attached to the ground.

Reconstructing watertight geometry has received considerable attention, as shown in Section 2. However, most related methods do not incorporate the fact that most surfaces in urban scenes are planar. This generally results in rounding of sharp corners or adding unnecessary complexity to the model. In Section 3, we present a novel method that constructs a concise watertight model using planar polygons approximating the local point cloud. We have evaluated our method on a number of urban LiDAR data sets and we describe these experiments in Sections 4 and 5. Finally, Section 6 presents a discussion of the broader implications of our method. Our key contributions are:

- A novel method that constructs watertight geometry from points and a soup of polygons. This geometry follows the polygons where possible and closes the object based on a line-of-sight based space carving.

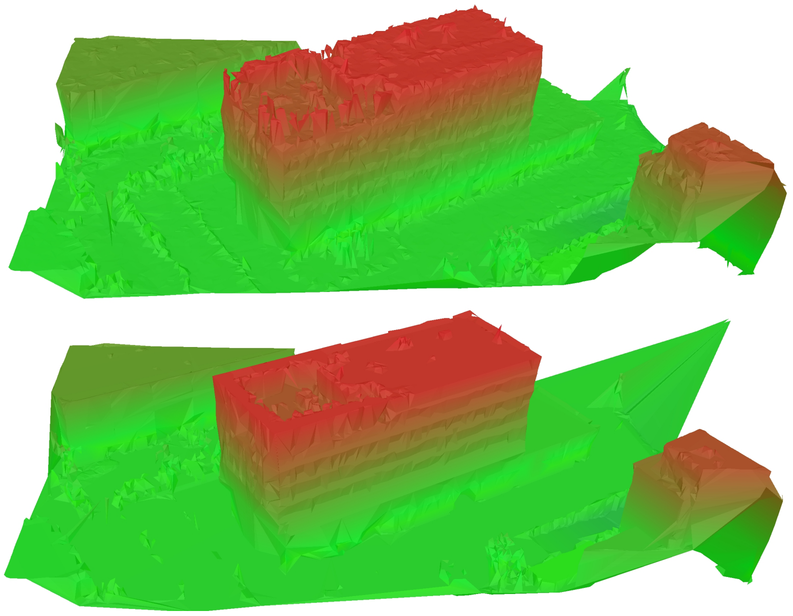
Figure 1: Watertight geometry from points (top) and surfaces (bottom)

- A novel method for constructing the constrained Delaunay tetrahedralization that avoids costly line-sphere intersection computations.
- A comparison of our method to the related method by Labatut *et al.* [20]. Our method produces concise and visually pleasing results using a third fewer triangles.

2 Related Work

For the purpose of constructing watertight geometry, implicit surface methods may be the most straightforward, because the implicit function already represents some notion of an inner and outer volume. Hoppe *et al.* [19] present an early method to explicitly construct the geometry of an implicit manifold. They first estimate the local surface near each data point using principal component analysis (PCA) on its k nearest neighbors. Then, they assign a consistent orientation to these local surfaces by traversing the minimal spanning tree of the Riemannian graph: a graph on the data points containing all edges $\overline{xy}$ where point y is in the k-neighborhood of x. From these oriented surfaces, their implicit surface function maps any point in space to the signed orthogonal distance from the closest surface. Finally, they construct the explicit surface using a variation on the marching cubes algorithm [2].

Many other methods use implicit surfaces in order to create watertight geometry. Mullen *et al.* [22] describe a method that improves the correctness of the local surface orientation in order to fill gaps in regions of missing data. Schnabel *et al.* [24] use implicit surfaces and combine this with graph-cuts to get a watertight model. They use an optimization scheme that iterates over multiple graph-cuts. Curless and Levoy [11] introduce the idea of using lines-of-sight between sensor and data point to indicate exterior regions during the implicit surface method. Similarly, Shalom *et al.* [27] use generalized cones with a data point at their apex to indicate exterior regions. Shen *et al.* [28] present a method for constructing implicit surfaces from a polygon soup. Alliez *et al.* [3] create an implicit surface using the anisotropy in Voronoi cells to estimate the normals of the surface.

Implicit surface methods have two inherent weaknesses: a) they approximate, rather than interpolate the point set and because of this b) they have problems reconstructing sharp features. Many implicit surface methods are also ill equipped to handle massive data sets.

There is also a wide range of methods that construct watertight geometry using the facets of the Delaunay tetrahedralization (DT). A survey of these Delaunay-based methods is given by Cazals and Giessen [8]. The basis for these methods is an observation by Boissonnat [6], which showed that a shape can be captured in the DT of the sample points. Dey and Goswami [12] provide a recent example by adjusting the Cocone method [5] to produce watertight models. Many of these Delaunay-based methods assume the object is r-sampled, e.g. [1], which requires the sampling to be sufficiently dense near important features. The assumption of r-sampling is too strong for urban reconstruction from LiDAR, because a data set may combine thin features and unpredictable sampling densities.

A recent approach by Labatut *et al.* [20] combines the DT, the minimal-weight graph-cut, and lines-of-sight in order to determine the regions inside and outside the objects. While this method produces nice results, it does not exploit the planarity in the scene. Chauve *et al.* [9] combine a partitioning of space into regions using planar primitives and a graph-cut to select interior and exterior regions. Although they achieve nice results, their method cannot reconstruct non-planar parts of a scene and lacks a theoretical basis.

Similar to [9, 20], our method partitions space and constructs watertight geometry by applying a graph-cut to this partitioning. We partition 3-space using an extension of the DT that can handle polygons, called the *conforming constrained Delaunay tetrahedralization* (CCDT) [29, 30, 32]. The CCDT is constructed from a collection of points and planar polygons: each point is contained as a vertex and each polygon is contained as a collection of triangular facets. This structure is described in Subsection 3.1.

Various methods can compute planar surfaces that locally approximate a point set, broadly divided into two approaches. Region growing methods start from a random point and grows a surface on the points as long as the surface is approximately planar. Once the surface can no longer be grown, a new surface grown from another location. Unfortunately, this approach cannot guarantee global planarity of the surfaces. RANSAC methods collect a random sample of three points and counts how many points lie in the plane through this sample. This is repeated until the chance of finding a plane containing more points is minimal.

While various methods exist to estimate planar or near-planar approximations of a subset of a point cloud [33, 25, 34], they generally do not bound these approximations based on the point cloud. Recent work by Van Kreveld *et al.* [35] uses Efficient RANSAC [25] to estimate planar approximations and an adaptation of the α-shape [14] to bound the shape such that sharp features are preserved. We use these polygons as input to our method.

3 Method

The goal of our method is to construct a watertight geometric model from a collection of planar polygons. This model should contain parts of these polygons where this is useful for separating the inner and outer volumes. In regions without polygons, an appropriate boundary is estimated from the point set. Our method achieves this in two steps, similar to Labatut *et al.* [20]. First we partition the complete space into many small regions and then we determine for each region whether it is inside or outside the model. Unlike [20], both of these steps are guided by the polygons.

We use an extension of the DT, called the conforming constrained Delaunay tetrahedralization (CCDT) to partition 3-dimensional space into tetrahedral cells. We describe the CCDT and how we construct it in Subsection 3.1. If all the cells are assigned to either the inner or outer volume, the facets separating inner and outer cells comprise a watertight model. The CCDT embeds input polygons in its facets, enabling partitioning along these surfaces.

We classify the cells of the CCDT into inner and outer cells using a minimum-weight graph-cut. This method takes a weighted graph that contains a source and a sink node and collects a number of edges to remove, the *cut*, such that the source and sink are no longer connected by a path on edges of the graph. The cut is chosen such that the sum of the weights of its edges is minimized. Subsection 3.2 describes the graph-cut method in more detail.

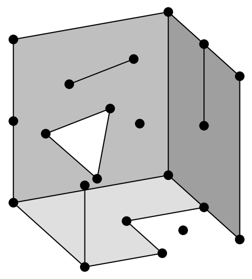

Figure 2: A piecewise linear complex.

We construct the graph to cut from the dual of the CCDT, which is similar to a Voronoi diagram. The graph-cut will correspond exactly to the facets of the CCDT separating outer and inner cells. This collection of facets constitutes a watertight model. It may be obvious that the shape of the model reconstructed by this method depends on the shape and location of the surfaces embedded in the CCDT. If a surface is not present in the CCDT, it cannot be selected by the graph-cut. Therefore, the geometry of the CCDT has a major influence on the geometry of the reconstruction.

However, there are two other considerations important to the cut. Firstly, a graph-cut only makes sense if there are a source and sink node. The connectivity between source and sink node is very important, because only edges on a path between source and sink are cut. We add two additional, abstract, nodes to the dual of the CCDT to act as source and sink. These two nodes are connected automatically to the other nodes in a careful fashion as described in Subsection 3.2.1.

Secondly, the weights of the edges in the graph determine the optimal positions to cut. We compute the weight of each edge based on four properties of its corresponding facet in the CCDT. The first three properties are also used by Labatut *et al.* [20], the last is novel to our method. Firstly, the laser does not penetrate solid surfaces, so facets intersected by the line from sensor to measurement point produce a higher weight. Secondly, the measurement locations indicate a reflection point on a surface, so the facets near points produce a lower weight. Thirdly, the regions separated by the surface should be largely empty of measurement points, so facets separating two cells with large circumspheres produce a lower weight. Finally, the reconstructed polygons indicate simpler representations of the surfaces, so the facets in a polygon produce a lower weight. The first two factors are described in more detail in Subsection 3.2.1 and the last two in Subsection 3.2.2.

3.1 Conforming Constrained Delaunay Tetrahedralization

In order to subdivide 3-space into regions that can be inside or outside the model, we use the conforming constrained Delaunay tetrahedralization (CCDT). This structure is closely related to the well-known 3-dimensional Delaunay tetrahedralization (DT). In order to properly describe the CCDT, we must first describe the piecewise linear complex (PLC) [21]. A PLC is a collection of points, straight line-segments, and planar straight-line polygons in 3-space, as shown in Figure 2. The polygons can be non-convex with any number of boundary segments and they may contain holes, interior segments, and isolated points. However, their boundary cannot be self-intersecting and all their vertices must be coplanar.

As its name implies, a PLC is a complex: the collection is both closed and non-intersecting. By closed we mean that the collection contains all faces of each of its elements. In other words, if a PLC contains a polygon, then it must also contain all the edges and vertices of the polygon. By non-intersecting we mean that the collection does not contain two elements that pairwise intersect in their interiors. That is, no two elements, whether polygon, segment, or point, may intersect except in the union of their shared vertices and edges.

The CCDT is based on the *constrained Delaunay tetrahedralization* (CDT). Some of the facets of this tetrahedralization are marked as *constrained* and the tetrahedrons are Delaunay with the exception that their circumscribing ball may contain points on the other side of a constrained

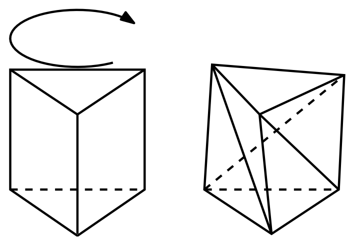

Figure 3: A prism and Schönhardt's polyhedron. The creases created by twisting the upper triangle are concave.

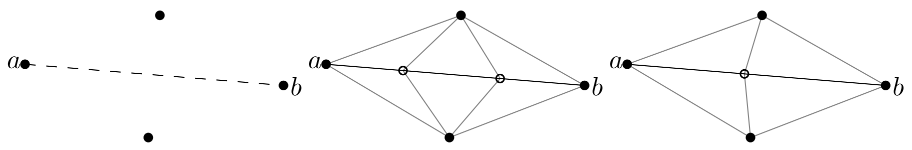

Figure 4: A configuration of points and two different configurations of Steiner points that force $\overline{ab}$ to appear in the Delaunay triangulation.

facet as seen from the interior of the tetrahedron. A CDT *embeds* a PLC if it contains all the points and segments of the PLC and each polygon of the PLC is exactly covered by a collection of constrained facets.

Shewchuk [29] showed that there are PLCs that do not admit an embedding CDT. Schönhardt's polyhedron [26] is a simple example, as shown in Figure 3. No tetrahedralization on the vertices of this polyhedron exists for which all the facets of the polyhedron are embedded.

Shewchuk goes on to prove that for any PLC X we can construct another PLC $\bar{X}$ that can be embedded in a CDT. This PLC $\bar{X}$ exactly covers X, but it contains additional points, called *Steiner points*, on the segments of X. These points are placed such that the DT of $\bar{X}$ contains edges that cover the segments of X. We call these edges *conforming* to the segments of X. Unlike Shewchuk, we prefer to call the CDT of $\bar{X}$ the conforming constrained Delaunay tetrahedralization of X, to indicate the similarity to the 2-dimensional conforming Delaunay triangulation.

3.1.1 Embedding Segments

There are many different methods for determining the positions of the Steiner points that ensure that the segments of $\bar{X}$ are present in the DT. Figure 4 gives an example PLC and two different configurations of Steiner points that ensure the embedding of the segment $\overline{ab}$.

There are also various ways of comparing the quality of configurations, e.g. by counting the number of Steiner points [32] or by bounding the minimal edge length [30]. Related work generally constructs Steiner points at the intersections of a sphere and a line, which is in practice either dangerous because of round-off errors or expensive to compute exactly. For this reason, we present a novel Steiner point insertion method, similar to [32], that does not require computing sphere-line intersections.

A recurring concept in methods for constructing conforming edges is the *protecting ball* of a point. The purpose of a protecting ball is to indicate a region around an input point where no Steiner point may be placed. Let us consider the balls $\mathcal{B}_p^q$ centered on p and touching a vertex or edge q of X in their boundary. The ball $\mathcal{B}_p$ is the smallest of these balls $\mathcal{B}_p^q$. The protecting ball $\mathcal{P}_p$ of a point p is an open ball centered at p no larger than $\mathcal{B}_p$. The radius of each $\mathcal{P}_p$ is generally chosen based on the local distribution of points and conforming edges. Let us assume for now that $\mathcal{P}_p = \mathcal{B}_p$.

These protecting balls are specifically important for *acute* points: points where at least two conforming edges meet at an acute angle. The insertion of a Steiner point on one edge may cause another conforming edge e to disappear from the DT, forcing the insertion of another Steiner point

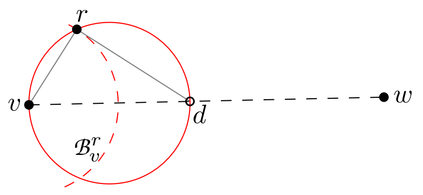

Figure 5: A protecting ball. The black disks are points of the PLC, the dashed black line is a segment of the PLC. The dashed red arc shows $\mathcal{B}_v^r$, the solid red circle is the largest empty ball with its diameter $\overline{vd}$ on $\overline{vw}$.

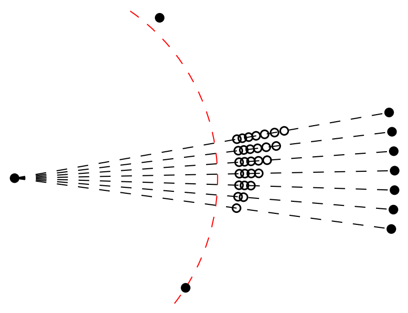

Figure 6: A pathological case when embedding conforming segments top to bottom between points (black disks). Steiner points (circles) are inserted at the intersection with the largest empty $\mathcal{D}_{\overline{vd}}$. Each Steiner point insertion invalidates all earlier conforming edges, leading to repeated Steiner point insertions.

to restore e. When Steiner points can be inserted arbitrarily close to an acute point, this process may cascade into an infinite loop.

In order to embed a conforming segment $\overline{vw}$ not present in the DT, we need to insert Steiner points. Let us consider the Steiner point s whose insertion results in the appearance of $\overline{vs}$ in the DT. This Steiner point may not be placed inside $\mathcal{P}_v$. However, in order for $\overline{vs}$ to appear in the DT, there must be a sphere through v and s with interior void of points. Most related methods will place s on the boundary of $\mathcal{P}_v$. Because $\mathcal{P}_v$ cannot contain any point, the open ball $\mathcal{D}_{\overline{vs}}$ with diameter $\overline{vs}$ must be empty. Therefore s meets both criteria: it is outside $\mathcal{P}_v$ and shares the boundary of an empty ball with v.

Instead of computing the intersection of the boundary of $\mathcal{P}_v$ and $\overline{vw}$, we will work directly with the empty open balls incident to v that have their diameter on $\overline{vw}$, as shown in Figure 5. Specifically, we choose the largest empty open ball $\mathcal{D}_{\overline{vd}}$. This ball must have a point r on its boundary and it is easy to see that $\|vr\| < \|vd\|$. Note that r may be a point of X or a Steiner point inserted earlier.

Because d is not inside $\mathcal{B}_v^r$ and because all Steiner points are constructed on the segments of X, d must be outside $\mathcal{P}_v$. Because $\mathcal{D}_{\overline{vd}}$ is empty, inserting d as Steiner point would result in the $\overline{vd}$ appearing in the DT. Additionally, v and r must be adjacent in the DT as witnessed by $\mathcal{D}_{\overline{vd}}$. Finally, according to Thales' Theorem we can compute d as the intersection of $\overline{vw}$ and the plane through r orthogonal to $\overline{vr}$, removing the need for a line-sphere intersection.

Unfortunately, inserting a Steiner point at d has two disadvantages. Firstly, if we want to be able to insert conforming edges incrementally, this method may result in an extreme number of Steiner points, as shown in Figure 6. Secondly, if there are two other points x, y cospherical with v, r, d, $\overline{vd}$ may not appear in the DT because the tetrahedra on v, r, x, y and d, r, x, y have an empty circumsphere. The free choice of which tetrahedra to use may not result in $\overline{vd}$ appearing.

In order to overcome these disadvantages, we want to insert the Steiner point at another location s on $\overline{vw}$ such that $\|vr\| \leq \|vs\| < \|vd\|$. We choose as s the midpoint m of d and the projection p of r onto $\overline{vw}$, as shown in Figure 7 and Algorithm 1. We refer to this procedure as

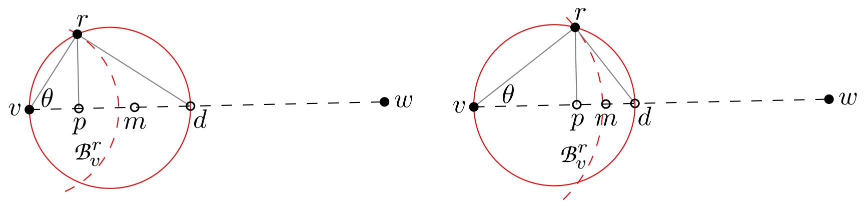

Figure 7: Two protecting balls. The black disks are vertices of the PLC, the dashed black line is a segment of the PLC, and the black circles indicate other significant locations. The dashed red arc shows $\mathcal{B}_v^r$, the solid red circle is the largest empty ball with its diameter $\overline{vd}$ on $\overline{vw}$, p is the projection of r onto $\overline{vw}$, and m is the mid-point of p and d.

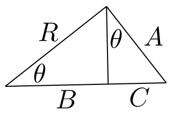

Figure 8: The setup for the proof of Lemma 1.

the *protection method.*

Algorithm 1 Compute the position of a new Steiner point

Input: two vertices v, w.

Output: the Steiner point inserted on $\overline{vw}$ to protect v

```
1:  procedure PROTECT($v, w$)
2:      $x \leftarrow w$
3:      $y \leftarrow w$
4:      $E \leftarrow$ edges incident to $v$
5:      for all $\overline{vv_i} \in E$ do
6:          if $\angle wvv_i < \frac{\pi}{2}$ then
7:              $P \leftarrow$ plane through $v_i$ orthogonal to $\overline{vv_i}$
8:              $d \leftarrow \overline{vw} \cap P$
9:              if $\|vd\| < \|vy\|$ then
10:                 $x \leftarrow$ projection of $v_i$ on $\overline{vw}$
11:                 $y \leftarrow d$
12:             end if
13:         end if
14:     end for
15:     return MIDPOINT($x, y$)
16: end procedure
```

It is straightforward to see that $\|vp\| < \|vm\| < \|vd\|$ and $\|vp\| < \|vr\|$. However, the difference between $\|rv\|$ and $\|mv\|$ depends on the angle $\theta = \angle rvw$, as shown in Figure 7. The following lemma shows that for any θ, $\|vr\| < \|vm\|$ and therefore m must lie strictly outside the protecting ball and strictly inside $\mathcal{D}_{\overline{vd}}$. This means that $\overline{vm}$ must appear in the DT.

Lemma 1. *Given the triangle vrd where r lies on a corner with right angle, let p be the orthogonal projection of r onto $\overline{vd}$, let m be the midpoint between p and d, and let $\theta = \angle rvd$, as shown in Figure 7. If $\theta \neq 0$ then $\|vr\| < \|vm\|$.*

Proof. For readability's sake, let us rename the distance involved as follows: $R = \|vr\|$, $A = \|rd\|$, $B = \|vp\|$, $C = \|pd\|$, $D = \|rp\|$, as shown in Figure 8. Observe that $\angle prd = \theta$, because of triangle similarity. This proof builds on the following trigonometric functions:

$$B = R\cos\theta \tag{1}$$
$$D = R\sin\theta \tag{2}$$
$$D = A\cos\theta \tag{3}$$
$$C = A\sin\theta \tag{4}$$

We will show that the following relation between the distances $\frac{R-B}{C} < \frac{1}{2}$ holds for any $\theta \neq 0$. In order to achieve this result, we will apply the above equalities in order.

$$\begin{aligned}\frac{R-B}{C} &= \frac{R - R\cos\theta}{C} = \frac{R(1-\cos\theta)}{C} \\ &= \frac{D(1-\cos\theta)}{C\sin\theta} \\ &= \frac{A\cos\theta(1-\cos\theta)}{C\sin\theta} \\ &= \frac{A\cos\theta(1-\cos\theta)}{A\sin^2\theta} = \frac{\cos\theta - \cos^2\theta}{\sin^2\theta} \\ &< \frac{1}{2}\end{aligned}$$

The final inequality follows from evaluating this function at $\lim\limits_{\theta\to 0}$, where it reaches its maximum because of the similarity to $\tan(\theta)^{-2}$. The function cannot be evaluated at $\theta = 0$, but in this case $r = d$ and $\overline{vr}$ already exists in the DT. $\square$

The protection method automatically constructs a protecting region around the input vertices based on the local point distribution. This region is roughly ball-shaped with dents at nearby vertices of X. There are two remaining issues when conforming a DT to a PLC X. Firstly, the parts of the segments of X between these protecting regions also need to be embedded in the DT. Secondly, it may occur that a segment $\overline{vw}$ of X is not embedded in the DT if the balls $\mathcal{B}_v$ and $\mathcal{B}_w$ intersect. In order to conform the DT to $\overline{vw}$, a Steiner point must be constructed inside either $\mathcal{B}_v$ or $\mathcal{B}_w$.

The parts of the segments of X between protecting regions are easily embedded by recursively applying the protection method for the newly inserted Steiner points. Starting from a segment $\overline{vw}$, Steiner points s_v, s_w are constructed such that the edges $\overline{vs_v}, \overline{ws_w}$ appear in the DT. Then, the segment $\overline{s_v s_w}$ is embedded similarly. Note that no Steiner point s inserted on $\overline{s_v s_w}$ can result in removing $\overline{vs_v}$ or $\overline{ws_w}$ from the DT, because s must be placed outside the protecting regions of v and w.

When two points v, w lie sufficiently close together and $\overline{vw}$ is not present in the DT, it may be necessary to insert a Steiner point inside $\mathcal{B}_v$ or $\mathcal{B}_w$. Related methods achieve this by choosing the protecting balls $\mathcal{P}_v$ and $\mathcal{P}_w$ smaller than $\mathcal{B}_v$ and $\mathcal{B}_w$ [30, 32]. Similarly, we will construct a Steiner point s such that $\overline{vs}$ and $\overline{sw}$ appear in the DT, while at the same time forcing $\mathcal{B}_v$ or $\mathcal{B}_w$ to shrink. We must choose s with caution in order to make sure no $\mathcal{B}_v$ can shrink too much.

We will distinguish three different cases based on the vertices v, w and the Steiner points s_v, s_w that would be constructed to protect them. Case i) occurs when the segments $\overline{vs_v}$ and $\overline{ws_w}$ do not overlap. In this case we compute the Steiner points for both ends using our protection method. Note that if $\mathcal{B}_v$ and $\mathcal{B}_w$ overlap, then $\overline{vs_v}$ and $\overline{ws_w}$ must overlap.

Case ii) occurs when $\overline{vs_v}$ and $\overline{ws_w}$ overlap and one of the points v, w is acute and the other is not. Note that Steiner points are never acute. Let us assume without loss of generality that the acute point is v. We insert only the Steiner point s_v into the PLC. This does not reduce the size of $\mathcal{B}_v$, and because w is not acute, none of the conforming edges incident to w can be removed by this insertion.

Case iii) covers all remaining configurations of v and w where $\overline{vs_v}$ and $\overline{ws_w}$ overlap. Again, we will insert one Steiner point s inside $\mathcal{B}_v$ or $\mathcal{B}_w$. We base s on s_v, s_w, and the midpoint m of

v, w. Let us assume without loss of generality that $\|vs_v\| \leq \|ws_w\|$. If $\|vs_v\| < \|vm\|$ then $s = s_v$; otherwise $s = m$. After the insertion of s, the radius of both $\mathcal{B}_v$ and $\mathcal{B}_w$ either remains the same or is at least $\frac{\|vw\|}{2}$. The pseudo-code of the complete method is shown in Algorithm 2.

Algorithm 2 Embed a conforming edge

Input: two vertices v, w.

```
 1: procedure CONFORM($v, w$)
 2:     if exists($\overline{vw}$) then
 3:         return
 4:     end if
 5:     $s_v \leftarrow$PROTECT($v, w$)
 6:     $s_w \leftarrow$PROTECT($w, v$)
 7:     if $\|vs_v\| + \|ws_w\| > \|vw\|$ then                          ▷ Case i)
 8:         insert $s_v$
 9:         insert $s_w$
10:         CONFORM($s_v, s_w$)
11:     else if acute($v$) and $\neg$acute($w$) then                  ▷ Case ii)
12:         insert $s_v$
13:         CONFORM($s_v, w$)
14:     else if acute($w$) and $\neg$acute($v$) then                  ▷ Case ii)
15:         insert $s_w$
16:         CONFORM($v, s_w$)
17:     else                                                          ▷ Case iii)
18:         $m \leftarrow$MIDPOINT($v, w$)
19:         if $\|vs_v\| < \|ws_w\|$ and $\|vs_v\| < \|vm\|$ then
20:             insert $s_v$
21:             CONFORM($s_v, w$)
22:         else if $\|ws_w\| < \|vs_v\|$ and $\|ws_w\| < \|wm\|$ then
23:             insert $s_w$
24:             CONFORM($v, s_w$)
25:         else
26:             insert $m$
27:         end if
28:     end if
29:     re-embed all conforming edges removed by this call
30: end procedure
```

So far we have mainly considered inserting a single conforming segment. However, as Figure 6 showed, complications may occur when inserting multiple conforming edges. Figure 9 shows the Steiner points constructed by our method. There are still point sets for which our protection method places a Steiner point such that an earlier conforming edge incident to an acute vertex is removed from the DT. Figure 10 shows the results of our embedding method on a difficult PLC.

It remains to show that our method always terminates. Related methods generally prove this using the *local feature size* (lfs) [23]. Given a PLC X and a point p in 3-space, *lfs*(p) is the radius of the smallest ball $\mathcal{B}$ centered on p such that $\mathcal{B}$ intersects two vertices and/or segments of X that do not intersect each other. Note that this measure is independent of Steiner points. Termination can then be proven by showing that there is some constant c such that after any Steiner point insertion $c\|e\| \geq \text{lfs}(p)$ holds for any edge e and any point p on e.

However, many of these proofs incorrectly discount the influence of Steiner points. Consider the example of applying Shewchuk [30] to a problematic case shown in Figure 11. The gray disks show the shortest edge lengths and four times that distance. Recall that for lfs(p) both Steiner points and intersecting segments of the PLC are ignored. This means that for most points x inside

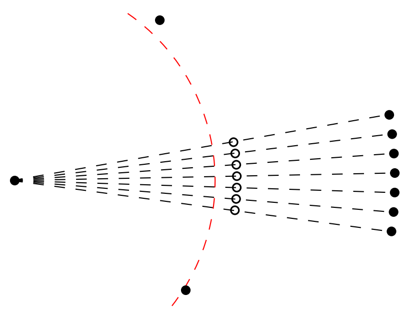

Figure 9: The pathological case shown in Figure 6 when inserting Steiner points using our protection method.

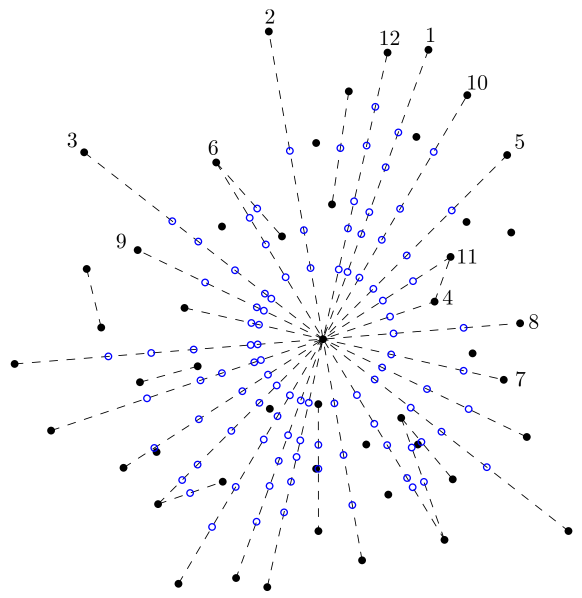

Figure 10: A pathological case. The black disks are points of the PLC, the dashed black lines are the segments of the PLC, and the blue circles make a collection of Steiner points that embed the segments in the DT. The numbers indicate the order in which the segments are embedded; inserting the segments in a different order may result in a different set of conforming edges.

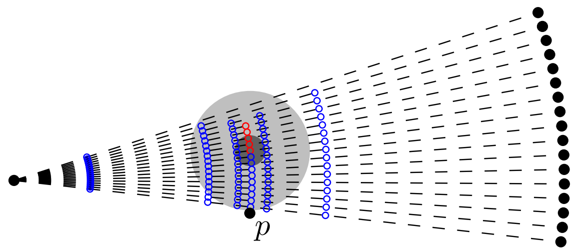

Figure 11: Shewchuk's edge protection applied to a problem case. The PLC contains the black disks and dashed segments, the small circles show the Steiner points constructed to embed the segments. The red Steiner points are incident to two edges that are too short, the gray disks show the edge length and four times this distance disproving his Theorem 2.

the wedge of conforming segments, $\mathrm{lfs}(x)$ is the distance to p. For this reason, we define the *star local feature size* ($\mathrm{lfs}^\star$) for subspaces.

Definition 1. *Given a fixed PLC X and a subspace S, the star local feature size $lfs^\star_X(S)$, or simply $lfs^\star(S)$, of S is the radius of the smallest ball centered within S that contains two points or segments of X that do not intersect in S.*

We are mainly interested in the cases where the subspace is an edge on a segment of X. Note that for edges incident to an acute vertex, $\mathrm{lfs}^\star$ is the minimum lfs over the points on the edge. For other edges, all segments except its supporting segment influence $\mathrm{lfs}^\star$.

Even with this definition, we cannot bound the minimum edge length for our method: case i) can produce arbitrarily short edges when s_v and s_w lie very close to the boundary of the ball. Similarly, in case ii) s_v can lie arbitrarily close to w. Nonetheless, the following three lemmas prove our method always terminates. Their proofs are given in the appendix.

Lemma 2. *Given a PLC X and a segment $\overline{vw}$ of X not embedded in the DT, let us apply the protection method to conform $\overline{vw}$ without re-embedding edges on other segments that were destroyed. This operation can add only a number of Steiner points linear in the number of vertices of the DT.*

Proof. Any time a Steiner point is placed on any segment $\overline{vw}$ by the protection method, this creates a segment $\overline{vs}$ based on another point r. As witnessed by the emptiness of the diametral ball touching r, $\overline{vs}$ cannot be destroyed while conforming $\overline{sw}$. It is also straightforward to see that r cannot be touched by any ball with part of $\overline{sw}$ as diameter.

This means that in case i) two points can no longer be inside $\mathcal{D}_{\overline{s_v s_w}}$. In case ii) there is one point that can no longer be in $\mathcal{D}_{\overline{s_v w}}$. In case iii) there are two options: either we insert s_v and one point can no longer be in $\mathcal{D}_{\overline{s_v w}}$, or we insert m. We only insert m if it lies on both $\overline{vs_v}$ and $\overline{ws_w}$, meaning that both $\mathcal{D}_{\overline{vm}}$ and $\mathcal{D}_{\overline{mw}}$. must be empty.

Each operation must either terminate the algorithm, or both construct a segment that will not be destroyed and remove a least one point from the collection of points encroaching upon the remaining part of the segment. This collection starts with a linear number of points and once it is empty the segment must exist in the DT. □

Lemma 3. *Given a PLC X and a collection of Steiner points S constructed using the protection method, there cannot be a point $s \in S$ and an acute vertex v such that $\|sv\| < \frac{1}{2}lfs(v)$.*

Proof. We will prove this by contradiction. Let us assume there is a Steiner point s and an acute vertex v such that $\|sv\| < \frac{1}{2}\mathrm{lfs}(v)$ and let us consider the different cases in which s could have been constructed.

First let us consider the case where s does not lie on a segment incident to v. By the definition of the local feature size, $\|sv\| \geq \frac{1}{2}\mathrm{lfs}(v)$.

If s lies on a segment $\overline{vw}$ incident to v, it could only have been constructed when conforming this segment. It could have been constructed to protect either v, w, or any Steiner point on $\overline{vw}$ in any of the three cases. If there is a Steiner point s' between v and s, let us consider this point instead, because $\|vs'\| < \|vs\|$.

Recall that according to Lemma 1, from any Steiner point s constructed by our protection method on any segment $\overline{vw}$ from point w and r, $\|vs\| > \|vr\|$. This means that s must lie outside $\mathcal{B}_v^r$. Either r is a Steiner point, or r is a point of X and it is straightforward that $\|vs\| > \|vr\| \geq \text{lfs}(v)$. If r is a Steiner point, the same reasoning can be applied to find the point r' used to construct r. Because each next point must be closer to v, at some point we will find a non-Steiner point r_i' such that $\|vs\| > \|vr_i'\| \geq \text{lfs}(v)$.

This means that in case i) and ii), no Steiner point can be constructed closer to v than $\text{lfs}(v)$. In case iii), either v is not acute, or w is also acute. By our assumption, w must be acute and therefore cannot be a Steiner point. in this case, s must have been constructed such that $\|vs\| \geq \mathcal{B}_v$ and $\|vs\| \geq \frac{\|vw\|}{2}$. In either case, $\|vs\| \geq \frac{1}{2}\text{lfs}(v)$.

All possible cases result in a distance $\|vs\| \geq \frac{1}{2}\text{lfs}(v)$. As this contradiction holds for any combination of v and s, our proof is complete. □

Lemma 4. *Given a PLC X and a collection of Steiner points S constructed using the protection method, any edge e embedding a segment of X and not incident to an acute point can only be removed from the DT if lfs$^\star(e) < \|e\|$.*

Proof. This proof follows directly from three facts: 1) e can only be removed if there is a Steiner point s constructed inside the ball with e as diameter, 2) by its construction s must lie on a segment t of X, and 3) t cannot intersect e, because e is not incident to an acute point. □

Theorem 5. *Out protection method always terminates.*

Proof. By Lemma 3, edges incident to an acute point have a minimum edge length. By Lemmas 2 and 4 each time an edge is embedded, this adds a finite number of edges and only destroys conforming edges longer than their $\text{lfs}^\star$. □

If we know all segments of the PLC X in advance, we can choose the embedding order such that the occurrence of these short edges is minimized. To achieve this, the Steiner points should be inserted in a strict order. When inserting the next Steiner point s, the segment $\overline{vw}$ and its endpoint v should be chosen such that the length of the conforming edge $\overline{vs}$ resulting from the protection method to v is minimized over all segments and endpoints. In practice, this minimizes the number of changes in $\mathcal{B}_v$ for all point v of X.

3.1.2 Embedding Polygons

After conforming the DT to the segments of the PLC, the interiors of the polygons can be inserted incrementally. Shewchuk [31] describes a method for recovering the polygons using an ordered sequence of bistellar flips. Unfortunately, this method seems numerically unstable. We incrementally embed the interiors of the polygons using a method that changes larger collections of cells simultaneously very similar to [32].

Like Si and Gärtner, we insert each polygon $\mathcal{P}$ per *cavity*. A cavity is a maximal facet-connected collection of cells intersected by $\mathcal{P}$. Shewchuk [31] showed that these cavities contain exactly the cells that need to change to embed the polygon. For each cavity $\mathcal{C}$, we construct two new tetrahedralizations T_a, T_b: one using the vertices of $\mathcal{C}$ on or above $\mathcal{P}$ and the other using the vertices on or below $\mathcal{P}$. When we replace the cells of $\mathcal{C}$ by the overlapping cells of T_a, T_b, the part of $\mathcal{P}$ inside $\mathcal{C}$ appears, because this part can be built from facets on the convex hull of both T_a and T_b.

We must take special caution when embedding polygons near facets constrained earlier. There are simple configurations of points and constraints such that some facets on the boundary of $\mathcal{C}$ are not contained in $T = T_a \cup T_b$, as shown in Figures 12 and 13. When boundary facets are missing from T, the cavity cannot be neatly filled with cells of T.

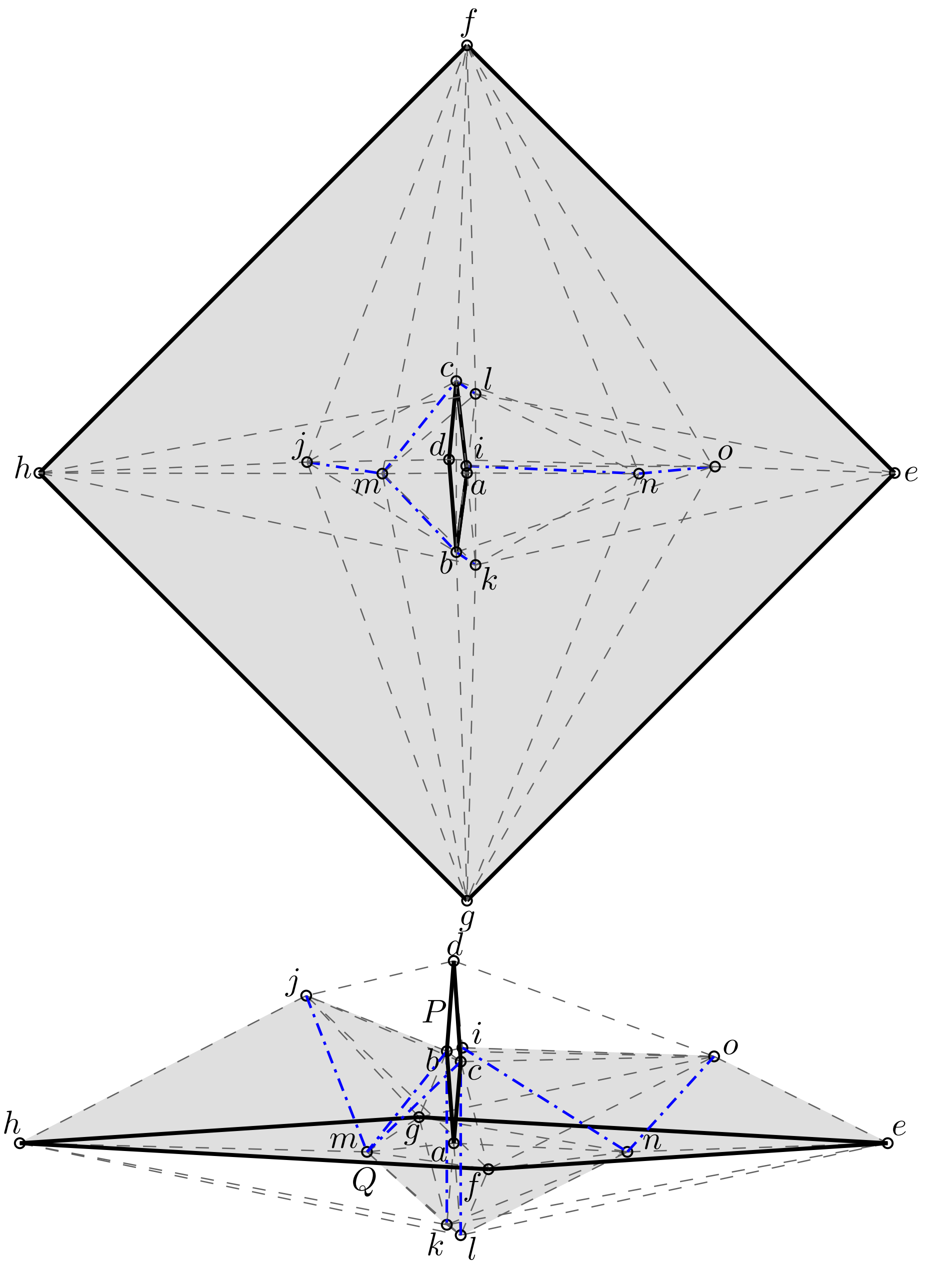

Figure 12: A small example where a partial tetrahedralization of a cavity cannot be glued along the boundary. This figure shows the CCDT of a point set in which polygon $P = abdc$ is embedded and polygon $\mathcal{Q} = efhg$ should be embedded. Note that n, m lie just below $\mathcal{Q}$ and the blue dash-dotted edges intersect $\mathcal{Q}$. Because the cavity $\mathcal{C}$ (gray region) contains exactly the cells incident to these edges, it does not contain d or the cell $abci$. Figure 13 shows the CCDT of the upper half of the cavity

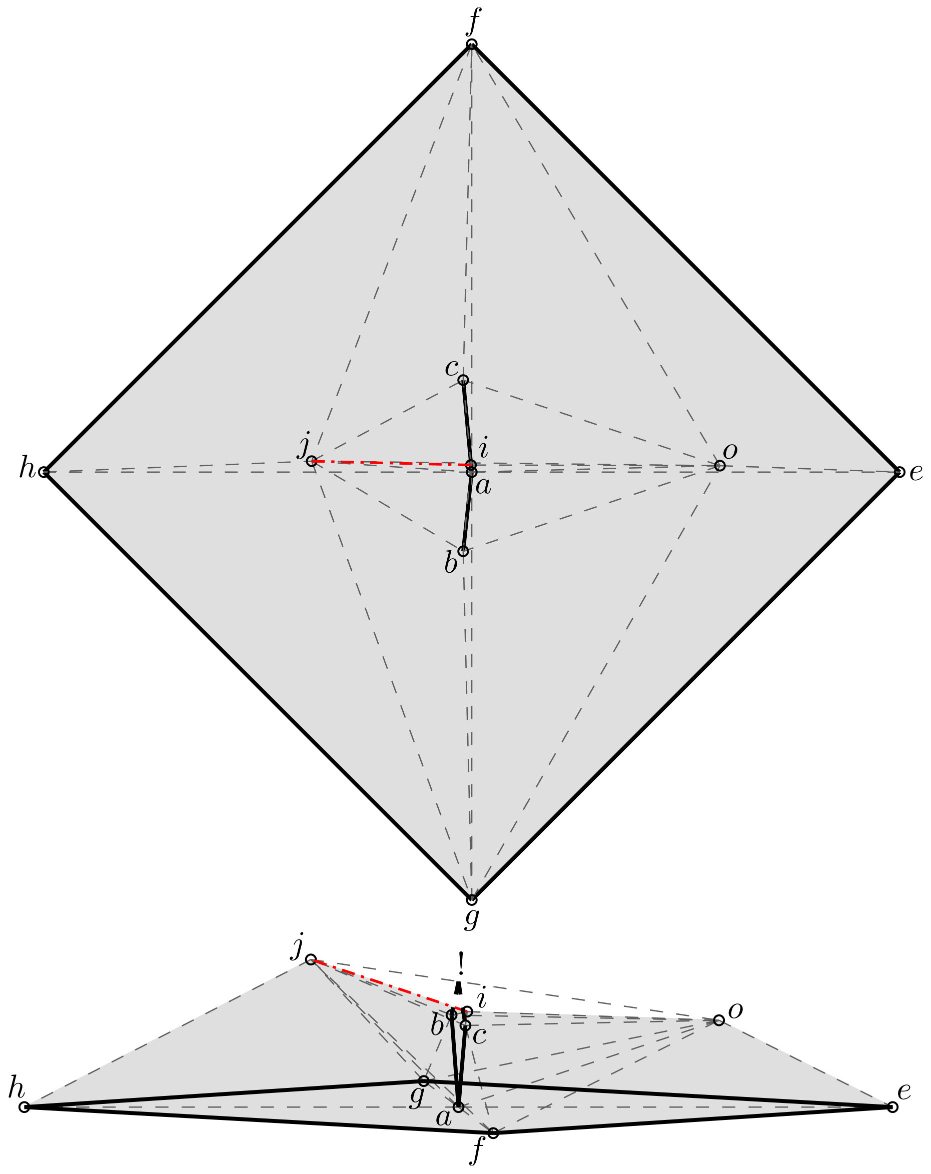

Figure 13: A small example where a partial tetrahedralization of a cavity cannot be glued along the boundary. This figure shows the CCDT T_a of the points of the cavity on or above $\mathcal{Q}$. Note that because T_a does not contain d, three problems occur: a) the facets abc and bcj on the boundary of $\mathcal{C}$ are not in T_a, b) the red edge $\overline{ij}$ intersects P, and c) P cannot be embedded in T_a, because there is no loop of coplanar edges.

As the figure shows, this problem can occur in cavities with a small number of points. However, there are a few conditions necessary to cause the problem. Most importantly, there must be an existing embedded polygon $\mathcal{P}$ inside the cavity or on its boundary, with a point d that is not part of the cavity. There must also be two points i, j on opposite sides of $\mathcal{P}$ for which the absence of d allows a Delaunay edge that intersects $\mathcal{P}$. Finally, i, j must be part of the DT T_a used to fill the cavity on the same side as $\mathcal{P}$. When these conditions are met it may occur that $\mathcal{P}$ cannot be embedded in T_a and some of the boundary facets of the cavity are not in T_a. While these conditions may seem restrictive, the configuration may occur in realistic data sets.

We overcome this problem similar to Si and Gärtner [32]. Before computing T_a, T_b, we grow the cavity until all its boundary facets must be contained in T. We do this by checking for each facet on the boundary of $\mathcal{C}$ whether it has a circumscribing ball that does not contain any point of $\mathcal{C}$. Note that this is a weak Delaunay criterion and unlike [32] it is restricted to $\mathcal{C}$. For each facet f not meeting this criterion, we grow $\mathcal{C}$ by adding the cell on the other side of f. We repeat this procedure until all boundary facets meet the criterion. In extreme cases this may force the cavity to grow to the complete CCDT, but in practice most cavities will not grow significantly.

If a cavity $\mathcal{C}$ of a polygon $\mathcal{P}$ to be embedded contains part $\hat{\mathcal{Q}}$ of a polygon embedded earlier, $\hat{\mathcal{Q}}$ should still be embedded after embedding $\mathcal{P}$. In order to retain the embedding of earlier polygons, we embed $\hat{\mathcal{Q}}$ into T_a or T_b, depending on the location of $\hat{\mathcal{Q}}$. The following lemma shows that embedding $\hat{\mathcal{Q}}$ is a strictly smaller problem than embedding $\mathcal{P}$ and so this recursion is finite. The proof is given in the appendix.

Lemma 6. *Given a cavity $\mathcal{C}$ in CCDT T formed while embedding polygon $\mathcal{P}$ and a polygon $\mathcal{Q}$ intersecting $\mathcal{C}$ in $\mathcal{Q}_c$, let T_p be the new CCDT formed by embedding $\mathcal{P}$ without enforcing $\mathcal{Q}_c$ to be embedded. The cavity of $\mathcal{Q}_c$ in T_c is a strict subspace of $\mathcal{C}$.*

Proof. We will build this proof in three steps. First, we will show that $\mathcal{Q}_c$ is bounded by a loop of coplanar edges of T_c. Then, we will show that the initial cavity of $\mathcal{Q}_c$, before growing it, is a strict subspace of $\mathcal{C}$. Finally, we will show that this cavity can neither grow outside $\mathcal{C}$ nor to the other side of $\mathcal{P}$.

In order to show that $\mathcal{Q}_c$ is bounded by a loop of coplanar edges of T_c, we will look at the different ways in which $\mathcal{Q}_c$ can be bounded. Let us denote the boundary of X by ∂X and recall that $\mathcal{Q}_c$ is $\mathcal{Q} \cap T_c$. We may divide $\partial \mathcal{Q}_c$ into the part ∂_q of $\partial \mathcal{Q}$ inside T_c and the part ∂_t of ∂T_c intersection $\mathcal{Q}$. Because $\mathcal{Q}$ is embedded in T, ∂_q must be covered by conforming edges and these must also exist in T_c. We may further divide ∂_t into the part ∂_p on $\mathcal{P}$ and the part ∂_c on $\partial \mathcal{C}$. By the PLC criteria, ∂_p must also be covered by conforming edges.

Finally, we show that ∂_c must also be covered by edges in T_c. Let us assume by contradiction that there is a part of ∂_c that is not covered by edges of T_c. This part must then pass through the interior of facets of $\mathcal{C}$. Let us look at an arbitrary facet f of this collection. Note that f intersects $\mathcal{Q}$ and f is a facet on the boundary of $\mathcal{C}$. However, recall that $\partial \mathcal{C}$ was grown in the cells of T and can only contain facets of T. Because $\mathcal{Q}$ is embedded in T, the facets of T cannot transversely intersect $\mathcal{Q}$, a contradiction. Because we took an arbitrary facet f, this contradiction holds for all facets intersected in their interior by ∂_c.

Next, we will show that the initial cavity of $\mathcal{Q}_c$ is a strict subspace of $\mathcal{C}$. Recall that the initial cavity is the collection of cells I of T_c intersected by $\mathcal{Q}_c$. As shown above, this intersection is bounded by edges of T_c. In fact, all these edges must also be edges of $\mathcal{C}$. Because $\mathcal{Q}_c$ is strictly inside $\mathcal{C}$, I must be a subspace of $\mathcal{C}$. Because $\mathcal{Q}_c$ must lie inside T_c and T_c lies on one size of $\mathcal{P}$, I must be a strict subspace of $\mathcal{C}$.

Finally, we will show that the cavity of $\mathcal{Q}_c$ can neither grow outside $\mathcal{C}$ nor to the other side of $\mathcal{P}$. Recall that we grow the cavity within T_c one cell at a time at facets that are not weakly Delaunay. In order for the cavity to grow outside $\mathcal{C}$, it must at some point have grown through a boundary facet of $\mathcal{C}$, since T_c contains all these boundary facets. However, since all these facets are weakly Delaunay, the cavity cannot grow outside of $\mathcal{C}$. Since we grow the cavity in the cells of T_c and T_c does not contain a point on the other side of $\mathcal{P}$, the cavity cannot grow to the other size of $\mathcal{P}$. □

Using a method very similar to our polygon embedding method, we can also remove a polygon $\mathcal{P}$ from a CCDT. In order to achieve this, we start a cavity from the two cells incident to one of the facets embedding $\mathcal{P}$. We grow this cavity as described above. Then, instead of constructing two new CCDTs, we construct one that contains the points and constraints on both sides $\mathcal{P}$. We repeat this for all facets embedding $\mathcal{P}$ that were not contained in an earlier cavity.

The growing procedure will ensure that the facets on the boundary of each cavity are Delaunay in the points on the opposite side of $\mathcal{P}$. The construction of the new CCDTs will ensure that the cells constructed to fill the cavity are Delaunay in the points on both sides of $\mathcal{P}$.

3.2 Minimum-Weight Graph-Cut

We use a minimum-weight graph-cut algorithm to determine which regions are inside or outside of the objects. A graph-cut is an operation on a weighted, directed graph $G\langle V, E\rangle$ containing two special nodes: a source s and a sink s'. The graph-cut C, also called a cut, is a subset of the edges E whose removal disconnects the source and sink. In other words, when all the edges of the cut are removed from the graph, there is no path from source to sink. A minimum-weight graph-cut C^- is an optimal cut in the sense that the summed weight $w(e)$ of the edges in the cut is minimal over all possible cuts. Specifically, C^- minimizes:

$$\sum_{e \in C} w(e) \tag{5}$$

over all possible cuts C, under the constraint that no path connecting s and s' exists in $C \subset E$.

Ford and Fulkerson [15] show that determining this minimum-weight graph-cut is equivalent to determining the maximum-flow. To visualize the maximum-flow, imagine that the graph represents a network of pipes where the weight of an edge represents its capacity. The maximum-flow is the maximal amount of water that can be pushed through this network from source to sink. One can imagine that this maximal amount depends on the bottlenecks in the network, the pipes that are maximally utilized. Ford and Fulkerson [15] show that these pipes are exactly the minimum-weight cut.

We use the method developed by Boykov and Kolmogorov [7] to compute the minimum-weight graph-cut. This method consistently outperforms the other state-of-the-art methods [13, 17] in practice, even though its asymptotic running time is worse. The method is an *augmenting path* based method [13], a group of methods that iteratively identify augmenting paths from source to sink and mark them in a residual graph. A path is augmenting if all its edges in the residual graph have non-zero weights. This residual graph starts identical to the original graph. Each time a new augmenting path from source to sink is found, the weights of all edges along this path in the residual graph are reduced by the same amount such that at least one of the edges gets weight 0. This can be imagined as sending a certain amount of flow along the path.

This method is optimized for image-based clustering and the regular nature of this problem. However, in practice it works well for any graph that describes the adjacency of a partitioning of low-dimensional space. For our application, this partitioning appears when we view the graph in 4-space. The fourth dimension is separated into three layers. The first and last layer contain only the source and sink respectively. The middle layer contains the CCDT.

Boykov and Kolmogorov [7] have optimized their method mainly by minimizing repetition. They search for new augmenting paths using two breadth-first search trees, rooted respectively in the source and sink. Unlike earlier methods, these search trees are persistent instead of being constructed multiple times. While searching for a new path from source to sink, both search trees are grown simultaneously on the non-zero edges of the residual graph until they touch each other. This indicates a new augmenting path and the weights along this path are decreased, reducing the weight of at least one edge to 0. This may turn the search trees into forests and where possible, the parts no longer connected to source or sink are reconnected along another edge. Once the trees can no longer be grown, there are no more paths possible from source to sink and the trees are separated by the minimum-weight graph-cut. Additionally, the nodes of the graph can be labeled according to the tree they are in, with a third label denoting ‘free’ nodes which are in neither tree.

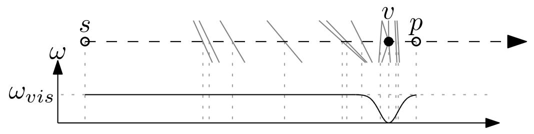

Figure 14: The influence of visibility on the weights. From the ray from sensor s through point v, we construct point p inside the object. The weight ω of a facet depends on the distance to v.

As the observant reader may have noticed, the result of a minimum-weight graph-cut depends on several factors. It depends on the connectivity of the graph, and its weights. The dual of the CCDT defines the connectivity between all nodes other than the source and sink nodes. The connectivity with the source and sink nodes and the weights of all edges is determined by the geometry of the scene and the measurement process, as described in the next two subsections.

3.2.1 Visibility

It is sometimes considered a disadvantage of laser light that it cannot pass through most solid surfaces, particularly opaque surfaces. These surfaces reflect the laser light, which is the basis for most laser scanners: they capture the reflected light in order to determine the distance to the surface. Another important feature is that laser light travels in a straight line, which is used to determine the position of the point of reflection, the LiDAR data point. From these features, we can infer that the line segment connecting data point and sensor can only pass through (semi-)transparent space. We use this information to adjust the weights of the graph the same as Labatut *et al.* [20].

For each data point v and its sensor location s, we construct a point p inside the object. We place p on the ray $\overrightarrow{sv}$ at a distance 3σ beyond v, where σ is a parameter based upon the measurement precision and the expected minimum thickness of the object. We connect $n(s)$ to the source node and $n(p)$ to the sink node, both with a weight of ω_{vis}, where $n(x)$ is the node in the graph of the CCDT cell containing x.

We know from the scanning procedure that a laser traveling along $\overrightarrow{sv}$ hit a solid surface near v. We express the exterior space traversed in the weights of the graph. Each edge corresponding to a facet f of the CCDT intersected by $\overline{ps}$ gains weight $\omega_{vis}(1 - e^{-d^2/2\sigma^2})$, where $d = \|vq\|$ and q is the intersection of f and $\overline{ps}$, as shown in Figure 14. Note this weight approaches 0 near v, and ω_{vis} at $d > 3\sigma$.

These weights drive the graph-cut to include a facet near v. The function always returns a value smaller than ω_{vis} and the procedure constructs a non-zero weight path in the graph connecting source and sink. The edges to the source and sink nodes are not cut because there is always a smaller weight on the connecting path.

This procedure is repeated for each data point, not only the CCDT vertices, and in all cases the assigned weights are aggregated. This resembles a voting procedure using evidence for the surface location and interior/exterior regions. Cells containing the source or sink for multiple data points get a stronger connection to that source or sink, representing the accumulation of evidence for them being outside or inside the object respectively. Facets intersected by multiple lines of sight get a higher weight, representing a lower chance that these facets are part of the surface of the object.

3.2.2 Facet Quality

The visibility-based weights term to the weights given in the previous section places sources and sinks in the interior and exterior of the objects respectively, and constructs a strong connection between cells that are unlikely to separate interior from exterior. However, this measure is not very robust to degenerate CCDT geometry. For example, long and thin cells are easily missed by all lines-of-sight, meaning they have a large chance of being mislabeled.

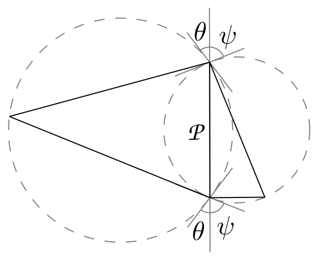

Figure 15: The influence of facet quality on the weights. The circumscribing balls intersect the supporting plane $\mathcal{P}$ at angles θ and ψ.

We use an inherent quality of each CCDT facet f and its local geometry to adjust its weight. Like Labatut *et al.* [20], we add $\omega_{qual} = 1 - min(\cos(\theta), \cos(\psi))$ to the weight of each facet, where θ and ψ are the angles between the plane $\mathcal{P}$ supporting f and the circumspheres of the two incident cells at the circle where they intersect $\mathcal{P}$, as shown in Figure 15.

Constrained facets are generally better qualified to be part of the surface. Irrespective of the visibility and facet quality terms, constrained facets are assigned a weight 0.

3.3 Implementation

We have implemented our method in C++ using CGAL 4.0.2 [10] and an implementation of Boykov and Kolmogorov's graph-cut method [18]. In order to construct a CCDT of a collection of points and polygons, we have built two classes on top of CGAL's Delaunay tetrahedralization. Firstly, we have implemented the conforming Delaunay tetrahedralization that extends the DT to handle conforming segments. Secondly, we have implemented the constrained Delaunay tetrahedralization to handle constrained polygons. Both of these classes are straightforward to program based on Section 3.1, although like the DT, careful programming is required to correctly handle degeneracies.

The conforming Delaunay tetrahedralization inherits most of its functionality from the basic Delaunay tetrahedralization. Additionally, conforming line segments can be inserted, as detailed in Subsection 3.1.1. The insertion of a Steiner point may force another conforming segment to be removed from the DT, in which case it is automatically reinserted after handling the current segment.

The constrained Delaunay tetrahedralization inherits most of its functionality from the conforming Delaunay tetrahedralization. Additionally, polygon constraints can be inserted once all its edges are conforming, as detailed in Subsection 3.1.2.

When a new polygon is embedded in the CCDT, we mark its boundary edges to distinguish them from other coplanar edges that do not belong to the polygon's boundary. Then, all cells intersected by the supporting plane of the polygon and inside the marked edges are collected. This is achieved by identifying a seed cell that intersects the interior of the polygon in a convex corner. All other intersected cells can be found by traversing intersected facets and unmarked coplanar edges. This collection is finally subdivided into cavities.

Recall that polygons of the PLC may contain holes. However, the segments bounding a hole were not marked when embedding the polygon. After embedding the interior of a polygon, its holes are marked and deconstrained as detailed at the end of Subsection 3.1.2.

In our implementation, it is not possible to insert points in regions near constrained facets. The reason for this is that the insertion of a point may force new Steiner points to make sure that the conforming edges would be in the DT. However, once cells are no longer strictly Delaunay, it is exceedingly difficult and expensive to identify which edges should receive new Steiner points. For this reason, we always insert all the points and conforming segments before embedding the polygon interiors.

	$\lvert P_d \rvert$	$\lvert P_s \rvert$	$\lvert S \rvert$
1	471,203	17,194	42
2	2,199,122	145,201	106
3	436,297	42,067	69
4	1,094,277	27,165	25
5	866,767	32,549	51
6	6,493,599	28,759	184
7	204,160	20,648	30

Table 1: The sizes of the chunks: the number of points ($|P_d|$), the number of points after subsampling ($|P_s|$), and the number of shapes ($|S|$) identified by Efficient RANSAC.

4 Experimental Setup

Unfortunately, there are no benchmark data sets available of urban point data, let alone benchmark data with known lines of sight. We use a data set provided by Fugro [16]. This data set contains aerial LiDAR data points that were measured during several flights. Each data point has a registered 3D coordinate and a time stamp that can be matched with the flight path to estimate sensor positions. The massive data sets are separated into smaller chunks based on a regular horizontal grid, with all points collected in one region combined into one chunk. We perform our experiments on seven different chunks, shown in Figure 16.

Our 32-bit implementation experienced memory problems with excessive data sizes. Therefore, we subsample the data by superimposing a 3D grid with fixed cell dimensions. We sample one random data point per grid cell [4]. Most chunks use a grid edge length of 1 m, but chunk 6 uses 3 m.

From this subsample, we estimate the local surface normal at each point using PCA on its 12 nearest neighbors. Subsequently, we apply Efficient RANSAC to the chunk to identify planar clusters and the remaining points. Simultaneously, the least-squares optimal approximating plane is computed. We have determined the parameter settings of Efficient RANSAC empirically based on clustering performance. We settled on support threshold 6.5 cm, normal threshold $20°$, bitmap size 1.5 m (2 m on chunk 1 and 4 m on chunk 6), minimal support 25 (10 on chunk 6), and probability 0.0001 of missing a better cluster. Table 1 shows the number of points and shapes per chunks.

For each pair of surfaces with data points within 1 m of each other, the intersection line is computed. We compute the guided α-shape [35] of each cluster using the intersection lines of that cluster as guides with the α-value equal to the RANSAC bitmap size. The resulting polygons and the remaining unclustered points are shown in Figure 17.

To compare our method to Labatut *et al.* [20], we construct both the DT and CCDT of the chunk. We insert all the data points into the DT. We insert the collection of guided α-shapes and the unclustered points into the CCDT. Note that for the planar clusters, the CCDT will only contain the points on the boundary. The CCDT also contains a number of Steiner points.

We construct a graph of these arrangements and we determine a graph-cut as explained in Section 3.2. This cut determines the facets that comprise the surfaces of the scene.

5 Results

Figure 16 shows some of the chunks of urban LiDAR we have performed our experiments on. Figures 18 and 19 show the watertight geometry constructed using [20] and our method. It is clear that our method constructs geometry that is simpler and that conforms more to our idea what an urban scene should look like. The small details are represented by free-form meshes, while the large roofs and facades are represented by flat surfaces. An unfortunate side-effect to the concise description of the planar surfaces is that it makes the other, noisy regions stand out more.

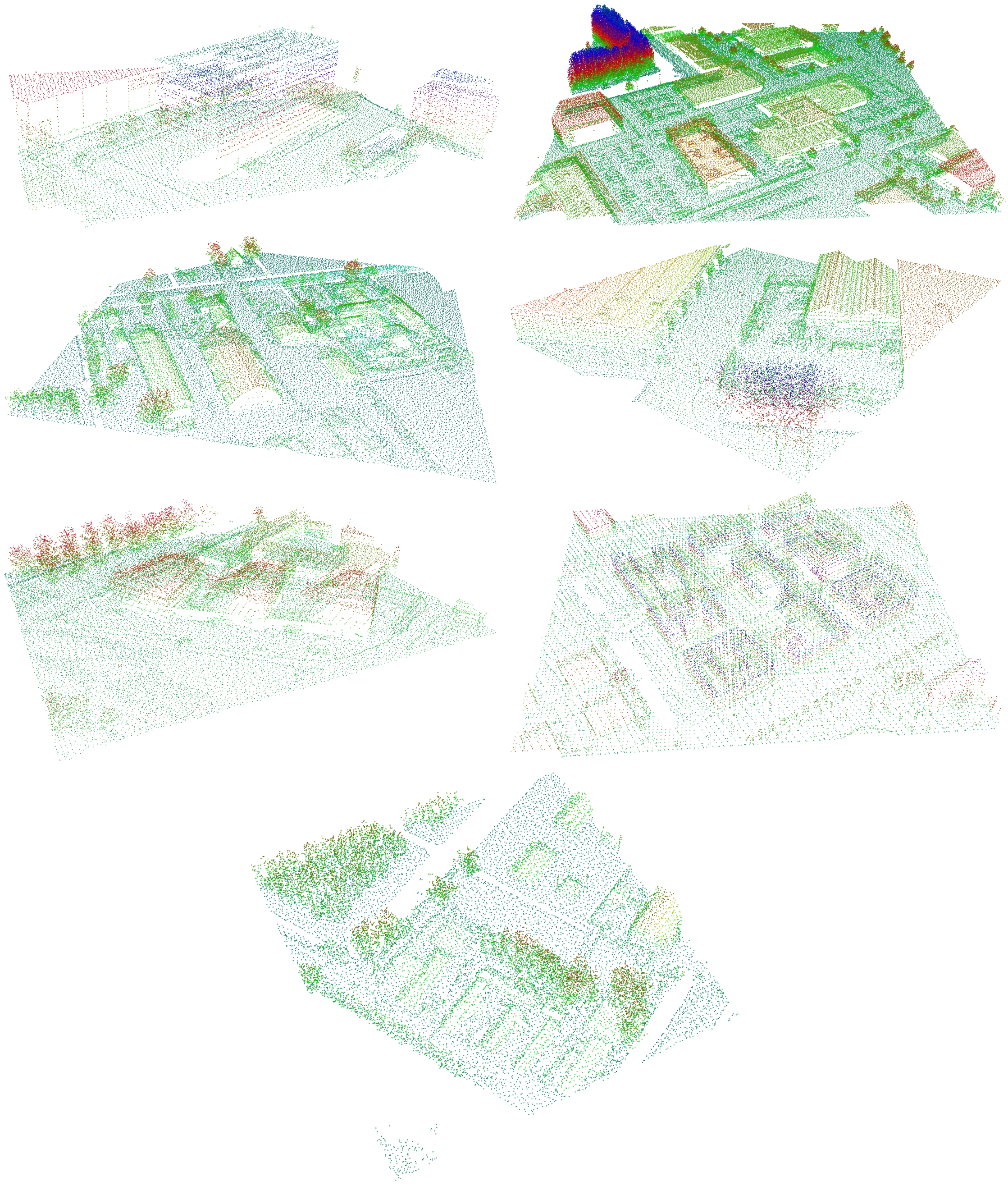

Figure 16: The points of the chunks.

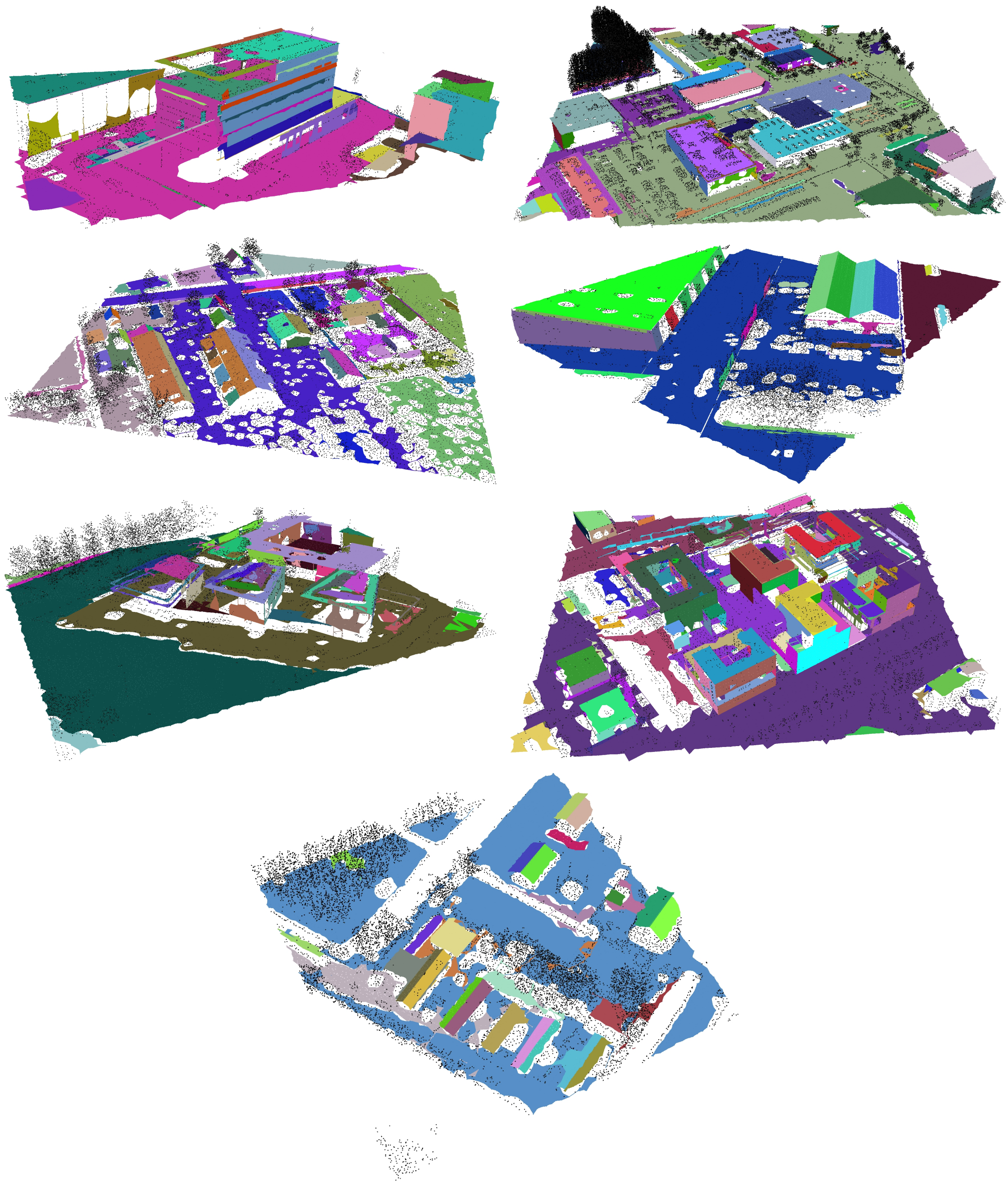

Figure 17: The guided α-shapes of the chunks.

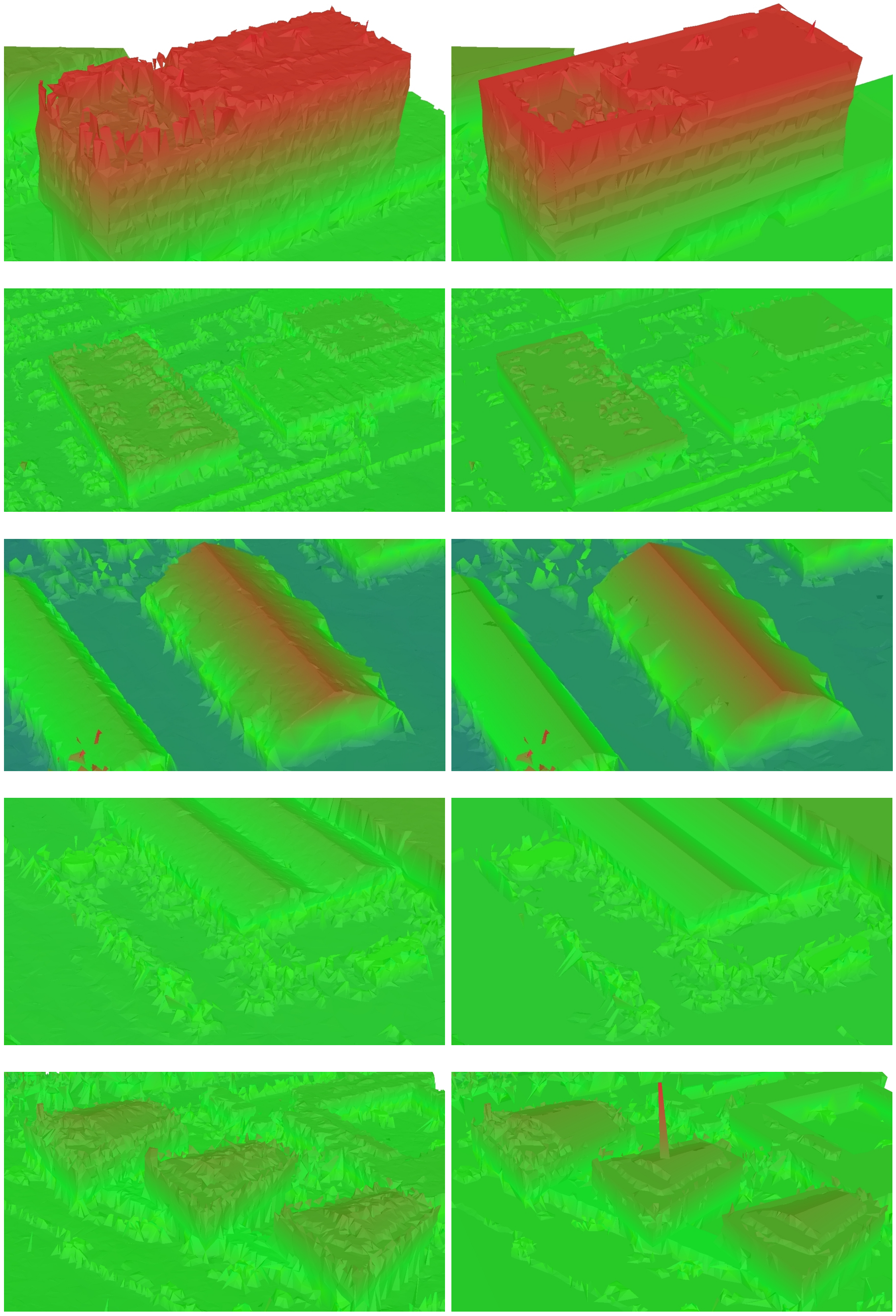

Figure 18: Details of the watertight geometry produced using [20] (left), and our method (right) of chunks 1–5. The colors are based on vertex height and triangle normal.

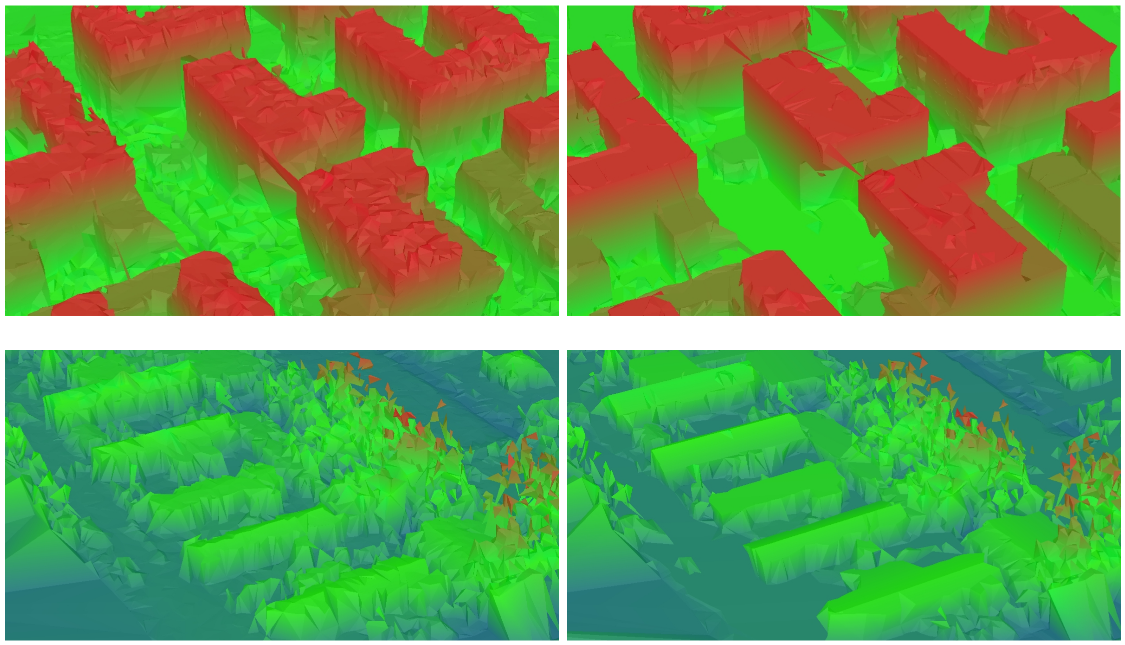

Figure 19: Details of the watertight geometry produced using [20] (left), and our method (right) of chunks 6–7. The colors are based on vertex height and triangle normal.

Note that vegetation is also reconstructed in various places. The quality of the reconstruction of vegetation is based both on the number of points in the trees and bushes and on the value of σ. Larger values of σ increase the chance that cells outside the object are connected to the sink. This results in this connection being cut, instead of an edge between Delaunay cells. The spike on a rooftop in chunk 5 seems to be the result of a similar process on some outliers.

Apart from producing a geometric model that better captures the planar nature of an urban scene, this geometry is also generally more concise, as Table 2 shows. In most cases, the CCDT has roughly a sixth fewer elements and the watertight surface generated from it uses roughly a third fewer triangles than when using the DT.

Chunk 6 is an exception in this regard: in this case the CCDT-based surface uses more triangles. We expect this is caused by the much higher than average shape to point ratio, as shown in Table 1. This chunk has roughly three times as many shapes as chunks of similar size. It is not surprising that the reconstruction would require more triangles to cover these shapes. This conjecture is supported by the higher than average number of Steiner points in the CCDT of chunk 6.

The disadvantage of our method is that it takes significantly more time to compute the geometry, as shown in Table 3. Once the CCDT is computed, traversing the lines of sight to set the weights of the graph and performing the graph-cut takes roughly the same amount of time for the CCDT as for the DT. We are currently working on improving the efficiency of constructing the CCDT.

6 Conclusions

We have presented a method to construct watertight geometry for urban scenes. Our method extends an earlier method by incorporating the assumption that many of the surfaces in urban scenes are piecewise planar. The method achieves this by performing a minimum-weight graph-cut on a conforming constrained Delaunay tetrahedralization. Although our method is slower than the earlier method that uses the Delaunay tetrahedralization, the results are more concise and

	DT				CCDT					
	$\lvert V\rvert$	$\lvert F\rvert$	$\lvert C\rvert$	$\lvert T\rvert$	$\lvert V\rvert$	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert C\rvert$	$\lvert T\rvert$	$\lvert T\rvert_r$
1	17k	220k	110k	29k	12k	4k	158k	79k	16k	0.55
2	145k	1874k	937k	197k	109k	26k	1398k	699k	119k	0.61
3	42k	534k	267k	69k	36k	7k	463k	232k	52k	0.75
4	27k	343k	172k	42k	14k	1k	188k	94k	20k	0.46
5	33k	416k	208k	50k	23k	6k	295k	148k	29k	0.59
6	29k	371k	185k	52k	46k	29k	552k	276k	59k	1.12
7	21k	268k	134k	33k	16k	1k	208k	104k	23k	0.70

Table 2: The number of vertices ($|V|$), of which Steiner ($|S|$), facets ($|F|$), and cells ($|C|$) in the DT and CCDT for the various chunks, and the number of triangles of the watertight surface ($|T|$). Each of these values is given in thousands. Finally, the ratio of triangles between CCDT and DT ($|T|_r$).

	DT				CCDT						
	T	t_P	t_L	t_G	T	t_P	t_l	t_b	t_p	t_L	t_G
1	5.11	12%	64%	25%	37.68	24%	13%	21%	34%	7%	2%
2	48.20	10%	64%	26%	442.56	30%	36%	11%	15%	6%	2%
3	12.16	10%	64%	25%	87.78	45%	12%	12%	20%	8%	3%
4	7.92	11%	64%	25%	20.84	30%	5%	16%	28%	16%	5%
5	9.27	11%	64%	26%	54.27	25%	25%	19%	19%	8%	3%
6	8.13	13%	61%	27%	361.86	34%	29%	21%	14%	1%	1%
7	5.63	12%	62%	27%	17.48	45%	10%	10%	13%	16%	6%

Table 3: The total time (T) in seconds to compute the watertight geometry using the DT and CCDT and the percentage of time taken by the steps in the process: inserting the points (t_P), inserting the intersection lines between shapes (t_l), inserting the boundaries of the shapes (t_b), inserting the interiors of the shapes (t_p), traversing the lines of sight to adjust the weights (t_L), and performing the graph-cut (t_G).

they are more visually pleasing.

One of the most interesting directions to improve on this method is to reduce the time needed to compute the CCDT, as this is an order of magnitude worse than computing the DT. Although our implementation is based on the earlier work by Si and Gärtner [32] and Shewchuk [30], some clever usage of multi-core processing may greatly speed up the process.

It is also interesting to consider the recurrence of Delaunay structures in this method and those used to pre-process the data. Most preceding methods, like estimating the point normals, already use a 3-dimensional DT. Others, like the guided α-shape, use a 2-dimensional constrained DT. Perhaps it would be beneficial to combine these methods so they could all use the same data structure. This may save a lot of time by reducing the amount of redundant work.

Our method reconstructs the scene as a whole, ignoring the properties of different objects in the scene. Tuning the parameters to the different objects may produce better results. For example, we may wish to reduce σ for points on vegetation to indicate the smaller expected object width.

Finally, a major issue in urban reconstruction is the general lack of ground truths and benchmark data sets. This makes it difficult to quantify results. We have expressed the quality of our method in the number of triangles. However, establishing benchmarks and comparing various methods on these will increase insight into their respective strengths.

Acknowledgments This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie). We would like to thank Pierre Alliez, Ramsay Dyer, and Mariette Yvinec for their insightful comments.

References

[1] O. Aichholzer, F. Aurenhammer, B. Kornberger, S. Plantinga, G. Rote, A. Sturm, and G. Vegter. Recovering structure from r-sampled objects. *Comp. Graph. Forum*, 28(5):1349–1360, 2009.

[2] E. L. Allgower and P. H. Schmidt. An algorithm for piecewise-linear approximation of an implicitly defined manifold. *SIAM Journal on Numerical Analysis*, 22(2):322–346, 1985.

[3] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *SGP*, pages 39–48, 2007.

[4] P. Alliez, L. Saboret, and N. Salman. Point set processing. 2013. `http://www.cgal.org/Manual/4.0/doc_html/cgal_manual/Point_set_processing_3/Chapter_main.html`.

[5] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *SoCG*, pages 213–222, 2000.

[6] J.-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Trans. Graph.*, 3(4):266–286, 1984.

[7] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, 2004.

[8] F. Cazals and J. Giesen. Delaunay triangulation based surface reconstruction. In *Effective Computational Geometry for Curves and Surfaces*, pages 231–276. 2006.

[9] A.-L. Chauve, P. Labatut, and J.-P. Pons. Robust piecewise-planar 3D reconstruction and completion from large-scale unstructured point data. In *CVPR*, pages 1261–1268, 2010.

[10] Computational Geometry Algorithms Library. `http://www.cgal.org/`, 2013.

[11] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH '96*, pages 303–312, 1996.

[12] T. K. Dey and S. Goswami. Tight cocone: a water-tight surface reconstructor. In *SM '03*, pages 127–134, 2003.

[13] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[14] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. In *IEEE Trans. Inf. Th.*, volume 29, pages 551–559, 1983.

[15] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010.

[16] Fugro EarthData, Inc. `http://www.fugroearthdata.com/`, 2012.

[17] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[18] Graph Cut optimization library. `http://cbia.fi.muni.cz/user_dirs/gc_doc/`, 2012.

[19] H. Hoppe, T. DeRose, T. Duchamp, J. A. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH '92*, pages 71–78, 1992.

[20] P. Labatut, J.-P. Pons, and R. Keriven. Robust and efficient surface reconstruction from range data. *Comp. Graph. Forum*, 28(8):2275–2290, 2009.

[21] G. L. Miller, D. Talmor, S.-H. Teng, N. Walkington, and H. Wang. Control volume meshes using sphere packing: generation, refinement and coarsening. In *IMR*, pages 47–61, 1996.

[22] P. Mullen, F. De Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez. Signing the unsigned: robust surface reconstruction from raw pointsets. *Comp. Graph. Forum*, 29(5):1733–1741, 2010.

[23] J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Alg.*, 18(3):548–585, 1995.

[24] R. Schnabel, P. Degener, and R. Klein. Completion and reconstruction with primitive shapes. *Comp. Graph. Forum*, 28:503–512, 2009.

[25] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. *Comp. Graph. Forum*, 26(2):214–226, 2007.

[26] E. Schönhardt. Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 98(1):309–312, 1928.

[27] S. Shalom, A. Shamir, H. Zhang, and D. Cohen-Or. Cone carving for surface reconstruction. *ACM Trans. Graph.*, 29(6):150:1–150:10, 2010.

[28] C. Shen, J. F. O'Brien, and J. R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph.*, 23(3):896–904, 2004.

[29] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. In *SoCG*, pages 76–85, 1998.

[30] J. R. Shewchuk. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *IMR*, pages 193–204, 2002.

[31] J. R. Shewchuk. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *SoCG*, pages 181–190, 2003.

[32] H. Si and K. Gärtner. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *IMR*, pages 147–163, 2005.

[33] F. Tarsha-kurdi and P. Grussenmeyer. Hough-transform and extended RANSAC algorithms for automatic detection of 3D building roof planes from lidar data. In *IAPRS*, volume 36, 2007.

[34] Y.-H. Tseng, K.-P. Tang, and F.-C. Chou. Surface reconstruction from LiDAR data with extended snake theory. In *EMMCVPR '07*, pages 479–492, 2007.

[35] M. van Kreveld, T. van Lankveld, and R. C. Veltkamp. On the shape of a set of points and lines in the plane. *Comp. Graph. Forum*, 30(5):1553–1562, 2011.