Identifying rectangles in laser range data for urban scene reconstruction

Thijs van Lankveld[a,*], Marc van Kreveld[a], Remco Veltkamp[a]

[a] *Utrecht University, Department of Information and Computing Sciences, Princetonplein 5, 3584CC, Utrecht, The Netherlands*

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 evaluate our method experimentally on thirteen urban data sets and we compare the rectangles found by our algorithm to the α-shape as a surface boundary.

1. Introduction

Automatic reconstruction of 3D geometric models is currently seeing an increase in demand. Applications like navigation, serious games for training, and urban planning require detailed and accurate models of very large data sets. Because of the 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. Photographs are easily obtained and ground-based and aerial views can quickly cover a large region. 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 data.

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

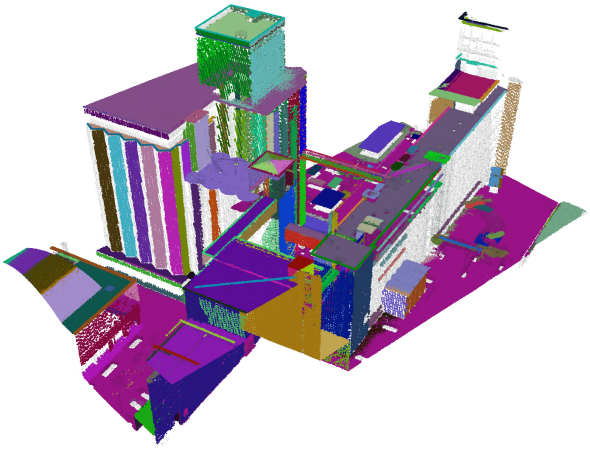

Figure 1: One of the regions used in the experiments. Each cluster has its own color and remaining unclustered points are black. A detail of this scene is shown in Figure 4.

We present an efficient algorithm that identifies the planes for which a rectangular boundary fits the data. We define "fitting" in a one-parameter coverage criterion that ensures a minimum local density across the whole rectangle. For each rectangular surface the algorithm produces the range of rotation angles for which a rectangle fits the point set. Experimental results show the quality of the algorithm in identifying 'rectangular' point sets.

*Corresponding author

Email addresses: thijsvl@cs.uu.nl (Thijs van Lankveld), marc@cs.uu.nl (Marc van Kreveld), Remco.Veltkamp@cs.uu.nl (Remco Veltkamp)

1.1. Problem

We aim to find the clusters of points in a plane for which a rectangle is the correct boundary. Intuitively, a rectangle fits 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 use the δ-coverage region to determine the correctness of a boundary, because it has a clear geometric meaning. Besides indicating the location of the point set, this structure also is a way to handle any noise model if we can expect an error less than δ. The value of δ has another intuitive geometric meaning: the choice of δ is tied to the resolution of the laser scanner, as each disk inside the boundary should ideally contain multiple data points.

Obviously, 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. When reconstructing a complete scene the different rectangles should be connected along an edge. Therefore, we don't seek one optimal rectangle, but the class of all δ-covered rectangles from which an appropriate rectangle can be selected.

1.2. Overview

After examining related work in Section 2, we present our algorithm in Section 3. Our algorithm identifies 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. We present the setup and results of our experiments in Section 4. We evaluate these results in Section 5 and discuss the broader implications 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. Our resulting geometry is simpler than smooth surfaces, and our paradigm is more general than building-model fitting, 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.
- An analysis of the parameter settings for which the algorithm works best. The 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 be divided into interactive and automatic methods. While interactive methods like SmartBoxes [14] significantly speed up manual reconstruction, their reliance on a human operator makes them less appropriate for handling massive datasets 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, which 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. 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 modeling only roof planes and vertical walls [24]. This way the scene can still be reconstructed 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 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 this fits the data. In other cases, our building shapes are not limited to models determined a priori.

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 creating an ex-

plicit geometric boundary for these clusters. Schnabel *et al.* used the primitives to create an implicit model of the scene and made it explicit using marching cubes [15]. However, marching cubes output lacks the simplicity and elegance of the primitive shape geometry.

Various methods have been developed for computing 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 [22], 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, and our results are compared to the α-shape in Section 4.1.

3. Rectangular boundaries

Our algorithm determines whether a cluster of points in a plane can be bounded by a rectangle that 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 [21]. 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, but for completeness we first explain the preprocessing step used to cluster the data as well as the α-shape.

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.

More details on Efficient RANSAC are provided in [16], including its usage to identify and remove points in vegetation. Note that identifying planar clusters and projecting the noisy points onto these planes 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 $\alpha = \delta$. Additionally, the δ-coverage region has a buffer around the α-shape, which may contain a rectangular boundary.

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]. These events can be determined from the combination of two structures: the δ-coverage region, and the trajectory of the corners of $\mathcal{R}$, as shown in Figure 2.

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].

There are two ways for $\mathcal{R}$ to enter or exit the δ-coverage region. Either during rotation an edge of $\mathcal{R}$

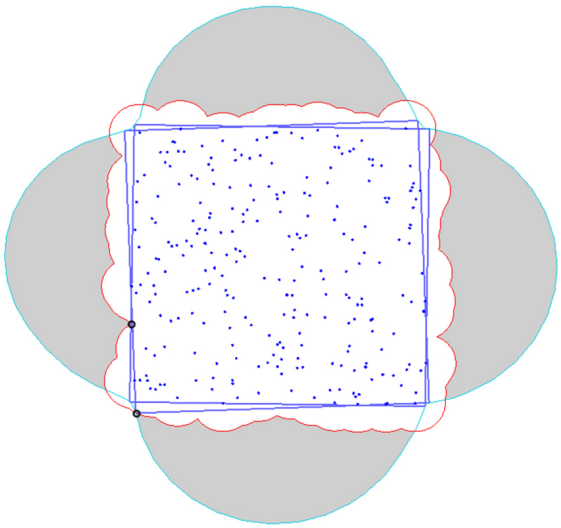

Figure 2: The different structures used during our algorithm and the extrema of the range of allowed rectangles. The blue points were sampled uniformly in a unit square. The δ-coverage region, with $\delta = 0.1$, is bounded by the red arcs, the trajectory region is bounded by the blue arcs. The grey 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.

passes over a vertex of $\mathcal{U}_\delta$, or a corner of $\mathcal{R}$ crosses the boundary of $\mathcal{U}_\delta$, shown by the upper and lower event in Figure 2 respectively. Each vertex of $\mathcal{U}_\delta$ will be inside the rectangle at some rotation angle, if and only if it is inside $\mathcal{T}$. Each vertex produces at most 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 $\mathcal{U}_\delta$ 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 $\mathcal{U}_\delta$, but some intersections produce two events if the boundaries of $\mathcal{U}_\delta$ and $\mathcal{T}$ touch, but do not intersect.

All events are inserted into a queue sorted on the angle of rotation at which they occur. After determining the situation 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, these points 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 angles are shown by blue rectangles.

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 our technical report [21].

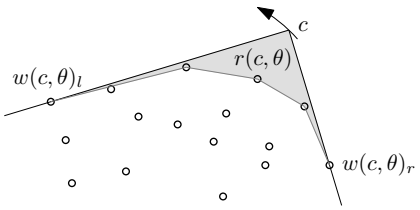

Figure 3: Corner c during rotation, together with the wedge it defines, the two points $w(c,\theta)_l$ and $w(c,\theta)_r$ on the corner's edges, and the grey region $r(c,\theta)$ visible to c.

3.2. *Efficiency*

Our algorithm is based on the rotating calipers algorithm [19] and processes a point set of size n in $O(n \log n)$ time. Proving this time bound requires four parts: constructing the used 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 handling the events takes $O(n)$ time.

The structures we use are the convex hull $\mathcal{CH}$, the δ-coverage region $\mathcal{U}_\delta$, and the trajectory region $\mathcal{T}$. $\mathcal{CH}$ can be constructed in $O(n \log n)$ time [7]. $\mathcal{U}_\delta$ can be directly constructed from the α-shape, which takes $O(n \log n)$ time to construct [8]. Both structures have $O(n)$ vertices. The terms used in the following lemmas and observations are shown in Figure 3. We use ∂X to refer to the boundary of X. Omitted proofs can be found in our technical report [21]

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 $w(c,\theta)_l$,$w(c,\theta)_r$.*

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

Proof sketch. Constructing $\mathcal{T}$ uses a variation of rotating calipers [19] with two orthogonal lines. □

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 sketch. The proof uses the fact that two piecewise simple functions with n curves have $O(n)$ intersections. By simple we mean that two such curves can intersect each other only a constant number of times. We obtain these functions by rotation around $\mathcal{CH}$ and recording the distance to $\mathcal{U}_\delta$ and $\mathcal{T}$. The complete proof is in our technical report [21]. □

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 sketch. Each vertex of $\mathcal{U}_\delta$ and $\partial\mathcal{U}_\delta \cap \partial\mathcal{T}$ can cause events. Events occur when a line tangent to such a vertex and $\mathcal{CH}$ contains an edge of the rectangle. Computing these tangents takes $O(n \log n)$ time. □

From the lemmas given above, we conclude:

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

3.3. 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.

To greatly reduce the size of the involved point set, we first calculate the α-shape, using the same value for α as set for δ. This produces the same results, because all structures can be computed using a subset of the vertices of the α-shape. More details on the implementation can be found in our technical report [21].

4. Experiments

We evaluated our algorithm using thirteen dense urban laser range data sets. The first six sets were scanned 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 sets were scanned using a ground-based laser scanner. Note that no benchmark data sets of urban scenes are available to compare our algorithm on.

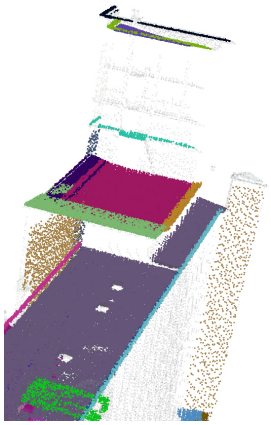

Figure 4: Vertical surfaces have a lower density than the other surfaces. This is a detail of Figure 1.

	NN	Δd	Δn	Cb	$\|S\|$	$P(miss)$
Aerial	12	6.5 cm	20°	25 cm	250	0.001
Ground	12	3.5 cm	20°	15 cm	250	0.001

Table 1: The settings for Efficient RANSAC: the number of nearest neighbors for normal estimation (NN), the maximal deviation between plane and support in distance (Δd) and normal (Δn), the connected component bitmap size (Cb), the minimal support set size ($|S|$), and the probability of missing a better cluster ($P(miss)$).

The data sets do not have a globally consistent density, as shown in Figure 4. This inconsistency has various causes, including occlusion, scanning angle, and aggregating multiple scans. In aerial data, vertical surfaces have a significantly lower density than horizontal and diagonal surfaces. In ground-based data, the density is very high near the scanner. We present separate results for the aerial vertical (sparse) and non-vertical (dense) data. We have subsampled the ground-based data by overlaying a 3D grid with 5 cm edge lengths and keeping one random point per grid cell.

As a pre-processing step, we applied the Efficient RANSAC algorithm [16] to cluster the point data into planes. The settings for aerial and ground based data are shown in Table 1. These parameters showed good clustering performance on the data set when compared to other values. One data set is shown in Figure 1 with its points colored by cluster.

4.1. Results

We test our algorithm on three criteria: correctness of identification, correctness of the boundary, and correct-

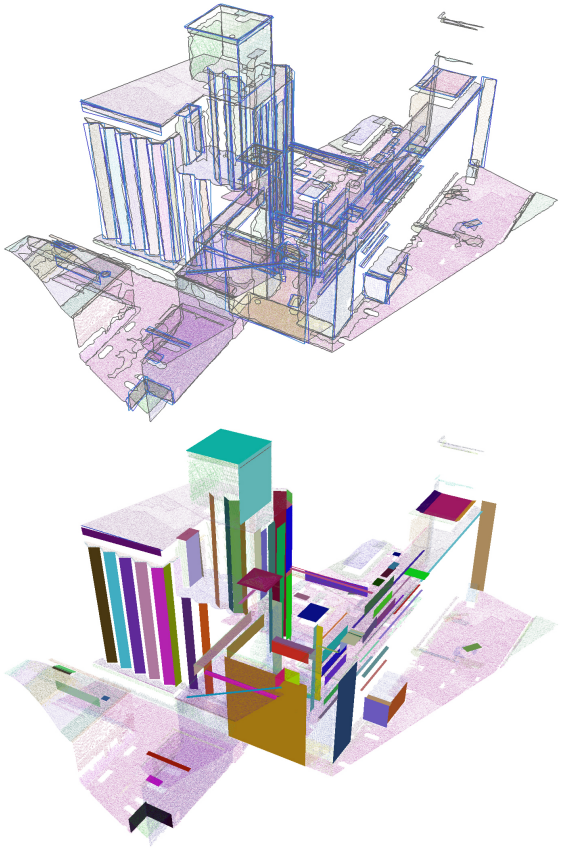

Figure 5: Identified rectangles, for $\delta = 60$ cm for dense and 125 cm for sparse surfaces. Top: the α-shape of each surface (grey outlines) and the extrema of the ranges of rectangles for surfaces on which they are identified (blue outlines). Bottom: appropriate δ-covered rectangles: these rectangles have an edge parallel to a neighboring surface.

ness 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 5 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 determined classification C_M of 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 C_M, but it is interesting to analyze the differences in the results of the approaches. For convenience, we assume C_M is fully correct. Therefore our algorithm can err by producing falsely identified (I_f) and falsely rejected (R_f) rectangles, as shown in Figure 7.

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. The means of these sizes are shown in Figure 8.

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

Table 2: The percentage of rectangular planes with a nearby neighbor that have a boundary edge parallel to one of these neighbors; $\delta = 60$ cm (dense), 125 cm (sparse), and 40 cm (ground-based).

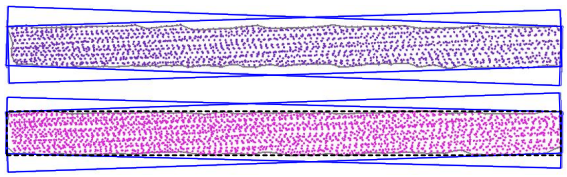

Figure 6: Some clusters of about 2200 points each, with their α-shape (grey outlines) and the extrema of the δ-covered rectangles (blue outlines). The black dashed outline in the bottom cluster shows another allowed rectangle that nicely bounds the data.

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 2.

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 contrast, we produce all fitting rectangles. Figure 6 shows both the α-shape and the range of δ-covered rectangles for some clusters.

4.2. 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. 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. On the same computer, preprocessing the data using Efficient RANSAC took roughly 15 minutes per data set.

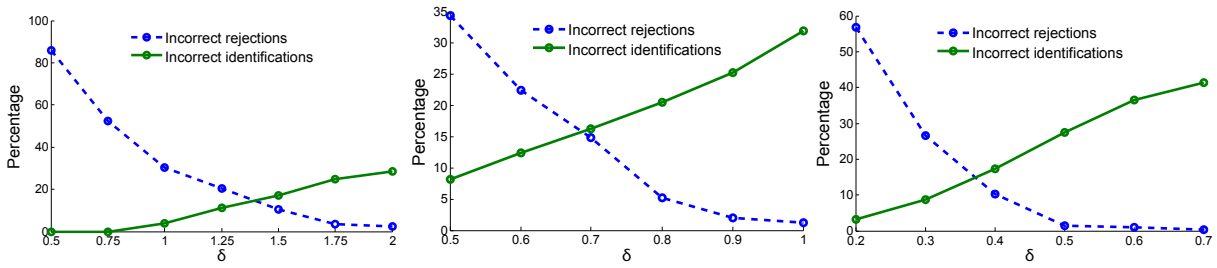

Figure 7: The percentage of aerial sparse (left), aerial dense (center), and ground-based (right) clusters for which a rectangle was falsely identified or rejected at different δ values. The percentages of correctly identified or rejected boundaries are not shown.

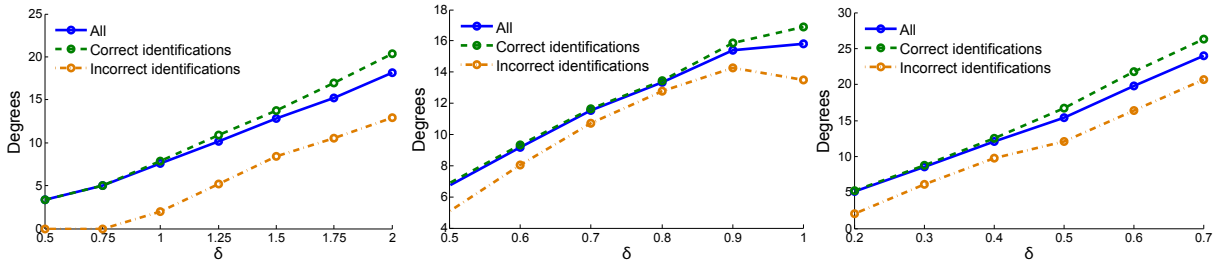

Figure 8: The mean range of allowed rotation angles for aerial sparse (left), aerial dense (center), and ground-based (right) clusters for which a rectangle was identified correctly or incorrectly at different δ values. Although not shown, the ranges vary widely at one δ.

5. Evaluation of results

Figure 7 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. Classifying these points as outliers during preprocessing may remove many of these incorrect rejections.

The ranges for which the rectangles are δ-covered grow as δ increases, as shown in Figure 8. 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 an important factor. Taking these criteria together, we selected δ at 60 cm, 125 cm, and 40 cm for respectively the aerial dense, aerial sparse, and ground-based surfaces. Using this parameter setting, the average angle range of the correct rectangles is about 10 degrees, while identifying about 84% of the rectangular surfaces and producing about 15% incorrect identifications.

Visual comparison with α-shapes leads to the following observations, shown in Figure 6. It is clear from the figure that these clusters should be bounded by a rectangular shape. Rectangles are a much simpler boundary shape: they have four edges compared to the hundreds of edges of the α-shapes. When these 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 two α-shapes, because laser scan points are rarely located exactly on the intersection line.

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. 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 7. 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. We may use a density measure to automatically compute the appropriate δ value for each surface or surface region, similar to the weighted α-shape [1, 5, 12]. This should remove the need to tweak the parameters for a specific data set.

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 problems with sparse regions near the border of the cluster. Similarly, we may allow a small number of the points to be outside the rectangle.

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

[1] Akkiraju, N., Edelsbrunner, H., Facello, M., Fu, P., Mücke, E. P., Varela, C., 1995. Alpha shapes: definition and software. In: Proc. 1st Int. Comp. Geom. Software Workshop. pp. 63–66.

[2] Alliez, P., Cohen-Steiner, D., Tong, Y., Desbrun, M., 2007. Voronoi-based variational reconstruction of unoriented point sets. In: Proc. SGP '07. pp. 39–48.

[3] Amenta, N., Choi, S., Dey, T. K., Leekha, N., 2000. A simple algorithm for homeomorphic surface reconstruction. In: Proc. SoCG '00. pp. 213–222.

[4] Brenner, C., 2005. Building reconstruction from images and laser scanning. Int. J. App. Earth Obs. Geoinf. 6 (3–4), 187–198.

[5] Cazals, F., Giesen, J., Pauly, M., Zomorodian, A., 2005. Conformal alpha shapes. In: Proc. VGTC Symp. on Point-Based Graph. pp. 55– 61.

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

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

[8] Edelsbrunner, H., Kirkpatrick, D. G., Seidel, R., 1983. On the shape of a set of points in the plane. In: IEEE Trans. Inf. Th. Vol. 29. pp. 551–559.

[9] Hoffmann, F., Icking, C., Klein, R., Kriegel, K., 2001. The polygon exploration problem. SIAM J. Comp. 31 (2), 577–600.

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

[11] Lipman, Y., Cohen-Or, D., Levin, D., 2007. Data-dependent MLS for faithful surface approximation. In: Proc. SGP '07. pp. 59–67.

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

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

[14] Nan, L., Sharf, A., Zhang, H., Cohen-Or, D., Chen, B., 2010. Smartboxes for interactive urban reconstruction. ACM Trans. Graph. 29, 1–10.

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

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

[17] Schwalbe, E., Maas, H.-G., Seidel, F., 2005. 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. pp. 12–14.

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

[19] Toussaint, G., 1983. Solving geometric problems with the rotating calipers. In: Proc. IEEE MELECON '83. pp. A10.02/1–4.

[20] Tseng, Y.-H., Tang, K.-P., Chou, F.-C., 2007. Surface reconstruction from LiDAR data with extended snake theory. Vol. 4679 of Lecture Notes in Computer Science. pp. 479–492.

[21] van Lankveld, T., van Kreveld, M., Veltkamp, R., 2011. Identifying rectangles in laser range data for urban scene reconstruction. Tech. Rep. UU-CS-2011-004, Utrecht University, Department of Information and Computing Sciences.

[22] Veltkamp, R. C., 1992. The γ-neighborhood graph. Comp. Geom. Th. App. 1 (4), 227–246.

[23] Zebedin, L., Bauer, J., Karner, K., Bischof, H., 2008. Fusion of feature- and area based information for urban buildings modeling from aerial imagery. In: Proc. ECCV '08. pp. 873–886.

[24] Zhou, Q.-Y., Neumann, U., 2008. Fast and extensible building modeling from airborne LiDAR data. In: Proc. ACM GIS '08. pp. 1–8.