Identifying rectangles in laser range data for urban scene reconstruction

Thijs van Lankveld

Marc van Kreveld

Remco Veltkamp

Technical Report UU-CS-2011-004
February 2011

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

Identifying rectangles in laser range data for urban scene reconstruction

Thijs van Lankveld   Marc van Kreveld   Remco Veltkamp

March 4, 2011

Abstract

In urban scenes, many of the surfaces are planar and bounded by simple shapes. In a laser scan of such a scene, these simple shapes can still be identified. We present a one-parameter algorithm that can identify point sets on a plane for which a rectangle is a fitting boundary. These rectangles have a guaranteed density: no large part of the rectangle is empty of points. We prove that our algorithm identifies all angles for which a rectangle fits the point set of size n in $O(n \log n)$ time. We experimentally evaluate our method on thirteen urban data sets. We also compare the rectangles found by our algorithm to the α-shape as a surface boundary.

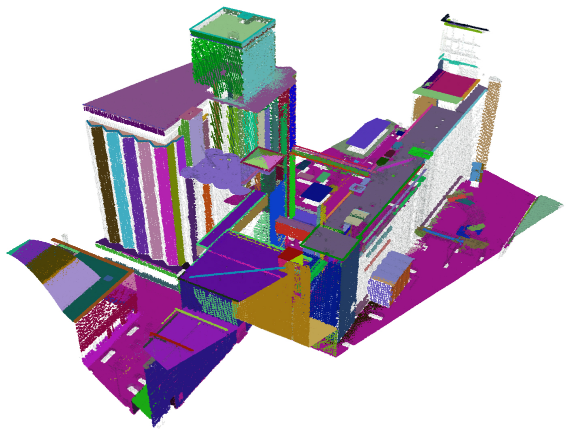

Figure 1: One of the regions used in the experiments. Each cluster has its own color and remaining unclustered points are black. A detail of this scene is shown in Figure 6.

1 Introduction

Automatic reconstruction of 3D geometric models of the world is currently seeing an increase in demand. Applications like navigation, serious games for training, and urban planning require very large data sets to be processed into a detailed and accurate model. Because of the number and size of the data sets and the constant changes to the scenes, the reconstruction process should be automated as much as possible.

Some data sources are directly useful for reconstruction of urban scenes. Ground-based photographs are easily obtained and together with aerial photographs, a large region can quickly be covered. However, while photographs are very suitable for finding the edges of structures, they are less suitable for accurate positioning of the surfaces in the scene. By contrast, laser range scans enable accurate positioning, but do not have high quality color information. In recent years, the resolution of laser range scanners has improved drastically and currently a single aerial scanning pass can produce a dataset with a typical density of 50-100 data points per square meter [10]. Figure 1 shows the data in one of the regions used in the experiments with the data points colored by surface. Some methods have successfully combined images and laser range scans [23], but to allow applications where no images are available we use only laser range scans.

Because of the large number of planar surfaces in urban scenes and limited number of major directions, many of the surface boundaries have sharp, straight edges and right angles. In the urban data sets we use, rectangles are the most prevalent shapes, bounding about 35% of the surfaces on average. Efficiently identifying these rectangles will solve a significant part of the geometry reconstruction problem.

We present an efficient algorithm that can identify the point sets in a plane for which a rectangular boundary fits well. We define "well fitting" in a one-parameter coverage criterion that ensures a minimum local density across the whole rectangle. For each identified point set in a plane the algorithm produces the range of rotation angles for which the rectangle fits the point set. If some of the points are known to be outliers, the algorithm can also be used to separate the remaining point set from these outliers. Experimental results show the quality of the algorithm in identifying 'rectangular' point sets and when used as an intermediate step in the overall reconstruction process. We leave identifying other shapes for future research.

1.1 Problem

We aim to find the clusters of points in a plane for which a rectangle is an appropriate boundary. Intuitively, a rectangle would fit as a boundary if no large parts are empty, i.e. have a low local density. We base a measure for this local density on the radius of an empty disk.

Definition 1. The δ-coverage region of a point set S in a plane is the union of disks in the plane with radius δ and center $c \in S$. A polygon $\mathcal{P}$ is δ-covered by point set S if and only if it is inside the δ-coverage region of S.

We choose the δ-coverage region to determine the correctness of a boundary, because it has a clear geometric meaning. Apart from indicating the location of the point set, this structure can also be seen as a way to handle any noise model under the assumption that the resulting error is smaller than δ. The value of δ also has an intuitive geometric meaning, and the choice of δ is tied to the resolution of the measurement device, as each disk inside the boundary should ideally contain multiple data points.

It is straightforward to see that any rectangle bounding a point set must contain the convex hull $\mathcal{CH}$ of the set. Therefore if the δ-coverage region does not contain $\mathcal{CH}$, no rectangle bounding the set can be δ-covered. If the δ-coverage region does contain $\mathcal{CH}$, there is a buffer in which a δ-covered rectangle may be placed. In the overall reconstruction process the different rectangles should be connected along an edge. For this reason, we are not looking for an optimal rectangle, but for the class of all δ-covered rectangles from which an appropriate rectangle can be selected.

Problem. Given a point set in the plane and a parameter δ indicating how well the rectangular boundary should fit, identify the δ-covered rectangles that contain the point set.

1.2 Overview

After examining related work in Section 2, we present our algorithm in Section 3. The primary application of our algorithm is identifying rectangles that fit the data, and the way this is achieved is explained in Subsection 3.1. The algorithm scales well with the data size, which is proven in Subsection 3.2. An extension of the algorithm to more general convex polygons is straightforward, as shown in Subsection 3.3. In many reconstruction methods from laser range scans, vegetation poses a significant problem. Our algorithm can easily be adapted to also separate the point set from points scanned in vegetation once these have been classified as outliers, as presented in Subsection 3.4. We present the setup and results of the experiments on the algorithm in Section 4.

We evaluate these results in Section 5 and discuss the broader implications of the results in Section 6.

Our key contributions are:

- A novel approach for automatic urban reconstruction as the creation of a geometric model consisting of simple surfaces that fit the data. Compared to smooth surfaces our models are simpler and handle inconsistency between data and model as noise. Compared to complete building-model fitting, our paradigm is more general by allowing more building types.
- A new one-parameter algorithm that identifies clusters for which a rectangular boundary is appropriate and all angles for which a rectangle fits well. The algorithm has a time efficiency of $O(n \log n)$, where n is the size of the point set. Extensions of the algorithm that can handle outliers and other shapes are given.
- An analysis of the parameter settings for which the algorithm works best on the urban data sets provided. These same settings can be used in the α-shape [8], to which we compare our results.

2 Related work

Geometry reconstruction from laser range scans can broadly be divided into interactive and automatic methods. While interactive methods like SmartBoxes [14] greatly speed up manual reconstruction, their reliance on a human operator makes them less appropriate for handling massive datasets of cities within limited time.

Recent research into automatic geometry reconstruction from laser range scans has focused on smooth surfaces [2, 11, 20]. A likely reason for using smooth surfaces is that traditionally most high-density laser range scans were made using close range measurements of natural objects in a controlled environment. The Stanford Bunny, Dragon, and Happy Buddha [18] are well known examples of this type of objects.

Unlike these methods, we process urban scenes that mainly consist of surfaces that are part of simple primitives like planes, spheres, and cylinders. The transitions between these surfaces are not expected to be smooth and the points are not expected to be exactly on the surface due to noise. Representing the surfaces as primitives has the additional advantage of greatly reducing the disk space of the model. Another important consideration is that urban scenes can contain a large number of outliers, generated by vegetation and a generally less controlled environment. Usually, smooth surface reconstruction methods have difficulty identifying these artifacts and use heuristics to solve this problem.

Related research in geosciences has focused on fitting models of complete buildings to laser range data [4, 17] or extruding roof planes vertically up from the ground [24]. This has the advantage that the methods are still able to reconstruct the scene from very sparse data if the possible shapes of the buildings are modeled a priori. However, this approach is limited by the number and complexity of the different building models and is greatly influenced by vegetation. Unlike these methods, we use dense data sets and reconstruct each individual surface. Combining the surfaces will produce a geometric model equivalent to the predefined building models when such a model fits the data. Additionally, our building shapes are not limited to a priori determined models.

Some earlier methods for urban reconstruction classify data points into vegetation and buildings, and cluster the data set into a point set per surface [16, 20]. However, limited effort has been invested in using these clusters to create an explicit geometric boundary for each surface. Schnabel *et al.* used the primitives to create an implicit model of the scene and made it explicit using marching cubes [15]. However, the output of marching cubes lacks the simplicity and elegance of the primitive shape geometry.

Various methods have been proposed for creating the boundary of a set of unordered points in the plane. These methods can broadly be divided into two groups. The first group, including methods like the crust [3] and γ-neighborhood graph [21], assumes all points lie on the boundary. The second group, including the α-shape [8] and $\mathcal{A}$-shape [13], assumes the boundary contains all points within its interior.

Our problem is most related to the second group, because we want to identify a boundary containing all points in its interior. The main difference between the α-shape and $\mathcal{A}$-shape is that the α-shape determines the interior region based on a local density criterion, while the $\mathcal{A}$-shape determines the interior region based on a second point set that is completely exterior. Because we also bound the interior based on a local density criterion, the boundary constructed by our algorithm is compared to the α-shape in Section 4.2.

3 Rectangular boundaries

Our algorithm determines whether a cluster of points in a plane can be bounded by a rectangle in such a way that the rectangle does not have a large region void of the points. The problem of identifying all δ-covered rectangles can be reduced to identifying the angles of rotation for which there is a δ-covered rectangle. All other δ-covered rectangles will have edges parallel to an identified rectangle.

Because a δ-covered rectangle must lie within a buffer around the convex hull, at any given angle we choose the minimal area bounding rectangle. Our algorithm can easily be adapted to find other predefined convex shapes such as triangles, as shown in Subsection 3.3. We chose rectangles because these occur most often in urban scenes. The algorithm may be adapted to identify non-convex shapes with predefined angles, like L-shapes, but at the cost of a decreased efficiency and simplicity.

The algorithm is presented in Subsection 3.1. Although roughly a third of the surfaces in our urban scenes can be correctly bounded by a rectangle, confining the algorithm to rectangles is somewhat restrictive. An extension to general convex shapes is given in Subsection 3.3.

Vegetation or scanning artifacts may cause points that are incorrectly included in a cluster, called outliers. Handling outliers is an important part of processing real data. While most of these points are removed by RANSAC during preprocessing, the remaining outliers may disrupt our algorithm. With minor adjustments, our algorithm can force the rectangle to exclude outliers, as described in Subsection 3.4.

For completeness we first explain the preprocessing step used to cluster the data. We also give a short description of the α-shape, which is related to our δ-coverage region.

The input of our algorithm is a point set in a plane. As the laser range data sets consist of unordered points in 3-space measured from multiple surfaces, some clustering has to be done before passing the data to our algorithm. For this clustering, we use Efficient RANSAC proposed by Schnabel *et al.* [16]. This algorithm can identify surfaces on different types of primitives, like spheres and cones, but we restrict the search to planar surfaces.

Efficient RANSAC iteratively identifies the surface containing the largest connected component of points. In each iteration, a random sample of three points defines a plane, and the support set of this plane contains all points near the surface. This support set is divided into disconnected components if the inter-component distance is too large. This process is re-iterated until the probability of finding a larger connected component is very small. This component is identified as a cluster, its points are removed from the unclustered point set, and the process continues until no cluster of sufficient size is found. The algorithm results in a collection of primitives with supporting point clusters. Different clusters either support a different primitive, or are far apart.

While filling the support set of a cluster, the process also identifies points measured from vegetation. This is done by fitting a plane to each point's nearest neighbors and comparing this plane to the cluster's local surface. This algorithm can efficiently cluster the data into sets likely to originate from the same surface, while at the same time identifying and removing points that are probably measured from vegetation. This means the effects of both noise and outliers are mitigated. More details on Efficient RANSAC are provided in [16], including its usage to identify and remove points in vegetation. Note that identifying a surface for each cluster and projecting

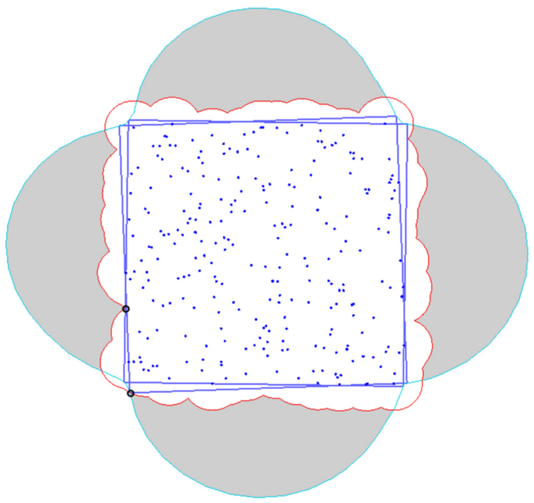

Figure 2: The different structures used during our algorithm and the extrema of the range of allowed rectangles. The blue points were sampled uniformly in a unit square. The δ-coverage region, with $\delta = 0.1$, is bounded by the red arcs, the trajectory region is bounded by the blue arcs. The grey regions show where the rectangle is outside the δ-coverage region. The blue rectangles show the rotations for which the rectangle starts and stops being δ-covered and the two events that caused this are emphasized by a black circle.

the noisy points onto this surface reduces the problem of determining a rectangular boundary to 2D.

The structure used by our method to determine if a rectangle is δ-covered is related to the 2-dimensional α-shape [8]. The α-shape of a point set is the collection of edges between two points that share the boundary of an empty disk with radius α. This collection of edges is a subset of the edges in the Delaunay triangulation of the point set, and the collection bounds an interior region that equals the union of Delaunay triangles with a circumcircle of radius at most α. Note that any triangle in the Delaunay triangulation is δ-covered if it has a circumcircle with radius at most δ. This is exactly the interior region of the α-shape if $\alpha = \delta$. The difference between the δ-coverage region and the α-shape, with $\alpha = \delta$, is covered by the union of disks with radius δ centered on the vertices of the α-shape. Two of these disks share a vertex in this union if their centers share an edge in the α-shape.

3.1 Algorithm

Our rectangular boundary algorithm borrows ideas from the rotating calipers algorithm [19]. However, we use a rectangle $\mathcal{R}$ instead of the traditional parallel lines. We rotate $\mathcal{R}$ around point set S as tightly as possible and handle important events when they occur, like in sweep-line algorithms [7].

A rectangle is only allowed if it is δ-covered. An *event* occurs at a rotation where the coverage of the rectangle changes structurally. To determine at which angle this happens two structures are introduced: the δ-coverage region, and the trajectory of the corners of $\mathcal{R}$, as shown in Figure 2. The events can be determined from the combination of these structures.

The δ-*coverage* region $\mathcal{U}_\delta$ is the maximal region that is δ-covered by S. It is the union of all δ-disks centered on a point in S, as shown by the red arcs in Figure 2. To satisfy the δ-coverage criterion, $\mathcal{R}$ must be completely inside $\mathcal{U}_\delta$. If part of $\mathcal{CH}$ is not in $\mathcal{U}_\delta$, then no bounding rectangle exists that is δ-covered. Conversely, if $\mathcal{CH} \subseteq \mathcal{U}_\delta$, then $\mathcal{U}_\delta$ has only one connected component.

The *trajectory region* $\mathcal{T}$ is the union of minimal bounding rectangles over all rotation angles, as shown by the blue arcs in Figure 2. During rotation all corners of the minimal bounding rectangle remain on the boundary of this region. This concept has previously been used by Hoffmann *et al.* [9]. The algorithm can be adjusted to find convex shapes other than rectangles, as shown in Subsection 3.3. This only requires changing $\mathcal{T}$: the new structure can be found by rotating each corner around the point set and applying Thales' theorem.

There are two ways in which $\mathcal{R}$ can start and stop being inside the δ-coverage region. Either during rotation an edge of $\mathcal{R}$ passes over a vertex of $\mathcal{U}_\delta$: the upper event in Figure 2, or a corner of $\mathcal{R}$ crosses the boundary of $\mathcal{U}_\delta$: the lower event in Figure 2. For a vertex of $\mathcal{U}_\delta$ to be inside the rectangle at some rotation angle, it is necessary and sufficient to be inside $\mathcal{T}$. Each of these vertices produces two events: when it enters and when it leaves $\mathcal{R}$. Because the corners of the rectangle follow the boundary of $\mathcal{T}$, a corner enters or leaves $\mathcal{U}_\delta$ exactly at an intersection of the boundaries of $\mathcal{U}_\delta$ and $\mathcal{T}$. Most of these intersections produce one event: the corner either enters or leaves $\mathcal{U}_\delta$. However, some intersections produce two events if the boundaries of $\mathcal{U}_\delta$ and $\mathcal{T}$ touch, but do not intersect.

Both types of events are inserted into a queue sorted on the angle of rotation at which they occur. After determining which vertices of $\mathcal{U}_\delta$ are inside $\mathcal{R}$ and which corners of $\mathcal{R}$ are outside $\mathcal{U}_\delta$ before rotation, the events are handled sequentially. The algorithm keeps track of the number of vertices of $\mathcal{U}_\delta$ inside $\mathcal{R}$ and the number of corners of $\mathcal{R}$ outside $\mathcal{U}_\delta$. The angles at which $\mathcal{R}$ becomes or stops being δ-covered are collected. In Figure 2, the points that caused these events are emphasized by a black circle. When all events have been handled, all rotation angles for which $\mathcal{R}$ is δ-covered are known. In Figure 2, the extrema of these ranges of angles are shown by blue rectangles. The overall structure of our method is summarized in Algorithm 1.

Algorithm 1 IDENTIFYING δ-COVERED RECTANGLES

Identify the range of angles of rotation for which the minimal bounding rectangle is δ-covered.

in P: the set of points in the plane that should be bounded.

out A: the set of angles for which the minimal bounding rectangle is δ-covered, or $\emptyset$ if there is no such rectangle.

```
Construct $\mathcal{CH}$
Construct $\mathcal{U}_\delta$
Construct $\mathcal{CH} \setminus \mathcal{U}_\delta$
if $\mathcal{CH} \setminus \mathcal{U}_\delta \neq \emptyset$ then
  return $\emptyset$
end if
Construct $\mathcal{T}$
Construct $\mathcal{T} \setminus \mathcal{U}_\delta$
Identify events $E$ from $\mathcal{T} \setminus \mathcal{U}_\delta$
Sort events $E$
Handle events $E$ in order to fill allowed angles $A$
return $A$
```

3.2 Efficiency

Our algorithm is based on the rotating calipers algorithm [19] and also has an asymptotic running time of $O(n \log n)$, where n is the size of the point set. To prove this we will prove four parts: constructing all necessary structures takes $O(n \log n)$ time, these structures generate $O(n)$ events, the angles at which the events occur can be computed in $O(n \log n)$ time, and each event can be handled in $O(1)$ time.

The structures used by the algorithm are the convex hull $\mathcal{CH}$, the δ-coverage region $\mathcal{U}_\delta$, and the trajectory region $\mathcal{T}$. It is well known that $\mathcal{CH}$ can be constructed in $O(n \log n)$ time [7]. $\mathcal{U}_\delta$ can be constructed directly from the α-shape, which can also be constructed in $O(n \log n)$ time citeedelsbrunner1983. Both structures have $O(n)$ vertices.

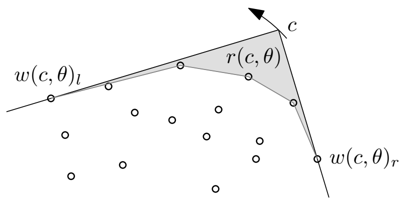

Figure 3: Corner c during rotation, together with the wedge it defines, the two points $w(c,\theta)_l$ and $w(c,\theta)_r$ on the corner's edges, and the grey region $r(c,\theta)$ visible to c.

The terms used in the following lemmas and observations are shown in Figure 3. We will use ∂X to refer to the boundary of X. If $\mathcal{CH} \setminus \mathcal{U}_\delta \neq \emptyset$ Observation 2 and Lemma 3 are incorrect, but in this case the algorithm is terminated before computing $\mathcal{T}$. Therefore, for these lemmas we assume $\mathcal{CH} \setminus \mathcal{U}_\delta = \emptyset$.

Lemma 1. *At any rotation angle θ, the corner c of $\mathcal{R}$ defines a wedge $w(c,\theta)$ with c on its apex and bounded by rays following edges of $\mathcal{R}$; these rays touch $\mathcal{CH}$ at two points $w(c,\theta)_l$,$w(c,\theta)_r$.*

Proof. This follows directly from the fact that we choose $\mathcal{R}$ as the bounding rectangle at each rotation angle. □

Lemma 2. *$\mathcal{T}$ can be constructed from $\mathcal{CH}$ in $O(n)$ time and contains $O(n)$ vertices and circular arcs.*

Proof. Let c be a corner of $\mathcal{R}$. It follows from Thales' theorem that during rotation c follows the smallest circumcircle of $w(c,\theta)_l$ and $w(c,\theta)_r$, while $w(c,\theta)_l$ and $w(c,\theta)_r$ remain unchanged. A new circular arc starts only when $w(c,\theta)_l$ or $w(c,\theta)_r$ changes, and this happens $O(n)$ times. The sequence of the $O(n)$ arcs forms $\mathcal{T}$. □

Observation 1. Any wedge $w(c,\theta)$ contains a closed region $r(c,\theta)$ between c and $\mathcal{CH}$, i.e. if $\mathcal{CH}$ obstructs visibility, $r(c,\theta)$ is the part of $w(c,\theta)$ visible to c. Because the angle of c is fixed, there is no rotation other than θ for which $r(c,\theta)$ contains c; otherwise the tangents of c with $\mathcal{CH}$ would have an incorrect angle. Therefore, the open line segment between any point on $\partial\mathcal{T}$ and its closest point on $\mathcal{CH}$ cannot intersect $\partial\mathcal{T}$.

Observation 2. The circumcenter of each arc in $\partial\mathcal{U}_\delta$ is on $\mathcal{CH}$ and no arc intersects $\mathcal{CH}$. Therefore, the open line segment between any point on $\partial\mathcal{U}_\delta$ and its closest point on $\mathcal{CH}$ cannot intersect $\partial\mathcal{U}_\delta$.

There are two causes for events: a vertex of $\mathcal{U}_\delta$ enters or leaves $\mathcal{R}$, or a corner of $\mathcal{R}$ enters or leaves $\mathcal{U}_\delta$. As stated earlier, $\mathcal{U}_\delta$ has $O(n)$ vertices.

Lemma 3. *The corners of $\mathcal{R}$ enter and leave $\mathcal{U}_\delta$ $O(n)$ times.*

Proof. This proof builds on the property that two piecewise simple functions with n curves have $O(n)$ intersections, which can be computed in $O(n)$ time. By simple we mean that two such curves can intersect each other only a constant number of times. We denote the boundary of X by ∂X.

To use this property, we must define two functions that have the same value at some argument if and only if $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ intersect. To create these functions, an extra structure is introduced to parameterize these boundaries, and thereby give the argument to define the functions. We will show that the functions created using this structure "detect" all intersections of $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$, which completes the proof. The structures used for this proof are shown in Figure 4.

Let $\mathcal{C}$ be the boundary of the morphological opening, the dilation of the erosion, of the region inside $\mathcal{CH}$ using an ϵ-disk. The radius $\epsilon > 0$ is chosen such that for all vertices of $\mathcal{U}_\delta$ and $\mathcal{T}$ that are not on a vertex of $\mathcal{CH}$, the closest point on $\mathcal{CH}$ is also on $\mathcal{C}$. Note that $\mathcal{C}$ is only needed to have a smooth boundary and thus a coherent definition of the normal direction.

Let $\mathcal{C}(t)$ be the point reached by following $\mathcal{C}$ from its topmost point for the fraction t of its length, and let $r(\mathcal{C}(t))$ be the outward ray emanating perpendicularly from $\mathcal{C}(t)$. Now $\partial\mathcal{U}_\delta$ and

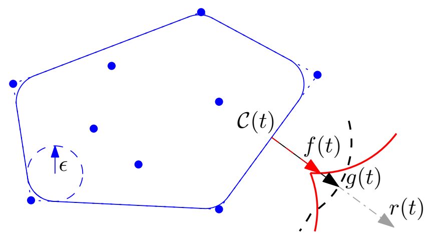

Figure 4: The structures used in Lemma 3 to prove $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ intersect $O(n)$ times. The radius ϵ of the disk used to create $\mathcal{C}$ will in practice be much smaller than shown.

$\partial\mathcal{T}$ define functions by taking $f(t) = dist(\mathcal{C}(t), r(\mathcal{C}(t)) \cap \partial\mathcal{U}_\delta)$ and $g(t) = dist(\mathcal{C}(t), r(\mathcal{C}(t)) \cap \partial\mathcal{T})$. These functions $f(t)$ and $g(t)$, are well-defined: for each t, $r(\mathcal{C}(t))$ intersects $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ exactly once. This is true, because Observations 1 and 2 can be extended to $\mathcal{C}$ to give a one-to-one relation between $\mathcal{C}$ and either one of $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$, additionally this relation always connects points on $r(\mathcal{C}(t))$.

The properties of $f(t)$ and $g(t)$ ensure that each point on $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ occurs for some t as the intersection of $r(\mathcal{C}(t))$ and $\partial\mathcal{U}_\delta$, respectively $\partial\mathcal{T}$, and this is the only intersection point for that ray. Hence, $f(t)$ and $g(t)$ have the property that a value of t for which $f(t) = g(t)$ corresponds one-to-one to an intersection point of $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$. Since $f(t)$ and $g(t)$ are also piecewise simple, $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ intersect $O(n)$ times. As each corner of $\mathcal{R}$ is always on $\partial\mathcal{T}$ it enters or leaves $\mathcal{U}_\delta$ at these $O(n)$ intersection points. □

Lemma 4. *Each vertex of $\mathcal{U}_\delta$ and intersection of $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ generates at most two events and these can be computed in $O(n \log n)$ time.*

Proof. The events either signify a vertex of $\mathcal{U}_\delta$ entering of leaving $\mathcal{R}$, called a vertex event, or a corner of $\mathcal{R}$ entering or leaving $\mathcal{U}_\delta$, called a corner event. After rotating $\frac{\pi}{2}$ radians $\mathcal{R}$ coincides with its initial position and further rotations need not be checked.

Each vertex of $\mathcal{U}_\delta$ has two lines tangent to $\mathcal{CH}$ and the angles of these tangent lines modulo $\frac{\pi}{2}$ are the angles of rotation at which the vertex could enter or leave $\mathcal{R}$. This means each vertex generates at most two vertex events.

Each point p where a corner passes over $\mathcal{U}_\delta$, also has two lines tangent to $\mathcal{CH}$, but because p is on the corner of a minimal bounding rectangle, the tangent lines make a right angle. However, a corner may leave and re-enter $\mathcal{U}_\delta$ at the same location. Therefore, any such point p generates at most two events.

A tangent line and its angle can be computed from $\mathcal{CH}$ in $O(\log n)$ time by performing a binary search on the vertices of $\mathcal{CH}$. Because both $\mathcal{U}_\delta$ and $\mathcal{T} \setminus \mathcal{U}_\delta$ have $O(n)$ vertices, all tangent lines can be computed in $O(n \log n)$ time. □

Theorem 1. *For a point set S of size n, and scalar δ, all angles $\theta \in [0, \frac{\pi}{2})$ for which there is a δ-covered rectangle $\mathcal{R}$ at rotation angle θ can be computed in $O(n \log n)$ time.*

Proof. Lemma 2, 3, and 4 prove that the rotation of a tight rectangle around S causes $O(n)$ events that can be generated in $O(n \log n)$ time. Sorting these events by angle takes $O(n \log n)$ time. During rotation, after each event it is checked if the rectangle has become empty or non-empty because of the vertex or corner that caused the event. This takes $O(1)$ time per event. □

3.3 Convex polygons

Our method can be adapted with minimal changes to a general convex k-gon with fixed angles and stretchable edges, which we will refer to simply as a convex k-shape. We say two k-shapes are equivalent if and only if each pair of edges with the same index in the clockwise sequence of edges starting at their top vertices have the same direction. Similar to the rectangle case, we only compute the rotation angles for which the minimal area k-shape containing all points is δ-covered.

The δ-coverage region depends on the point distribution and so is the same for different shapes. However, $\mathcal{T}$ depends on the angle of the corners of the k-shape. This can be handled by constructing a separate trajectory region for each corner. Each corner specifies the angle of a wedge. This wedge is rotated 2π radians independently to give the ranges of rotation angles for which that corner is allowed. Finally, these ranges can be combined by correcting for the relative rotations of the wedges in the k-shape. This produces the ranges for which all corners are allowed. We prove that the allowed rotation angles of a convex k-shape can be computed in $O(kn\log(kn))$ time in Theorem 2.

Before proceeding to this proof, there is a degeneracy to consider: an edge of a k-shape can shrink to length 0, causing corners to overlap and the angles to be taken together. We will prove our algorithm can handle this degeneracy, because an edge will only ever have length 0 if it contains a vertex of $\mathcal{CH}$. Note that this implies Observation 1 holds for all convex corners.

Lemma 5. *For any minimal area convex k-shape $\mathcal{P}$ and any rotation angle of $\mathcal{P}$, each edge contains a vertex of $\mathcal{CH}$.*

Proof. By contradiction, assume there is an edge e of $\mathcal{P}$ that does not contain a vertex of $\mathcal{CH}$. Without loss of generality, we assume this edge is above $\mathcal{CH}$. We can move e down by shortening other edges, while keeping $\mathcal{CH}$ inside the k-shape, until e contains a vertex of $\mathcal{CH}$. Because we only shorten edges, the new k-shape is a subset of $\mathcal{P}$, while having the same angles. This means $\mathcal{P}$ is not the smallest area k-shape. □

Theorem 2. *Given a point set S of size n, scalar δ, and convex k-shape $\mathcal{P}$, all angles $\theta \in [0, 2\pi)$ for which there is a δ-covered k-shape equivalent to $\mathcal{P}$ rotated by θ radians can be computed in $O(kn\log(kn))$ time.*

Proof. For each corner c of $\mathcal{P}$, we rotate $w(c,\theta)$ around S independently and we identify the rotations for which all $r(c,\theta)$ are δ-covered. Most of this proof is the same as the proof of Theorem 1 and we do not repeat this here. However, we do need to prove that for any convex corner c, $\mathcal{T}_c$ can be computed in $O(n\log n)$ time, and $\mathcal{T}_c$ intersects $\mathcal{U}_\delta$ $O(n)$ times, thereby producing $O(n)$ events. To show this, we will prove Lemma 2 and 3 hold for convex k-shapes.

The proof of Lemma 2 is based on Thales' theorem, which describes the trajectory of a corner irrespective of angle. However, unlike the proof of Lemma 2, a convex corner follows a circumcircle of $w(c,\theta)_l$ and $w(c,\theta)_r$ that is not necessarily the smallest. However, because Observation 1 holds for any convex corner with fixed angle, no $r(c,\theta)$ can contain c, except at rotation θ. Therefore Lemma 2 still holds.

Observation 1 shows the one-to-one relation between $\mathcal{T}$ and $\mathcal{C}$ used in the proof of Lemma 3 holds for corners with other angles. The remainder of the proof of Lemma 3 remains the same. □

3.4 Outliers

The preprocessing done on the data set will correctly classify most points scanned in vegetation as outliers and ignore them. However, some outliers may remain in the clusters. Common examples are points from nearby surfaces that were misclustered, points measured through windows, and solitary points with all their nearest neighbors in the cluster. These outliers can adversely affect our algorithm by significantly changing the convex hull.

There are different methods for classifying outliers. Some methods for semi-automatic outlier classification and removal in point data are given by Weyrich *et al.* [22]. They also explain that these different methods are appropriate for different types of outliers. For this reason, they leave the choice in method to be determined interactively. Because we are interested in automatic reconstruction and because our method starts from pre-clustered data, we assume outlier detection is done during pre-processing. We leave the choice of outlier detector to the user.

The algorithm presented in Subsection 3.1 provides an effective way to determine the angles for which a rectangle fits a complete cluster. The algorithm can also be extended to handle outliers.

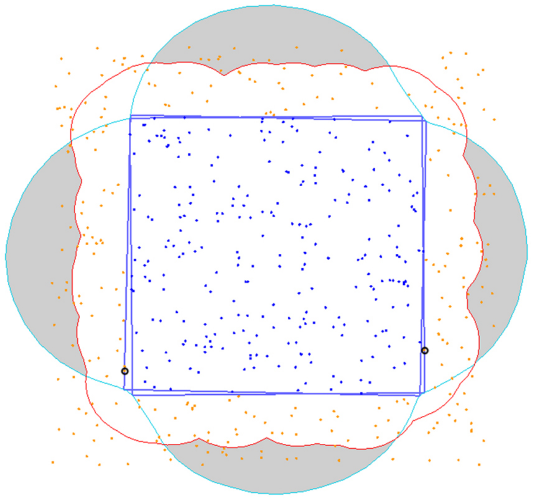

Figure 5: The different structures used during our algorithm and the range of allowed angles. The blue points were sampled uniformly in a unit square and the orange points are outliers. The δ-coverage region, with $\delta = 0.2$, is bounded by the red arcs, the trajectory region is bounded by the blue arcs. The grey regions show where the rectangle is outside the δ-coverage region. The blue rectangles show the rotations for which the rectangle starts and stops being δ-covered and the events that caused this are emphasized by a black circle.

The way outliers are handled, depends on the definition of outlier and two different definitions may be appropriate. Either (1) outliers are points that should not influence the reconstruction, or (2) outliers are points that should be outside the boundary of a surface. Our algorithm can handle outliers under definition 1 without needing an extension: removing these outliers from the point set is sufficient to ignore them.

If outliers are points in the cluster that must be outside the rectangle, our algorithm can be adapted to find the range of outlier-free, allowed angles. For these angles there is a rectangle containing all inliers and no outliers that is also δ-covered by the inliers. Only one change needs to be made to the algorithm to implement this extension, assuming the input points S are divided into inliers S_I and outliers S_O. The algorithm is run normally, with S_I taking the place of S and with the points in S_O treated as if they were vertices of $\mathcal{U}_\delta$. Figure 5 shows the structures and range of rotation angles in the presence of outliers.

3.5 Implementation

We implemented the algorithm in C++ using the CGAL library [6] for most of the computations. For most structures we used CGAL's utility of lazy exact rational coordinates to speed up the calculations. However, the intersection of two circular arcs cannot be expressed by rational coordinates, while the coordinates need to be exact to correctly check if a vertex of $\mathcal{U}_\delta$ is inside $\mathcal{T}$. The coordinates can be expressed exactly by using numbers with one root factor, e.g. $x = a + b\sqrt{c}$. However, vectors and line segments, which play a crucial role in our algorithm, cannot use one-root coordinates, because the plus and minus operations are not defined on one-root numbers, i.e. $(a + b\sqrt{c}) - (d + e\sqrt{f})$ can only be expressed using one root if $c = f$. This problem is solved by approximating the points by rational coordinates after constructing the structures. If the rectangles must be computed exactly, it is also possible to use algebraic numbers, but this significantly slows down computations.

To greatly reduce the number of points that have to be considered, we first use CGAL to calculate the α-shape of the point set, using the same value for α as is set for δ. This may be done, because all other structures need only a subset of the vertices in the α-shape to be computed.

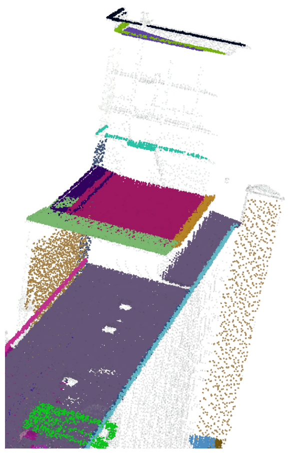

Figure 6: The surfaces do not have a globally consistent density. Specifically, vertical surfaces have a lower density than the other surfaces. This is a detail of Figure 1.

The convex hull of the point set uses a subset of the α-shape irrespective of the value of α. The trajectory region can be created from the convex hull using an adaptation of the rotating calipers algorithm. The algorithm is modified to compute which pairs of points have tangent lines with a right angle and where the corners of the rectangle are when the points on their incident edges change.

The vertices and circular arcs of the δ-coverage region can be computed using boolean set-operations. Experiments showed computing $\mathcal{U}_\delta$ this way cost about 24 minutes, as compared to 2 minutes when constructing $\mathcal{U}_\delta$ from the α-shape. The region has a circular arc around each vertex v of the α-shape, with $\alpha = \delta$. The arc's source and target are at distance δ from both v and its adjacent vertices in the α-shape. We use CGAL's boolean set-operations to check if the convex hull is δ-covered and to compute the part of the trajectory region that is not δ-covered.

The events are sorted by angle to the positive x-axis. For two events with the same angle, an event that makes the rectangle more δ-covered has precedence. This is because we define rectangles for which the strict interior is inside the δ-coverage region as δ-covered. If the rectangle becomes δ-covered at a certain angle and stops being δ-covered at the same angle, at this rotation angle the rectangle is δ-covered.

4 Experiments

As can be seen in Figure 6, the aerial data sets do not have a globally consistent density. There are many reasons for this difference in density, including occlusion, surface reflectiveness, measurement angle, and the combination of different flight paths into one data set. In general, vertical surfaces have a significantly lower density than horizontal and diagonal surfaces. For this reason separate results are presented for vertical (sparse) and non-vertical (dense) surfaces. In the ground-based data sets, the surface density fluctuates less, but because of the differences in the scanning device the results from the ground-based data sets are also presented separately.

The experiments performed using the algorithm have two main goals. Firstly, we aim to find a value of δ for which the algorithm can correctly identify which planes are rectangular. To quantify this, we have manually classified thirteen data sets of different regions into rectangular and non-rectangular surfaces and we compare the identification done by our algorithm to this manual classification. Another way of determining the quality of the algorithm is from the produced ranges of rotation angles. If many of the ranges are large, δ may very well have been chosen too large for the data set.

Secondly, we aim to inspect whether the rectangles produced by our algorithm are useful for creating a closed geometric model of the world. This can be determined if the plane has neighboring surfaces nearby. To create a closed geometric model incorporating both surfaces, they should be connected along an edge of both boundaries. Therefore, a rectangle is probably a correct boundary if the range of allowed angles includes the angle of the intersection lines with its neighboring surfaces.

4.1 Data sets

We evaluated our algorithm using thirteen large laser range data sets of Rotterdam, Vlaardingen, and Enschede in The Netherlands. The first six regions were scanned in Rotterdam using an aerial laser scanner that complies with the specifications given in [10]. This data is the combination of ten flights over one district and each data set is restricted to a range of x- and y-coordinates. The other seven regions were scanned in Vlaardingen and Enschede using a ground-based laser scanner. Because of the uneven density of these ground-based data sets, they were subsampled using a 3D grid with 5 cm edge lengths. One random point per grid cell was kept. Note that no benchmark data sets of urban scenes were available to compare our algorithm on.

As a pre-processing step, we applied the Efficient RANSAC algorithm [16] to cluster the point data into planes. Because of the differences in the scanners, we used slightly different settings for the aerial and ground-based data. In the aerial sets, we estimated the normal of a point based on its 12 nearest neighbors; points within 6.5 cm of a plane are counted as support; the point normals could deviate from the local surface normal by 20 degrees; the connected component bitmap size was 25 cm; the minimal support set size was 250 points; and the maximum probability of overlooking a better cluster was 0.001. In the ground based sets, points within 3.5 cm of a plane are counted as support; the connected component bitmap size was 15 cm; all other settings were the same. These parameters showed good clustering performance on the data set when compared to other values.

4.2 Results

One of the regions of Rotterdam is shown in Figure 1 with its points colored by cluster. Figure 7 shows the rectangles that were identified in the same data set with δ set to 60 cm for the dense surfaces and 125 cm for sparse surfaces. For each of the "rectangular" clusters, a range of bounding rectangles is δ-covered and Figure 7 shows the extrema of these ranges and a rectangle that fits well with the neighboring surfaces.

Figures 11 and 12 show the results in some more of our data sets. In these figures, the displayed rectangles are chosen from the ranges of allowed rectangles. This choice is based mainly on the neighboring surfaces. Note that in Figure 11 (center row, right image) the blue and turquoise balconies are correctly estimated, even though their neighboring surface does not have any points near the connecting edge.

Because a ground truth for our laser scans is absent, we have manually classified our data sets into rectangular and non-rectangular planes. About 35% of the planes were classified as rectangular. The remaining planes had various other shapes, like trapezia, L-shapes, etc. However, no other shape was as prevalent as the rectangle. This percentage of rectangular planes is for our data sets of European cities. We expect that the prevalence of rectangles in American cities is even higher.

We test our algorithm on three criteria: correctness of identification, correctness of the boundary, and correctness within the scene. We also compare our results to the α-shape. However, there is no clear metric for this comparison, so they are compared visually. Figure 7 shows the data of Figure 1 with identified rectangles.

For the correctness of identification, we compare our results to another classification. Because there is no ground truth, we have manually classified which clusters are rectangular or not. Rectangles are the most prevalent shapes at 35% of the planes; the remainder has various other shapes, like trapezia, L-shapes, etc. We cannot guarantee the correctness of the manual classification, but it is interesting to analyze the differences in the results of the approaches. For convenience, we

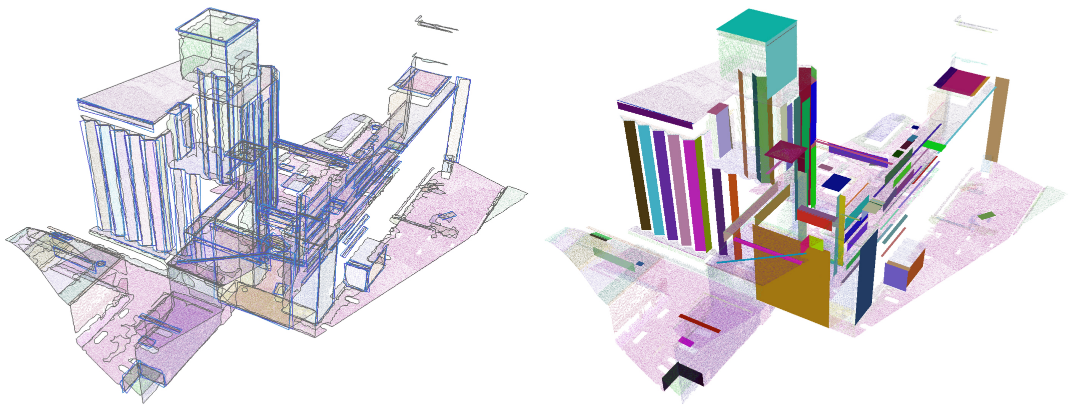
Figure 7: Identified rectangles, for $\delta = 60$ cm for dense and 125 cm for sparse surfaces. Top: the α-shape of each surface (grey outlines) and the extrema of the ranges of rectangles for surfaces on which they are identified (blue outlines). Bottom: appropriate δ-covered rectangles: these rectangles have an edge parallel to a neighboring surface.

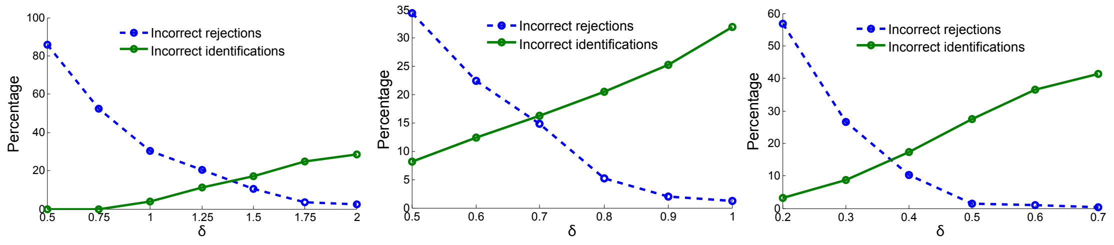

Figure 8: The percentage of aerial sparse (left), aerial dense (center), and ground-based (right) clusters for which a rectangle was falsely identified or rejected at different δ values. The percentages of correctly identified or rejected boundaries are not shown.

assume our manual classification is fully correct. Therefore our algorithm can err by producing falsely identified (I_f) and falsely rejected (R_f) rectangles, as shown in Figure 8.

We determine the correctness of the boundary from the freedom we have in choosing its size and rotation. A large freedom may indicate the value of δ is chosen too large for the data set. We measure this freedom by the size of the angle ranges for which a rectangular boundary is allowed. Smaller ranges indicate there is less leeway in choosing a rectangle for the point set and for some small δ the point set can no longer correctly be bounded by a rectangle. On the other hand, larger ranges indicate there is more freedom in choosing a rectangle and for some large δ all minimum bounding rectangles are δ-covered by the point set. The means of these sizes are shown in Figure 9. Although not shown, the standard deviations of these ranges have about the same value as their mean. This large variation may be caused by the large difference in density and size of the rectangles.

The correctness within the scene expresses whether boundary fits in the scene. Any boundary fits within the scene, if it can be connected to its neighbors along its edges. In our case, there should be a rectangle with an edge parallel to a neighbor. All sparse and 67% of the dense and ground-based rectangles have a neighbor; roughly 80% of these has an edge parallel to at least one of these neighbors, as shown in Table 1. These results were obtained at $\delta = 60$ cm for dense surfaces, 125 cm for sparse surfaces, and 40 cm for the ground-based data. Once the correctness of an intersection line is determined, it is straightforward to align the rectangle to its neighboring surface.

Most related methods in 2D shape reconstruction require different input and output, as described in Section 2. They usually produce one shape with all edges between data points. In

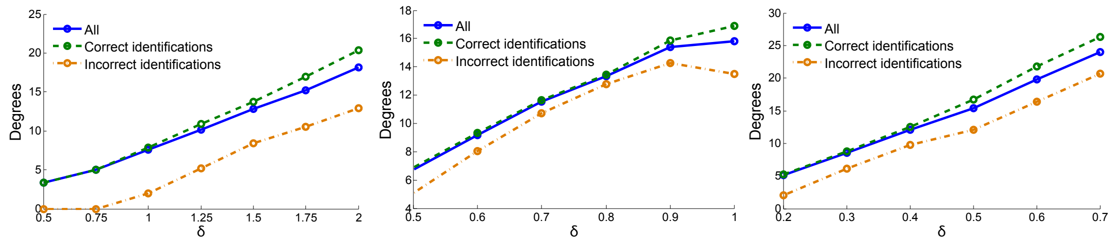

Figure 9: The mean range of allowed rotation angles for aerial sparse (left), aerial dense (center), and ground-based (right) clusters for which a rectangle was correctly or incorrectly identified at different δ values.

	Dense	Sparse	Ground
Correct identifications	89.167%	92.105%	72.464%
Incorrect identifications	68.421%	46.154%	41.667%

Table 1: The percentage of rectangular surfaces with a nearby neighbor that contain the angle of the intersection line with one of these neighbors in the range of angles.

contrast, we produce all fitting rectangles. The efficiency of our algorithm given in Subsection 3.2 is at least as good as that of the methods presented in Section 2, and one could argue that the related methods are more robust to outliers. Besides efficiency and robustness, our algorithm may be compared to these other methods on their results. Because there is no metric that expresses how well a boundary fits a cluster for 3D geometry reconstruction, we compare our results to the 2-dimensional α-shape by visual inspection. Figure 10 shows both the α-shape and the range of δ-covered rectangles for some clusters.

4.3 Speed

The algorithm has been timed using thirteen real-world data sets of different regions. These sets have an average of 147 planes, containing an average of 7432 points per surface, as shown in Table 2. The average running time using one processor of a 64bit Core 2 Duo 3.0 GHz with 2GB of RAM memory was 5.6 minutes per data set. The largest time investment was computing the α-shape at 38% of the total time cost, followed by constructing the δ-coverage region at 21% of the time cost. The relative times for some of the computations are shown in Table 3. The time cost could be halved by making the algorithm multi-threaded. On the same computer, preprocessing

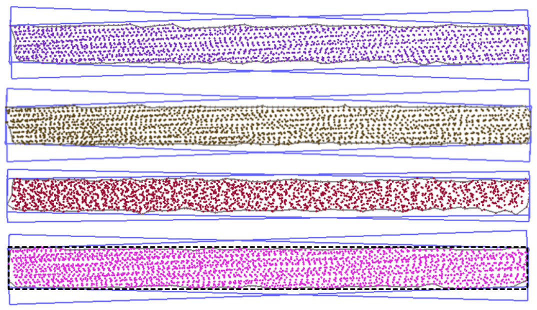

Figure 10: A few surfaces in the data set with an average of 2163 points per surface. The grey outlines show the 2-dimensional α-shape of each cluster. The blue outlines show the extrema in the range of δ-covered rectangles. The black dashed outline around the bottom cluster shows another rectangle within the allowed range that nicely bounds the data. The lower edge of the black rectangle is parallel to the cluster's neighboring surface (not shown here).

Aerial set	$\lvert P\rvert$	$\lvert S\rvert$	$S:\mu\lvert P\rvert$	$\lvert S_d\rvert$	$S_d:\mu\lvert P\rvert$	$\lvert S_s\rvert$	$S_s:\mu\lvert P\rvert$
1	1087158	137	5099.41	119	5568.79	18	1877.44
2	1327707	170	5343.84	148	5790.25	22	2340.68
3	1752129	150	9250.53	109	12001.45	41	1937.10
4	985160	73	10804.97	68	11494.54	5	1426.80
5	1776077	147	8916.46	118	10321.73	29	3198.48
6	1912999	241	6507.19	159	8946.23	82	1771.83
Total	8841230	918	7257.48	721	8671.27	197	2083.16

Ground set	$\lvert P\rvert$	$\lvert S\rvert$	$S:\mu\lvert P\rvert$
1	470771	86	3385.01
2	971637	167	3569.02
3	1166711	196	3615.04
4	1072442	190	3473.08
5	439726	105	2221.17
6	723115	178	2029.77
7	502920	69	5242.36
Total	5347322	991	3240.98

Table 2: The point distribution per region. $|P|$ denotes the number of points per set. $|S|$, $|S_d|$, and $|S_s|$ denote the number of surfaces, dense surfaces, and sparse surfaces respectively. $S : \mu|P|$ denotes the mean number of points per surface.

Aerial set	1	2	3	4	5	6	Total
α-shape	44.35%	41.60%	54.74%	53.32%	47.48%	42.70%	46.96%
$\mathcal{CH}$	4.76%	4.69%	4.31%	4.83%	5.44%	4.80%	4.81%
$\mathcal{U}_\delta$	20.03%	18.83%	14.01%	14.04%	13.72%	17.18%	16.17%
$\mathcal{T}$	3.58%	3.89%	2.08%	1.69%	2.36%	3.64%	2.94%
$\mathcal{CH} \setminus \mathcal{U}_\delta$	7.95%	7.37%	6.12%	6.67%	7.37%	7.02%	7.03%
$\mathcal{T} \setminus \mathcal{U}_\delta$	2.93%	3.07%	1.99%	1.72%	2.83%	3.69%	2.82%
Sort E	0.46%	0.47%	0.27%	0.25%	0.40%	0.59%	0.42%
Handle E	0.00%	0.00%	0.00%	0.01%	0.00%	0.00%	0.00%

Ground set	1	2	3	4	5	6	7	Total
α-shape	24.97%	29.42%	30.34%	28.48%	22.71%	21.41%	38.83%	28.08%
$\mathcal{CH}$	5.68%	5.00%	4.01%	5.32%	4.91%	4.41%	4.77%	4.82%
$\mathcal{U}_\delta$	26.16%	25.75%	28.38%	24.42%	29.40%	30.29%	21.09%	26.58%
$\mathcal{T}$	3.41%	4.22%	4.10%	4.06%	4.87%	5.42%	3.30%	4.21%
$\mathcal{CH} \setminus \mathcal{U}_\delta$	9.09%	8.31%	7.71%	8.29%	8.90%	8.69%	7.38%	8.30%
$\mathcal{T} \setminus \mathcal{U}_\delta$	3.98%	3.89%	3.71%	4.16%	3.87%	4.72%	3.23%	3.99%
Sort E	0.64%	0.65%	0.68%	0.67%	0.63%	0.81%	0.52%	0.67%
Handle E	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Table 3: Time investment for some subroutines. The first six rows show the time investment for constructing the structures; the last two rows show time investment for sorting and handling the events. The columns contain the data sets.

the data using RANSAC took roughly 15 minutes per data set.

5 Evaluation of results

The main application of our algorithm is to identify the clusters that should be bounded by a rectangle, given a point set clustered per plane. Figure 8 shows that the value of δ has a large influence on the number of planes for which a rectangular boundary is appropriate. Many of the incorrect identifications are relatively small planes with a high density; many incorrect rejections are clusters that contain a few points that should have been in another cluster.

At larger values of δ the algorithm produces rectangles that are δ-covered over larger ranges as shown in Figure 9. The main usage of our algorithm is as a first step in the boundary reconstruction process, before handling more complex shapes. Therefore, our algorithm should give few incorrect rectangles while still bounding as many of the planes as possible. Furthermore, the algorithm should produce a good indication for the orientation of the rectangles. Therefore, minimizing the range of rotation angles for the correctly bounded planes is also an important factor. Taking these criteria together, we selected δ at 125 cm, 60 cm, and 40 cm for the aerial sparse, aerial dense, and ground-based surfaces respectively. Using this parameter setting, the average angle range of the correct rectangles is about 10 degrees, while identifying 84% of the rectangular surfaces and producing 15% incorrect identifications.

In the complete reconstruction process, the boundaries should be connected to neighboring surfaces along their edges. Our algorithm produces a range of allowed angles, possibly including the angle of intersection with a neighboring surface. In our data sets almost all sparse and about two thirds of the dense surfaces have a nearby neighbor. Table 1 shows that roughly 80% of the correctly identified angle ranges contain the intersection with at least one neighbor. Visual inspection shows that planes with multiple neighbors can usually be connected to all of these neighbors.

Visual comparison with α-shapes leads to the following observations, shown in Figure 10. It is clear from the figure that the clusters should be bounded by a rectangular shape. Unlike the α-shape, the corners of the rectangles need not lie on a point in the cluster. Furthermore, a rectangle is a much simpler boundary shape: it has four edges compared to the hundreds of edges the α-shapes have. When these surface boundaries are used to reconstruct the scene, they should connect to neighboring planes along straight edges. Connecting two boundaries with long straight edges is easier than connecting two α-shapes, because laser scan points are rarely located exactly on the intersection line. Finally, many of the applications that use geometric reconstruction are web-based. Because sending large data sets through the internet can be slow, more concise representations such as rectangles are preferred.

6 Concluding remarks

We presented a one-parameter algorithm that computes all rectangles that tightly cover a point set in the plane while not containing a part that is too far away from any point. An extension of this algorithm can also handle outliers, points that must remain outside the rectangle. The algorithm is efficient and relatively simple to implement.

We performed a number of experiments with the algorithm on a number of urban data sets. These experiments show there are parameter settings for which sparse and dense planes can be handled properly. When using this parameter setting, there is usually an angle of rotation within the range of δ-covered rectangles for which the rectangle can be connected with the neighboring surfaces along an edge.

Even though there are parameter settings that either minimize incorrectly identified or rejected rectangles, no setting can minimize both, as shown in Figure 8. This is most likely caused by the differences in sampling density of the different surfaces and remaining outliers near the clusters.

A number of extensions may be considered that will make the algorithm more robust to incorrectly clustered point sets. We may automatically compute the appropriate δ value for each surface, possibly based on some density measure. This should greatly reduce the number of errors in the results (both incorrect identifications and rejections), while at the same time removing the need to tweak the parameters for a specific data set. We could even allow different values of δ within the same cluster and change the coverage region accordingly, comparable to a weighted α-shape [1]. However, computing the appropriate value for δ without greatly increasing the computation time is a difficult problem that has been previously studied for the α-shape [5, 12]. A method of automatically computing α for the α-shape will probably work for our algorithm without change, because we only use δ for computing the δ-coverage region; the remainder of the algorithm is independent of δ and instead uses the geometry of the δ-coverage region.

We may include a classifier in the preprocessing that identifies remaining outliers in the cluster. We expect this will greatly reduce the amount of incorrect rejections for smaller values of δ. However, different classifiers produce different results and an extensive analysis should identify the classifier most suited to our method and data. Weyrich *et al.* [22] present three outlier removal methods for interactive point set processing and these methods may also be incorporated in automatic reconstruction.

We may allow a small part of the rectangle to be non δ-covered. Due to minor occlusion, it may be that some points are missing in a region. While ignoring holes in the δ-coverage region is a straightforward solution, this does not solve the problem for sparse regions near the border of the cluster. It seems possible to incorporate this extension using an approximation version.

We may allow a small number of the points to be outside the rectangle. This extension appears considerably harder, and we do not expect to achieve the same running time in this case. Splitting a point set into multiple rectangular clusters may seem like a viable way to handle various rectilinear shapes. However, this procedure suffers from the same problems as allowing some points outside:

determining which points should be selected as inliers for a rectangle is a difficult problem.

It would be useful to extend the algorithm to finding different shapes than rectangles, like L-shapes, without complicating the algorithm too much. An extension to convex polygons with fixed corner angles is achieved by changing the trajectory region and handling the events for each corner separately, as shown in Subsection 3.3. Non-convex polygons pose more difficult problems.

While we have visually compared our results to the α-shape, it may be interesting to develop a metric that expresses how well a boundary fits a cluster for 3D geometry reconstruction. This quality of fit measure should award boundaries that fit with the available neighboring surfaces, and punish boundaries that are more jagged than necessary. Using this metric, we can quantify comparisons between our results and the α-shape or other shapes.

Acknowledgements

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 Ron Wein and Efi Fogel for their help concerning the CGAL boolean set-operations library.

References

[1] Akkiraju, N., Edelsbrunner, H., Facello, M., Fu, P., Mücke, E. P., and Varela, C. Alpha shapes: definition and software. In *Proc. International Computational Geometry Software Workshop* (1995), pp. 63–66.

[2] Alliez, P., Cohen-Steiner, D., Tong, Y., and Desbrun, M. Voronoi-based variational reconstruction of unoriented point sets. In *Proc. Symposium on Geometry Processing* (2007), pp. 39–48.

[3] Amenta, N., Choi, S., Dey, T. K., and Leekha, N. A simple algorithm for homeomorphic surface reconstruction. In *Proc. Symposium on Computational Geometry* (2000), pp. 213–222.

[4] Brenner, C. Building reconstruction from images and laser scanning. *International Journal on Applied Earth Obseravtion and Geoinformation 6*, 3–4 (2005), 187–198.

[5] Cazals, F., Giesen, J., Pauly, M., and Zomorodian, A. Conformal alpha shapes. In *Proc. IEEE VGTC Symposium on Point-Based Graphics* (2005), pp. 55– 61.

[6] Computational Geometry Algorithms Library. http://www.cgal.org/, 2010.

[7] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. *Computational Geometry: Algorithms and Applications*, 3rd ed. ed. Springer-Verlag, 2008.

[8] Edelsbrunner, H., Kirkpatrick, D. G., and Seidel, R. On the shape of a set of points in the plane. In *IEEE Transactions on Information Theory* (1983), vol. 29, pp. 551–559.

[9] Hoffmann, F., Icking, C., Klein, R., and Kriegel, K. The polygon exploration problem. *SIAM Journal on Computing 31*, 2 (2001), 577–600.

[10] John Chance Land Surveys, and Fugro. Fli-map specifications. http://www.flimap.com/site47.php, 2009.

[11] Lipman, Y., Cohen-Or, D., and Levin, D. Data-dependent MLS for faithful surface approximation. In *Proc. Symposium on Geometry Processing* (2007), pp. 59–67.

[12] Mandal, D. P., and Murthy, C. A. Selection of alpha for alpha-hull in R^2. *Pattern Recognition 30*, 10 (1997), 1759–1767.

[13] Melkemi, M., and Djebali, M. Computing the shape of a planar points set. *Pattern Recognition 33*, 9 (2000), 1423–1436.

[14] Nan, L., Sharf, A., Zhang, H., Cohen-Or, D., and Chen, B. Smartboxes for interactive urban reconstruction. *ACM Transactions on Graphics 29* (2010), 1–10.

[15] Schnabel, R., Degener, P., and Klein, R. Completion and reconstruction with primitive shapes. *Computer Graphics Forum 28* (2009), 503–512.

[16] Schnabel, R., Wahl, R., and Klein, R. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (2007), 214–226.

[17] Schwalbe, E., Maas, H.-G., and Seidel, F. 3D building model generation from airborne laser scanner data using 2D GIS data and orthogonal point cloud projections. In *Proceedings of ISPRS WG III/3, III/4, V/3 Worksh. Laser Scanning* (2005), pp. 12–14.

[18] The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/, 2010.

[19] Toussaint, G. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON* (1983), pp. A10.02/1–4.

[20] Tseng, Y.-H., Tang, K.-P., and Chou, F.-C. Surface reconstruction from lidar data with extended snake theory. In *Proc. International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition* (Berlin, Heidelberg, 2007), EMMCVPR'07, Springer-Verlag, pp. 479–492.

[21] Veltkamp, R. C. The γ-neighborhood graph. *Computational Geomerty Theory and Applications 1*, 4 (1992), 227–246.

[22] Weyrich, T., Pauly, M., Keiser, R., Heinzle, S., Scandella, S., and Gross, M. Post-processing of scanned 3D surface data. In *VGTC Symposium on Point-Based Graphics* (2004).

[23] Zebedin, L., Bauer, J., Karner, K., and Bischof, H. Fusion of feature- and area based information for urban buildings modeling from aerial imagery. In *Proc. European Conference on Computer Vision* (2008), pp. 873–886.

[24] Zhou, Q.-Y., and Neumann, U. Fast and extensible building modeling from airborne LiDAR data. In *Proc. ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2008), pp. 1–8.

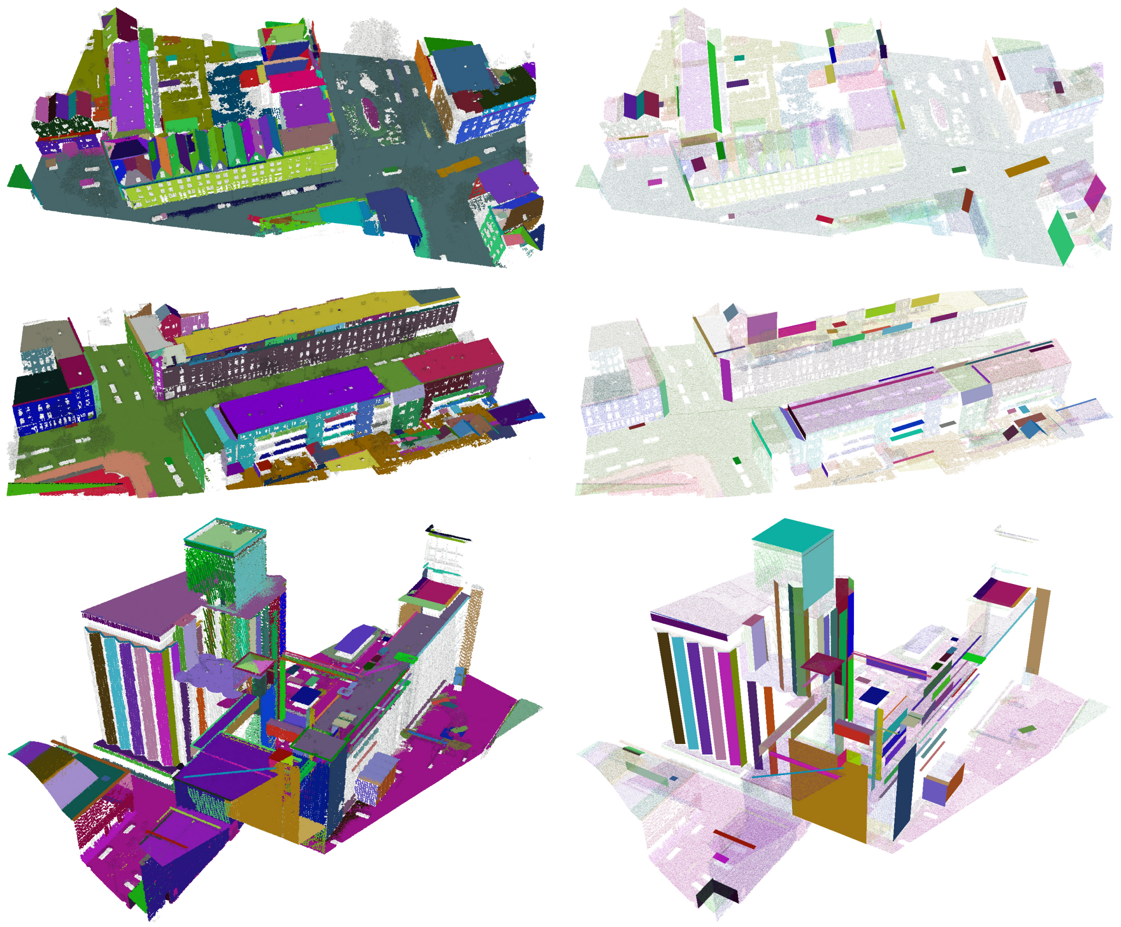

Figure 11: Some examples of our aerial data sets. The left figures show the point data colored by cluster; Points that could not be assigned to a cluster are shown transparent black. The right figures emphasize the rectangular surfaces. All rectangles shown are δ-covered. If there is a δ-covered rectangle that fits the intersection with a neighboring surface, this rectangle is shown. Otherwise, rectangles with a horizontal edge are preferred; if there is no δ-covered rectangle with a horizontal edge, an arbitrary δ-covered rectangle is shown. For these examples, δ is set to 60 cm for dense surfaces and 125 cm for sparse surfaces.

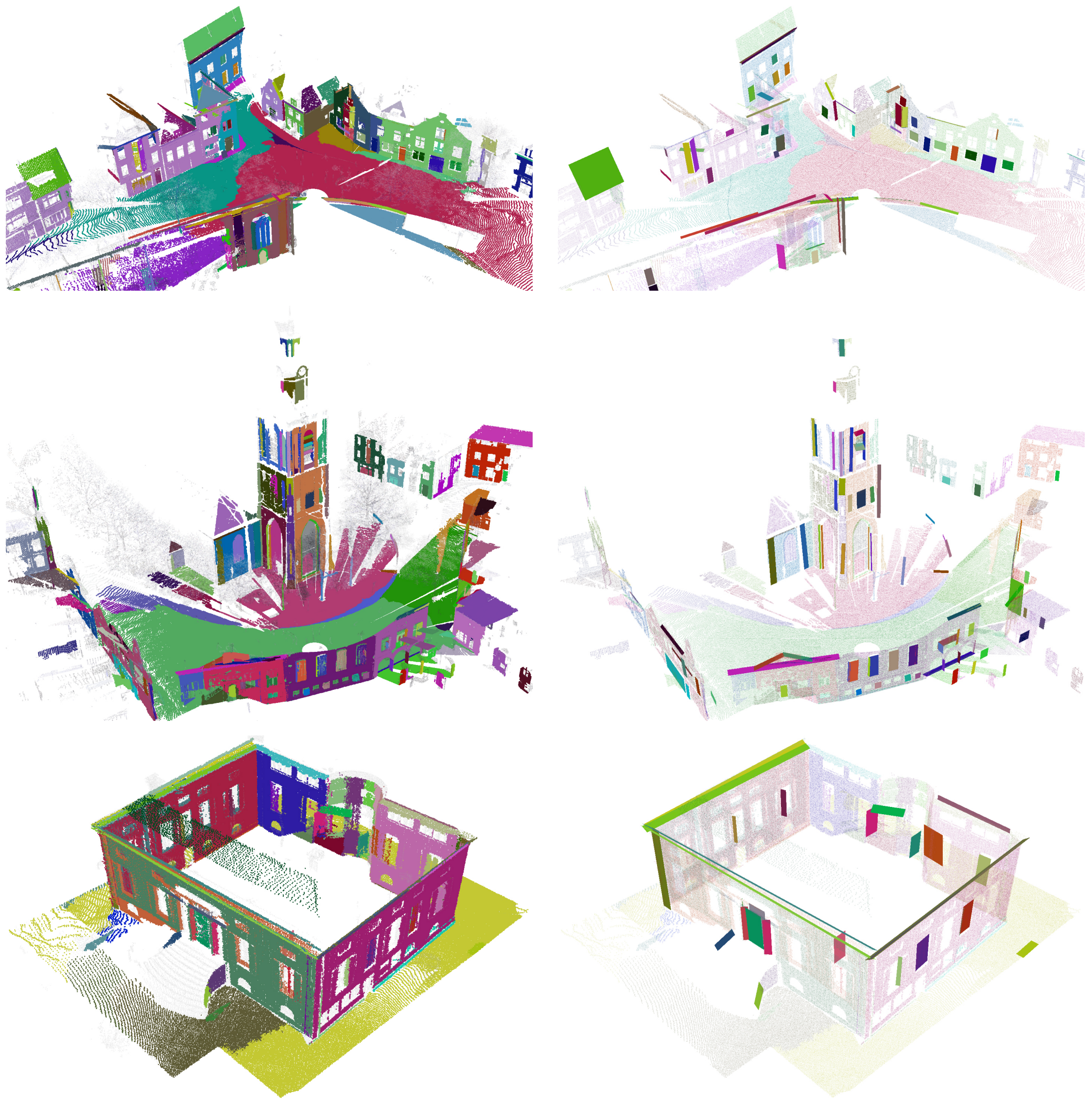

Figure 12: Some examples of our ground-based data sets. The left figures show the point data colored by cluster; Points that could not be assigned to a cluster are shown transparent black. The right figures emphasize the rectangular surfaces. All rectangles shown are δ-covered. If there is a δ-covered rectangle that fits the intersection with a neighboring surface, this rectangle is shown. Otherwise, rectangles with a horizontal edge are preferred; if there is no δ-covered rectangle with a horizontal edge, an arbitrary δ-covered rectangle is shown. For these examples, δ is set to 40 cm.