Large Scale Shape Reconstruction from Urban Point Clouds

T. van Lankveld

ISBN 978-90-393-6038-5

Cover design by Thijs van Lankveld.

Typeset in LaTeX
Printed in the Netherlands by Drukkerij De Pandelaar,
http://www.drukkerijdepandelaar.nl/.

The research presented in this thesis 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).

Large Scale Shape Reconstruction from Urban Point Clouds

Vorm reconstructie op grote schaal
vanuit punt data van stedelijke gebieden
(met samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen op
vrijdag 25 october 2013
des middags te 2.30 uur

door

Thijs van Lankveld

geboren op 9 juli 1983
te Veghel

Promotoren: Prof.dr. M.J. van Kreveld
Prof.dr. R.C. Veltkamp

Contents

Chapter 1

Introduction

"The computer can't tell you the emotional story. It can give you the exact mathematical design, but what's missing is the eyebrows."

— Frank Zappa

This dissertation strives to collect a body of research and to present the resulting insights in a coherent way. As the lengthy title may suggest, the subject matter presented in this work comprises a specific, focused challenge. We can split this challenge, or problem, into more manageable chunks by considering it one part at a time. The core of the problem is shape reconstruction, the goal of which is to recreate shapes and spatial relationships from measured data. This aspect lies at the intersection of several fields of research such as Computer Vision and Geometry Processing. The data we feed into these methods comes from large, urban, architectural scenes. Using this type of data means we touch on the field of Photogrammetry and the size of the data sets encourages efficient processing using techniques from the field of Computational Geometry.

As the previous paragraph shows, the research problem tackled in this dissertation brings together a number of fields in Computer Science. The following sections aim to familiarize the reader with the different aspects of our main problem. A reader who is already familiar with the fields may skip to Chapter 2 for a more technical description of our methodology and related work. Section 1.1 gives a brief description of shape reconstruction and its related research fields. While we build our methods on a strong theoretical basis, we also require them to work on practical data and within useful applications. Section 1.2 presents a selection of application types that may benefit from our methods. We use urban LiDAR as our main source of data, which is described in Section 1.3. Finally, Section 1.4 describes the organization of the remainder of this thesis and Section 1.5 lists our publications supporting the research in this work.

1.1 Shape Reconstruction

"I saw the angel in the marble and carved until I set him free."

— Michelangelo

This section briefly describes some aspects of shape reconstruction. More detailed information and related work can be found in Section 2.1. However, before we get to the meat of things we provide brief descriptions of the various fields of research related to shape reconstruction.

Geometry is the branch of mathematics devoted to spatial objects and their relations. While this field is theoretical in nature, it is easy to imagine practical examples of it. Example spatial objects include points, lines, and balls. These are usually described using coordinates and other numbers expressing their location and dimensions. Generally, we are interested in determining properties of these objects or relations between them, such as length, distance, or angle. When we describe a triangle by its corner points and the angles in these corners, we are using geometry.

Computational Geometry is a branch of Computer Science concerned with efficient algorithms to solve geometric problems. An algorithm is a formal description of a method that does not require human supervision; it can be viewed as a detailed recipe that a computer can use to turn an input into an output. For geometric problems, this input is generally some collection of spatial objects and the output is either some other collection of objects, or some property of the input. For example, the input could be a collection of points and the output could be the the largest distance between any two points of the input.

The efficiency of an algorithm is generally expressed in its computational complexity. The complexity describes the rate of growth of requirements on valuable resources such as computation time or working memory. The complexity is often given in big-Oh notation: a function on the size of the input. For example, if the time complexity of an algorithm is $O(n)$, where n is the size of the input, the time needed to produce the output is linear in the size of the input. If the input is twice as large, the computations take no more than twice as long. For a time complexity of $O(n^2)$, the time taken grows quadratically in the size of the input. If the input is twice as large, the computations take at most four times as long. This efficiency becomes increasingly important as input data sizes increase, sometimes making the difference between taking a minute or a day of computations.

Geometry Processing deals with the efficient processing of complex geometric models.

It can be seen as a more focused and numerical brother of Computational Geometry. The models are often embedded in 3-space, meaning that they can be manifested as a physical object. Diverse types of processing are covered, such as analysis, simplification, aggregation, and surface reconstruction. Reducing a large collection of points to a small set of characteristic points that best describe the point distribution is an example of Geometry Processing.

Computer Vision concerns methods to process visual data of the real world. This field covers a broad spectrum of processes such as acquisition, analysis, and condensation into high-level knowledge. The visual data can also come in various forms. Images or video are the most common types, but more exotic forms of vision are also covered, such as infra-red distance sensors and multi-dimensional medical scanners. Some common themes to the field are that the data generally comes in the form of regular grids, such as colored pixels, and many techniques mimic human vision systems.

Photogrammetry concerns methods to measure geometric features of objects in photographic images. Example geometric features include the height of a door and the distance between two persons in an image. Many of the techniques employed by Photogrammetry try to measure these properties using concepts such as scaling and perspective. The methods generally aim to maximize the precision of the measurements.

Shape reconstruction lies on the intersection of these fields. We focus on urban scenes and in this case the shapes could be blocks, half-spheres, polygons and line segments, representing buildings, domes, fences, and lamp-posts. Reconstruction is the process of determining which combination of object shapes, called a geometric model, best describes the scene as captured by the measured data. This includes determining if a building should be described by a cuboid, a cylinder, or multiple objects, where these objects should be placed, and at which size and orientation. Reconstruction may include assigning color values to the shapes to produce a 'photo-realistic' model. However, we restrict ourselves to reconstructing the shapes, and ignoring colors.

Within this task, we have to accept that it is unlikely that we can produce a perfect replica of the scene. For example, antennae and other small features may not be present in the data, so we cannot expect these to be reconstructed. Similarly, the data may be so detailed that it describes the creases between the bricks. However, including this detail in the scene geometry would require so much memory that no consumer computer could render it. Within these constraints, we try to reconstruct a model that is both descriptive and concise.

Shape reconstruction can be done manually. For example, a human modeler may study a collection of photographs to determine the shape of the buildings depicted. Another

option is automatic reconstruction, where you let the computer do the work for you. Both options have their advantages. For example, a human is much better at recognizing the buildings in an image and separating them from their background, while a computer never gets bored when it has to repeat this process a hundred times. Perhaps more importantly, the computer can do this work in the background or at night, while the human modeler handles other important tasks. These approaches can also be combined. For example, the computer produces an initial, rough model and the human modeler molds this into a final product.

Public interest in virtual cityscapes has been increasing for some years. This interest manifests itself in various applications, some of which are described in Section 1.2. In reaction to this increased interest, the quality, complexity, availability, and general rate of collection of urban datasets are increasing. The production of data has reached a level where no amount of manual modeling effort could ever hope to process it all. Automatic shape reconstruction is an active field of research, working to overcome this problem.

1.2 Example Applications

> *"In the fields of observation chance favors only the prepared mind."*
>
> — Louis Pasteur

There is a wide range of applications that use virtual cities, or that could benefit from their usage. Some applications have strict restrictions on the type of input or the quality of the output. Others can freely select these to best suit their needs. A general theme to most applications for urban reconstruction seems to be that they benefit from the user recognizing the particular scene.

An application where this recognition aspect is at the core is navigation. Services like Google Maps make easy navigation ubiquitous as long as there is an internet connection available. Google Streetview and Google Earth add a 3D aspect to the traditional 2D maps. While these applications currently make limited use of detailed geometry in their models, this addition could greatly increase the ease of recognizing specific buildings. You may imagine that recognizing the buildings may play an integral role in determining your current location. Mobile navigation devices are also starting to tap the potential of 3D navigation.

Another important application is in serious games: computer games designed to teach the player something. An example is a serious game that trains firefighters or other

emergency response units. In these professions quick decision making may be crucial to succeed and there is a steep cost to failure. This means that rigorous training is important. However, training in an actual burning building may be just as dangerous as going into the field for a new recruit. When certain large events require special attention from these emergency response units, it is beneficial to train for anticipated emergency scenarios in the actual location of the event.

The models can also be used for health care. For example, virtual reality exposure therapy tries to rid people of anxiety disorders or phobias by having them experience the triggering factors in a safer environment. Specifically, the environment used in this case is virtual reality. People experiencing post-traumatic stress disorder may benefit from being exposed to the specific scene in which the trauma took place.

Urban planning and architecture promotion are applications where the simulation aspect of virtual cities may take less precedence than visualization. Architecture firms usually wish to give a preview of their design in the current environment. This is generally done using 2D drawings, but 3D previews may provide a clearer presentation of the vision of the architect.

Finally, many regular computer games wish to place their story in an existing city. These games generally require very high detail from their expansive environments, together with a large amount of controlled interactivity. Even though it is unlikely that automatic shape reconstruction can replace the need for manual work in this application, automatic methods may save months of work.

1.3 Urban Point Data: LiDAR

> *"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."*
>
> — Isaac Asimov

Some data sources are directly useful for reconstruction of urban scenes. Photographs are easily obtained and ground-based and aerial images can quickly cover a large region. Similar to how a person can estimate distance by combining the images from both eyes, certain automatic methods can combine multiple images to reconstruct the scene. 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. There are various laser ranging schemes, but we focus on scanners that measure travel times. These laser scanners measure distance by emitting a laser and measuring the time it takes for the laser to reflect off a surface and return to the sensor. The sensor is further equipped with a precise GPS to transform the distance measurement into world coordinates for the point of reflection. Repeating this many times in different directions and from different locations creates a collection of these reflection points, which we call a point cloud. This process is called LiDAR, which was originally a portmanteau from light-radar but is now commonly taken to abbreviate light detection and ranging.

There are three common platforms for LiDAR. Static terrestrial LiDAR employs a tripod mounted device that scans in a ball pattern. Dynamic terrestrial and aerial LiDAR employ one or more devices mounted on a car, airplane or helicopter that scan in a circular line pattern. These devices achieve a cylinder-like pattern from forward movement. We focus on aerial LiDAR, because this is generally the fastest way to get massive data sets of urban scenes. However, we also evaluate our method in Chapter 3 on terrestrial LiDAR.

In recent years, the resolution of LiDAR scanners has improved drastically. A single aerial scanning pass of a few minutes can produce a dataset with millions of points and a density of 50–100 data points per square meter (John Chance Land Surveys and Fugro 2009). Figure 1.1 shows the data in one of the scenes used in our experiments.

Each point in the data sets we use has three coordinates describing its location. These coordinates are the same as those used by other GPS devices, so the points are already registered to each other and the rest of the world. The device that produced this data set has an accuracy of 8 cm horizontal and 5 cm vertical. This means that the data point may have slightly different coordinates than the actual position of the reflection point, which we call measurement error or noise. Each point also has a color that was measured from a simple integrated camera. As Figure 1.1 shows, the colors of this data set are slightly shifted towards green.

The way the points are measured introduces some interesting patterns in the data. Specifically, between measuring two consecutive points, the sensor is rotated a fixed angle in the plane orthogonal to the direction of travel. What this means in practice is that the points are measured in 'scan lines' orthogonal to the direction in which the aircraft travels. This is not apparent on the horizontal surfaces, which are seen straight on. However, on some vertical surfaces, where the surface was scanned at a small angle, these scan lines are separated by large empty regions. One of the roof surfaces on the left in Figure 1.1 shows this effect.

Figure 1.1: A view of one of the data sets used in our experiments. This image contains roughly three million LiDAR points that were collected during two passes of a helicopter.

Apart from this scan line effect, there are two other effects that result in differences in scanning density. Firstly, it may be obvious that only surfaces seen from the sensor are measured. This effect is called occlusion. Even surfaces that were largely seen from the sensor may have occluded parts under balconies and other obstacles. This occurs to a lesser extent behind trees, because the laser can partially pass through the leaves and the sensor can measure multiple reflection points per laser emitted. Finally, our data sets combine multiple passes and surfaces scanned multiple times will have a higher point density. While generally this results in better models, the variation in local density may also cause problems.

1.4 Organization and Contributions

"Science is organized knowledge. Wisdom is organized life."

— Immanuel Kant

In this chapter we have described the main problem that is the focus of this thesis and we have briefly explored some aspects of this problem, which we we will call urban

reconstruction for simplicity's sake. The next chapter gives an overview of recent research related to urban reconstruction. This is followed by a description of the novel methodology we use to approach urban reconstruction. An important difference with related work is that we reconstruct the scene per surface. This is a type of divide-and-conquer approach, which makes the problem more tractable by dividing it into subproblems. Since most surfaces in urban scenes are planar, these subproblems are 2-dimensional instead of 3-dimensional, further simplifying the problems. Some related methods are used throughout this thesis and Chapter 2 ends with a detailed description of these. Chapters 3 to 5 describe different methods we developed to reconstruct the shape of individual surfaces.

We found that many of the surfaces in urban scenes are not only planar, they are also rectangular. Chapter 3 presents a novel method that determines if a planar point set originates from a rectangular surface. If this is the case, the method constructs a range of rectangles that are plausible candidates for the shape of the surface. The extent of this range is based on a novel concept we introduce, called δ-coverage. The rectangle reconstruction method has only a single parameter that has a clear geometric meaning related to the sampling density. We show that our method is both efficient and capable of computing nice results on real-world data.

Apart from the rectangles, many other surfaces in urban scenes have only straight corners. These shapes are called rectilinear and Chapter 4 describes a novel method to compute fitting rectilinear shapes for any planar point cloud. The fitting criterion is based on a new concept, called the (α,δ)-sleeve. This method has two parameters, related to the sampling density and noise in the point set. We also explore a technique to set these parameters based on the geometry of the data, in case no useful a priori knowledge is available.

Where Chapters 3 and 4 produce shapes with some fixed properties, Chapter 5 handles general shapes based on the point distribution. Unlike related work, we present novel shape concepts that are based on lines as well as points. For the guided α-shape we use the data from neighboring surfaces as additional cues to determine the boundary. When two surfaces lie next to each other, they should meet in the intersection line of their supporting planes. In other words, both their shapes should use this line as part of their boundary. As a thought experiment, we also present the generalized α-shape, which views lines as infinite sets of points that should be part of the interior of the shape. We show that both these novel geometric objects can be efficiently constructed. We also show that guided α-shapes are well-suited to urban reconstruction.

Chapter 6 covers the end of the divide-and-conquer methodology by presenting a novel method that combines the shapes of the surfaces back into a single geometric model.

Where surfaces do not neatly touch, this method connects them. Similarly, in regions with occlusion or otherwise lacking reconstructed shapes, the method tries to use any available data to fill the gap with an appropriate surface. To achieve this, the method uses both the geometry of the data and the observation that the region between sensor and data must be outside the objects. In order to correctly incorporate the geometry of the data, we require the conforming constrained 3D Delaunay triangulation (CCDT). We present a novel method for constructing the CCDT that avoids computing line-sphere intersections which are often expensive to compute. We also show that our gap-filling method performs well in a comparison with related work.

We conclude this thesis in Chapter 7 with a discussion on the wider implications of our work. Many of the questions that we started from or that came up during our research are answered in this dissertation. However, others remain open. While we have made some steps in the right direction, the problems of urban reconstruction are far from solved. Similarly, it remains important to consider the impact to urban reconstruction outside its own niche.

1.5 Publications

The work presented in this dissertation is based on a number of papers that have been published in conference proceedings, in scientific journals, or as technical reports. The following is an overview of these underlying works.

Chapter 3

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2009). Identifying well-covered minimal bounding rectangles in 2D point data. In *Proc. EuroCG*, pages 277–280

van Lankveld, T., van Kreveld, M., and Veltkamp, R. C. (2011a). Identifying rectangles in laser range data for urban scene reconstruction. *Computers & Graphics*, 35(3):719–725

van Lankveld, T., van Kreveld, M., and Veltkamp, R. C. (2011b). Identifying rectangles in laser range data for urban scene reconstruction. Technical Report UU-CS-2011-004, Utrecht University, Department of Information and Computing Sciences

Chapter 4

van Kreveld, M., van Lankveld, T., and de Rie, M. (2013a). (α,δ)-sleeves for reconstruction of rectilinear building facets. In *Proceedings of the 7th International 3D GeoInfo Conference*, Lecture Notes in Geoinformation and Cartography, pages 231–248. Springer Berlin Heidelberg

Chapter 5

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

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2011b). On the shape of a set of points and lines in the plane. Technical Report UU-CS-2011-017, Utrecht University, Department of Information and Computing Sciences

Chapter 6

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2013b). Watertight scenes from urban lidar and planar surfaces. In *Proc. SGP*, number 12, pages 217–228

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2013c). Watertight scenes from urban lidar and planar surfaces. Technical Report UU-CS-2013-007, Utrecht University, Department of Information and Computing Sciences

Chapter 2

Preliminaries

Over the last few decades, a lot of research effort has been aimed at recreating the world in virtual environments. While expert modelers can accurately transfer the geometry of buildings and streets into a computer model, this takes a significant amount of time and effort. Measurement devices like LiDAR are able to collect data quicker than could feasibly be processed without some sort of automation. With the increasing demand for virtual building models, various companies have been amassing raw data, ready to be processed into city-wide computer models.

This chapter will present a number of earlier methods to automate the shape reconstruction process. While we do not presume to to give an exhaustive overview of the material, we do touch on the different directions in which the state of the art is moving. The next section presents a broad selection of methods aimed at reconstructing shapes, either urban or otherwise closely related.

Unlike most other methods, our methodology has at its core the assumption that urban scenes are mainly composed of simple shapes, particularly planar surfaces. We exploit this assumption by decomposing the point data into planar clusters and reconstructing the model per surface. This divide-and-conquer strategy enables us to process large data sets more efficiently. Additionally, while not employed in the experiments of this dissertation, this strategy makes it straightforward to parallelize a large part of the pipeline, decreasing the time investment even further and optimizing the usage of multi-core processing power. Section 2.2 presents the pipeline that structures our methodology from raw data to completed models and Section 2.3 describes the method we use to cluster the data into planar patches.

Finally, Section 2.4 describes the α-shape, an early yet very elegant method for constructing the boundary of a planar point set. Many of our methods are either based on the α-shape or make use of it. For this reason, Section 2.4 also contains a number of theoretical results on the α-shape, which will be referred back to from later chapters.

2.1 Shape Reconstruction

By shape reconstruction, we refer to the construction of a virtual shape model that captures the geometry of a real world object or scene. This dissertation focuses on urban scenes and we aim to construct geometric models that are detailed and realistic enough to be used for simulation purposes. This means that a resulting model should contain the 3-dimensional geometry of the scene such that a virtual actor could walk around in the model and recognize the reconstructed buildings.

Urban scene reconstruction can be approached from different perspectives and brings together research fields like geo-sciences, computer-aided design, photogrammetry, computer vision, and geometry processing. A recent survey by Musialski et al. (2012) gives an overview of the state-of-the-art in urban scene reconstruction methods. Many of these methods process photographic images and we will limit the overview given here to point-based methods. Even though many reconstruction methods focus on smaller objects scanned in controlled environments, more recent work improves robustness to noise and outliers and the potential of these methods should not be ignored for urban reconstruction.

For the purpose of this overview, we initially differentiate the related work into semi-automatic and automatic methods. This grouping is based on the amount of interaction with a human modeler the method requires. Even though automatic methods can process data sets without intermediate human interaction, most methods require tweaking some parameters to the data. Most automatic methods can be divided into three groups based on the type of model they produce: parametric building models, triangular meshes, or implicit surface models.

2.1.1 Semi-automatic Methods

Semi-automatic methods aid a human modeler in reconstructing a scene. In general, these methods will leave high-level tasks like recognizing objects and determining their geometry to the operator, and they contribute by snapping surfaces so they minimize the distance to the point set and other computationally expensive tasks.

A seminal semi-automatic reconstruction method is Façade (Debevec et al. 1998). Even though this method is image-based it cannot be ignored in this overview, because of the long series of semi-automatic image-based reconstruction methods that followed up on this method. Recent examples include Lee and Nevatia (2004) and Sinha et al. (2008). In the Façade tool, the user constructs a building by combining simple polyhedra. The shape and position of these polyhedra is based on a few images from different view-

points.

A recent point-based semi-automatic reconstruction tool is SmartBoxes (Nan et al. 2010). In SmartBoxes, the user selects part of a point cloud and the tool will fit a rectangular box to this part of the data. Alternatively, the user may define the topology of a collection of boxes and the tool will fit this to the data. Finally, SmartBoxes can use repetition in the point cloud to quickly fit recurring shapes in the data, even in sparse regions.

Semi-automatic methods can greatly speed up the reconstruction process. Additionally, the human modeler can guarantee the resulting model meets very high quality standards. Unfortunately, the inclusion of a human modeler means that these methods cannot scale well with very large data sets.

2.1.2 Parametric Buildings

Parametric buildings are predefined polyhedral shapes with a number of parameters that define their geometry. For example, a rectangular box may have parameters for its length, width, height, position, and horizontal rotation. Parametric building methods have a database of parametric shapes for different types of buildings or parts of a building. These methods select shapes and their parameters to form an optimal fit for the point cloud. For an example, see Lafarge et al. (2010a).

Parametric building methods are mainly employed within the geo-sciences. The reasons for this may be two-fold. Firstly, geo-sciences mainly uses point clouds in the form of a Digital Elevation Model (DEM). A DEM is a 2.5D height model, consisting of a regular grid of height values on the horizontal plane. Most of these DEMs are created from LiDAR data by averaging or selecting one point for each grid cell. Parametric building methods are well-suited for this data type, because they are able to accurately reconstruct buildings from sparse point data with significant noise. A second reason may be that reconstructing complete buildings makes it easier to gather semantic information of the scene, e.g. whether an area is urban, suburban, or industrial.

For convenience, this category also includes the methods that reconstruct only roof planes, because these basically reconstruct parametric sheared prisms (Rottensteiner 2003; Schwalbe et al. 2005; Vosselman et al. 2004; Zhou and Neumann 2008). In general, these methods use some classification procedure on a DEM to extract the points inside each building or part of a building. They then compute the 2D boundary of each building part, possibly supported by a ground plan. Many of the earlier methods only support 90 degree corners for this boundary. Finally, they compute either a fixed height

or a plane that best fits the roof of each building part. These roof patches are connected to the ground using vertical walls by extending the 2D boundary upward.

Two recent papers showcase the strengths of this methodology. Poullis and You (2009) reconstruct buildings with horizontal roofs in massive data sets containing hundreds of buildings. Pu and Vosselman (2009) present a method that reconstructs buildings consisting of solid blocks from their facades in a single terrestrial LiDAR scan, using similar 2.5D techniques.

Parametric building methods have several disadvantages when used to reconstruct detailed scenes. Firstly, they are limited by the type of buildings in their database. Earlier methods used only a handful of building types, but even with considerable effort it seems improbable that all possible building types could ever be parametrized. Secondly, most parametric buildings contain only vertical walls. This means that structures like window sills, alcoves, balconies, and arcades cannot be modeled. Finally, grid-based methods usually have problems coping with irregular sampling causing empty grid cells or multiple measurements per cell.

2.1.3 Triangular Meshes

In geometry processing, most shape reconstruction methods produce a collection of connected triangles that cover the surfaces of the scene. This collection of triangles is generally called a triangular mesh. In general, shape reconstruction methods that produce a triangular mesh will strive to create a mesh that is not self-intersecting, i.e. any pair of its triangles intersects at most in their shared edge.

A large part of the triangular mesh methods are based on the 3-dimensional Delaunay triangulation (DT). This well known geometric structure fills the space between the points with tetrahedra connecting four points that share a sphere with empty interior.

Boissonnat (1984) showed that a shape can be captured in the Delaunay triangulation of a dense point sample of the shape. This observation has resulted in a large number of methods that select a collection of triangles in the DT to estimate the surface of the shape. A recent survey by Cazals and Giesen (2006) describes these methods in more detail, including methods like the lower dimensional localized Delaunay triangulation (Gopi et al. 2000), the Greedy algorithm (Cohen-Steiner and Da 2004), the Crust (Amenta and Bern 1998), Cocone (Amenta et al. 2000), the Power crust (Amenta et al. 2001), the Ball-pivoting algorithm (Bernardini et al. 1999), and Conformal α-shapes (Cazals et al. 2005). Dey and Goswami (2003) have adjusted the Cocone method to produce models that are also watertight, i.e. each triangle edge is incident to exactly

two triangles. Many of these methods require the shape to be r-sampled, which guarantees a certain point density to shape feature size ratio. For a more recent example of reconstruction of r-sampled shapes, see Aichholzer et al. (2009).

Other triangular mesh methods focus on constructing a mesh that follows a smooth surface, e.g. (Lipman et al. 2007). 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 (The Stanford 3D Scanning Repository 2010) are well known examples of this type of objects.

Some triangular mesh methods show that this methodology can also be used to reconstruct buildings (Carlberg et al. 2009; Labatut et al. 2009; Marton et al. 2009; Tseng et al. 2007). These methods can reconstruct buildings of any shape, but most have difficulty dealing with the artifacts inherent in urban point data, like measurement noise and outliers. Additionally, most triangle mesh methods reconstruct smooth surfaces, removing the sharp edges and simple shapes widely present in urban scenes. Other methods take into account the piecewise nature of the scene (Hoppe et al. 1994; Ling et al. 2008). In this case identifying the sharp features becomes the main problem.

A recent method by Lafarge et al. (2010b) combines both triangular meshes and simple primitive shapes, like planes and cylinders. Their method starts from a triangular mesh and iteratively identifies regions that are approximated well by a simple primitive using a simulated annealing procedure they call jump-diffusion. This iterative refinement is guided by a quality term that measures the image-based reprojection error, mesh smoothness, and angles between each pair of primitives. When images of the scene are available, this method may be used to improve the results of a triangular mesh method to better suit an urban scene.

Recent work by Chauve et al. (2010) also falls within the group of triangular mesh methods. However, they construct the mesh following a novel methodology. Their method first identifies points lying on planar surfaces and additional 'ghost' planes that promote vertical walls and orthogonal corners. The identified planes are then used by a greedy heuristic to construct an arrangement that divides the 3-dimensional space into different polyhedral regions. Finally, they assign a weight to each facet of the arrangement and they use a minimum-weight graph-cut algorithm to determine which cells lie in the interior of the buildings or in the exterior, 'free' space. The polygons that separate interior from exterior are triangulated to provide the triangular mesh of the surfaces of the scene.

2.1.4 Implicit Surfaces

Implicit surfaces are structures that are often used in the computer vision field and medical imaging. An implicit surface is a continuous function that is defined over the complete 3-dimensional space and that returns a value close to zero near the surface of the reconstructed object. Usually, this function is discretized over the cells in a voxel grid. An iso-surfacing method like Marching Cubes (Lorensen and Cline 1987) can then be used to construct an explicit surface near the zero-values of the implicit surface.

The goal of an implicit surface method is to construct the function of the implicit surface from the distribution of the point cloud (Hoppe et al. 1992), and possibly an estimation of the surface normal near the points (Alliez et al. 2007). These implicit surfaces can also be constructed using polygons (Shen et al. 2004) and simple primitives like planes and cylinders (Schnabel et al. 2009). Curless and Levoy (1996) have introduced a method that uses lines of sight from the sensor to the measurement points to indicate regions of 3-space that must be outside the object.

Implicit surface methods have three main problems when applied to urban LiDAR. Firstly, like triangular mesh methods, most implicit surface methods will reconstruct smooth surfaces. However, recently Salman et al. (2010) have presented a method that also detects sharp features. Secondly, most implicit surface methods require an input point set with normals oriented outward from the object. Constructing correctly oriented normals from a raw point set remains an important problem for implicit surface models, e.g. (Mullen et al. 2010). Finally, implicit surface methods are not easily scalable to large data sets. In general, the function must be computed for each voxel cell in the bounding box of the scene. This results in a lot of calculations for cells that are far from the surface of the object.

2.2 Pipeline

As the previous section shows, few automatic reconstruction methods exploit the observation made by Debevec et al. (1998) that most buildings consist of simple primitive shapes. Parametric building models are generally limited to certain building shapes and to purely 2.5D representations, i.e. containing no overhanging structures. Some triangular mesh methods strive for local planarity, but cannot guarantee this feature globally. The notable exceptions, which explicitly use planar structures, are (Chauve et al. 2010; Schnabel et al. 2009). Lafarge et al. (2010b) process triangular meshes to use primitive shapes where appropriate, but besides point data, they require images to guide this process.

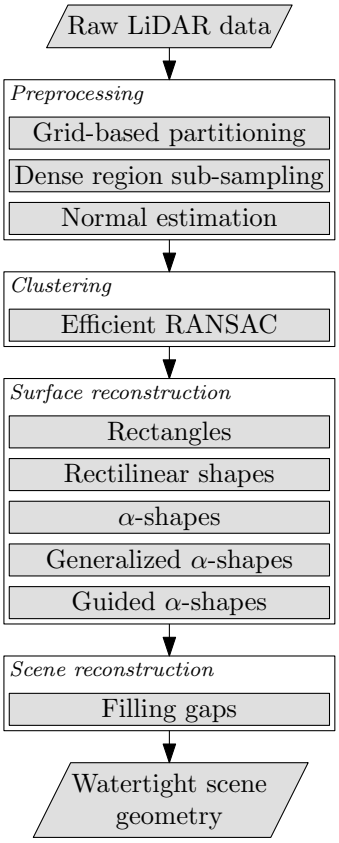


Figure 2.1: The pipeline that structures the methods in this dissertation.

The pipeline structuring our methods to process unordered point data into a geometric model of the scene is similar to (Chauve et al. 2010; Poullis and You 2009; Schnabel et al. 2009; Pu and Vosselman 2009). We first identify clusters of planar point sets within the data. We then construct a polygonal boundary for each cluster. Finally, we stitch these polygons back together and fill the gaps between them to create a watertight geometric model of the scene. The complete pipeline is shown in Figure 2.1.

We identify planar clusters in the point data using Efficient RANSAC (Schnabel et al. 2007). Region growing techniques, e.g. (Tseng et al. 2007), can also be used to identify planar clusters. However, these techniques generally cannot guarantee global planarity within the cluster. Section 2.3 describes Efficient RANSAC in more detail.

Efficient RANSAC is a highly optimized method for identifying point clusters near planes and other simple primitive shapes. However, it cannot be directly applied to raw LiDAR. Most importantly, it needs an estimation of the unoriented normal of the local surface each point was measured from. We compute these normal estimates using the k nearest neighbors of the point, with k generally set to 12.

When the sampling density is very high it may be impossible to get a reliable estimate of the local surface normal because of measurement noise. This can be solved by increasing k, at the cost of processing time. Instead, we choose to sub-sample very dense point sets (Alliez et al. 2013). We do this by superimposing a cube grid over the data and selecting one random point per grid cell. Note that this only reduces the density where cells contain multiple points and the resulting density is based on the size of the grid.

A second drawback of Efficient RANSAC is that it can place a heavy load on processing memory. On a PC with 4GB of working memory, we found that Efficient RANSAC could not process more than roughly 2 million points. This can easily be solved by splitting very large data sets into smaller chunks based on horizontal point coordinates.

Once the clusters are identified, we can project the noisy points onto the plane that optimally interpolates each cluster in a least-squares sense. This reduces the reconstruction problem to a number of 2-dimensional problems of determining the boundary for each planar surface. In Chapters 3, 4, and 5, we present different methods to compute a planar polygonal boundary for each cluster. All of these methods either use or are based upon the well known α-shape method. We describe this method in Section 2.4.

After constructing a polygonal boundary for each surface, these are placed back into 3D space to construct the scene geometry. There will invariably be surfaces missing from the scene because of occlusion or under-sampling. We detect and fill these gaps in the model, while simultaneously stitching together the reconstructed surfaces, using our method presented in Chapter 6. This will result in a watertight geometric model of the scene.

2.3 Clustering

Debevec et al. (1998) observed that buildings are usually composed of simple geometric volumes, like rectangular boxes and prisms. Likewise, we observe that the surfaces of these simple volumes are primitive shapes, particularly planes. In order to exploit this observation during point-based reconstruction, we need to determine which points belong to the same surface. This process of grouping elements based on some shared

property is called clustering.

Recently Schnabel et al. (2007) presented Efficient RANSAC, a method that efficiently identifies subsets of a 3D point cloud that are measured from the same primitive shape. Efficient RANSAC is based on RANSAC (Fischler and Bolles 1981), which is in essence an outlier detection method. First, we give a short description of RANSAC, after which we describe the extensions that optimize Efficient RANSAC for clustering 3D point clouds.

RANSAC is a method that estimates a 'model' from point data in the presence of 'outliers'. In the context of geometry reconstruction, the model to estimate is a plane or other primitive shape. Outliers are points in the data set that do not support the model, e.g. points far from the plane. RANSAC performs an iterative procedure to determine the plane that has the largest support set, i.e. that has the most points nearby. The points in the support set are called 'inliers'.

Each iteration, RANSAC randomly selects a minimal sample of the point cloud. Any sample uniquely determines a model. When estimating a plane, a sample contains either three points or one point and a surface normal. The plane consistent with this sample is constructed and all points within a given threshold Δd of the plane, called the support set, are collected. The number of points collected is used as an indicator for the quality of the model.

RANSAC repeats this process a number of times while storing the largest support set and its plane. The method terminates when the probability of finding a plane with a larger support set becomes smaller than a predetermined value P. This probability of finding a better plane is based on the size of the data set, the size of the support set, and the number of samples that has been tried previous to the current iteration.

Efficient RANSAC expands on the basic RANSAC method in order to efficiently estimate all planar clusters in the point cloud. RANSAC estimates only one plane, i.e. the plane with the largest support set. Efficient RANSAC solves this problem by repeating the RANSAC procedure while there are planes left with enough support. Each time the plane with the largest support set is determined, it is stored and the support set is removed from the point cloud. This process continues until the largest support set found contains less than a given number of points $|S|$.

A number of other extensions either improve the running time of Efficient RANSAC, or improve the resulting clusters. For example, the clusters produced by Efficient RANSAC only contain points for which the angular difference between their normal and the normal of the plane is smaller than a given value Δn. Additionally, Efficient

RANSAC enforces the clusters to be connected components. If two groups of points on the same plane lie further apart than a given distance Δc, they are collected in separate clusters.

After Efficient RANSAC terminates, all the points remaining in the cloud are classified as outliers. This includes points on very small surfaces, but also points measured from cars, vegetation, antennae, and cables. We ignore these points while reconstructing surface boundaries, but they are used later when filling gaps in the model.

2.4 Shape of a set of planar points

Few papers explicitly define what the 'shape' of a set of points in the plane is, but there are some considerations on the intuitive concept of this shape that most papers have in common. A shape is the outer form of an object; it is the outline of the region that makes up the object. In order to capture the space covered by the object, we require the shape to contain the interior region that makes up the object as well as its boundary. A shape is invariant to rotations, translations, and isometric scaling. For this dissertation, we confine shapes to regions bounded by straight line segments, which may be used to approximate curved shapes.

Based on our data type and preprocessing, we are mainly interested in the shape of a reasonably uniform point sampling of a physical object. We do not require the shape to be connected, but points that are close together are in the same component of the shape. Conversely, a component of the shape cannot contain a large region void of points. What constitutes 'close' and 'a large region' depends both on the density of the point distribution and the abstraction level at which the shape is viewed. Therefore, the shape of a point set is linked to some scale parameter; when the scale of the shape increases, the amount of detail in the shape decreases. Figure 2.2 shows an example of the influence of a scale parameter.

Even though related work does not clearly define what the shape of a point set actually is, different methods have been proposed to find a region containing a point set. These methods can broadly be divided into two groups. The first group, including methods like the crust, β-skeleton and γ-neighborhood graph (Amenta et al. 1998, 2000; Veltkamp 1992) assume the points were sampled from one closed curve, the boundary of the region. The second group, including the α-shape and $\mathscr{A}$-shape (Edelsbrunner et al. 1983; Melkemi and Djebali 2000), assumes the boundary polygon contains all points within its interior.

For our purpose of surface reconstruction from LiDAR, the second group is more appro-

Figure 2.2: Some possible shapes of a tree at three different scale levels are shown by the blue outlines. At a high scale level (left) the shape of the tree is very rough, while at a lower scale level (right) the shape contains much more detail.

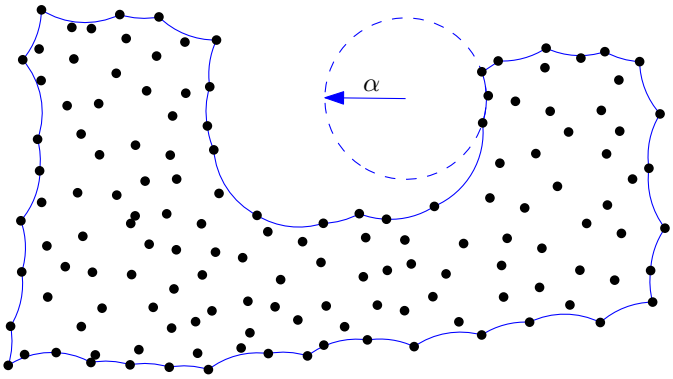


Figure 2.3: The α-hull. Its complement contains all empty α-disks. The α-shape is the straight line version of the α-hull.

priate. Of this group, the α-shape is used most often (Cazals et al. 2005; Mandal and Murthy 1997). More complex methods based on the α-shape can produce the shape of non-evenly distributed point sets (Chevallier and Maillot 2011; Mandal and Murthy 1997; Teichmann and Capps 1998).

An intuitive method often used to describe the α-shape uses its curved-arc dual, the α-hull, shown in Figure 2.3. Imagine the plane as a tub of ice-cream with small nuts at the positions of the points. Now imagine someone allergic to nuts eating the ice-cream with a circular scoop of radius α. The scoop can be used to eat parts of the ice-cream as long as the scoop doesn't contain nuts. After eating as much ice-cream as possible, what remains is the α-hull.

The α-shape is based on the Delaunay triangulation (DT). The DT is a well known

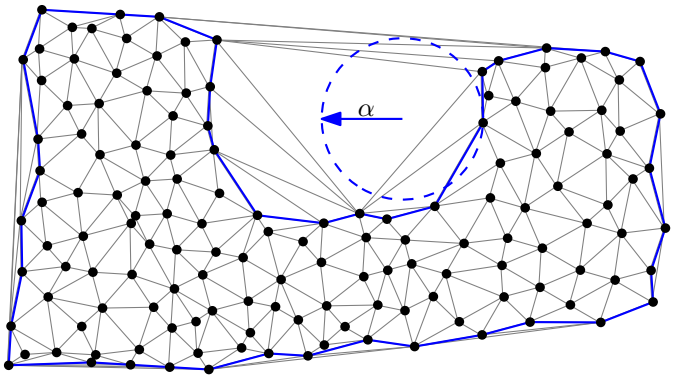


Figure 2.4: The α-shape is a selection of the edges of the Delaunay triangulation.

graph on the point set that connects each pair of points that share the boundary of a disk empty of points. This graph is called a triangulation, because each edge with points on both sides is part of exactly two triangles. For each triangle in the DT, its circumscribing sphere has no points in its interior. The Voronoi diagram is a planar partitioning closely related to the DT. This partitioning has a cell for each point p that contains the part of the plane for which p is the closest point. The Voronoi diagram is dual to the DT, because each triangle connecting the points p,q,r in the DT indicates that the cells of p,q,r in the Voronoi diagram touch in a shared point and vice versa.

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, as shown in Figure 2.4. 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. After the definition of the α-shape, given in Definition 2.4.1, we prove a number of features of the α-shape that are used to prove features of our own methods in Chapters 3, 4, and 5.

Definition 2.4.1 (α-shape (Edelsbrunner et al. 1983))**.** *Given a point set S and a real-valued parameter $\alpha > 0$, a point $p \in S$ is α-extreme if there exists an empty open disk (i.e., not containing any point from S) of radius α with p on its boundary. Two points $p,q \in S$ are called α-neighbors if they share such an empty disk. The* α-shape of S *is the straight-line graph whose vertices are the α-extreme points and whose edges connect the respective α-neighbors.*

Theorem 2.4.2. (Edelsbrunner et al. 1983) *The α-shape of a point set S partitions the plane into interior and exterior faces. The interior faces do not contain the center of an empty open disk with radius α. The interior faces of the α-shape are exactly the union*

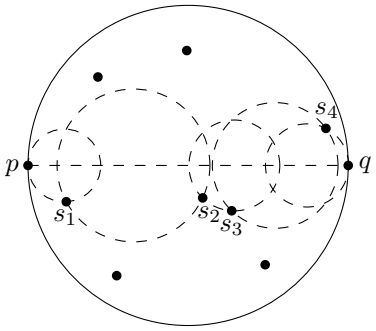


Figure 2.5: Illustration of the proof of Lemma 2.4.4.

of all triangles of a Delaunay triangulation with a circumscribed circle of radius $\leq \alpha$. The α-shape can be constructed from the Delaunay triangulation of the point set in $O(n)$ time, where n is the number of points.

Lemma 2.4.3. *Given a planar point set S and a value $\alpha > 0$, let $\mathcal{A}$ be the union of the α-shape of S and its interior regions. Any two points $p, q \in S$ that share the boundary of an empty open disk d of radius $\leq \alpha$ are connected by a line segment inside $\mathcal{A}$.*

Proof. According to Theorem 2.4.2, the interior of the α-shape is the union of all Delaunay triangles with a circumcircle of radius $\leq \alpha$.

The Delaunay triangulation of S must contain the edge $\overline{pq}$ as witnessed by d. Now consider the Delaunay triangle t incident to $\overline{pq}$ on the side of the center of d; we distinguish two cases. Either (i) this triangle has a circumcircle of radius $\leq \alpha$ and is part of the interior of $\mathcal{A}$, or (ii) there is an empty radius-α disk with p, q on its boundary and p, q are α-neighbors. In both cases, $\overline{pq}$ must be either in the interior or on the boundary of $\mathcal{A}$. □

Lemma 2.4.4. *Given a planar point set S and a value $\alpha > 0$, let $\mathcal{A}$ be the union of the α-shape of S and its interior regions. Any two points $p, q \in S$ at distance at most 2α are connected by a path that lies both inside $\mathcal{A}$ and inside the disk with p, q as its diameter.*

Proof. In this proof, we use a property of α-shapes that is easily inferred from Lemma 2.4.3: for two points $p, q \in S$ at distance at most 2α, either (1) they are α-neighbors, or (2) the connecting line segment is interior to the α-shape, or (3) the disk with $\overline{pq}$ as its diameter contains another point of S.

In cases (1) and (2), it is clear there is a connecting path within $\mathcal{A}$ and within the disk with $\overline{pq}$ as its parameter. In case (3), we consider the collection of empty disks with their center on $\overline{pq}$ and touching two points of S, as shown in Figure 2.5. These disks impose an ordering $(p, s_1, s_2 \dots q)$ on a subset of the points inside the disk. For any two subsequent points x, y in this ordering, the line segment connecting x, y is contained in $\mathcal{A}$, according to Lemma 2.4.3. □

Chapter 3

Rectangles

Generally, urban scenes contain mostly planar surfaces oriented along a limited amount of major directions. This characteristic means that many of the surface boundaries have sharp, straight edges and right angles. Amongst those rectilinear shapes, rectangles are the most prevalent, bounding roughly a third of the surfaces on average. Efficiently identifying these rectangles will solve a significant part of the geometry reconstruction problem.

In this chapter, we present an efficient algorithm that identifies the planar point sets for which a rectangular shape fits the data. In Section 3.1, we define 'fitting' using a one-parameter coverage criterion that ensures a minimum local density across the whole shape. The results of our algorithm are twofold, as described in Section 3.2. Firstly, our algorithm tests whether there exists a fitting rectangle that bounds the point set. Secondly, for any point sets that passes this test, our algorithm produces the range of all rotation angles for which a rectangle fits. Experimental results show the quality of the algorithm, as described in Section 3.3. We discuss the broader implications of these results in Section 3.4.

The key contributions of this chapter are:

- A new one-parameter algorithm that identifies planar point sets 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 (Edelsbrunner et al. 1983), to which we compare our results.

3.1 Fitting boundaries

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 3.1.1. *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 and contains all the points.*

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 the expected noise is less than $\frac{\delta}{2}$. 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.

Any rectangle bounding a point set must contain the convex hull $\mathcal{H}$ of the set. Therefore, if the δ-coverage region does not contain $\mathcal{H}$, no rectangle bounding the set can be δ-covered. If the δ-coverage region contains $\mathcal{H}$, 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 later be selected based on the neighboring surfaces.

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

3.2 Rectangular boundaries

Our algorithm determines whether a set 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 key 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.

For an allowed angle, there is generally a family of δ-covered rectangles with varying width, height, and position. While it is possible to produce allowed ranges for these

parameters, in practice we are only interested in the rectangles compatible with nearby surfaces. Each δ-covered rectangle at a given angle contains the minimal-area bounding rectangle of the convex hull at that angle. We choose these minimal rectangles to represent the family at each angle.

Our algorithm is presented in Subsection 3.2.1 and its efficiency is proven in Subsection 3.2.2. 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 3.2.3. 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. We leave this extension for future research. However, we present another algorithm to construct rectilinear shapes in Chapter 4.

Vegetation or scanning artifacts in the LiDAR data or incorrect clustering parameter settings 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 Efficient 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 3.2.4.

3.2.1 Algorithm

Our rectangular boundary algorithm borrows ideas from the rotating calipers algorithm (Toussaint 1983). 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 a sweep-line algorithm (de Berg et al. 2008). 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 3.1.

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 3.1. To satisfy the δ-coverage criterion, $\mathcal{R}$ must be completely inside $\mathcal{U}_\delta$. If part of $\mathcal{H}$ is not in $\mathcal{U}_\delta$, then no bounding rectangle exists that is δ-covered. Conversely, if $\mathcal{H} \subseteq \mathcal{U}_\delta$, 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 3.1. 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.* (Hoffmann et al. 2001). The algorithm can

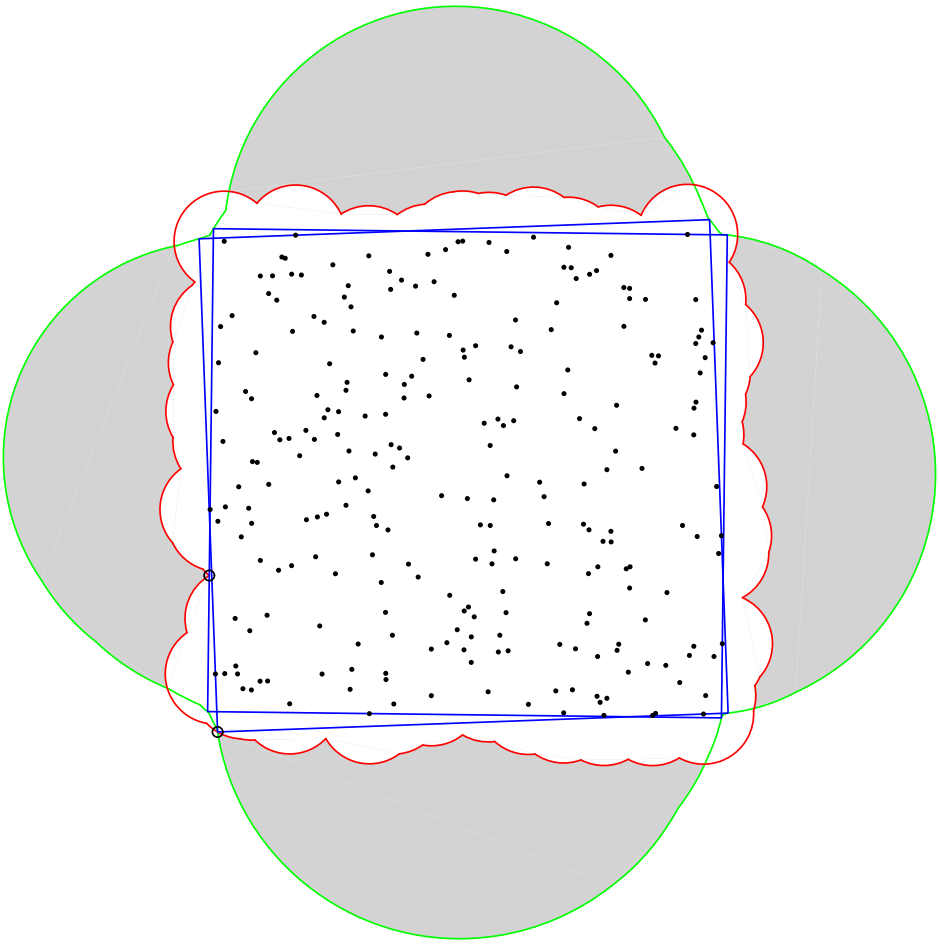


Figure 3.1: The different structures used during our algorithm and the extrema of the range of allowed rectangles. The points were sampled uniformly in a unit square. The δ-coverage region, with $\delta = 0.1$, is bounded by the red arcs, the trajectory region is bounded by the green arcs. The gray regions 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.

be adjusted to find convex shapes other than rectangles, as shown in Subsection 3.2.3. 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 for $\mathcal{R}$ to enter or exit the δ-coverage region. Either during rotation an edge of $\mathcal{R}$ 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 3.1 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 3.1, 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 3.1, the extrema of these angles are shown by blue rectangles. Algorithm 3.1 presents the overall structure of our method.

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 3.2.4.

3.2.2 Efficiency

Our algorithm is based on the rotating calipers algorithm (Toussaint 1983) 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 $\mathcal{H}$, the δ-coverage region $\mathcal{U}_\delta$, and the trajectory region $\mathcal{T}$. $\mathcal{H}$ can be constructed in $O(n \log n)$ time (de Berg et al. 2008). $\mathcal{U}_\delta$ is the region containing all parts of the plane close to a point and for this reason, $\mathcal{U}_\delta$ is closely

Algorithm 3.1 Identify the angles of rotation for which the minimal bounding rectangle is δ-covered.

Input: a set P of points in the plane.
Output: the set of angles for which $\mathcal{R}$ is δ-covered by P, or $\emptyset$ if there is no such rectangle.

```
1: procedure IDENTIFY(P)
2:     H ← 𝓗(P)
3:     U ← 𝓤_δ(P)
4:     if H \ U ≠ ∅ then
5:         return ∅
6:     end if
7:     T ← 𝓣(P)
8:     E ← events from T \ U
9:     Sort E
10:    Handle E in order to fill allowed angles A
11:    return A
12: end procedure
```

related to the Voronoi diagram and its dual, the Delaunay triangulation, described in Section 2.4. Any triangle in the Delaunay triangulation with a circumscribed circle of radius at most δ is completely δ-covered. Incidentally, these triangles exactly comprise the interior region of the α-shape if $\alpha = \delta$. Apart from this region, the δ-coverage region contains a buffer around the α-shape, which can also be directly constructed from the α-shape. This leads to the following lemma.

Lemma 3.2.1. *Given a planar point set P of size n, the δ-coverage region of P can be constructed in $O(n \log n)$ time and contains $O(n)$ vertices.*

Determining the efficiency of constructing $\mathcal{T}$ requires some novel theoretical results. The terms used in the following lemmas and observations are shown in Figure 3.2. We use ∂X to refer to the boundary of X. If $\mathcal{H} \setminus \mathcal{U}_\delta \neq \emptyset$ Observation 2 and Lemma 3.2.4 are incorrect, but in this case the algorithm is terminated before computing $\mathcal{T}$. Therefore, for these lemmas we assume $\mathcal{H} \setminus \mathcal{U}_\delta = \emptyset$.

Lemma 3.2.2. *At any rotation angle θ, a corner $c(\theta)$ of $\mathcal{R}$ defines a wedge $w(\theta)$ with $c(\theta)$ on its apex and bounded by rays following edges of $\mathcal{R}$; these rays touch $\mathcal{H}$ at two points $w(\theta)_l$,$w(\theta)_r$.*

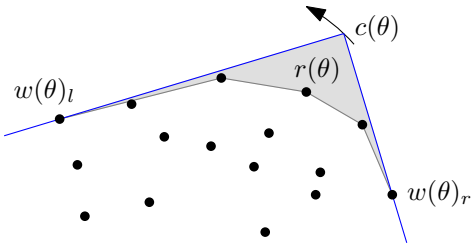


Figure 3.2: Corner $c(\theta)$ during rotation, together with the wedge it defines, the two points $w(\theta)_l$ and $w(\theta)_r$ on the corner's edges, and the gray region $r(\theta)$ visible to $c(\theta)$.

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

Lemma 3.2.3. *$\mathcal{T}$ can be constructed from $\mathcal{H}$ in $O(n)$ time and contains $O(n)$ vertices and circular arcs.*

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

Observation 1. Any wedge $w(\theta)$ contains a closed region $r(\theta)$ between $c(\theta)$ and $\mathcal{H}$, i.e. if $\mathcal{H}$ obstructs visibility, $r(\theta)$ is the part of $w(\theta)$ visible to $c(\theta)$. The open line segment $l(\theta)$ between $c(\theta)$ and its closest point on $\mathcal{H}$ must lie within $r(\theta)$. By definition $\partial\mathcal{T}$ cannot intersect $r(\theta)$ and therefore $l(\theta)$ cannot intersect $\partial\mathcal{T}$, irrespective of θ.

Observation 2. If $\mathcal{H} \setminus \mathcal{U}_\delta = \emptyset$, the circumcenter of each arc in $\partial\mathcal{U}_\delta$ is on $\partial\mathcal{H}$ and no arc intersects $\mathcal{H}$. Therefore, the open line segment between any point on $\partial\mathcal{U}_\delta$ and its closest point on $\mathcal{H}$ 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 3.2.4. *If $\mathcal{H} \setminus \mathcal{U}_\delta = \emptyset$, 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 $\mathcal{X}$ by $\partial\mathcal{X}$.

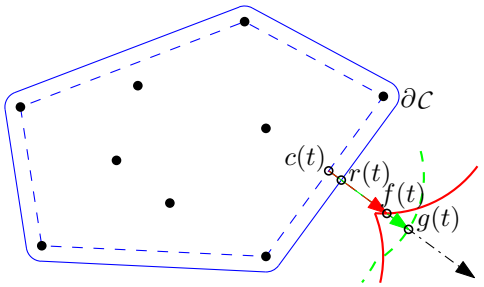


Figure 3.3: The structures used in Lemma 3.2.4 to prove $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ intersect $O(n)$ times.

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 3.3.

Let $\partial\mathcal{C}$ be the boundary of the dilation of $\mathcal{H}$ using a unit-disk. Let $r(t)$ be the point reached by following $\partial\mathcal{C}$ from its topmost point for the fraction t of its length and let $c(t)$ be the point on $\partial\mathcal{H}$ closest to $r(t)$. Finally, let $\overrightarrow{cr}(t)$ be the ray emanating from $c(t)$ trough $r(t)$.

According to Observations 1 and 2, $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$ intersect $\overrightarrow{cr}(t)$ exactly once. We call these intersection points $f(t)$ and $g(t)$ respectively. Finally, we define the functions $F(t) = \|c(t)f(t)\|$ and $G(t) = \|c(t)g(t)\|$. Because of the one-to-one relations between $\partial\mathcal{C}$ and both $\partial\mathcal{U}_\delta$ and $\partial\mathcal{T}$, the functions $F(t)$ and $G(t)$ are well-defined.

$F(t)$ 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 3.2.5. *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 $\mathcal{H}$ 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 $\mathcal{H}$, 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 $\mathcal{H}$ in $O(\log n)$ time by performing a binary search on the vertices of $\mathcal{H}$. 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 3.2.6. *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.*

Proof. Lemma 3.2.3, 3.2.4, and 3.2.5 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. □

3.2.3 Convex Polygons

Our method can be adapted with minimal changes to produce 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 only 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 3.2.8.

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 $\mathcal{H}$. Note that this implies Observation 1 holds for all convex corners.

Lemma 3.2.7. *For any minimal area convex k-shape $\mathcal{P}$ and any rotation angle of $\mathcal{P}$, each edge contains a vertex of $\mathcal{H}$.*

Proof. By contradiction, assume there is an edge e of $\mathcal{P}$ that does not contain a vertex of $\mathcal{H}$. Without loss of generality, we assume this edge is above $\mathcal{H}$. We can move e down by shortening other edges, while keeping $\mathcal{H}$ inside the k-shape, until e contains a vertex of $\mathcal{H}$. Because, apart from e, 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 3.2.8. *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 3.2.6 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 3.2.3 and 3.2.4 hold for convex k-shapes.

The proof of Lemma 3.2.3 is based on Thales' theorem, which describes the trajectory of a corner irrespective of angle. However, unlike the proof of Lemma 3.2.3, 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 3.2.3 still holds.

Observation 1 shows the one-to-one relation between $\mathcal{T}$ and $\mathcal{C}$ used in the proof of Lemma 3.2.4 holds for corners with other angles. The rest of the proof of Lemma 3.2.4 remains the same. □

3.2.4 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. (2004). 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 3.2.1 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. 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 the 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 additional vertices of $\mathcal{U}_\delta$. Figure 3.4 shows the structures and range of rotation angles in the presence of outliers.

3.2.5 Implementation

We implemented the algorithm in C++ using the CGAL library (Computational Geometry Algorithms Library 2013) 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,

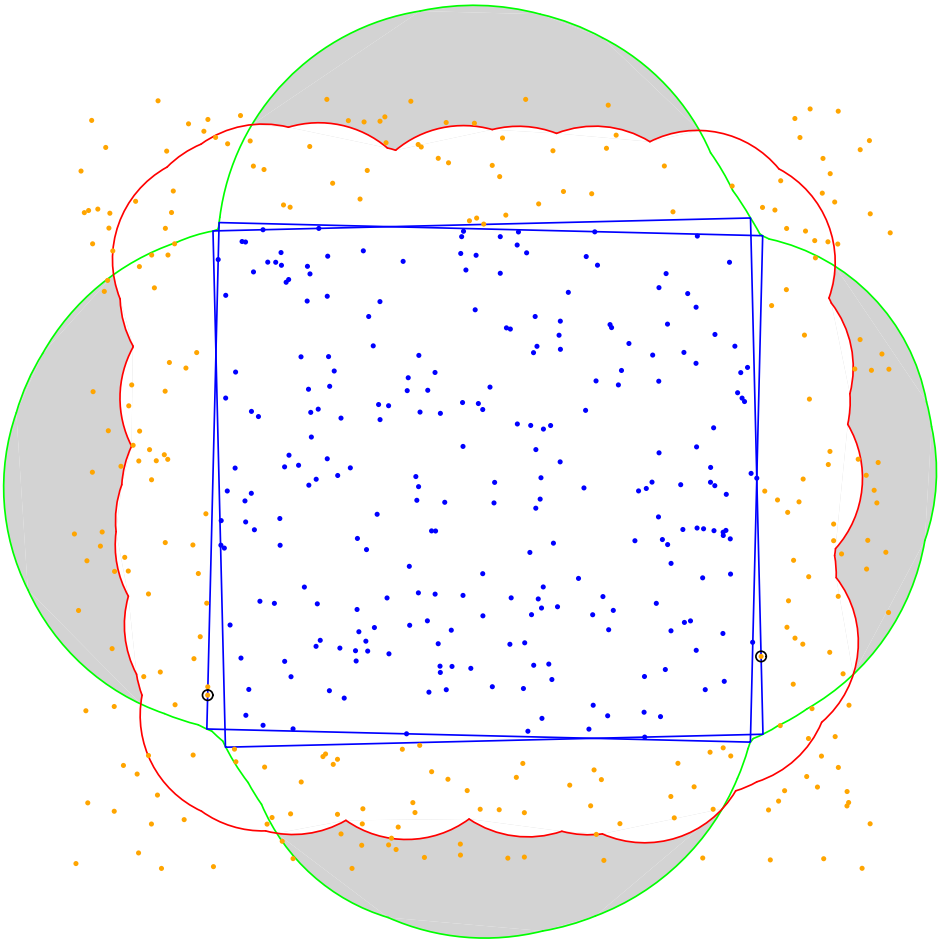


Figure 3.4: The different structures used during our algorithm and the range of allowed angles. The blue points were sampled uniformly in a unit square and the orange points are outliers. The δ-coverage region, with $\delta = 0.2$, is bounded by the red arcs, the trajectory region is bounded by the green arcs. The gray 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.

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 does not change the result of our method, because all other structures need only a subset of the vertices in the α-shape to be computed.

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.

3.3 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 (John Chance Land Surveys and Fugro 2009). 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

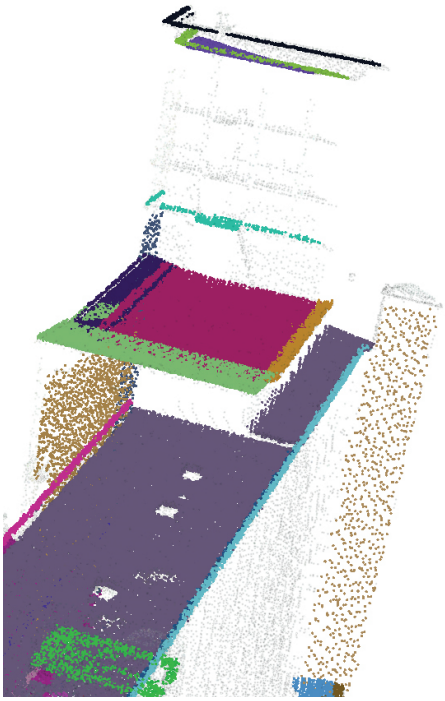


Figure 3.5: Vertical surfaces have a lower density than the other surfaces. This is a detail of Figure 1.1.

scanner. Note that no benchmark data sets of urban scenes are available to compare our algorithm on.

The data sets do not have a globally consistent density, as shown in Figure 3.5. 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 (Schnabel et al. 2007) to cluster the point data into planes. The settings for aerial and ground based data are shown in Table 3.1. These parameters showed good clustering performance on the data set when compared to other values. One data set is shown in Figure 1.1 with its points colored by cluster.

	NN	$\Delta d(cm)$	Δn	$Cb(cm)$	$\|S\|$	P_{miss}
Aerial	12	6.5	20°	25	250	0.001
Ground	12	3.5	20°	15	250	0.001

Table 3.1: 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}).

3.3.1 Results

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 3.6 shows the data of Figure 1.1 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 3.8.

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 3.9. 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

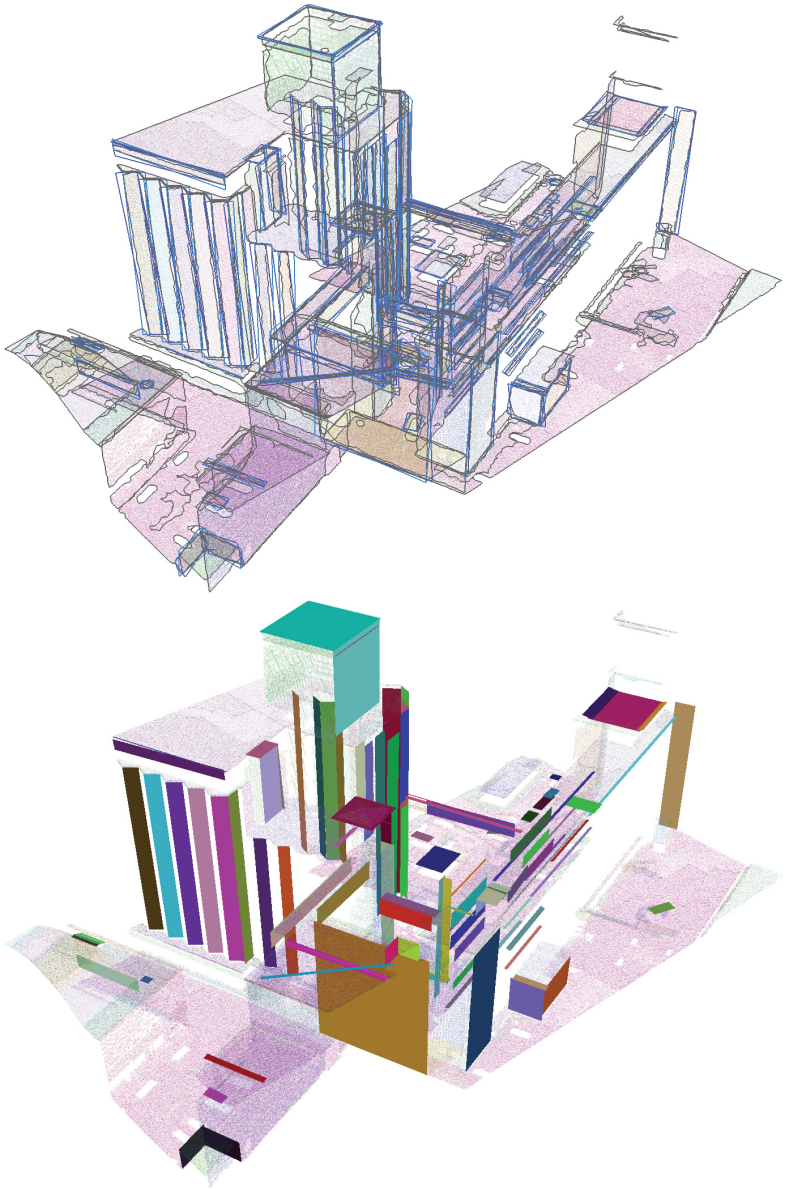


Figure 3.6: Identified rectangles, for $\delta = 60$ cm for dense and 125 cm for sparse surfaces. Top: the α-shape of each surface (gray outlines) and the extrema of the ranges of rectangles for surfaces on which they are identified (blue outlines). Bottom: appropriate δ-covered rectangles: these rectangles have an edge parallel to a neighboring surface.

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

Table 3.2: The percentage of rectangular planes with a nearby neighbor that have a boundary edge parallel to one of these neighbors; δ = 60 cm (dense), 125 cm (sparse), and 40 cm (ground-based).

these has an edge parallel to at least one of these neighbors, as shown in Table 3.2.

Most related methods in 2D shape reconstruction assume different input or output, as described in Section 2.4. They usually produce one shape with all edges between data points. In contrast, we produce all fitting rectangles. The efficiency of our algorithm given in Subsection 3.2.2 is at least as good as that of the methods presented in Section 2.4. One could argue that our method is less robust to outliers, because one outlier may result in an incorrectly rejecting the rectangular boundary. Besides efficiency and robustness, our algorithm may be compared to these other methods based 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 3.7 shows both the α-shape and the range of δ-covered rectangles for some clusters.

3.3.2 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, as shown in Table 3.3. 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 3.4. On the same computer, preprocessing the data using Efficient RANSAC took roughly 15 minutes per data set.

3.3.3 Evaluation of Results

Figure 3.8 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

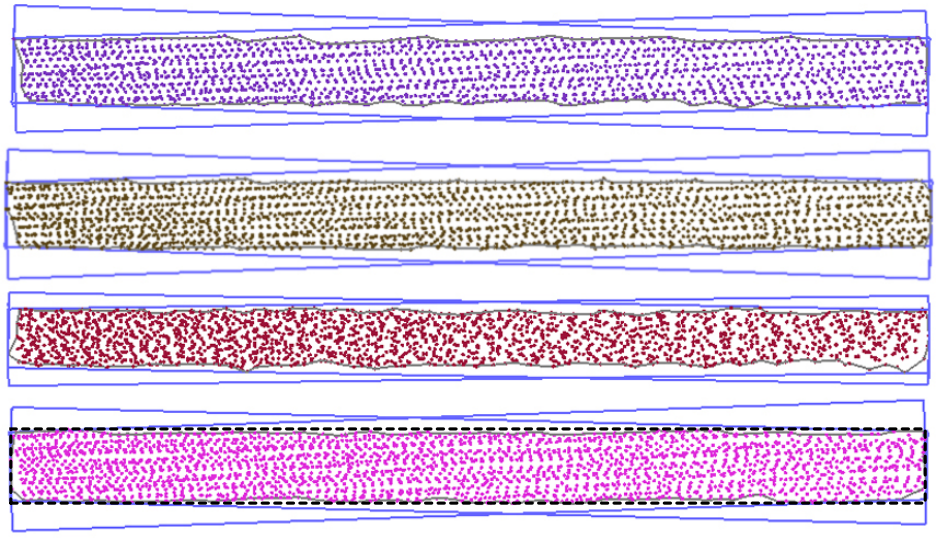


Figure 3.7: Some clusters of roughly 2200 points each, with their α-shape (gray outlines) and the extrema of the δ-covered rectangles (blue outlines). The black dashed outline in the bottom cluster shows another allowed rectangle that nicely bounds the data.

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

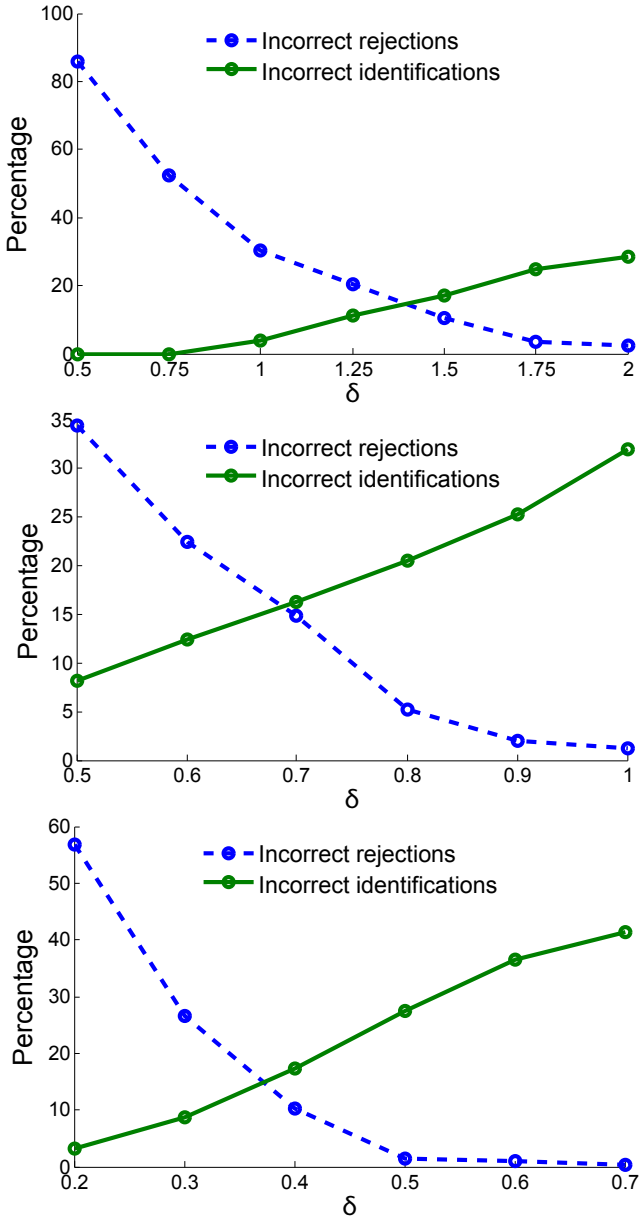


Figure 3.8: The percentage of aerial sparse (left), aerial dense (center), and ground-based (right) clusters for which a rectangle was falsely identified or rejected at different δ values. The percentages of correctly identified or rejected boundaries are not shown.

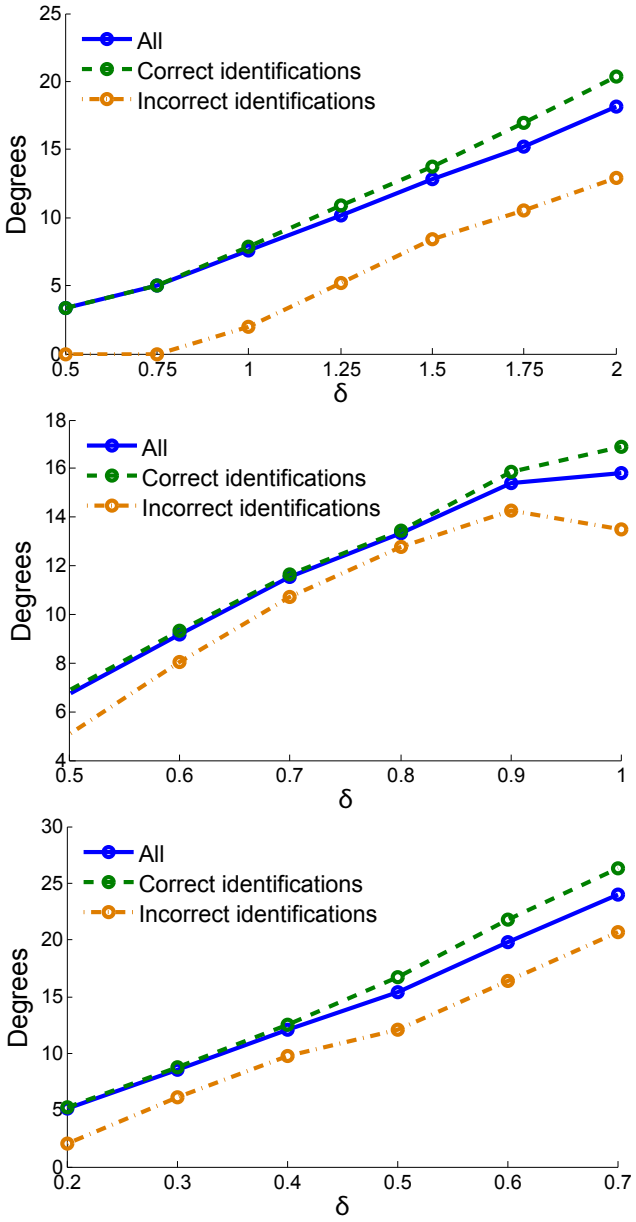


Figure 3.9: The mean range of allowed rotation angles for aerial sparse (left), aerial dense (center), and ground-based (right) clusters for which a rectangle was identified correctly or incorrectly at different δ values. Although not shown, the ranges vary widely at one δ.

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%
$\mathcal{H}$	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%
$\mathcal{H} \setminus \mathcal{U}_\delta$	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	Average
α-shape	24.97%	29.42%	30.34%	28.48%	22.71%	21.41%	38.83%	28.08%
$\mathcal{H}$	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%
$\mathcal{H} \setminus \mathcal{U}_\delta$	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 3.4: 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.

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 3.9. 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 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.

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 neigh-

bor. Table 3.2 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 3.7. 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.

3.4 Conclusions

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 known outliers by forcing the rectangle to avoid them. 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 3.8. 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. 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 (Akkiraju et al. 1995; Cazals et al. 2005; Mandal and Murthy 1997). We could even allow different values of δ within the same cluster and change the coverage region accordingly, comparable to a weighted α-shape (Akkiraju et al. 1995). 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 (Cazals et al. 2005; Mandal and Murthy 1997). 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 pre-processing 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. (2004) 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 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.

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 3.2.3. Extension of this algorithm to non-convex polygons poses more difficult problems. In the next chapter, we present an algorithm for computing an arbitrary rectilinear boundary.

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.

Chapter 4

Rectilinear Shapes

In the previous chapter, we mentioned that roughly a third of the surfaces in many city scenes are rectangles. Many of the remaining two-thirds of the surfaces have a rectilinear shape: all the corners of the shape are 90 degrees. In fact, this occurrence is so common that many urban reconstruction methods operate under the assumption of a 'Manhattan world', where all lines and planes are axis-aligned. Unfortunately, when reconstructing from urban LiDAR, particularly from data sets of older cities, the buildings may not align with the coordinate axis and different rectilinear surfaces may not align with each other.

In this chapter, we broaden the rectangularity premise of the previous chapter to general rectilinear shapes that tightly bound a point set. To determine these shapes, we present a simple variation of the α-shape, called the (α,δ)-sleeve in Section 4.1. This structure creates a buffer around the shape and we search for a rectilinear shape within this buffer, as detailed in Subsection 4.1.2. Figure 4.1 shows an overview of our method. In Section 4.2, we describe some experiments we subjected our method to in order to determine if it is suited for reconstructing surfaces and parameter settings under which it performs optimally. Finally, we discuss some broader implications of this method in Section 4.3.

Finding a rectilinear shape that is close to a given shape is a problem that has been studied in different contexts. For example, restricted-orientation line simplification has been studied for the purpose of schematized map computation (Buchin et al. 2011; Swan et al. 2007; Wolff 2007). Another example is squarifying, an operation that occurs in ground plan generalization (Mayer 2005; Regnauld et al. 1999; Ruas 1999). In these cases, the starting point is a polygonal line or planar subdivision, whereas we start from a set of points. Furthermore, since sampled points are assumed to be inside the rectilinear polygon to be found, we wish to find an outer approximation of the boundary of the point set. Hence, an area-preserving method like presented by Buchin et al. (2011) does not appear suitable. Our method guarantees that the rectilinear polygon

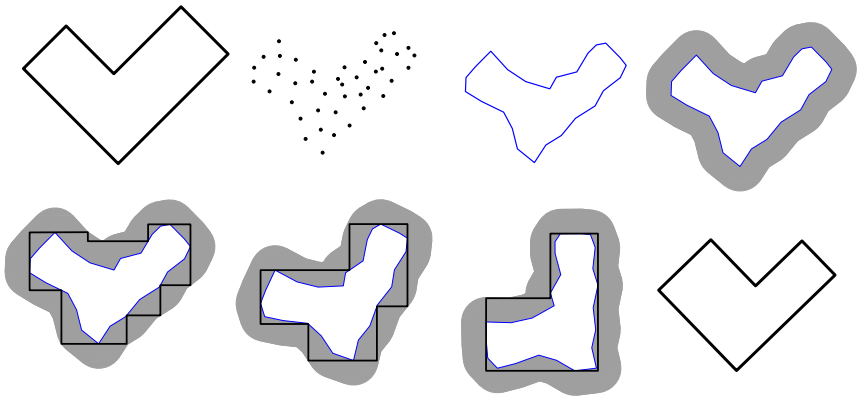


Figure 4.1: An overview of our method. From top left to bottom right: the surface, a point sample of the surface, the α-shape, the (α,δ)-sleeve, three rectilinear minimum-link paths within the (α,δ)-sleeve for different rotations, and the cycle with the fewest links over all rotations.

is within a specified distance of the α-shape but still outside it; other methods do not have this feature.

The key contributions of this chapter are:

- A novel two-parameter concept to model a buffer tightly surrounding a point set.
- A method for constructing a rectilinear shape within this buffer that therefore contains all points. The shape has the fewest edges over a discretized set of rotation angles.
- An analysis of fixed or data-dependent parameter settings for which the method performs well.

4.1 (α,δ)-Sleeves and Minimum-Link Paths

In this section we describe the approach for computing a rectilinear polygon that corresponds to the shape of a set of points well. For this purpose, we introduce a new structure called the (α,δ)-sleeve that is based on the α-shape described in Section 2.4. We will show some properties of this new structure and give an efficient algorithm for its construction. Finally, we show how we can use the (α,δ)-sleeve to determine a suitable rectilinear polygon, and give an algorithm to compute it. An example of our method is shown in Figure 4.2.

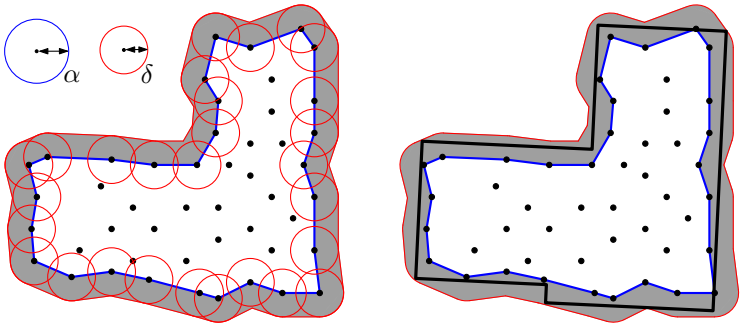


Figure 4.2: Left, values of α and δ shown by disks, and the (α,δ)-sleeve of the points shown. Right, a minimum-link rectilinear cycle in the sleeve.

Our objective is to bound the point set by a rectilinear shape with few edges. This shape must have all points to the inside or on it, but we must allow the shape to cover some area outside of the α-shape to accommodate a rectilinear shape with few edges.

Like was done for finding rectangles to fit a set S of points, as shown in Chapter 3, we will use the δ-coverage concept. There we define the δ-coverage region to be the union of the radius-δ disks centered on the points of S. Any point in the plane not in the δ-coverage region is at least at distance δ from all sample points. We require the approximating rectangle to contain all points of the sample, but not be outside the δ-coverage region. The value of δ should be chosen small enough so that the rectangle cannot be too far away from the sampled points and therefore from the likely shape. On the other hand, δ should be chosen large enough so that the irregularities in the sampling do not exclude the existence of a rectangle in the δ-coverage region that encloses the points as well.

4.1.1 Definition and Properties of (α,δ)-Sleeves

We adapt some ideas of the previous chapter on rectangles to rectilinear shapes. Let S be a set of sampled points in a plane; we wish to construct a rectilinear polygon containing S. Let $\alpha > 0$ be a real parameter related to the sampling density, typically between 20 and 50 cm for LiDAR data. Let $\delta > 0$ be another such parameter, also related to the sampling density.

We will compute the α-shape $\mathcal{A}$ of S and require that the rectilinear shape $\mathcal{P}$ contains $\mathcal{A}$ completely. If $\mathcal{P}$ is a rectangle, there is no difference between requiring S or $\mathcal{A}$ to be inside $\mathcal{P}$, but for other rectilinear shapes it can make a difference. Our main

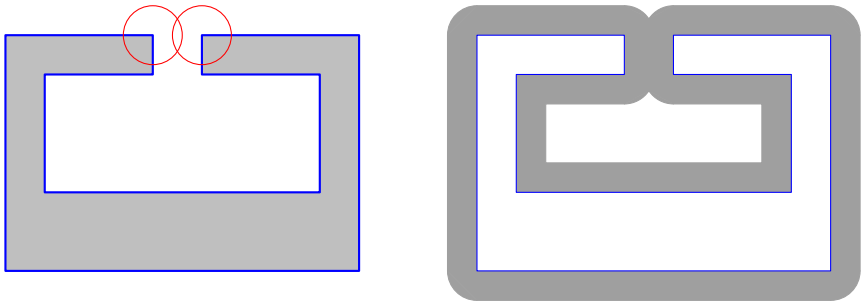


Figure 4.3: If the α-shape bounds the gray interior shown left, and δ corresponds to the radius of the disks shown left, then the (α,δ)-sleeve (shown in gray on the right) has more than one inner boundary.

reason for using the α-shape is the guarantee that $\mathcal{P}$ is not self-intersecting if the α-shape consists of one connected component. Under regular sampling conditions of an individual surface, we can make sure that the α-shape has only one component by carefully choosing α based on the sampling density.

From the α-shape $\mathcal{A}$ of S we will compute the (α,δ)-sleeve, defined as follows.

Definition 4.1.1. *For set S of points in the plane, the* (α,δ)-sleeve *is the Minkowski sum of the α-shape of S with a disk of radius δ centered at the origin, minus the interior faces of the α-shape.*

The Minkowski sum with a disk creates a buffer region around the shape. The (α,δ)-sleeve is an outer proximity region of the α-shape.

Not all values of α and δ, or combinations of values, give nice properties to the (α,δ)-sleeve. Let us assume that α is such that the α-shape is in principle a good approximation of the underlying shape. In particular, let us assume that it is connected and has no holes. This implies that the inner boundary of the (α,δ)-sleeve is the boundary of a relatively simple polygon. This polygon cannot have intersecting edges or holes, but it may have overlapping edges. If δ is sufficiently small, the outer boundary of the (α,δ)-sleeve will be the boundary of a simple polygon, but with circular arcs. For some shapes and larger values of δ, the (α,δ)-sleeve can have several inner boundaries: not just those created by subtracting the interior of the α-shape, but also ones where opposite sides on the outside of the α-shape are close. Figure 4.3 shows an example.

It turns out that if we set $\delta \leq \frac{4}{5}\alpha$, then the (α,δ)-sleeve will have the desired topology with one outer and one inner boundary.

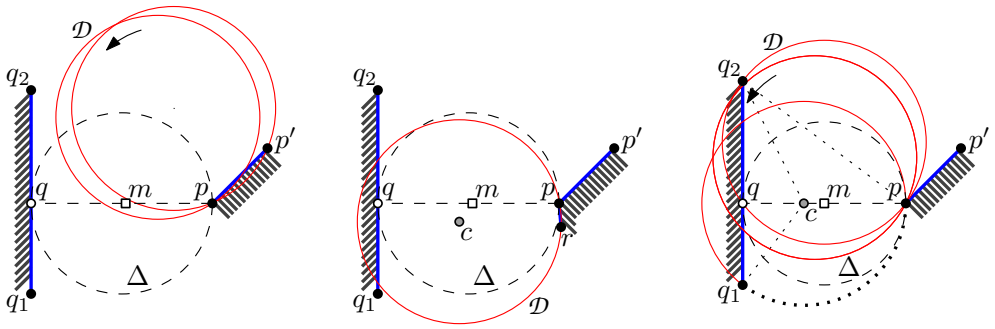


Figure 4.4: Illustration of the last case of the proof of Lemma 4.1.2.

Lemma 4.1.2. *For a given $\alpha > 0$, let $\mathcal{A}$ be an α-shape of a set S of points in the plane, and assume that $\mathcal{A}$ is the boundary of a relatively simple polygon. Then for $0 < \delta \leq \frac{4}{5}\alpha$, the (α,δ)-sleeve has the topology of an annulus.*

Proof. Since for very small δ, the topology of the (α,δ)-sleeve is an annulus, we can imagine growing δ to the first (smallest) value δ' where the topology is no longer an annulus. This happens only when there are two points p,q, not necessarily in S, on the α-shape at distance $2\delta'$. Let m be the midpoint of p,q. Then m is at distance δ' from p and q, and no point of the α-shape is closer to m. Hence, the disk Δ centered at m with radius δ' does not intersect the α-shape in points other than p and q.

Assume first that the two points are two vertices $p,q \in S$. According to Lemma 2.4.4, Δ contains an edge inside the α-shape. This contradicts the assumption that no point of the α-shape is closer to m than p or q.

Assume next that $p,q \notin S$. Then they lie in the interior of two different α-shape edges. If they are not parallel, it is impossible that the first topological change occurs at m. If they are parallel, then the topological change will occur simultaneously along a stretch of the two edges, and we can choose p and q such that at least one of them is an endpoint and therefore in S. So this case is treated together with the final case.

Assume finally that only one point is in S, say, $p \in S$ and $q \notin S$. Let q_1 and q_2 be the endpoints of the α-shape edge that q lies on, so $q_1,q_2 \in S$. The diametral disk Δ of p and q is tangent to the edge $\overline{q_1q_2}$ at q. Assume without loss of generality that $\overline{q_1q_2}$ is vertical, that q_1 is the lower endpoint of $\overline{q_1q_2}$, that q_1 is closer to p than q_2, and that p is to the right of $\overline{q_1q_2}$, see Figure 4.4.

Since the topology of the (α,δ)-sleeve changes for the first time (smallest δ') due to the

contact at m, the α-neighbors of p cannot lie to the left of the vertical line through p. Let p' be the α-neighbor of p that makes the smallest angle with the vertical upward direction. Since p and p' are α-neighbors, an empty α-disk $\mathcal{D}$ exists that has p and p' on its boundary and its center to the left of the directed edge from p to p'.

We now rotate $\mathcal{D}$ in contact with p in counterclockwise direction, see Figure 4.4. Initially $\mathcal{D}$ does not contain any point of S inside. Let r be the first point of S that is reached by the boundary of $\mathcal{D}$, such that r would be inside if we were to rotate $\mathcal{D}$ further. We distinguish several cases.

If $r = q_2$, then p, q_2 are α-neighbors (as witnessed by the emptiness and current position of $\mathcal{D}$). The implied α-shape edge will intersect Δ because q_2 lies left of p, a contradiction. The same contradiction is obtained when $r = q_1$, or when r is any point that lies left of the vertical line through p. Hence, r lies to the right of this vertical line or on it. This implies that the disk $\mathcal{D}$ has rotated beyond the situation where it has a vertical tangent at p. This is equivalent to stating that the center c of $\mathcal{D}$ lies below the horizontal line through p. Also, this center must lie right of the line through q_1 and q_2, because $\overline{q_1q_2}$ is an α-shape edge with the exterior to its right.

Because q_1, q_2 are α-neighbors, $\|q_1q_2\| \leq 2\alpha$. We argue that $\|pq_2\| > 2\alpha$. According to Lemma 2.4.4, if $\|pq_2\| \leq 2\alpha$, there is a path connecting p, q_2 inside their diametral disk. Because $\|pq_1\| \leq \|pq_2\|$, the same holds for p, q_1. This means that if $\|pq_2\| \leq 2\alpha$, then either there is a point on $\mathcal{A}$ closer to m than p or $\mathcal{A}$ is not a relatively simple polygon. Recall that $\mathcal{A}$ is relatively simple, so both cases contradict an assumption.

We are interested in the threshold case, where $\|pq\| = 2\delta'$ is as small as possible, and the (α,δ)-sleeve is an annulus for $\delta \geq \delta'$, but not if δ is infinitesimally smaller than δ'. Because $\|q_1q_2\| \leq 2\alpha$, $\|pq_2\| > 2\alpha$, and $\|pq_1\| \leq \|pq_2\|$ and $\angle pqq_2 = \frac{\pi}{2}$, the smallest $\|pq\|$ occurs when the angle $\angle q_1q_2p$ is as small as possible. At the same time, the radius-α disk $\mathcal{D}$ touching p, r cannot contain q_1 and cannot have its center above $\overline{pq}$. Finally, because r was the first point encountered by $\mathcal{D}$ during its rotation, either $\|pq_1\| \geq 2\alpha$ or c is above $\overline{pq_1}$. If $\|pq_1\| \geq 2\alpha$, then $\|pq\| \geq \sqrt{3}\alpha$ so $\delta' \geq \frac{1}{2}\sqrt{3}\alpha > \frac{4}{5}\alpha$. We continue to show that the other case can give a smaller lower bound for δ. Here $\|pq_1\| < 2\alpha$ and c is above $\overline{pq_1}$. By Lemma 2.4.4, $\mathcal{A}$ has a sequence of edges connecting p with q_1 via r inside the disk that has $\overline{pq_1}$ as its diameter.

The threshold case is shown in Figure 4.4 (right). If $\mathcal{D}$ stays empty while rotating beyond the point where c lies on $\overline{pq}$, it is possible for $\mathcal{A}$ to connect q_1 to r and p through a series of edges strictly outside Δ. This would result in a hole in the (α,δ)-sleeve just below m, meaning that the topology of the (α,δ)-sleeve is not an annulus.

In the threshold case, c lies on $\overline{pq}$ and $\|q_1q_2\| = \|pq_2\| = 2\alpha$. Because $\mathcal{D}$ touches both p and q_1, $\triangle q_2cq_1$ and $\triangle q_2cp$ are mirrored triangles, so the angles $\angle q_2q_1c = \angle q_2pc$, and $\triangle q_2qp$ and $\triangle cqq_1$ are similar triangles. This implies that $\frac{2\alpha}{\alpha} = \frac{\|q_2q\|}{\|cq\|}$ or $\|q_2q\| = 2\|cq\|$. If we combine this with the Pythagorean theorem on $\triangle q_2qp$ we can derive that $2\delta' = \|pq\| = \frac{8}{5}\alpha$:

$$
\begin{aligned}
(2\alpha)^2 &= \|q_2q\|^2 + (\|cq\| + \alpha)^2 \\
4\alpha^2 &= 4\|cq\|^2 + \|cq\|^2 + 2\|cq\|\alpha + \alpha^2 \\
3\alpha^2 - 2\|cq\|\alpha - 5\|cq\|^2 &= 0 \\
(\alpha + \|cq\|)(3\alpha - 5\|cq\|) &= 0 \\
\|cq\| = -\alpha \text{ or } \|cq\| &= \tfrac{3}{5}\alpha
\end{aligned}
$$

The first option leads to a degenerate triangle $\triangle q_1q_2p$. The second option leads to $\|pq\| = \|cq\| + \alpha = \frac{8}{5}\alpha$.

The threshold case presented results in an (α,δ')-sleeve with the topology of an annulus: the value of δ' does not allow $\mathcal{D}$ to rotate beyond a vertical tangent at p when it reaches r but before it reaches q_1. Hence, r is not to the right of p. If r is vertically below p (a degenerate situation), then we can repeat the whole construction with r instead of p, which means we can ignore this case. Hence, in order to get a different topology, r must be strictly right of p, and c must be strictly below $\overline{pq}$. Because $\mathcal{D}$ cannot contain q_1, $\|q_1q_2\| \leq 2\alpha$, and $\|pq_2\| > 2\alpha$, this implies that $\|pq\| = 2\delta' > \frac{8}{5}\alpha$. □

We can use known algorithms to compute the (α,δ)-sleeve for given values of α and δ. For a set S of n points in the plane, the α-shape can be computed directly from the Delaunay triangulation of S (Edelsbrunner et al. 1983). Then we use a buffer computation algorithm on the α-shape. Such an algorithm can be based on computing the Voronoi diagram of the line segments of the α-shape first (de Berg et al. 2008; Yap 1987). Then the buffer boundary can be found in each Voronoi cell, and these can be merged into the boundaries of the δ-buffer of the α-shape. Converting this to the (α,δ)-sleeve is then straightforward. This procedure takes $O(n \log n)$ time in total.

4.1.2 Minimum-Link Cycles in (α,δ)-Sleeves

The (α,δ)-sleeve gives a region in which we want to determine a rectilinear shape. The rectilinear shape should separate the inner boundary from the outer boundary. We will

show how to use a minimum-link path algorithm to find the shape. We will assume that the (α,δ)-sleeve has the topology of an annulus.

For any simple polygon and start- and endpoints s and t inside, we can find a minimum-link path that uses only horizontal and vertical edges. This problem has been well-studied in computational geometry, and a linear-time algorithm exists that finds such a path (Hershberger and Snoeyink 1994). Our problem is different in four aspects: (i) We do not have a simple polygon but a shape with the topology of an annulus. (ii) We do not have a start- and endpoint; instead of a path we want a rectilinear cycle. (iii) We do not know the orientation of the edges beforehand, we only know that the angles on the path are 90 degrees. (iv) Our polygon has circular arcs.

Lemma 4.1.3. *In an (α,δ)-sleeve, there exists a minimum-link axis-parallel cycle that passes through the lowest point of the inner boundary of the sleeve (the α-shape).*

Proof. Consider any minimum-link cycle in the sleeve, and let e be its lowest horizontal edge. If e contains a vertex of the α-shape the lemma is true, otherwise we can move e upwards while shortening the two adjacent edges of the cycle. During this move two things can happen: (i) An adjacent edge reduces to length 0, but then a cycle with two fewer links is found. (ii) Edge e is stopped by a vertex of the α-shape, which must be its lowest vertex because all vertices of the α-shape have a y-coordinate at least as high as the lowest edge of the cycle. This proves the lemma. □

Another property of the minimum-link rectilinear path is that it is non-selfintersecting. Otherwise, we could remove the extra loop and obtain a path with fewer links.

Lemma 4.1.4. *Any minimum-link rectilinear path in an (α,δ)-sleeve is non-selfintersecting.*

Suppose that we know the orientation of the minimum-link cycle. Then we can rotate the (α,δ)-sleeve so that this orientation becomes the axis-parallel orientation. By Lemma 4.1.3, we now know a point that we can assume to lie on the minimum-link cycle. To convert the (α,δ)-sleeve to a polygonal region we do the following. We find the lowest vertex v of the α-shape and insert a new edge vertically down from v, until it reaches the outer boundary, see Figure 4.5. This edge will split the annulus into a shape with the same topology as a simple polygon. We duplicate the new edge including its endpoints, splitting v into a left copy v_l and a right copy v_r. Now our shape does not have doubly used edges, and it can be treated as a normal simple polygon.

We will find a minimum-link axis-parallel path from v_r to v_l using the algorithm by Hershberger and Snoeyink (1994). This algorithm starts from an axis-aligned line

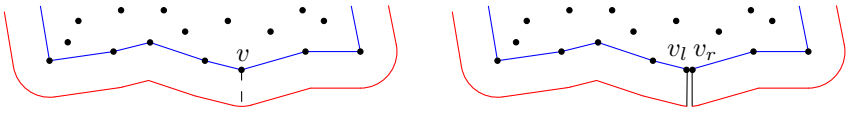


Figure 4.5: The bottom part of an (α, δ)-sleeve, the lowest vertex v of the α-shape (left), and the conversion of the sleeve to a simple polygon (right).

l_0 through the start-point and searches the edges of the path incrementally. Each step, l_i divides the polygon into two or more parts. We search for the longest orthogonal line l_{i+1} in the part containing the end-point. In order to find l_{i+1}, only orthogonal lines through the vertices visible from l_i need to be considered. If the end-point is visible, the line through it is chosen instead and the algorithm terminates. The start- and end-point together with the intersections of consecutive lines define the minimum-link path. Because in our case the start- and end-point overlap, we can easily convert the path into a cycle. This algorithm can be adapted to handle polygons with circular arcs by also checking lines tangent to circular arcs during the longest orthogonal line search.

In order to determine the orientation of the minimum-link cycle, we need to repeat the minimum-link path for a number of different directions while storing the current optimum. However, in practice it is generally sufficient to try a discrete set of meaningful angles.

4.2 Experiments

In the previous section we presented a method to construct a rectilinear minimum-link cycle that approximates the α-shape. Here, we evaluate this method on synthetic and real data with the main goal of checking whether the method is suitable for piecewise planar urban scene reconstruction. In particular, we wish to discover whether for a given point density, values of α and δ exist that lead to a rectilinear minimum-link cycle that is close to the true building facet shape.

This section describes both the setup of these experiments and the results. The first two subsections describe our experiments on synthetic data to determine a value for δ to get the correct number of edges or the best overlap with the ground truth, respectively. The last subsection describes an experiment on urban LiDAR data.

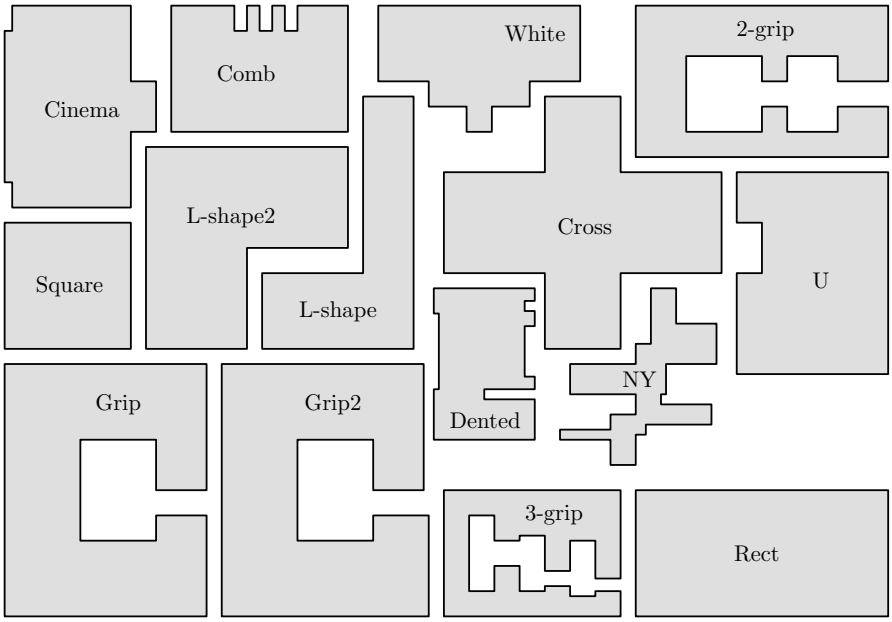


Figure 4.6: The different test cases used for the experiments on synthetic data. For the "Grip" and "Grip2" cases, note the difference in thickness of the bottom right part.

4.2.1 Universal δ

For the evaluation on synthetic data, we have constructed fifteen test cases of varying shape and complexity. Each test case comprises a ground truth polygon T that should represent a realistic urban surface shape with an area of between 30 and 60 m^2, as shown in Figure 4.6. To counter bias towards a certain initial orientation, T is rotated by a random real-valued angle for each test. Each rotated T then yields a point set of predetermined density by uniform sampling from its interior.

For each case, we compute the (α,δ)-sleeve using a fixed α based on the sampling density ϱ and a varying δ. We have chosen a fixed α such that the α-shape is connected and without holes irrespective of the test case. We vary δ to determine whether there is a single value of δ at each ϱ such that our method produces polygons similar to the ground truth in all test cases. To measure the impact of ϱ on the results of our method, we have repeated the experiments for three different densities. The chosen α for each density ϱ is shown in Table 4.1. We vary δ using increments of $1\,cm$.

A rectilinear polygon $\hat{\mathcal{P}}$ is constructed inside the (α,δ)-sleeve, according to the al-

ϱ (points/m^2)	α (cm)
50	60
100	20
200	17

Table 4.1: The different densities used for the point sampling and the associated values of α.

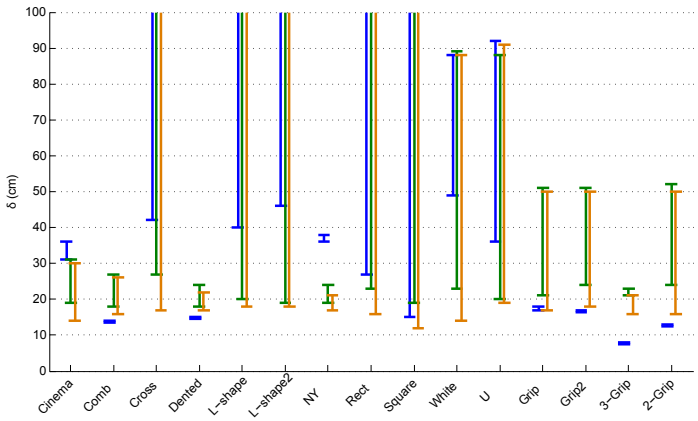


Figure 4.7: The range of δ for which the constructed polygon has the same number of edges as the ground truth at point densities 50 (blue), 100 (green), and 200 (orange) points/m^2.

gorithm given in Section 4.1, such that $\hat{\mathcal{P}}$ has the minimum number of edges over all rotation angles. We use an angular step size of 1 degree, meaning that we run the minimum-link algorithm 90 times for each (α, δ)-sleeve. To get a canonical result for each angle, we 'shrink' $\hat{\mathcal{P}}$ to a smaller polygon $\mathcal{P}$ by moving each edge inward until it touches the α-shape. We move edges in descending order of edge length. If there are multiple polygons with the minimum number of edges, created for different rotation angles, $\mathcal{P}$ is chosen as the one with the smallest area.

In urban reconstruction, the shape of a surface is not known a priori. For this reason, we have analyzed whether our method reconstructs the correct shapes. We measure the correctness of a shape by its number of edges and its angle of rotation, by comparing them to the ground truth. In the ideal case, there is a value for δ at which our method always produces the correct shape. However, it is very likely that the ideal value of δ depends on ϱ. To promote shape-invariance, we are looking for a δ for which our method performs well at the various settings for ϱ, irrespective of ground truth case.

Our experiments show that the chosen polygons $\mathcal{P}$ always have a rotation within

1 degree of the rotation of the ground truth. Figure 4.7 shows the ranges of δ values for which the constructed polygon has the same number of edges as the ground truth. At the higher densities (100 and 200 points/m^2), we can choose the value of δ at 24 cm and 20 cm respectively to construct a shape with the same number of edges for most cases.

At the lowest density (50 points/m^2), there is no δ value that consistently results in a polygon with the correct number of edges. Additionally, Figure 4.7 shows that in half of the cases the range of δ resulting in the correct number of edges is very small. These cases all have some small features that add edges to the ground truth while being difficult to make out in the rough point samples. Visual inspection showed that the openings of the "grip" cases were closed off in their α-shape, explaining the problems with reconstructing a shape with the correct number of edges. For the cases without a small feature, a value of δ between 50 and 85 results in polygons with the correct number of edges.

Considering the values of α and δ together, we observe that we should choose δ slightly larger than α. However, by Lemma 4.1.2, we are not guaranteed to get an (α,δ)-sleeve with the topology of an annulus in this case. Especially in "grip" cases, finding ways to correctly deal with (α,δ)-sleeves that do not have an annulus topology may yield better results.

4.2.2 Data-Dependent δ

The shapes encountered in urban scenes vary greatly. Small features of the shape combined with insufficient sampling density may make it very difficult to correctly estimate the number of edges of the shape, even for a human modeler. Additionally, the sampling density may vary greatly between surfaces. For this reason, it may be interesting to estimate the best value for δ from the point data itself instead of choosing some fixed value.

One way to determine which δ is best is to analyze how $\mathcal{P}$ changes as δ increases. At the smallest δ, $\mathcal{P}$ will approximate the α-shape and from some large δ onwards, $\mathcal{P}$ is a rectangle. Recall that $\mathcal{P}$ is the shrunken version of $\hat{\mathcal{P}}$, so we can expect the largest changes in $\mathcal{P}$ when its number of edges change. Additionally, given a fixed number of edges, it is likely that the polygon that approximates $\mathcal{T}$ best is found immediately after a jump to that number of edges. While growing δ, we use the polygon just after each jump as representative for the polygons with the same number of edges. This leads to a succession of representative polygons $\{\mathcal{R}_0 \dots \mathcal{R}_k\}$, where each consecutive polygon has fewer edges, culminating in four edges at $\mathcal{R}_k$.

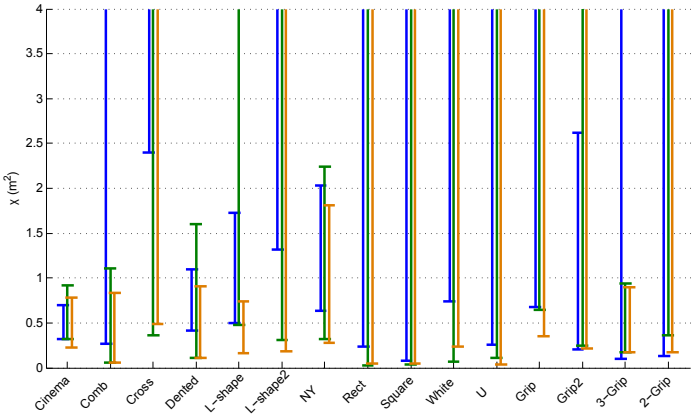


Figure 4.8: The range of $\bar{\chi}$ for which the polygon selected by visual inspection is constructed at point densities 50 (blue), 100 (green), and 200 (orange) points/m^2.

The best $\mathcal{R}$ should strike a balance between complexity and approximation of the shape of $\mathcal{T}$. The complexity can again be measured in the number of edges, with a preference for low complexity. How well $\mathcal{R}$ approximates $\mathcal{T}$ could be measured from their area of symmetric difference. Unfortunately, for real-world data the ground truth is not known, so the symmetric difference cannot be used to determine the best δ. However, because the area of $\mathcal{T}$ is fixed, a simple approximation for the symmetric difference is to use the area of $\mathcal{R}$.

Because $\mathcal{R}$ always covers the α-shape, we can assume that the reason for large jumps in the area of $\mathcal{R}$ is the removal of a large feature of $\mathcal{T}$ from $\mathcal{R}$. The goal then becomes to determine when the process of increasing δ stops reducing the complexity of the shape and starts removing large features. If we denote by χ_i the change in area between consecutive representative polygons $\mathcal{R}_i$ and $\mathcal{R}_{i+1}$, we search for a threshold value $\bar{\chi}$ for χ_i. The idea is that if the area of $\mathcal{R}$ makes a jump of $\chi_i > \bar{\chi}$ due to a decrease in the number of edges, then the polygon loses a key feature of $\mathcal{T}$ so $\mathcal{R}_i$ is the simplest polygon that contains all important features.

We determine $\bar{\chi}$ by selecting which reference polygon $\mathcal{R}_i$ best approximates $\mathcal{T}$ by visual inspection and computing χ_i. Choosing $\bar{\chi}$ such that it is smaller than χ_i but larger than χ_j for all $j < i$ would result in terminating the automatic search for δ at the preferred polygon. Because we want the method to be shape-invariant, we are looking for a value of $\bar{\chi}$ for which our method performs well, irrespective of ground truth case.

The ranges of $\bar{\chi}$ resulting in the polygon selected by visual inspection are shown in

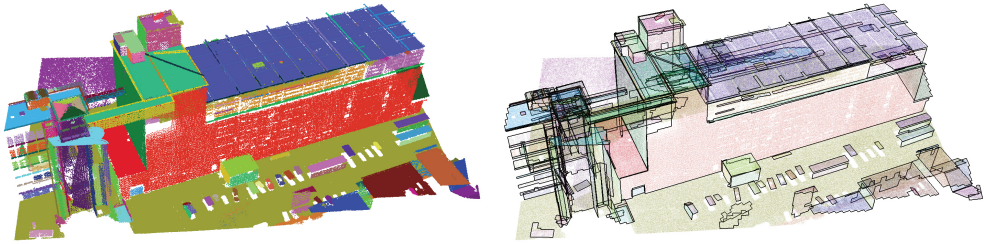


Figure 4.9: A LiDAR data set with points colored per surface and the rectilinear boundaries of those surfaces.

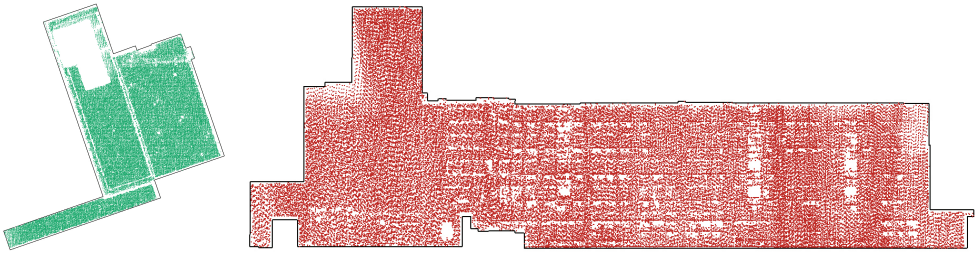


Figure 4.10: Two interesting rectilinear surfaces.

Figure 4.8. At the higher densities (100 and 200 points/m^2), there are values for $\bar{\chi}$ that result in the correct polygon in all the test cases. At a density of 50 points/m^2, there is no $\bar{\chi}$ that produces the correct polygon in all cases, although 0.8 m^2 is a good value for most cases.

4.2.3 LiDAR Data

Apart from synthetic data, we have also applied our method to an airborne LiDAR data set of a building, as shown in Figure 4.9. The point density varies greatly between surfaces, because of the scanning method. However, the same as in Chapter 3, we set α to 125 cm for surfaces close to vertical and 60 cm for the other surfaces.

The building contains many rectilinear surfaces, which we have reconstructed using $\delta = 60$ cm. The surfaces near the edge of the scene were given jagged edges because the buildings are not aligned with the scene. However, our results do favor long straight edges for the remainder of the shape.

Figure 4.10 shows two interesting surfaces and their rectilinear boundaries. Note how the rotation of the reconstructed polygons matches the neighboring surfaces. A human

modeler may construct a similar shape with fewer edges for these surfaces. It seems that there are two reasons for these 'faults' in our results. Firstly, parts of our shapes continue past neighboring surfaces. Secondly, if you look at these two surfaces in the scene in Figure 4.9, you may notice that many of the small edges of the red shape are in regions where the building is under-sampled. In both cases, a human modeler may choose to bend the proximity or coverage rules a bit to produce longer edges and thereby a simpler shape.

4.3 Conclusions

We have presented a novel concept, the (α,δ)-sleeve, which contains a proximity zone around a point set. When combined with a minimum-link path algorithm, this structure can be used to reconstruct simple shapes that contain the point set. We have presented a method aimed at reconstructing surfaces in urban scenes from a point set by combining the (α,δ)-sleeve with a rectilinear minimum-link path. The (α,δ)-sleeve can be computed in $O(n \log n)$ time and the contained minimum-link path can be constructed in $O(kn)$ time, where n is the number of points and k is the number of considered orientations. Our experiments showed that when the surfaces are sampled sufficiently dense, there are parameter settings for α and δ that lead to correct reconstructions for artificial data. Finally, we have shown on an urban LiDAR data set that our method produces plausible results.

A number of interesting possibilities for extensions and improvements remain. In most urban scenes there are some surfaces that are not rectilinear. By changing the rectilinear minimum-link path algorithm to allow a few edges that do not follow one of the principal directions, the (α,δ)-sleeve can be used to reconstruct such surfaces as well. Alternatively, we could use a post-processing method that replaces long stair-like parts in the rectilinear surface boundary by one line.

Another recurring problem we encountered is the rounding of concave corners. As Figure 4.11 shows, the α-shape can "round off" a concave corner, often going outside the original polygon. In such cases, constructing a polygon within the (α,δ)-sleeve requires either more edges or a larger δ. This problem may be overcome by using pieces of the α-hull (Edelsbrunner et al. 1983) as inner boundary of the (α,δ)-sleeve, instead of the α-shape. The α-hull uses concave circular arcs between its boundary vertices, allowing the polygon to go deeper into concave corners. Unfortunately, using α-hull arcs everywhere may cause the inner boundary to self-intersect, which in turn can result in a self-intersecting rectilinear path. Hence, this solution would require additional steps to ensure that the constructed polygon does not have self-intersections.

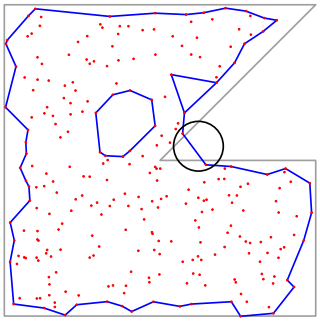


Figure 4.11: An example α-shape where a concave corner is rounded off.

Finally, for this work we have ignored holes in the α-shape. Reconstructing those holes can be done in a fashion similar to the method described in this paper. The (α, δ)-sleeve would consist of several components, each one with the topology of an annulus. We can run minimum-link path algorithms in each annulus with the same orientations in order to find an orientation that is best overall. This way we can easily ensure that the rectilinear directions of the outside boundary and of the hole boundaries are the same.

Chapter 5

Distribution-based shapes

In the previous two chapters, we presented different methods to construct rectangular or rectilinear shapes for a point set sampled from a planar surface. However, it is unlikely that all surfaces in an urban scene can be reconstructed correctly using only 90-degree corners. When shapes are required that follow the point distribution more closely, the α-shape may be an appropriate method, as presented in Section 2.4. However, as Figure 5.1 shows, the α-shape may surround its point set too closely, ignoring the region between neighboring surfaces.

In many cases the process of reconstructing the shape of a surface can be influenced by the intersection lines of the surface with its neighbors. After reconstruction these surfaces should fit together neatly to form a watertight geometric model of the scene. The α-shape can be influenced by lines in three basic ways. When we say 'lines', the concept can generally also be applied to half-lines and line segments.

Firstly, the lines may denote infinite clusters of sample points. In this case we call the lines *generalized points* and the lines should be interior to the *generalized α-shape*. This case is similar to the way line segments are handled in the Voronoi diagram generalized to points and line segments (Yap 1987). We describe a method for determining the α-shape of a set of generalized points in Section 5.1. This simple adaptation of the α-shape will always result in a shape containing the line segments. However, we will show that some artifacts, which occur mainly in corners, make this approach less than optimal when we want to compute a shape using the line segments as boundary.

Secondly, the line segments may indicate likely locations of the boundary of the shape. In this case we call the segments *guides* and the boundary of the *guided α-shape* should follow the line segments where appropriate, as described in Section 5.2. This problem is the main subject of this chapter, because the method will result in a polygon suited to bounding a surface in the geometric reconstruction process. The guided α-shape differs from the regular α-shape on another point: the vertices of the shape are not necessarily

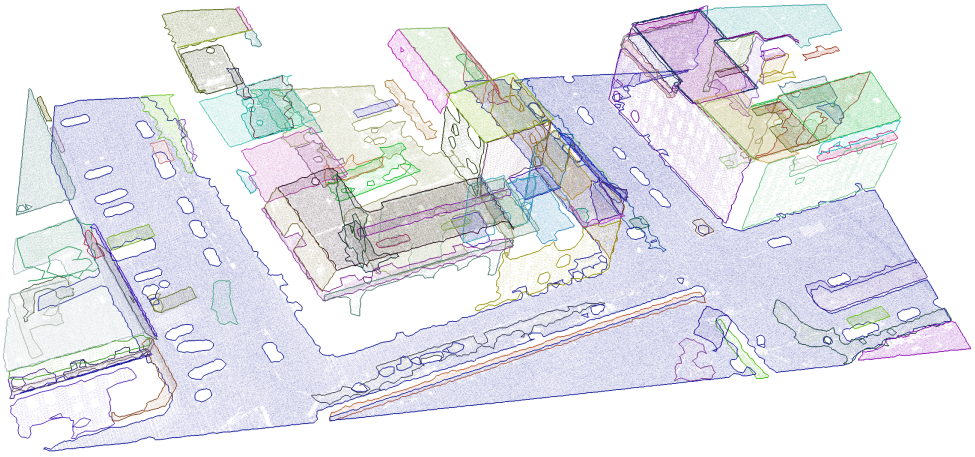


Figure 5.1: The α-shapes of the surfaces in an urban LiDAR scene. Note the gaps between neighboring surfaces.

part of the point set. The guided α-shapes of the point set of Figure 5.1 are shown in Figure 5.2. Experimental results on the guided α-shape are presented in Section 5.3.

Thirdly, the line segments may indicate forbidden regions where the shape is not allowed to be, without directly indicating the boundary of the shape. In this case we call the segments *shields* and the line segment must be outside the *shielded α-shape*. This shape may be interesting for applications like geometric interpolation of earth surface samples in the presence of fault lines. Because computing this shape has proven to be very complex and because the results are less interesting for urban reconstruction, this problem is left as future research. The reason for addressing it here is just for completeness.

Similar to the way α-shape construction is based on the Delaunay triangulation and Voronoi diagram, our methods are based on the constrained Delaunay triangulation (Chew 1989) and generalized Voronoi diagram (Yap 1987). The Delaunay empty-circle criterion and Voronoi closest-point criterion are natural ways of determining the interior of a shape of a point set. Unlike the standard α-shape, which is constructed purely from the data points, the generalized and guided α-shapes are also guided by lines and therefore constructed using structures that are influenced by lines.

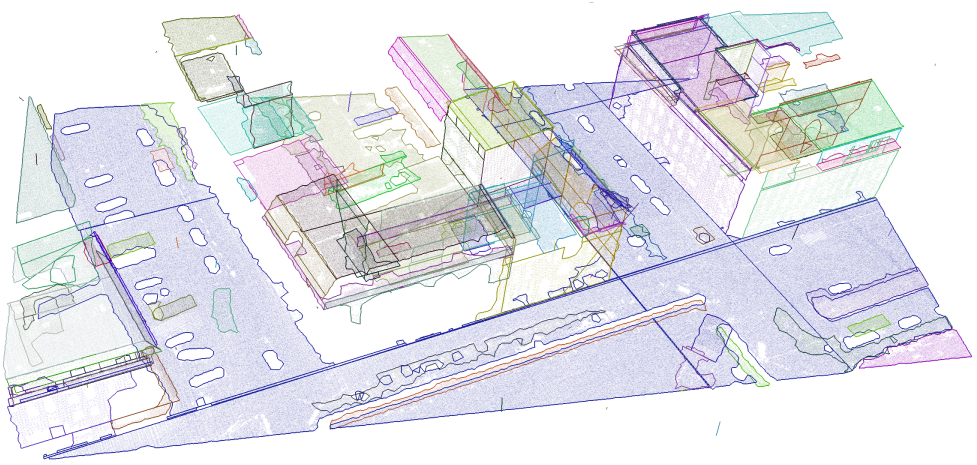


Figure 5.2: The guided α-shapes of the surfaces in an urban LiDAR scene. These connect neighboring surfaces along an edge.

5.1 Generalized α-Shape

Lines and line segments can indicate the interior of the shape. In this case the shape of a collection of points and lines contains all points and lines in its interior. Inversely, the exterior of the shape is directly related to the union of all empty disks of a fixed radius. This structure is in many ways similar to the α-shape.

Definition 5.1.1 (Generalized α-shape)**.** *Given a point set S and a set L of lines, half-lines, and line segments, the generalized α-shape of $S \cup L$ is the α-shape of $S \cup \{x | x \text{ is a point on } l \in L\}$.*

Similar to Theorem 2.4.2, the generalized α-shape separates the internal and external faces of the shape.

Lemma 5.1.2. *The generalized α-shape of a set of point S and a set of lines L partitions the plane into internal and external faces.*

Proof. The generalized α-shape of S is equal to the α-shape of S', where S' is the union of S and all points on a line of L. Like the α-shape, the generalized α-shape is bounded by edges between two points that share an empty open disk of radius α. Note that such an empty disk also cannot intersect a line of L, because otherwise its interior must contain a point on that line.

These edges may connect two points of S or a point of S and a point on a line of L just like the α-shape does. When connecting a point in s and a point p on a line $l \in L$, the empty disk must be tangential to l in p. A special case occurs when there is a segment s of a line of L such that there are no points within 2α of at least one side of s. In this case, the infinite point set on s is on the boundary of the generalized α-shape, with degenerate edges connecting the points. This part of the boundary may be regarded as if it were an edge connecting the endpoints of the segment, ignoring the intermediate points except for the purpose of the empty disk criterion where points not on s are involved.

We maintain Definition 5 (Edelsbrunner et al. 1983) for interior and exterior faces, with the popular adjustment that we read α to describe the radius of the disks. This definition first defines an edge e of a face F of the shape as positive based on the open disk d with radius α that touches the endpoints of e and has its center strictly on the same side of e as F. This edge e is positive if and only if d contains a point. Note that for the generalized α-shape, this point may be in S or lie on a line in L.

We extend the definition of positive edges to segments on a line of L. An edge e of a face F of the shape that lies on a line of L is positive if and only if there is an open disk d with radius α such that d is tangent to e in a point in its interior and d contains a point.

Finally, similar to Definition 5 (Edelsbrunner et al. 1983), F is interior if it has a positive edge and it is exterior otherwise. Similar to the α-shape, it is rather straightforward but lengthy to prove that either all or none of the edges of a face are positive. □

It seems safe to assume that the interior faces of the generalized α-shape cover a superspace of the interior faces of the α-shape without lines. However, just like adding points to the α-shape can reduce the region its interior faces cover, adding line segments to the shape can move the boundary inward, as shown in Figure 5.3.

The main problem generalized α-shapes are faced with is modeling the effects of the infinite collection of points on the lines without reducing the computational efficiency of constructing the shape.

5.1.1 Constructing the Generalized α-Shape

We will construct the generalized α-shape using line segments instead of lines, because we make use of the medial axis and its definition is not extended to lines. However, the extension to lines is straightforward. Just replace lines by line segments with

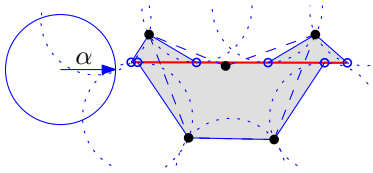


Figure 5.3: The generalized α-shape of a point set does not always bound a larger region than the shape without line segments. The black disks are the input points and the thick red line segment is added in the generalized α-shape. The dashed blue lines are in the α-shape and the generalized α-shape consists of the solid blue lines.

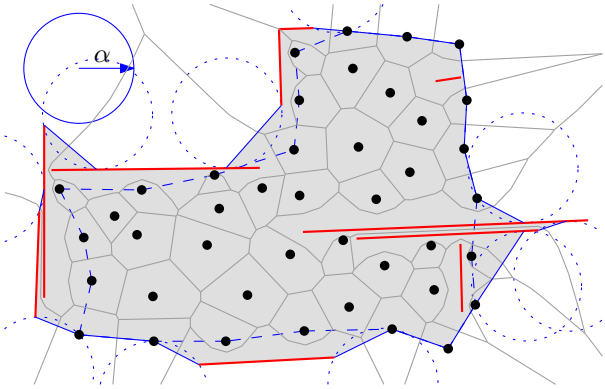


Figure 5.4: The construction of the generalized α-shape. The black disks are the input points and the thick red line segments are the input line segments. The dashed blue lines are in the original α-shape and the line segments add the solid blue lines between the points that are at distance α from the medial axis.

endpoints located very far away.

Conceptually, the generalized α-shape considers the line segments as an infinite number of points to add to the point set. In practice, the generalized α-shape is composed of a number of edges linear in the number of points n and line segments m, and can be constructed in $O((n+m)\log(n+m))$ time, using $O(n+m)$ space. The process is shown in Figure 5.4. An important observation to achieve an efficient algorithm is that there is a limited number of points on the line segments that influence the shape: all empty open disks with a point from a line segment on its boundary must either be tangent to the segment or the disk only touches one of the endpoints of the segment.

Theorem 5.1.3. *The generalized α-shape of n points S and m line segments L that intersect at most in their endpoints can be constructed using $O((n+m)\log(n+m))$ time and $O(n+m)$ space.*

Proof. The generalized α-shape has an edge between each pair of points on the boundary of an empty disk with radius α. These edges can be divided into three categories: point-point, line-point, and line-line. The case of point-point edges is the same as an edge formed in the regular α-shape, with the addition that the disk cannot intersect a line segment in its interior.

In the case of an edge connecting the infinite point set represented by a line segment to a point $p \in S$, the edge will be incident to either the endpoint of the segment or a point where a disk with p on its boundary is tangent to the line. Otherwise the disk must intersect the segment in its interior meaning it is not empty.

In the case of an edge connecting two line segments, both vertices can be either an endpoint of a segment or a point where the empty disk is tangent to the line.

In all three cases the center of the disk has distance α to both incident vertices, whether point or line segment. This information is captured in the medial axis of $S \cup L$.

The medial axis of N points and line segments can be computed in $O(N \log N)$ time (Yap 1987) and contains $O(N)$ straight or parabolic edges. A straight or parabolic edge can have at most two points at distance α to the elements it separates. A brute force search of all edges in the medial axis will produce the generalized α-shape in $O(n+m)$ time from the medial axis. □

5.2 Guided α-Shape

The α-shape cannot take into account the context of the shape. In surface reconstruction, we may know some of the neighboring surfaces, so we have an indication of where the surface ends. The guided α-shape takes advantage of these indicators to construct a better shape.

The guided α-shape is the shape of a set of points in the plane, which we will refer to as sites, where a set of non-intersecting line segments, called guides, indicate preferred locations for parts of the boundary. Guides determine the boundary of the shape only near sites. Further away from either guides or sites, the shape is the same as the α-shape. That is: without nearby sites the guides are not part of the guided α-shape and without nearby guides, the guided α-shape is the same as the α-shape.

The definition of the guided α-shape is based on two concepts: the connected α-disk and the α-projection. A connected α-disk is a connected component of a disk. This structure replaces the α-disk in determining the guided α-shape as compared to the

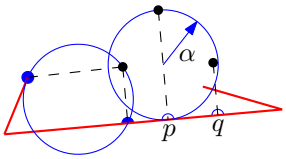


Figure 5.5: The α-projections are shown as blue (half) disks. The blue semi-circles are not α-projections, because one of the conditions of Definition 5.2.2 is not met: p does not share an empty connected disk with its source and the segment connecting q to its source intersects a guide.

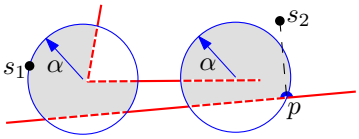


Figure 5.6: Connected α-disks of site s_1 and a projection p of s_2 are indicated by the gray regions. The dashed parts of the guides are part of the connected α-disks, but the circular blue boundaries are not.

original α-shape. This limits the influence of sites across a guide.

An α-projection is near the point on a guide closest to a site, its source, and it must conform to some conditions, as shown in Figure 5.5. These conditions, listed in Definition 5.2.2, are chosen to elegantly identify appropriate parts of the guides for the shape boundary.

Definition 5.2.1 (Connected α-disk)**.** *Given a point p, a set of guides G and a positive value α, let $\mathcal{D}$ denote an open disk of radius α with p on its boundary. A connected α-disk of p is the closure of a connected region of $\mathcal{D} \setminus G$ that contains p, minus the boundary of $\mathcal{D}$ (shown in Figure 5.6).*

Definition 5.2.2 (α-projections)**.** *Given a set of sites S, a set of guides G, and a positive real value α, a point p is an α-projection of $s \in S$, called the projection's source, onto a guide $g \in G$ if it conforms to the following three conditions.*

> ***Closest:*** *q is the point on g closest to s, and p lies on the open line segment $\overline{qs}$ infinitesimally close to q and on the same side of g as s.*
>
> ***Visible:*** *the open line segment $\overline{sp}$ does not intersect any guide in G.*
>
> ***Neighbor:*** *p and s share a connected x-disk empty of sites, where $x \leq \alpha$.*

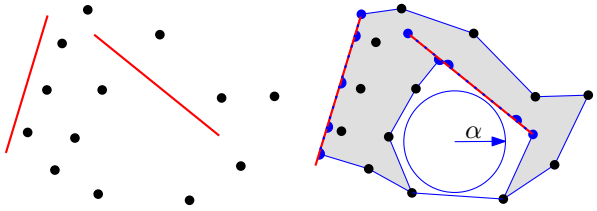


Figure 5.7: A set of sites and guides and the line-guided α-shape of this set. The blue disks and half-disks are α-projections.

$P^{\alpha}_{S,G}$ *is the set of all projections of the sites in* S *onto the guides in* G.

Where the point set is near a guide, the α-projections are the vertices of the shape boundary. This is achieved by using the α-projections as additional points, similar to the Steiner points added in conforming Delaunay triangulations (Edelsbrunner and Tan 1992; Ruppert 1995). However, unlike Steiner points, all projections lie infinitesimally close to a guide and strictly to one side of it.

There are three important subtleties to the α-projections. Firstly, the α-projection p of site s onto guide g is not exactly on g. Therefore, the connected component containing p of an α-disk is always on the same side of g as s. This, in turn, means all connected α-disks of p are on the same side of g as s. This is not the same for a site located on a guide; this site has connected α-disks on either side of the guide.

Secondly, an α-projection p of site s always has an empty connected α-disk. Imagine taking a sequence of connected α-disks of p, each next one further away from s than the current. At some point, the connected α-disk will be empty.

Finally, the point q on a guide closest to site s may be the guide endpoint. There is an α-projection of s near q if it follows the conditions of Definition 5.2.2. Multiple sites may project onto this endpoint, giving multiple α-projections near q. In an endpoint q where multiple guides meet, these guides divide the space around q into wedges. All connected α-disks of a projection onto q intersect the same wedge as its source.

Armed with the definitions for the connected α-disk and projections, we can now define the line-guided α-shape, shown in Figure 5.7. In the absence of guides, this shape is the same as the original α-shape; near guides, the shape is adapted following the shortest route to use the guides as boundaries.

Definition 5.2.3 (Line-guided α-shape)**.** *Given a set of sites* S*, a set of guides* G *that intersect at most in their endpoints, and a positive real value* α*, let* $P^{\alpha}_{S,G}$ *denote the set*

of α-projections of S onto G.

A point $s \in S \cup P^{\alpha}_{S,G}$ is α-extreme if there exists a connected α-disk of s empty of sites and α-projections. Two points in $S \cup P^{\alpha}_{S,G}$ are α-neighbors if they share a connected α-disk that is empty of sites and α-projections.

The line-guided α-shape of S and G is the straight line graph whose vertices are the α-extreme points and whose edges connect the respective α-neighbors.

Observation 3. Three observations follow naturally from the terms defined above.

1. If there are no guides, the α-extreme points are the α-extreme points of the original α-shape.
2. Projections are generally α-extreme.
3. Because a projection must share an α-disk with its source, these can be no further than 2α apart. This also means that each guide has a maximal reach of 2α in which it can influence the shape.

The observant reader may have noticed that the definition of the line-guided α-shape causes guides surrounded by sites to also be part of the boundary, even if the space between these boundary parts is degenerate. This is deliberate and follows the idea that guides indicate boundaries. Guides dividing the point set may indicate a boundary between different components, even if these components touch. Similarly, these guides may indicate where an intersecting surface meets this shape in the ambient 3-dimensional space.

The line-guided α-shape is like an α-shape that is influenced by guides. However, some situations produce undesirable artifacts in line-guided α-shapes, as shown in Subsection 5.2.1. This problem is solved by enforcing an additional constraint on the connected α-disks, leading to the definition of guided α-shapes. As an additional advantage, this constraint drastically reduces the number of projections, as explained in Subsection 5.2.2. In Subsections 5.2.2 to 5.2.5, we present our algorithm to construct the guided α-shape and we discuss some artifacts arising in practice.

5.2.1 The Influence of Endpoints

The primary goal of the guided α-shape is to adjust the α-shape such that it uses the supplied guides as boundary where appropriate. The definitions given in the previous section partially achieve this goal: the guides influence the shape if they are near

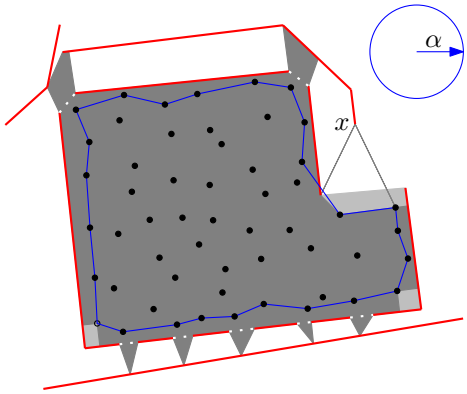


Figure 5.8: An example of realistic input where the line-guided α-shape (dark gray) gives undesired results. Slightly adjusting the criteria near guide endpoints gives the guided α-shape. This shape does not have α-projections on the other side of the small gaps indicated by the dotted lines, because they are no longer visible. The light gray regions are covered, and the shape does not connect to x. The original α-shape is blue.

it. However, in some practical cases, the influence of guides is not appropriate, see Figure 5.8.

The line-guided α-shape can be improved by taking into account the locations of the endpoints of the guides. In some cases, the locations where a guide starts and ends is not important, or not exactly known. In other cases there is a clear indicator for the endpoint of a boundary edge. An example is the intersection point where three neighboring surfaces meet.

For the remainder of the paper, we will assume all endpoints of the guides are such explicit indicators of boundary corners. If the endpoint of a guide is less explicit, it can be placed very far away and the orthogonal projection feature of the guided α-shape will make sure that only the part of the guide near the point set is used. To make the endpoints more explicit in guiding the shape, they disrupt empty connected α-disks and provide more projections.

Definition 5.2.4 (Guided α-shape)**.** *A connected α-disk is* valid *if it is empty and does not contain a guide endpoint. For α-projections each endpoint of a guide is handled as if it is a degenerate (length 0) guide. The guided α-shape is the line-guided α-shape with all references to empty connected α-disks replaced by valid connected α-disks.*

The guided α-shape follows the guides and their endpoints where they have sites nearby, while reverting to the α-shape in absence of such indicators. The validity con-

straint makes sure that the shape will not cross small gaps between endpoints, unless there are sites that indicate the shape continues on the other side.

The endpoints also provide additional α-projections, because they are handled as additional guides. If a site is near an endpoint, this endpoint is the point on the additional guide closest to the site. The result is that an α-projection is added in convex corners near sites.

Like the α-shape, the guided α-shape separates interior regions or faces from exterior faces. The interior faces do not contain the center of a valid connected α-disk. Similar to the α-shape, the interior faces of the guided α-shape are the union of the triangles in the triangulation of the sites, guides, and projections that have a circumdisk with radius smaller than α. This can be proven comparable to how Edelsbrunner et al. (1983) prove the faces of the α-shape are a subset of the Delaunay triangulation.

Lemma 5.2.5. *The interior faces of the guided α-shape are the union of the triangles of the constrained Delaunay triangulation (CDT) C on the sites, guides, and projections with a circumdisk of radius $x < \alpha$.*

Proof. It is straightforward to see from the definition of the α-neighbors that the edges and vertices of the guided α-shape are a subset of the edges and vertices of C. The interior and exterior faces are separated by edges of the guided α-shape. It is not uncommon for some of these exterior faces to be degenerate. For example, because projections are infinitesimally close to a guide, there are degenerate (area 0) faces between the projections and guides.

An interior face can be identified from the fact that it does not contain the center of a valid connected α-disk. The triangles in C that do not contain the center of an valid connected α-disk must have a circumdisk with radius smaller than α. By contrast, an exterior face only contains triangles of the CDT with a circumdisk with radius at least α: the triangles that contain the center of a valid connected α-disk. The edge between an interior and exterior triangle in C must connect two α-neighbors by Definition 5.2.4. □

5.2.2 Number of Projections

Validity, as described in Definition 5.2.4, improves the resulting shape. It also drastically reduces the worst-case bound on the number of projections. Because each site is projected onto any guide at most once, there can be at most $O(n * m)$ projections of n sites onto m guides. It is possible to construct examples that show $\Theta(n * m)$ is indeed

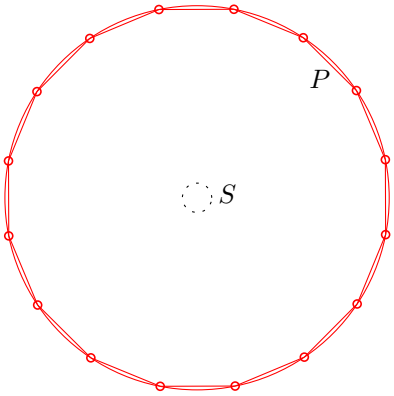


Figure 5.9: An example resulting in $\Theta(n * m)$ projections if validity is not enforced. The n sites are evenly spread over the dotted circle S and all endpoints of the m guides are spread evenly over the red circle P. If validity is not enforced, each guide contains projections for half of the sites.

the worst-case bound on the number of projections if validity is not enforced, as shown in Figure 5.9. However, we will show that if the connected α-disks must be valid, there are at most $O(n+m)$ projections.

We use two steps to prove there are a linear number of projections if the connected α-disks must be valid. We first prove that in the constrained Delaunay triangulation (CDT) on the sites and guides, each projection must be inside the circumdisk of a triangle incident to its source. We then show that any circumdisk of a CDT triangle can contain at most two projections of each of its incident vertices. Because the CDT has $O(n+m)$ triangles, there are $O(n+m)$ projections. Figure 5.10 shows most of the structures used in the proofs.

Lemma 5.2.6. *In the CDT of the sites and guides, each projection is inside the circumdisk of a triangle incident to its source.*

Proof. For any projection p and any of its sources s, take a disk $\mathcal{D}$ with radius $x \leq \alpha$ that has s and p on its boundary and that has a connected component $\mathcal{C}$ that does not contain any sites or guide endpoints, and has s on its boundary. Note that according to Definition 5.2.2, such a disk must exist for each pair of projection and source and according to Definition 5.2.1 $\mathcal{C}$ is an connected x-disk of s touching p.

Let $\triangle sxy$ be the unique triangle of the CDT intersected by the line segment $\overline{sp}$. If p

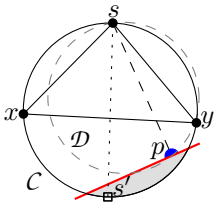


Figure 5.10: The different structures used in the proofs of Lemmas 5.2.6 and 5.2.7. Site s is a source of projection p, x and y are points adjacent to s and sharing the boundary of the CDT circumdisk $\mathcal{C}$ shown in black. The dashed gray disk $\mathcal{D}$ shows p is actually a projection. The point opposite of s on the disk is s'. The gray region is the part of the disk behind the guide.

lies on $\overline{xy}$, it is trivially inside the circumdisk of $\triangle sxy$. Otherwise, $\overline{sp}$ intersects $\overline{xy}$ and $\overline{xy}$ cannot be a constraint. This means that for any circumdisk $\mathcal{Q}$ of $\overline{sp}$, if x or y is inside $\mathcal{Q}$, then it must be in the same component of the connected disk in $\mathcal{Q}$ as s and p. Because $\mathcal{C}$ is a valid connected disk, $\mathcal{D}$ cannot contain x or y. Therefore, the circumdisk of $\triangle sxy$ must contain p. □

Lemma 5.2.7. *Given a set of n sites and m guides that intersect at most in their endpoints, there are $O(n+m)$ projections.*

Proof. We will show that each circumdisk $\mathcal{C}$ of a CDT triangle incident to a site s is divided into two parts by the diagonal between s and opposite point s'; each half of $\mathcal{C}$ can contain at most one projection of s. As there are $O(n+m)$ such circumdisks in the CDT and each is incident to at most three sites, this constitutes a proof. Furthermore, there are at most $2m$ projections on an endpoint of a guide.

Assume a half of disk $\mathcal{C}$ contains two projections p_1 and p_2 of s on guides g_1 and g_2 respectively. Both g_1 and g_2 must completely intersect $\mathcal{C}$ and we call the regions behind the guides relative to the site r_1 and r_2 respectively. Because the projections are not on an endpoint of g_1 or g_2, the lines connecting source and projection are orthogonal to their respective guide. According to Thales' theorem, both r_1 and r_2 must contain s'. Therefore one of three cases are possible: r_1 contains p_2 and therefore $\overline{sp_2}$ intersects g_1, or similarly r_2 contains p_1, or g_1 and g_2 intersect inside $\mathcal{C}$. As all three cases contradict the assumptions, both halves of $\mathcal{C}$ can contain at most one projection of s. □

5.2.3 Cumulus Tree

There is an interesting inherent structure in the collection of CDT triangles for which the circumdisks overlap a certain constraint. This structure can be used to both ef-

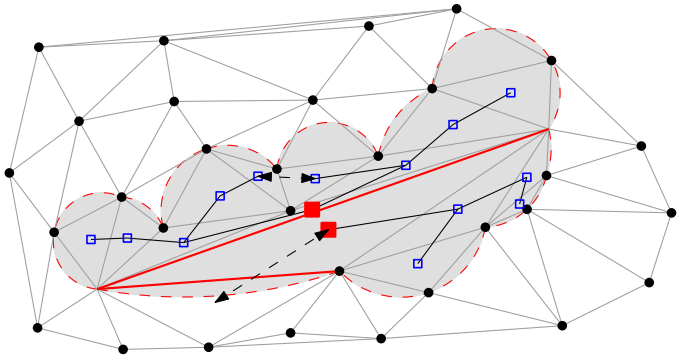


Figure 5.11: The cumulus trees of a constraint. The red squares are the roots of the trees and the black edges show which nodes are connected. The nodes connected by the dashed arrows do not share an edge in the cumulus tree, because the intersection of their disks does not overlap the constraint (top), or the edge in the CDT is a constraint (bottom).

ficiently find the projections and calculate the radius of the smallest valid connected disk they share with their sources, as explained in Subsection 5.2.4. We call this structure a *cumulus tree*, based on its cloud-like appearance. The two cumulus trees of a constraint are shown in Figure 5.11.

Definition 5.2.8 (Cumulus tree)**.** *A cumulus tree T_c associated with constraint c of a CDT is a subset of the dual graph of the CDT. Each node of T_c corresponds to a triangle in the CDT for which the circumdisk intersects c. Two nodes share an edge if their corresponding triangles share a non-constrained edge e in the CDT and the intersection of their circumcircles intersects c. Finally, T_c is the maximal, connected graph that contains a node corresponding to a triangle incident to c.*

Each constraint c is associated with two cumulus trees T_c, rooted in the two triangles incident to c. These cumulus trees have nice properties, given in the following two lemmas. Lemma 5.2.9 implies that the graph of a cumulus tree is actually a tree. Note that a cumulus tree T_c need not contain all circumcircles in the CDT that overlap c, only those not separated from c by another constraint.

Lemma 5.2.9. *Let T_c be a cumulus tree of a constraint c, let u be the node corresponding to the triangle incident to c, and let $\mathcal{D}_x$ denote the circumdisk of the triangle associated with node x. Then for any simple path on T_c from u to another node w, we have: if v lies on the path between u and w, then $\{\mathcal{D}_w \cap c\} \subseteq \{\mathcal{D}_v \cap c\}$.*

Proof. Because u is incident to c, $\{\mathcal{D}_u \cap c\} = c$. For any edge $\overline{vw}$ in the cumulus tree, both $\mathcal{D}_w \cap c$ and $\mathcal{D}_v \cap c$ lie on the same side of the supporting line of the edge e shared

by the triangles associated with w and v. Let v be the node associated with the triangle on the same side of e as $\mathcal{D}_v \cap c$. We can now parameterize the line segment l between the centers of $\mathcal{D}_w$ and $\mathcal{D}_v$ using parameter $t \in [0,1]$, such that $l(0)$ is the center of $\mathcal{D}_w$ and $l(1)$ is the center of $\mathcal{D}_v$. Using this parameterization, let $\mathcal{D}_t$ be the circumdisk of e centered on $l(t)$. Now for any $x, y \in [0,1]$, $\{\mathcal{D}_x \cap c\} \subseteq \{\mathcal{D}_y \cap c\}$ if and only if $x \leq y$. □

Lemma 5.2.10. *Given a CDT on a set of n points and m constraints, the total number of nodes in the cumulus trees of all the constraints is $O(n+m)$.*

Proof. This proof follows straightforward from the proof that there are $O(n+m)$ pairs of triangle and constraint for which the circumdisk of the triangle overlaps the constraint and the constraint is visible to the triangle.

If a CDT triangle's circumdisk overlaps a constraint, there is a disk $\mathcal{D}$ completely inside the circumdisk that is tangent to the constraint and touching a vertex of the triangle. This means that the center of $\mathcal{D}$ is equally distant to the vertex and the constraint and so this vertex and constraint share an edge in the segment-generalized Voronoi diagram (Yap 1987) of the vertices and constraints. As this diagram contains $O(n+m)$ edges and each of these edges bounds two cells, there are $O(n+m)$ pairs of triangles and constraint for which the circumdisk of the triangle overlaps the constraint. □

5.2.4 Constructing the Guided α-Shape

The guided α-shape can be computed in six simple steps. We show that, given n sites, m guides, and a value of α, these steps take $O((n+m)\log(n+m))$ time to determine the guided α-shape. The steps of our method are:

1. Compute the constrained Delaunay triangulation (CDT) of the sites and guides.
2. Compute all possible projections (for any α).
3. Compute the minimal α value for each projection.
4. Select the projections for the given α value.
5. Adjust the CDT to incorporate the projections.
6. Extract the guided α-shape.

The following subsections show the algorithms and running times for steps 2, 3, and 5. The other steps have trivial or well known time bounds:

- Step 1 takes $O((n+m)\log(n+m))$ time and gives an $O(n+m)$ size structure (Chew 1989).

- Step 4 can be done from the collection of all $O(n+m)$ projections in $O(n+m)$ time, by iterating over the projections and keeping only those for which the minimal α value is smaller than the specified α value.

- Step 6 can be done from the adjusted CDT in $O(n+m)$ time, by iterating over all $O(n+m)$ edges and triangles in the structure and keeping those that comply with two conditions. Firstly, they must have a circumcircle with radius smaller than α. Secondly, they cannot be incident to an endpoint of a guide that is not projected onto by a source in the wedge containing the edge or triangle. Note that the first condition is the same as the condition for computing the α-shape from the Delaunay triangulation. The second condition makes sure that only edges on the correct side of a guide are in the shape.

5.2.4.1 Finding Projections

In step 2 of our method, all possible projections are computed. At this stage, α is temporarily set to ∞ to provide the projections; the projections that are too far away will be removed in step 4. The pseudo code for computing the projections is given in Algorithm 5.1. This recursive algorithm starts from a triangle incident to a constraint c and traverses the cumulus tree. For each encountered triangle t, its incident sites are projected onto c. The projections in the circumcircle of t are the projections with an empty connected α-disk.

It is not necessary to explicitly construct the cumulus tree beforehand. Each triangle incident to a constraint c must be a root. After handling a triangle t during traversal, each triangle t_a that shares a non-constrained edge with t is checked. If the intersection of the circumdisks of t and t_a intersects c, then t and t_a share an edge in the cumulus tree. In this case, t_a is handled. The number of triangles that are checked, but not handled can be no more than twice the number of triangles that are handled. Because each triangle can be handled in $O(1)$ time, it follows from Lemma 5.2.6 and 5.2.10 that this algorithm constructs all projections in $O(n+m)$ time.

5.2.4.2 Selecting Projections

In step 3 of our method, the minimum α at which each projection appears is calculated. This is the radius $\tilde{\alpha}$ of the smallest valid connected disk shared by the projection and

Algorithm 5.1 Computing the projections of the guided α-shape.

Input: e: an edge in the CDT, t: a triangle incident to e, c: a constraint, and P the current collection of projections onto c.

```
1:  procedure PROJECT(e, t, c, P)
2:      s ← the vertex of t opposite to e
3:      D ← the circumdisk of t
4:      if c = e or c intersects D on opposite side of e to s then
5:          p ← the point on c closest to s
6:          if p is inside D and p is not an endpoint then
7:              s projects on D: add p to P
8:          end if
9:          e_cw ← the edge of t clockwise to s
10:         if e_cw is not a constraint then
11:             t_n ← the triangle incident to e_cw that is not t
12:             PROJECT(e_cw, t_n, c, P)
13:         end if
14:         e_ccw ← the edge of t counter clockwise to s
15:         if e_ccw is not a constraint then
16:             t_n ← the triangle incident to e_ccw that is not t
17:             PROJECT(e_ccw, t_n, c, P)
18:         end if
19:     end if
20: end procedure
```

one of its sources. These values can be efficiently computed by using the CDT and its cumulus trees.

Lemma 5.2.11. *All the $\tilde{\alpha}$ values at which a projection appears can be computed from the CDT in $O(n+m)$ time.*

Proof. For a projection p on an endpoint of a guide, $\tilde{\alpha}$ can be computed from the circumdisks of the triangles in the CDT p shares with its sources. Note that for any edge $\overline{ps}$ for which the centers of the circumdisks of its incident triangles are on opposite sides of $\overline{ps}$, the circumdisk with $\overline{ps}$ as its diameter is also valid and smaller than either circumdisk.

For a projection p in the interior of a guide, it is less obvious which is the smallest connected disk $\mathcal{D}$ that p shares with its source s. However, because of Lemma 5.2.6

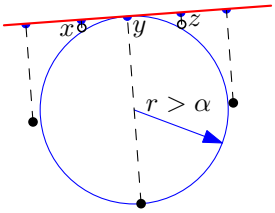


Figure 5.12: α-projections have a minimum α value; at smaller values they do not influence the guided α-shape.

we know that there is a connected disk Q shared by p and s that is completely inside the circumdisk of a triangle t incident to s. Any connected disk shared by s and p that is smaller than Q must have its center closer to the line segment $\overline{ps}$. Either the connected disk C with $\overline{ps}$ as diameter is valid, or there is a point x that also shares the boundary of $\mathcal{D}$; x can be either a site or an endpoint of a guide. We use $\mathcal{D}_y$ to denote the circumdisk of the triangle associated with node y of the cumulus tree.

Because we know $\mathcal{D}$ is a connected disk shared by s and x and because $\mathcal{D}$ is valid, we also know that $\overline{sx}$ is an edge in the CDT. Because p is on constraint c and inside the circumdisk of t and $\overline{ps}$ cannot intersect a constraint, we know that t has a node in cumulus tree T_c. According to Lemma 5.2.9, we know that there is an extreme node y in T_c, such that y is the node furthest away from the root of T_c that contains p in $\mathcal{D}_y$. We can now easily check if the disk C with diameter $\overline{ps}$ is valid. If the centers of Q and $\mathcal{D}_y$ lie on opposite sides of $\overline{ps}$, then C must be valid, because $\{Q \cup \mathcal{D}_y\}$ is valid and $C \subseteq \{Q \cup \mathcal{D}_y\}$.

Once we have determined x, $\tilde{\alpha}$ is the radius of either C or the circumcircle of $\triangle spx$. Which of these two options it is, depends on whether C contains x. □

The necessity of checking the size of the x-disk is shown in Figure 5.12. Here, projection y is a valid projection for larger α values, but at smaller α values x and z are α-neighbors and a false projection y should not interfere here.

5.2.4.3 Triangulated Structure

For step 5 of our method, the algorithm is similar to the algorithm for computing the constrained Delaunay triangulation. The main difference is that as far as predicates are concerned the projections are not on the guide, but on the side of their source. The CDT on n points and m constraints, including projections, can be constructed in

$O(N \log N)$ time, where $N = n + m$.

Some applications may require the shape of the same sites and guides for various α values. For example, we may want to know the smallest α for which the area of the interior faces of the guided α-shape is at least x. Unfortunately, as shown in Subsection 5.2.4.2, the guided α-shape at a certain α-value contains a subset of the projections; adding the part of the projections to an existing CDT is not known to be more efficient than recomputing the complete CDT.

If the value of interest is a monotone function in the value of α, such as the area of the interior of the shape, there is an easy way to compute it: create a sorted list of α values at which the guided α-shape changes and perform a binary search on this list. In this case, the appropriate CDT must be recomputed for each of the $O(\log N)$ α-values considered. This leads to a total running time of $O(N \log^2 N)$. Inspired by Guibas et al. (1992) and Chazelle et al. (2002), we can lower this bound to $O(n \log N + m \log^2 N)$. This may not seem like a major improvement, but note that in practice the number of constraints m is generally much smaller than the number of sites n.

Lemma 5.2.12. *Given n points and m guides that intersect at most in their endpoints, the optimal α for a function $f(\alpha)$ can be determined in $O(n \log N + m \log^2 N)$ time, where $N = n + m$, if $f(\alpha)$ is monotone in α and $f(\alpha)$ can be computed in linear time from the guided α-shape.*

Proof. Following the binary search paradigm, we can determine the optimal α by considering the guided α-shape for $O(\log N)$ values for α. To construct these guided α-shapes, we run steps 1-3 of our algorithm once, and repeat steps 4-6 for each value of α. Steps 4 and 6 take $O(N)$ time per α value, but step 5 costs more time. According to Guibas et al. (1992), inserting K new points into an existing Delaunay triangulation of size M will take update time $O(M \log K + K)$ when employing randomized incremental construction. In our case, $K = N$ and $M = m$. According to Chazelle et al. (2002), given point sets Y, of size N, and $X \subseteq Y$ location time for randomized incremental construction of Delaunay triangulation DT_X in the presence of a known DT_Y, is $O(N)$.

Both results are proven for Delaunay triangulations under randomized incremental construction. However, no algorithm is known for randomized incremental construction of constrained Delaunay triangulations. In case of our specific triangulations, these bounds still hold. This is because all points that are inserted are projections and all projections are next to a constraint. The update bound by Guibas et al. is based on flipping edges around a newly inserted point. As constraints cannot be flipped, there can only be fewer flips. The location bound by Chazelle et al. is based on deconstructing DT_Y in randomized vertex order and constructing DT_X in the reverse

order. This bound does not directly hold for any CDT, because there is no clear and consistent way to randomly insert the constraints. However, in our CDT we can restrict the procedure to the triangles of the cumulus trees, which are locally unconstrained Delaunay. This puts the total running time at $O(\log N) * [O(M \log K + K) + O(N)] = O((m \log N + N) \log N) = O(n \log N + m \log^2 N)$. □

5.2.5 Artifacts Arising in Practice

There are a few artifacts that occur when we apply the guided α-shape method to bound the surfaces for urban reconstruction. Notably, noise and outliers can cause measurement points on the other side of the boundary. The method can be used without change to reconstruct multiple shapes per plane, as long as the point sets are at least 2α apart.

Factors like noise may cause points outside a surface boundary, beyond a guide. Any method for geometric reconstruction should be able to handle these artifacts. While our method is able to handle these points without breaking down, it is preferable if it can also identify the noise and reduce its impact on the result. Because we incorporate guides to indicate preferable locations for boundary edges, we have an estimator for which regions are noisy. The guides partition the shape into multiple components. Components that are very thin compared to the point distribution, are likely caused by noise (see Figure 5.16 (bottom right), region 5). These components are easily identified by computing the Hausdorff distance between the guides and the remaining boundary of the component.

Another common source of artifacts in geometric reconstruction is outliers: points that were incorrectly assigned to a cluster. In the case of urban reconstruction, outliers are mainly caused by vegetation, but points measured through windows are another source. The parts of the shape with outliers are usually significantly less dense and in most cases a neighboring surface provides a convenient guide to separate out the outliers. Our method will automatically incorporate the guide into the boundary, thereby separating the real surface from the outliers. This separation makes identifying regions with lower density easy.

5.3 Results

The guided α-shape method presented in Section 5.2 is applied to synthetic and real-world dense LiDAR data.

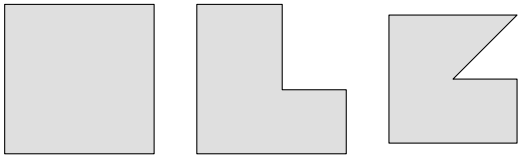


Figure 5.13: The different shapes used in the tests on synthetic data; from left to right: square, L-shape, C-shape.

5.3.1 Synthetic Data

Our goal of running experiments on synthetic data is to ascertain whether or not the guided α-shape is better than the α-shape at reconstructing a shape from a point sample of its interior. The quality of either method is measured by the area of the symmetric difference between the interiors of the produced shape and the original shape. We approach our goal by answering five questions on the quality of the shape produced by the different methods.

1. Is the guided α-shape better than the α-shape when guides are supplied?
2. Does the type of shape influence the individual quality of the α-shape, guided α-shape, or their relative quality?
3. Does the type of guides influence the quality of the guided α-shape?
4. Does the addition of a Gaussian noise model influence the individual quality of the α-shape, guided α-shape, or their relative quality?
5. Does the inclusion of a number of additional false guides influence the quality of the guided α-shape?

Each of these questions is visualized in a graph and tested using a t-test with ($p < 0.05$) as criterion for rejecting the null hypothesis that the quality of both methods was sampled from the same probability distribution. In the rest of this subsection, by 'significant' we indicate that the t-test rejects the null hypothesis.

All tests are done by comparing the α-shape to the guided α-shape on different sets of points and guides. These sets are created by sampling the interior and edges of a polygon. The interior is uniformly sampled up to a certain point density and each edge has 50% chance of being included as a guide. To incorporate variation in the type of shape, we use three different polygons to sample from, shown in Figure 5.13.

Because both the α-shape and guided α-shape are defined locally, we limit the number of shapes used. However, to cover a variety of cases we have included both shapes of varying simplicity and with different corner types (i.e. right convex, right concave, sharp convex and sharp concave). We expect any error in the α-shape and guided α-shape to be near the boundary of the shape. Therefore, we have normalized the errors by giving all shapes the same boundary length of 28. This leads to the interior areas of 49, 37, and 30.95 for the square, L-shape, and C-shape respectively.

The type of guides used may have an important influence on the guided α-shape. We use three different ways of creating guides from the sampled edges of the original shape. The first is using the supporting line of the edge as guide (Lines), the second is using the edge itself as guide (Segments), and the third simulates a guide with a gap in it by cutting the edge into three segments of equal length and using the outer two as guides (Pairs).

For most tests, we directly use the point samples of the interior of the original shape at different densities. However, in practical applications, noise is an unavoidable problem. Therefore, we have created samples with added Gaussian noise. The coordinates of each sample point are perturbed by adding a random value from a normal distribution with $\mu = 0$ and $\sigma = 0.05$. This noise conforms with the typical noise in real data. This noise may have influence mainly by extending the α- and guided α-shapes over the boundary of the original shape.

Finally, we expect that guides have a large influence on the quality of the guided α-shape. While correct guides should force the shape to be more like the original, incorrect guides may create holes, or guide the shape too far outside the shape. This simulates real data, where we are not sure of the correctness of the guides.

To measure the impact of incorrect guides, we have added five fake guides to each sample. Note that this means that the number of fake guides is larger than the expected number of correct guides (i.e. half of the original edges). Each fake guide is created by taking a random point p inside the shape and translating it in a random direction by a third of the shape diameter. The guide contains p and has an independent random direction. If the guide has endpoints (Segments or Pairs setting), one endpoint is p and the other lies the average length of the real edges away from p. Intersecting guides are split into multiple guides.

Because the shape of a point set is dependent on the density of the sampling, each of the tests is done for five different sampling densities. The α-value of both the α-shape and guided α-shape is set to the average distance to the sixth nearest neighbor. This choice is based on the kissing number of unit disks in the plane, and we expect this α

will result in shapes without holes.

The results for the different questions are shown in Figure 5.14. Most of these results are the average over 60 sample sets, but the overall results average 180 samples, and the fake guides per type average 20 samples. Whenever we indicate a (significant) difference, this is supported by t-tests. The α-shape and guided α-shape of one sample are shown in Figure 5.15.

Figure 5.14(a) shows the overall results. The guided α-shape consistently performs significantly better than the α-shape. The shape of the polygon does not have a significant impact on these results, as shown in Figure 5.14(b). The guided α-shape is better than the α-shape on each shape, and the only time the results within one method differ is for the α-shapes of the square and C-shape at density 9.

When we compare the influence of the different types of guides, as shown in Figure 5.14(c), t-tests support two observations. Firstly, the guided α-shape consistently performs significantly better than the α-shape. Secondly, the significance of the difference between the different ways of creating the guides depends on the density. However, for the densities of 9 and upward, the pairs perform the worst, and the results of the segments and lines are comparable.

When we introduce noise in the point coordinates or fake guides, paired t-tests indicate that the guided α-shape still always performs significantly better than the α-shape. A notable result is that within one method, noise in the points can actually improve the results, as is the case for density 25, shown in Figure 5.14(d).

Contrary to our expectations, introducing fake guides does not make the guided α-shape perform worse than the α-shape, as shown in Figures 5.14(e) and 5.14(f). In most cases, the difference between the guided α-shape with and without fake guides is significant. However, these shapes with fake guides still estimate the original polygon significantly better than the α-shape. For example, in Figure 5.14(f), the difference between the guided α-shape with the pairs setting and the α-shape at density 25 seems small, but the paired t-test gives a p-value of $5.79 * 10^{-7}$.

5.3.2 Urban LiDAR

We have applied our method to airborne LiDAR data, which is frequently used for urban reconstruction. We have applied efficient RANSAC (Schnabel et al. 2007) to partition the point set into planar clusters. We have calculated each surface's intersection lines with the other surfaces. Each cluster is projected onto its supporting plane to

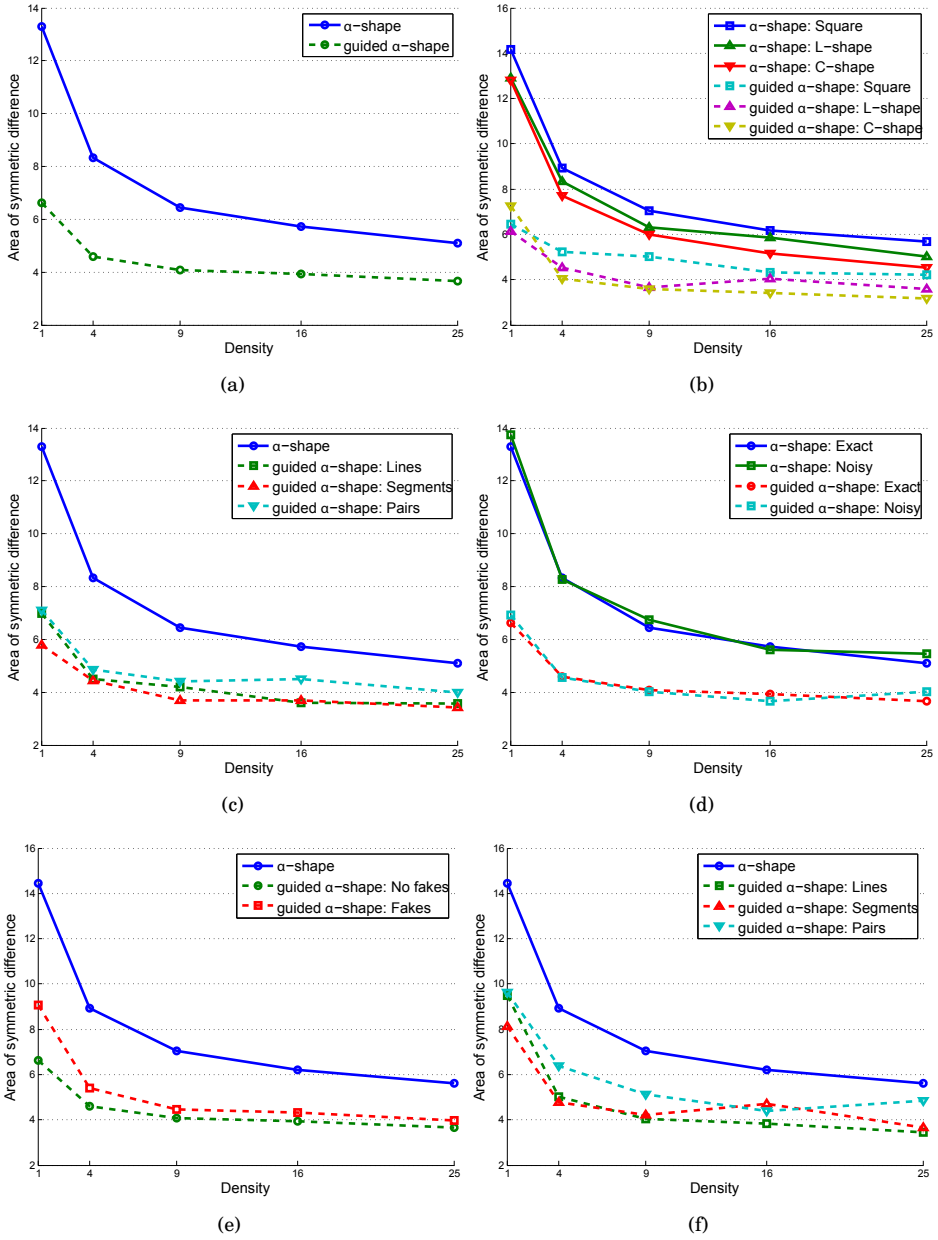
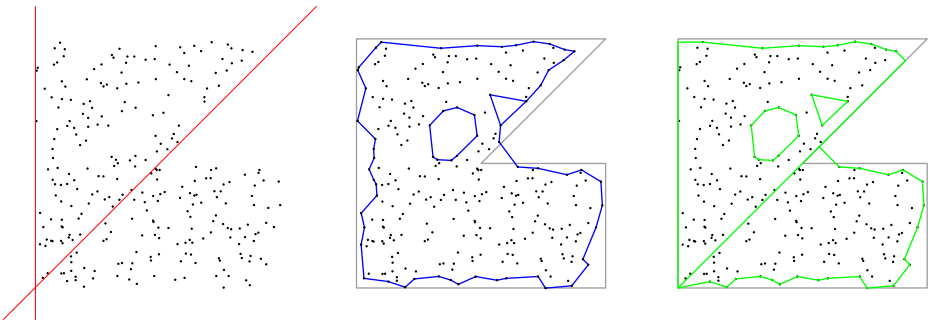


Figure 5.14: The area of symmetric difference between the original shape and the interior faces of either the α-shape or the guided α-shape.



(a) (b) (c)

Figure 5.15: A set of points and lines (a) and the resulting α-shape (b) and guided α-shape (c) together with the original shape in gray.

provide the input for the α-shape and guided α-shape. The guided α-shape also receives those intersection lines that have a point within 1 meter of both clusters. The α is the same for both methods and set to 0.5, based on the data distribution.

Note that the boundary of the surfaces cannot be directly inferred from the planes produced by RANSAC and their intersections. Even with this a priori knowledge, determining which parts of which intersections to use is not straightforward. Apart from identifying the parts of the intersections to use for the boundary of the surface, our algorithm also generates a fitting boundary in regions without appropriate intersection lines.

Constructing the shapes from a raw LiDAR data set of two million points took roughly 32 minutes on a consumer computer. This resulted in a model containing 101 planar shapes. Roughly half the points were classified as outliers, i.e. vegetation or surfaces too small to confidently reconstruct. As part of this process, computing the guided α-shapes of all surfaces in the data took seven minutes. Computing the α-shapes of the same surfaces requires two and a half minutes.

Figure 5.16 shows both the α-shape and guided α-shape of a number of surfaces. The shapes differ in some key aspects. The α-shape has jagged 'cracks' between neighboring surfaces (1). In the guided α-shape these neighboring surfaces share long boundary edges. Some cracks (2) may indicate a missing surface, but the lack of data may just as well be caused by invisibility of part of the surface. Because both surfaces are close enough to their intersection line, the guided α-shape uses this line, but only if no other surface is closer to the point set. In regions without intersection lines, the guided α-

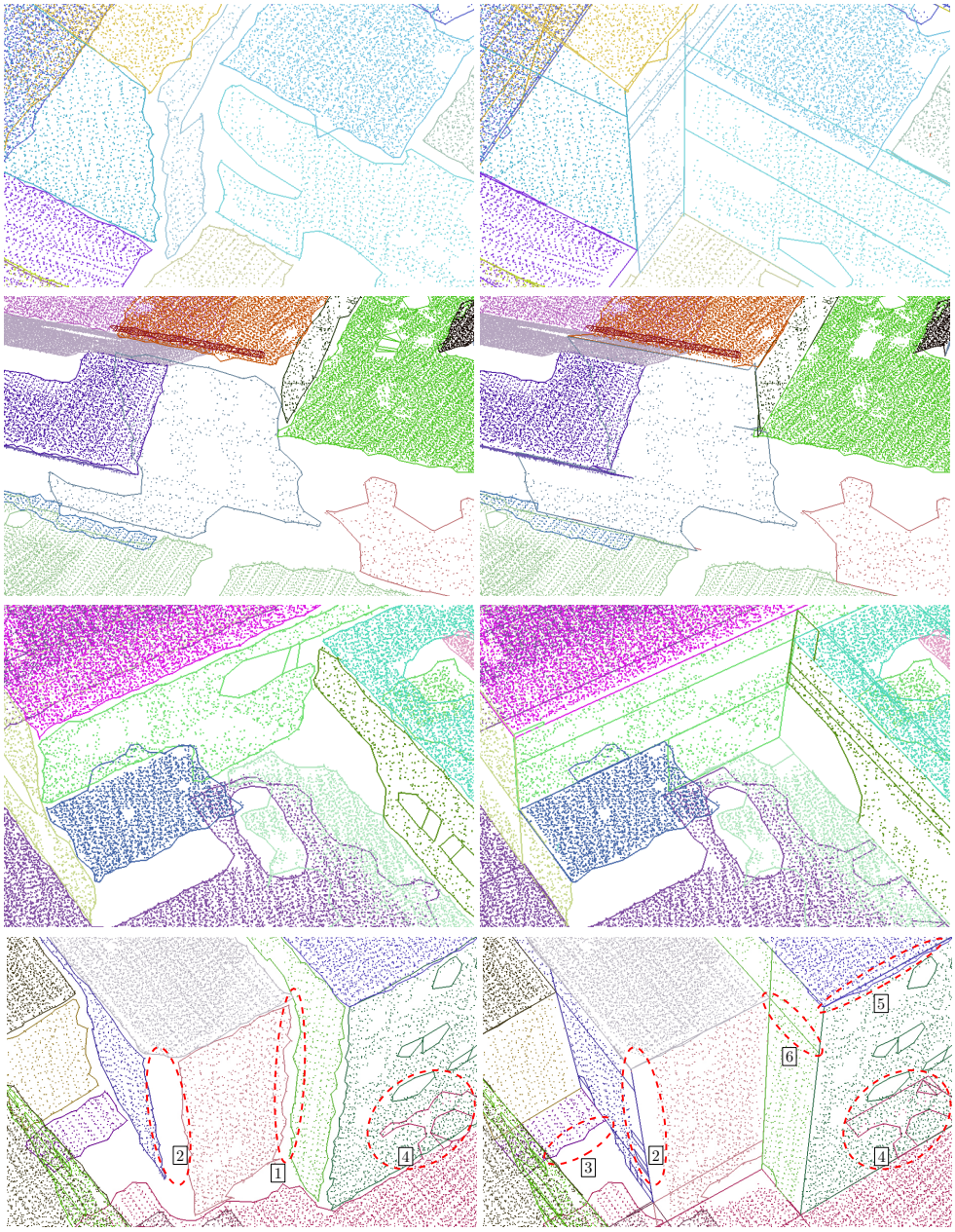


Figure 5.16: The α-shapes (left) and guided α-shapes (right) of part of various LiDAR data sets. Interesting regions of the last set are marked.

shape is the same as the α-shape (3).

Besides constructing boundaries that are more appropriate for urban reconstruction, the guided α-shape can also be used to identify artifacts in the data (4: red lines) and (5). The α-shape will not recognize these artifacts, making identifying them a difficult problem. The guided α-shape automatically separates these unwanted points from the main surface shape. This makes identifying and removing them straightforward: points measured through windows (4) usually give a sparser sample, and noise near a surface boundary (5) produces thin shapes near a guide.

A type of artifact specific to the guided α-shape is that the intersection lines may divide one surface into multiple shapes (6). However, after identifying unwanted shapes (eg. 4 and 5), it is easy to remove these edges (6) by joining the remaining shapes of a surface back together.

5.4 Conclusions

We have presented the generalized and the guided α-shapes, each a shape of a set of n points and m line segments, and algorithms to construct them in $O((n+m)\log(n+m))$ time. A comparison between the guided α-shape and the α-shape reveals that the guided α-shape is a better estimator for the interior of a point sampled polygon. This remains true for different shapes and when incorrect guides are provided. We have shown that it is not necessary to provide an accurate estimate of the appropriate boundary vertices; their supporting lines are sufficient.

A few interesting problems remain to be addressed. The presented shapes have one parameter. Like for the α-shape, the choice of this parameter determines the quality of the final shape. It remains an open problem how to optimally determine the value of α (Cazals et al. 2005; Mandal and Murthy 1997). This problem also affects the generalized and guided α-shape. However, we expect that as long as α is not prohibitively small, the generalized points and guides have a much larger influence than α.

We have evaluated the guided α-shape within the problem of urban reconstruction. For this application, it is safe to assume that surfaces are planar and guides are straight. However, the guided α-shape may be appropriate for computing the shape of a point set in other situations as well. These situations may use curved guides or curved surfaces. Extending the guided α-shape to encompass these shapes is an interesting problem. However, while extending the definitions of the shape is straightforward enough, achieving the same time bound may prove more challenging.

Similarly, we may wish to extend the generalized α-shape to curved generalized points. This will most likely depend on an extended generalized Voronoi diagram.

Other applications may pose different challenges. Indoor scenes and engineered objects are usually composed of simple surfaces. Reconstructing them as such may prove beneficial. These applications may include scenes that cannot be correctly reconstructed using only primitive shapes, like planes, cylinders, and spheres. In this case, the non-guide parts of the guided α-shape may give good edges to connect a mesh that reconstructs the irregular parts.

Finally, in our real world data sets, noise and outliers are a recurring problem. This may cause input points to lie outside the desired shape. An automatic way of detecting these parts of the shape should improve the reconstruction results and the guided α-shape may prove helpful for resolving this.

Chapter 6

Filling Gaps

The three previous chapters describe various methods for constructing a polygonal boundary that contains the points in a plane. These boundaries define the shape of the surfaces in the scene on which a sufficient number of points were measured. In most practical cases, there will be regions that do not contain sufficiently dense measurements to adequately reconstruct the surfaces. Because of this, some surfaces or parts of surfaces in the scene will not be reconstructed. In this chapter we present a method that identifies these missing surfaces and that constructs a collection of triangles that fill these gaps in the geometry.

An intuitive way of looking at this problem is to imagine the buildings in the scene as a volume of compressed water. We require closed surfaces in order to keep the water in its shape and any gaps in the geometry will allow the water to flow away. In essence, our goal when filling gaps is to create a *watertight* model. This model is a collection of surfaces that divides the space into two distinct volumes, inside and outside the model. Any place where these two volumes meet must be part of the surfaces. In practice, we ignore the surfaces needed to close the volume from below, because we assume the scene is firmly attached to the ground.

After presenting some related work in Section 6.1, we present our method for gap-filling in Section 6.2. We have evaluated our method on a number of urban LiDAR datasets and we present the setup and results of these experiments in Sections 6.3 and 6.4 respectively. Finally, Section 6.5 concludes this chapter with a discussion of the broader implications of our method.

The key contributions of this chapter are:

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

- A novel method for constructing the constrained Delaunay tetrahedralization that avoids costly line-sphere intersection computations.

- A comparison of our method to the related method by Labatut et al. (2009). Our method produces concise and visually pleasing results using a third fewer triangles.

6.1 Related Work

Reconstructing watertight geometry has received considerable attention. For the purpose of constructing watertight geometry, implicit surface methods, e.g. (Carr et al. 2001), may be the most straightforward, because the implicit function already represents some notion of an inner and outer volume. Hoppe et al. (1992) present an early example of explicitly constructing the geometry of an implicit manifold. They first estimate the local surface near each data point using principal component analysis (PCA) on its k nearest neighbors. After this, they assign a consistent orientation to these local surfaces by traversing the minimal spanning tree of the Riemannian graph: a graph on the data points containing all edges $\overline{xy}$ where point y is in the k-neighborhood of x. From these oriented surfaces, their implicit surface function maps any point in space to the signed orthogonal distance from the closest surface. Finally, they construct the explicit surface using a variation on the marching cubes algorithm (Allgower and Schmidt 1985).

Many other methods have used implicit surfaces in order to create watertight geometry. Mullen et al. (2010) describe a method that improves the correctness of the local surface orientation in order to fill gaps in regions of missing data. Schnabel et al. (2009) use implicit surfaces and combine this with graph-cuts to get a watertight model. They use an optimization scheme that iterates over multiple graph-cuts. Curless and Levoy (1996) introduce the idea of using lines-of-sight between sensor and data point to indicate exterior regions during the implicit surface method. Similarly, Shalom et al. (2010) use generalized cones with a data point at their apex to indicate exterior regions. Shen et al. (2004) present a method for constructing implicit surfaces from a polygon soup. Alliez et al. (2007) create an implicit surface using the anisotropy in Voronoi cells to estimate the normals of the surface.

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

There is also a wide range of methods that construct watertight geometry using the facets of the Delaunay tetrahedralization (DT). A survey of these Delaunay-based methods is given by Cazals and Giesen (2006). The basis for these methods is an observation by Boissonnat (1984), which showed that a shape can be captured in the DT of the sample points. Dey and Goswami (2003) provide a recent example by adjusting the Cocone method (Amenta et al. 2000) to produce watertight models. Many of these Delaunay-based methods assume the object is r-sampled, e.g. (Aichholzer et al. 2009), which requires the sampling to be sufficiently dense near important features. The assumption of r-sampling is too strong for urban reconstruction from LiDAR, because a data set may combine thin features and unpredictable sampling densities.

A recent approach by Labatut et al. (2009) combines the DT, the minimal-weight graph-cut, and lines-of-sight in order to determine the regions inside and outside the objects. While this method produces nice results, it does not exploit the planarity in the scene. Chauve et al. (2010) combine a partitioning of space into regions using planar primitives and a graph-cut to select interior and exterior regions. Although they achieve nice results, their method cannot reconstruct non-planar parts of a scene and lacks a theoretical basis.

Similar to (Chauve et al. 2010; Labatut et al. 2009), our method partitions space and constructs watertight geometry by applying a graph-cut to this partitioning. We partition 3-space using an extension of the DT that can handle polygons, called the *conforming constrained Delaunay tetrahedralization* (CCDT) (Shewchuk 1998, 2002; Si and Gärtner 2005). The CCDT is constructed from a collection of points and planar polygons: each point is contained as a vertex and each polygon is contained as a collection of triangular facets. This structure is described in Subsection 6.2.1.

6.2 Method

The goal of our method is to construct a watertight geometric model from a collection of planar polygons. This model should contain parts of these polygons where this is useful for separating the inner and outer volumes. In regions without polygons, an appropriate boundary is estimated from the point set. Our method achieves this in two steps, similar to Labatut et al. (2009). First we partition the complete space into many small regions and then we determine for each region whether it is inside or outside the model. Unlike (Labatut et al. 2009), both of these steps are guided by the polygons.

We use an extension of the DT, called the conforming constrained Delaunay tetrahedralization (CCDT) to partition 3-dimensional space into tetrahedral cells. We describe the CCDT and how we construct it in Subsection 6.2.1. If all the cells are assigned to

either the inner or outer volume, the facets separating inner and outer cells comprise a watertight model. The CCDT embeds input polygons in its facets, enabling partitioning along these surfaces.

We classify the cells of the CCDT into inner and outer cells using a minimum-weight graph-cut. This method takes a weighted graph that contains a source and a sink node and collects a number of edges to remove, the *cut*, such that the source and sink are no longer connected by a path on edges of the graph. The cut is chosen such that the sum of the weights of its edges is minimized. Subsection 6.2.2 describes the graph-cut method in more detail.

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

However, there are two other considerations important to the cut. Firstly, a graph-cut only makes sense if there are a source and sink node. The connectivity between source and sink node is very important, because only edges on a path between source and sink are cut. We add two additional, abstract, nodes to the dual of the CCDT to act as source and sink. These two nodes are connected automatically to the other nodes in a careful fashion as described in Subsection 6.2.3.

Secondly, the weights of the edges in the graph determine the optimal positions to cut. We compute the weight of each edge based on four properties of its corresponding facet in the CCDT. The first three properties are also used by Labatut et al. (2009), the last is novel to our method. Firstly, the laser does not penetrate solid surfaces, so facets intersected by the line from sensor to measurement point produce a higher weight. Secondly, the measurement locations indicate a reflection point on a surface, so the facets near points produce a lower weight. Thirdly, the regions separated by the surface should be largely empty of measurement points, so facets separating two cells with large circumspheres produce a lower weight. Finally, the reconstructed polygons indicate simpler representations of the surfaces, so the facets in a polygon produce a lower weight. The first two factors are described in more detail in Subsection 6.2.3 and the last two in Subsection 6.2.4.

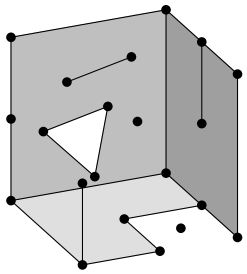


Figure 6.1: A piecewise linear complex.

6.2.1 Conforming Constrained Delaunay Tetrahedralization

In order to subdivide the 3-space into regions that can be interior or exterior, we use the conforming constrained Delaunay tetrahedralization (CCDT). This structure is closely related to the well-known 3-dimensional Delaunay tetrahedralization (DT). The DT is described in Section 2.4. To reiterate, the DT is a graph on the point set that connects each pair of points that share the boundary of a ball empty of points. This graph is called a tetrahedralization, because it partitions the convex hull of the points into tetrahedral cells. For each tetrahedron in the DT, its circumscribing ball has no points in its interior.

In order to properly describe the CCDT, we must first describe the piecewise linear complex (PLC) (Miller et al. 1996). A PLC is a collection of points, straight line-segments, and planar straight-line polygons in 3-space, as shown in Figure 6.1. The polygons can be non-convex with any number of boundary segments and they may contain holes, interior segments, and isolated points. However, their boundary cannot be self-intersecting and all their vertices must be coplanar.

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

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

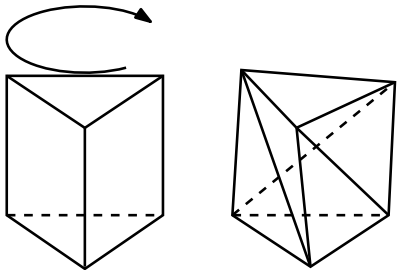


Figure 6.2: A prism and Schönhardt's polyhedron. The creases created by twisting the upper triangle are concave.

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

Shewchuk (1998) showed that there are PLCs that do not admit an embedding CDT. Schönhardt's polyhedron (Schönhardt 1928) is a simple example, as shown in Figure 6.2. No tetrahedralization on the vertices of this polyhedron exists for which all the facets of the polyhedron are embedded.

Shewchuk (1998) goes on to prove that for any PLC X we can construct another PLC $\bar{X}$ that can be embedded in a CDT. This PLC $\bar{X}$ exactly covers X, but it contains additional points, called *Steiner points*, on the segments of X. These points are placed such that the DT of $\bar{X}$ contains edges that cover the segments of X. We call these edges *conforming* to the segments of X. Unlike Shewchuk, we prefer to call the CDT of $\bar{X}$ the conforming constrained Delaunay tetrahedralization of X, to indicate the similarity to the 2-dimensional conforming Delaunay triangulation.

6.2.1.1 Embedding segments

There are many different methods for determining the positions of the Steiner points that ensure that the segments of $\bar{X}$ are present in the DT. Figure 6.3 gives an example PLC and two different configurations of Steiner points that ensure the embedding of the segment $\overline{ab}$.

There are also various ways of determining which configuration is better, for example by counting the number of Steiner points (Si and Gärtner 2005) or by bounding the minimal edge length within the CCDT (Shewchuk 2002). Related work generally constructs Steiner points at the intersections of a sphere and a line, which is in practice

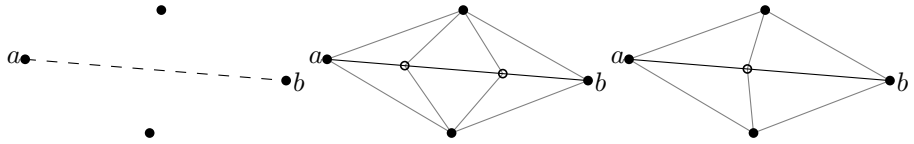


Figure 6.3: A configuration of points and two different configurations of Steiner points that force $\overline{ab}$ to appear in the Delaunay triangulation.

either dangerous because of round-off errors or expensive to compute exactly. For this reason, we present a novel Steiner point insertion method, similar to (Si and Gärtner 2005), that does not require computing sphere-line intersections.

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

These protecting balls are specifically important for *acute* points: points where at least two conforming edges meet at an acute angle. The insertion of a Steiner point on one edge may cause another conforming edge e to disappear from the DT, forcing the insertion of another Steiner point to restore e. When Steiner points can be inserted arbitrarily close to an acute point, this process may cascade into an infinite loop.

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

Instead of computing the intersection of the boundary of $\mathcal{P}_v$ and $\overline{vw}$, we will work directly with the empty open balls incident to v that have their diameter on $\overline{vw}$, as shown in Figure 6.4. Specifically, we choose the largest empty open ball $\mathcal{D}_{\overline{vd}}$. This ball must have a point r on its boundary and it is easy to see that $\|vr\| < \|vd\|$. Note that r may be a point of X or a Steiner point inserted earlier.

Because d is not inside $\mathcal{B}_v^r$ and because all Steiner points are constructed on the seg-

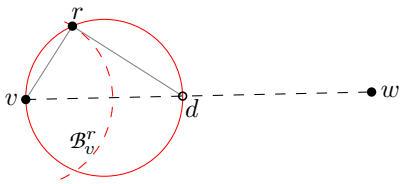


Figure 6.4: A protecting ball. The black disks are points of the PLC, the dashed black line is a segment of the PLC. The dashed red arc shows $\mathcal{B}_v^r$, the solid red circle is the largest empty ball with its diameter $\overline{vd}$ on $\overline{vw}$.

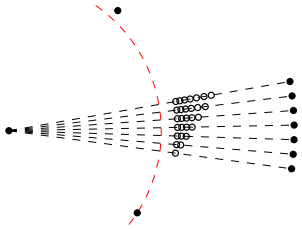


Figure 6.5: A pathological case when conforming segments top to bottom between points (black disks). Steiner points (circles) are inserted at the intersection with the largest empty $\mathcal{D}_{\overline{vd}}$. Each Steiner point insertion invalidates all earlier conforming edges, leading to repeated Steiner point insertions.

ments of X, d must be outside $\mathcal{P}_v$. Because $\mathcal{D}_{\overline{vd}}$ is empty, inserting d as Steiner point would result in the $\overline{vd}$ appearing in the DT. Additionally, v and r must be adjacent in the DT as witnessed by $\mathcal{D}_{\overline{vd}}$. Finally, according to Thales' Theorem we can compute d as the intersection of $\overline{vw}$ and the plane through r orthogonal to $\overline{vr}$, removing the need for a line-sphere intersection.

Unfortunately, inserting a Steiner point at d has two disadvantages. Firstly, if we want to be able to insert conforming edges incrementally, this method may result in an extreme number of Steiner points, as shown in Figure 6.5. Secondly, if there are two other points x, y cospherical with v, r, d, $\overline{vd}$ may not appear in the DT because the tetrahedra on v, r, x, y and d, r, x, y have an empty circumsphere. The free choice of which tetrahedra to use may not result in $\overline{vd}$ appearing.

In order to overcome these disadvantages, we want to insert the Steiner point at another location s on $\overline{vw}$ such that $\|vr\| \leq \|vs\| < \|vd\|$. We choose as s the midpoint m of d and the projection p of r onto $\overline{vw}$, as shown in Figure 6.6 and Algorithm 6.1. We refer to this procedure as the *protection method*.

It is straightforward to see that $\|vp\| < \|vm\| < \|vd\|$ and $\|vp\| < \|vr\|$. However, the

Algorithm 6.1 Compute the position of a new Steiner point.

Input: two vertices v, w.
Output: the Steiner point inserted on $\overline{vw}$ to protect v.

```
1:  procedure PROTECT(v, w)
2:      x ← w
3:      y ← w
4:      E ← edges incident to v
5:      for all v̄v_i ∈ E do
6:          if ∠wvv_i < π/2 then
7:              P ← plane through v_i orthogonal to v̄v_i
8:              d ← v̄w̄ ∩ P
9:              if ‖vd‖ < ‖vy‖ then
10:                 x ← projection of v_i on v̄w̄
11:                 y ← d
12:             end if
13:         end if
14:     end for
15:     return MIDPOINT(x, y)
16: end procedure
```

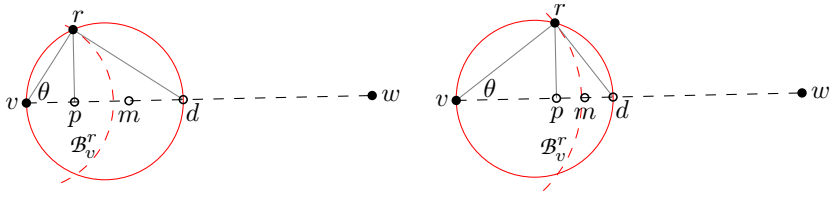


Figure 6.6: Two protecting balls. The black disks are points of the PLC, the dashed black line is a segment of the PLC, and the black circles indicate other significant locations. The dashed red arc shows $\mathcal{B}_v^r$, the solid red circle is the largest empty ball with its diameter $\overline{vd}$ on $\overline{vw}$, p is the projection of r onto $\overline{vw}$, and m is the mid-point of p and d.

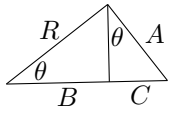


Figure 6.7: The setup for the proof of Lemma 6.2.1.

difference between $\|rv\|$ and $\|mv\|$ depends on the angle $\theta = \angle rvw$, as shown in Figure 6.6. The following lemma shows that for any θ, $\|vr\| < \|vm\|$ and therefore m must lie strictly outside the protecting ball and strictly inside $\mathcal{D}_{\overline{vd}}$. This means that $\overline{vm}$ must appear in the DT.

Lemma 6.2.1. *Given the triangle vrd where r lies on a corner with right angle, let p be the orthogonal projection of r onto $\overline{vd}$, let m be the midpoint between p and d, and let $\theta = \angle rvd$, as shown in Figure 6.6. If $\theta \neq 0$ then $\|vr\| < \|vm\|$.*

Proof. For readability's sake, let us rename the distance involved as follows: $R = \|vr\|$, $A = \|rd\|$, $B = \|vp\|$, $C = \|pd\|$, $D = \|rp\|$, as shown in Figure 6.7. Observe that $\angle prd = \theta$, because of triangle similarity. This proof builds on the following trigonometric functions:

$$B = R\cos\theta \tag{6.1}$$

$$D = R\sin\theta \tag{6.2}$$

$$D = A\cos\theta \tag{6.3}$$

$$C = A\sin\theta \tag{6.4}$$

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

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

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

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

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

When two points v, w lie sufficiently close together and $\overline{vw}$ is not present in the DT, it may be necessary to insert a Steiner point inside $\mathcal{B}_v$ or $\mathcal{B}_w$. Related methods achieve this by choosing the protecting balls $\mathcal{P}_v$ and $\mathcal{P}_w$ smaller than $\mathcal{B}_v$ and $\mathcal{B}_w$ (Shewchuk 2002; Si and Gärtner 2005). Similarly, we will construct a Steiner point s such that $\overline{vs}$ and $\overline{sw}$ appear in the DT, while at the same time forcing $\mathcal{B}_v$ or $\mathcal{B}_w$ to shrink. We must choose s with caution in order to make sure no $\mathcal{B}_v$ can shrink too much.

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

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

Case iii) covers all remaining configurations of v and w where $\overline{vs_v}$ and $\overline{ws_w}$ overlap. Again, we will insert one Steiner point s inside $\mathcal{B}_v$ or $\mathcal{B}_w$. We base s on s_v, s_w, and the midpoint m of v, w. Let us assume without loss of generality that $\|vs_v\| \leq \|ws_w\|$. If $\|vs_v\| < \|vm\|$ then $s = s_v$; otherwise $s = m$. After the insertion of s, the radius of both $\mathcal{B}_v$ and $\mathcal{B}_w$ either remains the same or is at least $\frac{\|vw\|}{2}$. The pseudo-code of the complete method is shown in Algorithm 6.2.

Algorithm 6.2 Embed a conforming edge.

Input: two vertices v, w.

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

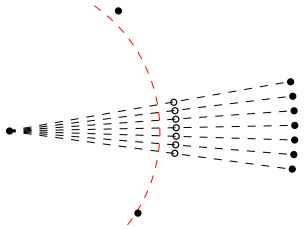


Figure 6.8: The pathological case shown in Figure 6.5 when inserting Steiner points using our protection method.

So far we have mainly considered inserting a single conforming segment. However, as Figure 6.5 showed, complications may occur when inserting multiple conforming edges. Figure 6.8 shows the Steiner points constructed by our method. There are still point sets for which our protection method places a Steiner point such that an earlier conforming edge incident to an acute vertex is removed from the DT. Figure 6.9 shows the results of our embedding method on a difficult PLC.

It remains to show that our method always terminates. Related methods generally prove this using the *local feature size* (lfs) (Ruppert 1995). Given a PLC X and a point p in 3-space, $lfs(p)$ is the radius of the smallest ball $\mathcal{B}$ centered on p such that $\mathcal{B}$ intersects two vertices and/or segments of X that do not intersect each other. Note that this measure is independent of Steiner points. Termination can then be proven by showing that there is some constant c such that after any Steiner point insertion $c\|e\| \geq \text{lfs}(p)$ holds for any edge e and any point p on e.

However, many of these proofs incorrectly discount the influence of Steiner points. Consider the example of applying Shewchuk (2002) to a problematic case shown in Figure 6.10. The gray disks show the shortest edge lengths and four times that distance. Recall that for $\text{lfs}(p)$ both Steiner points and intersecting segments of the PLC are ignored. This means that for most points x inside the wedge of conforming segments, $\text{lfs}(x)$ is the distance to p. For this reason, we define the *star local feature size* ($\text{lfs}^\star$) for subspaces.

Definition 6.2.2. *Given a fixed PLC X and a subspace S, the star local feature size $lfs^\star_X(S)$, or simply $lfs^\star(S)$, of S is the radius of the smallest ball centered within S that contains two points or segments of X that do not intersect in S.*

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

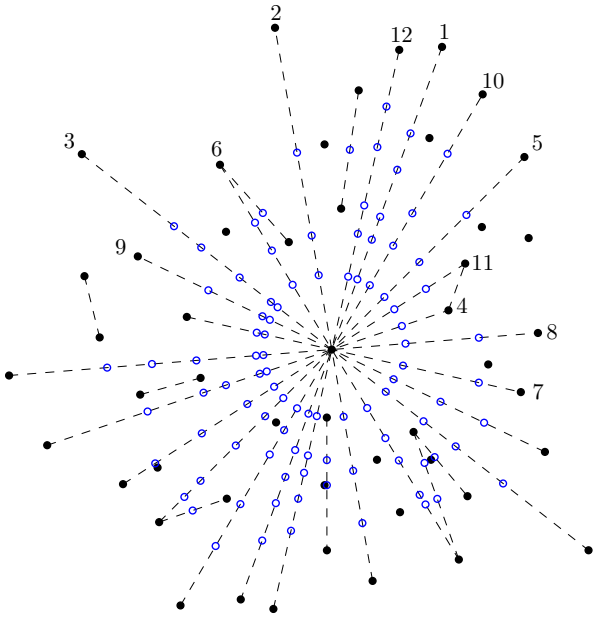


Figure 6.9: A pathological case. The black disks are points of the PLC, the dashed black lines are the segments of the PLC, and the blue circles make a collection of Steiner points that embed the segments in the DT. The numbers indicate the order in which the segments are embedded; inserting the segments in a different order may result in a different set of conforming edges.

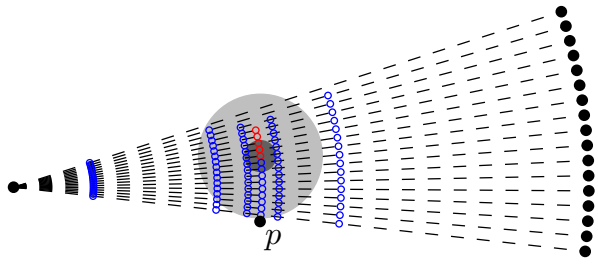


Figure 6.10: Shewchuk's edge protection applied to a problem case. The PLC contains the black disks and dashed segments, the small circles show the Steiner points constructed to embed the segments. The red Steiner points are incident to two edges that are too short, the gray disks show the edge length and four times this distance.

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

Lemma 6.2.3. *Given a PLC X and a segment $\overline{vw}$ of X not embedded in the DT, let us apply the protection method to conform $\overline{vw}$ without re-embedding edges on other segments that were destroyed. This operation can add only a number of Steiner points linear in the number of vertices of the DT.*

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

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

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

Lemma 6.2.4. *Given a PLC X and a collection of Steiner points S constructed using the protection method, there cannot be a point $s \in S$ and an acute vertex v such that $\|sv\| < \frac{1}{2} lfs(v)$.*

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

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

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

Recall that according to Lemma 6.2.1, from any Steiner point s constructed by our protection method on any segment $\overline{vw}$ from point w and r, $\|vs\| > \|vr\|$. This means that s must lie outside $\mathcal{B}_v^r$. Either r is a Steiner point, or r is a point of X and it is straightforward that $\|vs\| > \|vr\| \geq \text{lfs}(v)$. If r is a Steiner point, the same reasoning can be applied to find the point r' used to construct r. Because each next point must be closer to v, at some point we will find a non-Steiner point r'_i such that $\|vs\| > \|vr'_i\| \geq \text{lfs}(v)$.

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

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

Lemma 6.2.5. *Given a PLC X and a collection of Steiner points S constructed using the protection method, any edge e embedding a segment of X and not incident to an acute point can only be removed from the DT if $lfs^\star(e) < \|e\|$.*

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

Theorem 6.2.6. *Out protection method always terminates.*

Proof. By Lemma 6.2.4, edges incident to an acute point have a minimum edge length. By Lemmas 6.2.3 and 6.2.5 each time an edge is embedded, this adds a finite number of edges and only destroys conforming edges longer than their $\text{lfs}^\star$. □

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

6.2.1.2 Embedding polygons

After conforming the DT to the segments of the PLC, the interiors of the polygons can be inserted incrementally. Shewchuk (2003) describes a method for recovering the polygons using an ordered sequence of bistellar flips. Unfortunately, this method seems numerically unstable. We incrementally embed the interiors of the polygons using a method that changes larger collections of cells simultaneously very similar to (Si and Gärtner 2005).

Like Si and Gärtner, we insert each polygon $\mathcal{P}$ per *cavity*. A cavity is a maximal facet-connected collection of cells intersected by $\mathcal{P}$. Shewchuk (2003) showed that these cavities contain exactly the cells that need to change to embed the polygon. For each cavity $\mathcal{C}$, we construct two new tetrahedralizations T_a, T_b: one using the vertices of $\mathcal{C}$ on or above $\mathcal{P}$ and the other using the vertices on or below $\mathcal{P}$. When we replace the cells of $\mathcal{C}$ by the overlapping cells of T_a, T_b, the part of $\mathcal{P}$ inside $\mathcal{C}$ appears, because this part can be built from facets on the convex hull of both T_a and T_b.

We must take special caution when embedding polygons near facets constrained earlier. There are simple configurations of points and constraints such that some facets on the boundary of $\mathcal{C}$ are not contained in $T = T_a \cup T_b$, as shown in Figures 6.11 and 6.12. When boundary facets are missing from T, the cavity cannot be neatly filled with cells of T.

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

We overcome this problem similar to Si and Gärtner (2005). Before computing T_a, T_b, we grow the cavity until all its boundary facets must be contained in T. We do this by checking for each facet on the boundary of $\mathcal{C}$ whether it has a circumscribing ball that does not contain any point of $\mathcal{C}$. Note that this is a weak Delaunay criterion and unlike (Si and Gärtner 2005) it is restricted to $\mathcal{C}$. For each facet f not meeting this criterion, we grow $\mathcal{C}$ by adding the cell on the other side of f. We repeat this procedure until all boundary facets meet the criterion. In extreme cases this may force the cavity to grow to the complete CCDT, but in practice most cavities will not grow significantly.

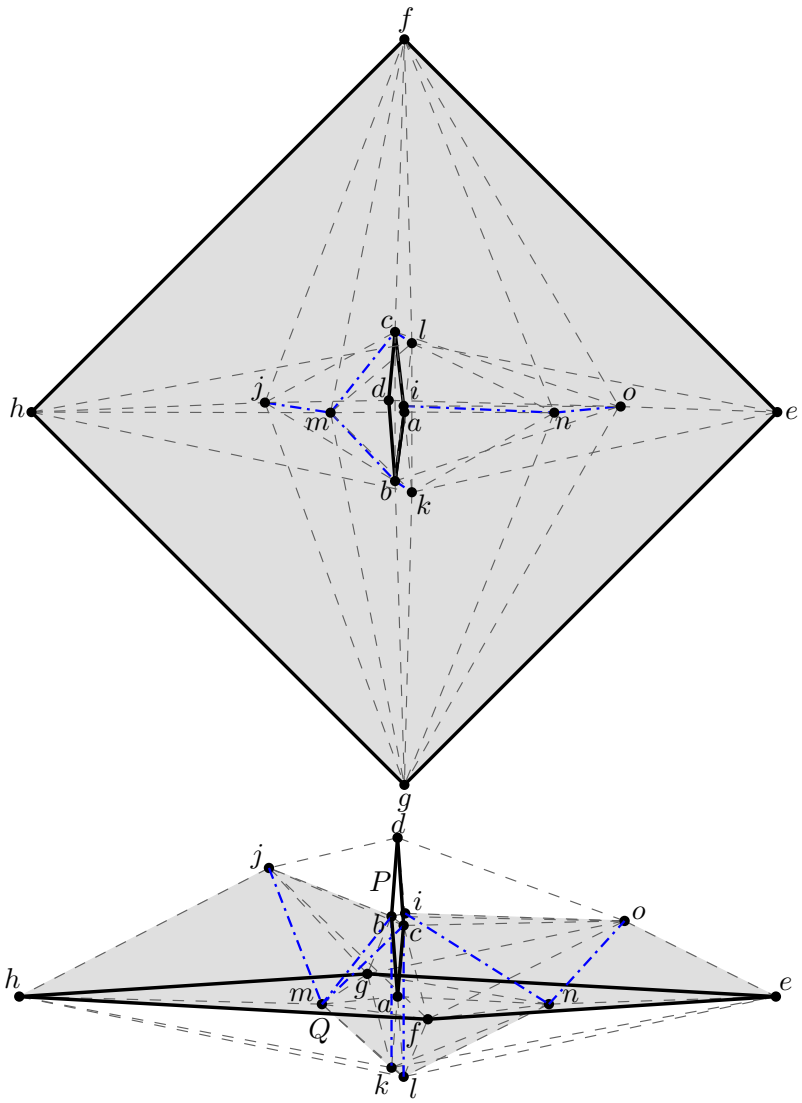


Figure 6.11: A small example where a partial tetrahedralization of a cavity cannot be glued along the boundary. This figure shows the CCDT of a point set in which polygon $P = abdc$ is embedded and polygon $Q = efhg$ should be embedded. Note that n, m lie just below Q and the blue dash-dotted edges intersect Q. Because the cavity C (gray region) contains exactly the cells incident to these edges, it does not contain d or the cell $abci$. Figure 6.12 shows the CCDT of the upper half of the cavity.

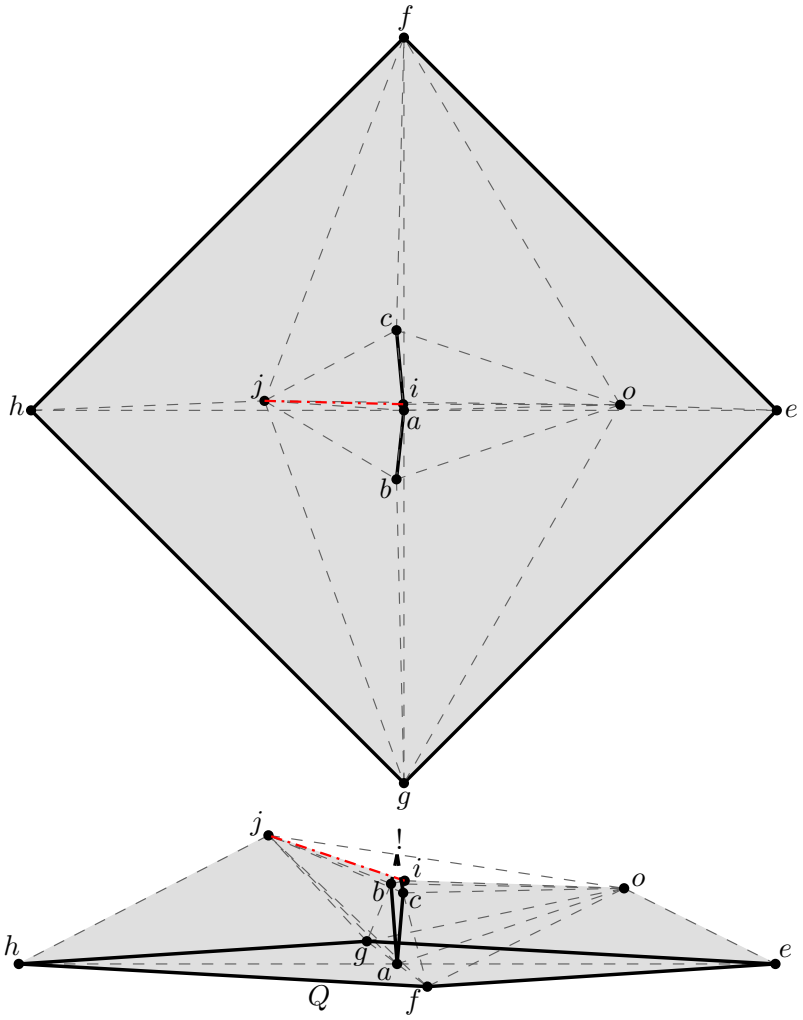


Figure 6.12: A small example where a partial tetrahedralization of a cavity cannot be glued along the boundary. This figure shows the CCDT T_a of the points of the cavity on or above Q. Note that because T_a does not contain d, three problems occur: a) the facets abc and bcj on the boundary of C are not in T_a, b) the red edge $\overline{ij}$ intersects P, and c) P cannot be embedded in T_a, because there is no loop of coplanar edges.

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

Lemma 6.2.7. *Given a cavity $\mathcal{C}$ in CCDT T formed while embedding polygon $\mathcal{P}$ and a polygon $\mathcal{Q}$ intersecting $\mathcal{C}$ in $\mathcal{Q}_c$, let T_p be the new CCDT formed by embedding $\mathcal{P}$ without enforcing $\mathcal{Q}_c$ to be embedded. The cavity of $\mathcal{Q}_c$ in T_c is a strict subspace of $\mathcal{C}$.*

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

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

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

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

Finally, we will show that the cavity of $\mathcal{Q}_c$ can neither grow outside $\mathcal{C}$ nor to the other side of $\mathcal{P}$. Recall that we grow the cavity within T_c one cell at a time at facets that are not weakly Delaunay. In order for the cavity to grow outside $\mathcal{C}$, it must at some point

have grown through a boundary facet of $\mathcal{C}$, since T_c contains all these boundary facets. However, since all these facets are weakly Delaunay, the cavity cannot grow outside of $\mathcal{C}$. Since we grow the cavity in the cells of T_c and T_c does not contain a point on the other side of $\mathcal{P}$, the cavity cannot grow to the other size of $\mathcal{P}$. □

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

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

6.2.2 Minimum-Weight Graph-Cut

We use a minimum-weight graph-cut algorithm to determine which regions are inside or outside of the objects. A graph-cut is an operation on a weighted, directed graph containing two special nodes: a source and a sink. The graph-cut, also called a cut, is a subset of the graph whose removal disconnects the source and sink. In other words, when all the edges of the cut are removed from the graph, there is no path from source to sink. A minimum-weight graph-cut is an optimal cut in the sense that the summed weight of the edges in the cut is minimal over all possible cuts.

Ford and Fulkerson (2010) show that determining this minimum-weight graph-cut is equivalent to determining the maximum-flow. To visualize the maximum-flow, imagine that the graph represents a network of pipes where the weight of an edge represents its capacity. The maximum-flow is the maximal amount of water that can be pushed through this network from source to sink. One can imagine that this maximal amount depends on the bottlenecks in the network, the pipes that are maximally utilized. Ford and Fulkerson (2010) show that these pipes are exactly the minimum-weight cut.

We use the method developed by Boykov and Kolmogorov (2004) to compute the minimum-weight graph-cut. This method consistently outperforms the other state-of-the-art methods (Dinic 1970; Goldberg and Tarjan 1988) in practice, even though its asymptotic running time is worse. The method is an *augmenting path* based method (Dinic 1970), a group of methods that iteratively identify augmenting paths

from source to sink and mark them in a residual graph. A path is augmenting if all its edges in the residual graph have non-zero weights. This residual graph starts identical to the original graph. Each time a new augmenting path from source to sink is found, the weights of all edges along this path in the residual graph are reduced by the same amount such that at least one of the edges gets weight 0. This can be imagined as sending a certain amount of flow along the path.

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

Boykov and Kolmogorov (2004) have optimized their method mainly by minimizing repetition. They search for new augmenting paths using two breadth-first search trees, rooted respectively in the source and sink. Unlike earlier methods, these search trees are persistent instead of being constructed multiple times. While searching for a new path from source to sink, both search trees are grown simultaneously on the non-zero edges of the residual graph until they touch each other. This indicates a new augmenting path and the weights along this path are decreased, reducing the weight of at least one edge to 0. This may turn the search trees into forests and where possible, the parts no longer connected to source or sink are reconnected along another edge. Once the trees can no longer be grown, there are no more paths possible from source to sink and the trees are separated by the minimum-weight graph-cut. Additionally, the nodes of the graph can be labeled according to the tree they are in, with a third label denoting 'free' nodes which are in neither tree.

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

6.2.3 Visibility

It is sometimes considered a disadvantage of laser light that it cannot pass through most solid surfaces, particularly opaque surfaces. These surfaces reflect the laser light, which is the basis for most laser scanners: they capture the reflected light in order to

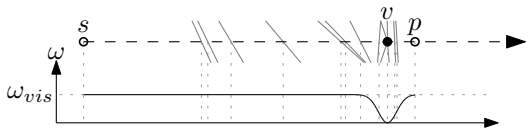


Figure 6.13: The influence of visibility on the weights. From the ray from sensor s through point v, we construct point p inside the object. The weight ω of a facet depends on the distance to v.

determine the distance to the surface. Another important feature is that laser light travels in a straight line, which is used to determine the position of the point of reflection, the LiDAR data point. From these features, we can infer that the line segment connecting data point and sensor can only pass through (semi-)transparent space. We use this information to adjust the weights of the graph the same as Labatut et al. (2009).

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

We know from the scanning procedure that a laser traveling along $\overrightarrow{sv}$ hit a solid surface near v. We express the exterior space traversed in the weights of the graph. Each edge corresponding to a facet f of the CCDT intersected by $\overline{ps}$ gains weight $\omega_{vis}(1-e^{-d^2/2\sigma^2})$, where $d = \|vq\|$ and q is the intersection of f and $\overline{ps}$, as shown in Figure 6.13. Note this weight approaches 0 near v, and ω_{vis} at $d > 3\sigma$.

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

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

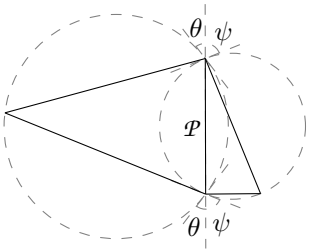


Figure 6.14: The influence of facet quality on the weights. The circumscribing balls intersect the supporting plane $\mathcal{P}$ at angles θ and ψ.

6.2.4 Facet Quality

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

We use an inherent quality of each CCDT facet f and its local geometry to adjust its weight. Like Labatut et al. (2009), we add $\omega_{qual} = 1 - min(\cos(\theta), \cos(\psi))$ to the weight of each facet, where θ and ψ are the angles between the plane $\mathcal{P}$ supporting f and the circumspheres of the two incident cells at the circle where they intersect $\mathcal{P}$, as shown in Figure 6.14.

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

6.2.5 Implementation

We have implemented our method in C++ using CGAL 4.0.2 (Computational Geometry Algorithms Library 2013) and an implementation of Boykov and Kolmogorov's graph-cut method (Graph Cut optimization library 2012). In order to construct a CCDT of a collection of points and polygons, we have built two classes on top of CGAL's Delaunay tetrahedralization. Firstly, we have implemented the conforming Delaunay tetrahedralization that extends the DT to handle conforming segments. Secondly, we have implemented the constrained Delaunay tetrahedralization to handle constrained polygons. Both of these classes are straightforward to program based on Section 6.2.1, although like the DT, careful programming is required to correctly handle degeneracies.

The conforming Delaunay tetrahedralization inherits most of its functionality from the basic Delaunay tetrahedralization. Additionally, conforming line segments can be inserted, as detailed in Subsection 6.2.1.1. The insertion of a Steiner point may force another conforming segment to be removed from the DT, in which case it is automatically reinserted after handling the current segment.

The constrained Delaunay tetrahedralization inherits most of its functionality from the conforming Delaunay tetrahedralization. Additionally, polygon constraints can be inserted once all its edges are conforming, as detailed in Subsection 6.2.1.2.

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

Recall that polygons of the PLC may contain holes. However, the segments bounding a hole were not marked when embedding the polygon. After embedding the interior of a polygon, its holes are marked and deconstrained as detailed at the end of Subsection 6.2.1.2.

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

6.3 Experimental Setup

Unfortunately, there are no benchmark data sets available of urban point data, let alone benchmark data with known lines of sight. We use a data set provided by Fugro EarthData, Inc. (2012). This data set contains aerial LiDAR data points that were measured during several flights. Each data point has a registered 3D coordinate and a time stamp that can be matched with the flight path to estimate sensor positions. The massive data sets are separated into smaller chunks based on a regular horizontal grid, with all points collected in one region combined into one chunk. We perform our experiments on seven different chunks, shown in Figure 6.15.

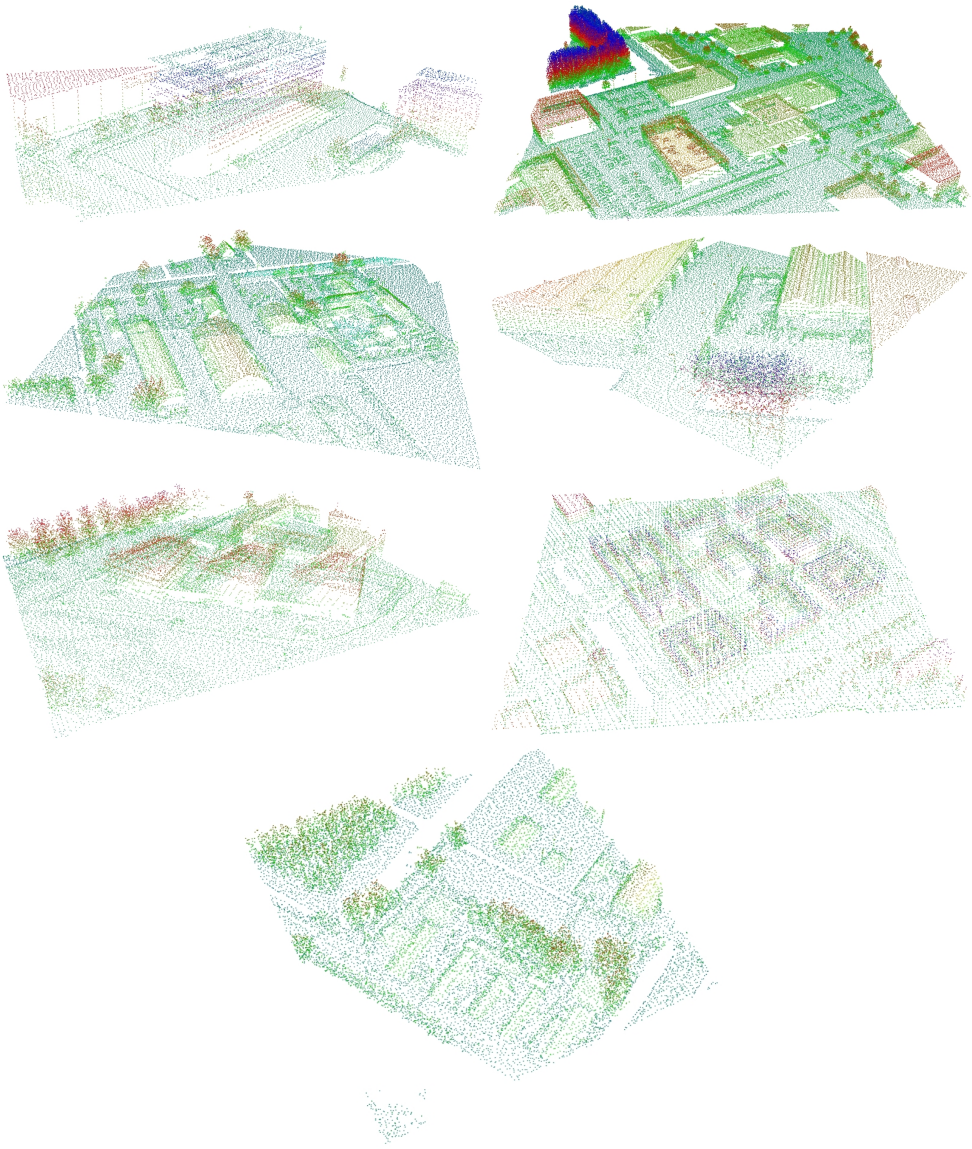


Figure 6.15: The seven different chunks used in our experiments.

	$s(m)$	NN	$\Delta d(cm)$	Δn	$Cb(m)$	$\|S\|$	P	$\alpha(m)$
1	1.0	12	6.5	20°	**2.0**	25	0.0001	**2.0**
2	1.0	12	6.5	20°	1.5	25	0.0001	1.5
3	1.0	12	6.5	20°	1.5	25	0.0001	1.5
4	1.0	12	6.5	20°	1.5	25	0.0001	1.5
5	1.0	12	6.5	20°	1.5	25	0.0001	1.5
6	**3.0**	12	6.5	20°	**4.0**	**10**	0.0001	**4.0**
7	1.0	12	6.5	20°	1.5	25	0.0001	1.5

Table 6.1: The settings used to preprocess the data for our method: the cell edge length for subsampling (s), 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). The exceptional values are shown bold face.

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

Table 6.2: The sizes of the data sets: the number of points in the raw dataset ($|P_d|$), the number of points after subsampling ($|P_s|$), and the number of shapes ($|S|$) identified by Efficient RANSAC.

In order to reduce the number of points, we subsample the data by superimposing a 3D grid with fixed cell dimensions (Alliez et al. 2013). The cell edge length used within each chunk is shown in Table 6.1. Inside each cell, one random data point is selected to construct the subsample. From this subsample, we estimate the local surface normal at each point using principal component analysis on their 12 nearest neighbors. Subsequently, Efficient RANSAC is performed on the point set to identify planar clusters and the remaining points. Simultaneously, the parameters of the least-squares optimal interpolating plane are computed. The parameters of Efficient RANSAC, shown in Table 6.1, were determined empirically based on clustering performance. Table 6.2 shows the number of points and shapes per chunk.

For each pair of planes with data points within 1 meter of each other, the intersection line is computed. We compute the guided α-shape of each cluster, using the α-value given in Table 6.1, using the intersection lines of that cluster as guides. The resulting

polygons and the remaining unclustered points are shown in Figure 6.16.

To compare our method to Labatut et al. (2009), we construct both the DT and CCDT of the chunk. We insert all the data points into the DT. We insert the collection of guided α-shapes and the unclustered points into the CCDT. Note that for the planar clusters, the CCDT will only contain the points on the boundary. The CCDT also contains a number of Steiner points.

We construct a graph of these arrangements and we determine a graph-cut as explained in Section 6.2.2. This cut determines the facets that comprise the surfaces of the scene.

6.4 Results

Figure 6.15 shows the different sets of urban LiDAR we have performed our experiments on. Figures 6.17 and 6.18 show the watertight geometry constructed using (Labatut et al. 2009) and our method. It is clear that our method constructs geometry that is simpler and that conforms more to our idea what an urban scene should look like. The small details are represented by free-form meshes, while the large roofs and facades are represented by flat surfaces.

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

Apart from producing a geometric model that better captures the planar nature of an urban scene, this geometry is also generally more concise, as Table 6.3 shows. In most cases, the CCDT has roughly a sixth fewer elements and the watertight surface generated from it uses roughly a third fewer triangles than when using the DT.

Chunk 6 is an exception in this regard, because in this case the CCDT-based surface uses more triangles. The most probable reason for this is the much higher than average shape to point ratio, as shown in Table 6.2. As this chunk has roughly three times as many shapes as chunks of similar size, it is not surprising that the reconstruction would require more triangles to cover these shapes. This conjecture is supported by the higher than average amount of Steiner points in the CCDT of chunk 6.

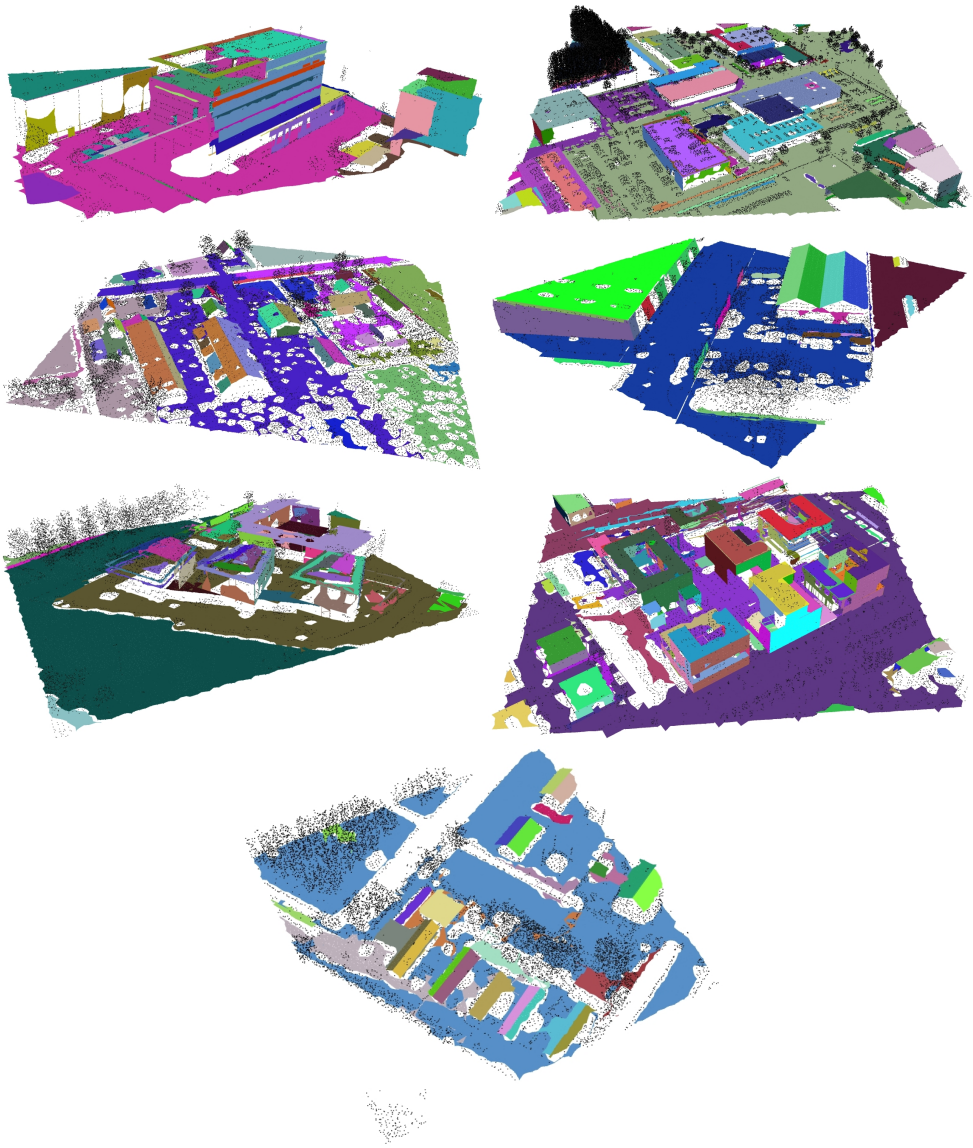


Figure 6.16: The guided α-shapes in the different chunks.

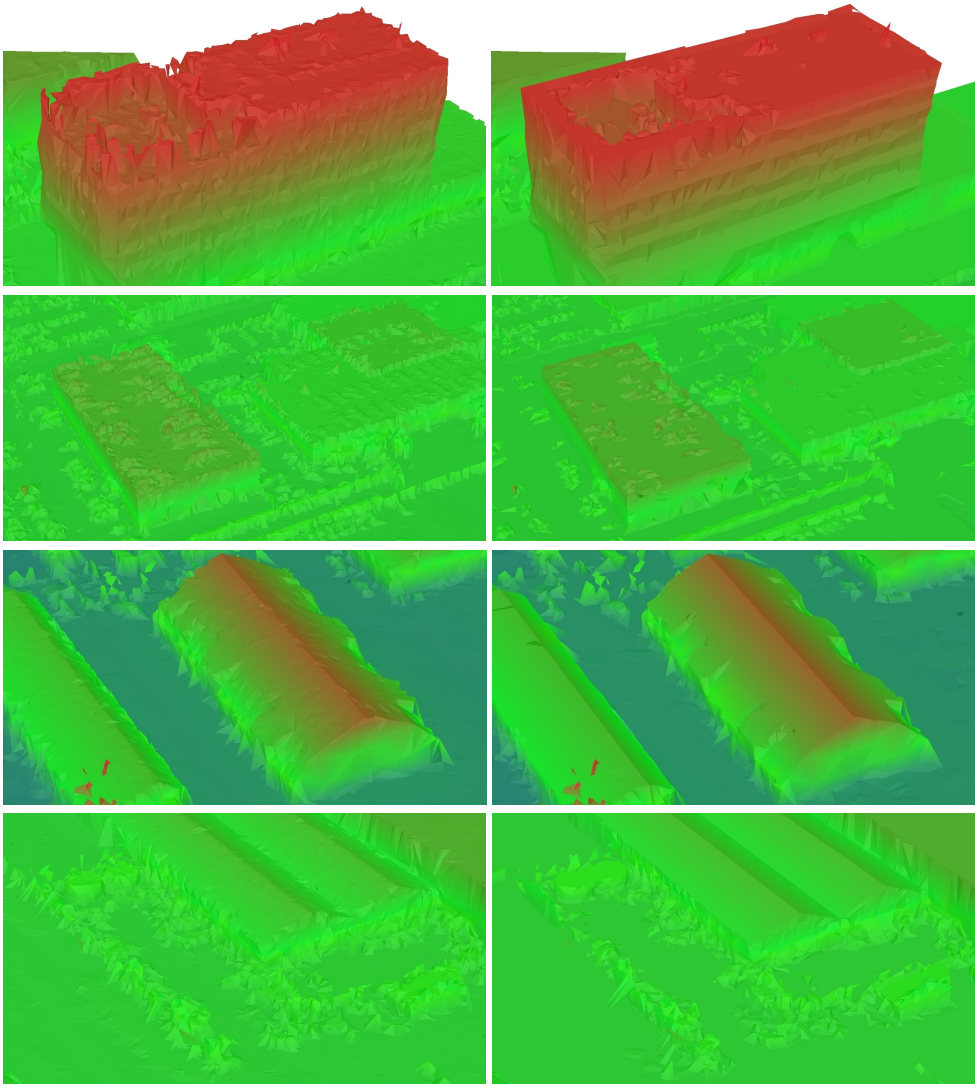


Figure 6.17: The watertight geometry produced using (Labatut et al. 2009) (left), and our method (right). The triangles are vertex-colored by height using a rainbow map combined with their surface normal orientation.

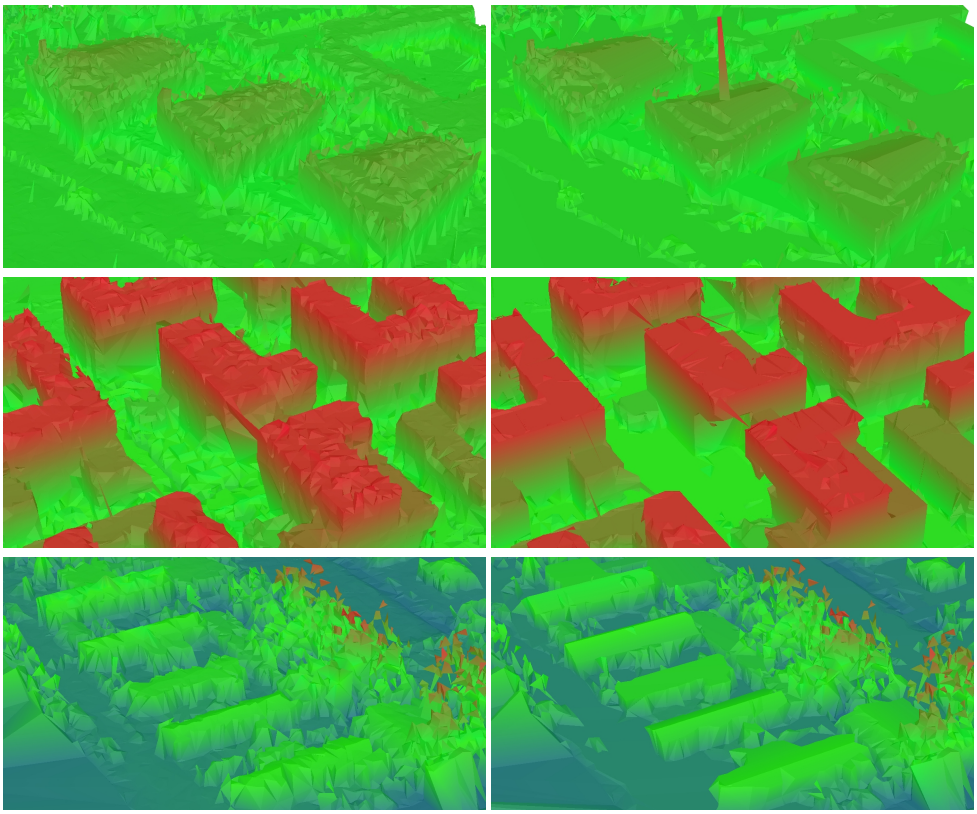


Figure 6.18: The watertight geometry produced using (Labatut et al. 2009) (left), and our method (right). The triangles are vertex-colored by height using a rainbow map combined with their surface normal orientation.

	DT				CCDT					
	$\|V\|$	$\|F\|$	$\|C\|$	$\|T\|$	$\|V\|$	$\|S\|$	$\|F\|$	$\|C\|$	$\|T\|$	$\|T\|_r$
1	17,202	220,476	110,235	29,207	12,146	4,289	158,000	78,997	16,014	0.55
2	145,209	1,874,214	937,104	196,569	109,310	25,885	1,398,058	699,026	119,441	0.61
3	42,075	533,938	266,966	69,386	36,002	7,156	463,100	231,547	52,196	0.75
4	27,173	343,386	171,690	42,321	14,359	1,542	187,592	93,793	19,521	0.46
5	32,557	416,248	208,121	50,086	23,180	6,163	295,080	147,537	29,445	0.59
6	28,767	370,912	185,453	52,462	46,312	28,571	551,930	275,962	59,018	1.12
7	20,656	268,002	133,998	32,810	15,808	1,203	207,524	103,759	23,073	0.70

Table 6.3: The number of vertices ($|V|$), facets ($|F|$), and cells ($|C|$) in the DT and CCDT for the various chunks, together with the number of triangles used for the watertight surface ($|T|$). The CCDT vertices also include a significant number of Steiner points ($|S|$). Finally, the ratio of triangles between CCDT and DT is given by $|T|_r$.

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

Table 6.4: The total time (T) in seconds to compute the watertight geometry using the DT and CCDT and the percentage of this time taken by the various steps in the process: inserting the points (t_P), inserting the intersection lines between shapes (t_l), inserting the boundaries of the shapes (t_b), inserting the interiors of the shapes (t_p), traversing the lines of sight to adjust the weights (t_L), and performing the graph-cut (t_G).

The disadvantage of our method is that it takes significantly more time to compute the geometry, as shown in Table 6.4. Once the CCDT is computed, traversing the lines of sight to set the weights of the graph and performing the graph-cut takes roughly the same amount of time for the CCDT as for the DT.

6.5 Conclusions

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

One of the most interesting directions to improve on this method is to reduce the time needed to compute the CCDT, as this is an order of magnitude worse than computing the DT. Although our implementation is based on the earlier work by Si and Gärtner (2005) and Shewchuk (2002), some clever usage of multi-core processing may greatly speed up the process.

It is also interesting to consider the recurrence of Delaunay structures in this method and those used to pre-process the data. Most preceding methods, like estimating the point normals, already use a 3-dimensional DT. Others, like the guided α-shape, use a 2-dimensional constrained DT. Perhaps it would be beneficial to combine these methods

so they could all use the same data structure. This may save a lot of time by reducing the amount of redundant work.

It is currently not possible to insert points into our CCDT once polygons have been constrained. At the same time, with massive datasets and changing scenes, it is important to be able to efficiently update a model based on new data.

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

Chapter 7

General Discussion

This dissertation has provided a novel methodology to reconstruct shapes from large urban point clouds. In Chapter 2 we described various related methods aimed at solving this problem. The pipeline that we presented in the same chapter indicates an important property of our methodology that differentiates it from the related work: we separate the scene into planar surfaces and we reconstruct the scene per surface. This divide-and-conquer approach provides manageable subproblems that can be solved efficiently.

In Chapters 3–5, we described several structures and methods to automatically reconstruct the boundaries of individual surfaces. Each of these methods constructs planar polygons to model the shape of each surface. We discuss these methods and their associated geometric structures in Section 7.1. In Chapter 6 we presented a method to consolidate these surfaces into a geometric model for the complete scene. We discuss this method in Section 7.2. Finally, in Section 7.3, we discuss our methodology in a broader scope together with some interesting problems that remain for future work.

7.1 Divide

When constructing boundaries per surface, there are several valid approaches. We may wish to reconstruct only surfaces that conform to some predefined shape. Conversely, we may wish to base the shape on the distribution of points and other data.

In either case, it is important not to construct shapes without data to support them. Chapter 3 presented δ-coverage to model the region near points. Chapter 4 presented the (α,δ)-sleeve to model a buffer zone around the point set.

Many of the surfaces in an urban scene are rectangular. Chapter 3 presented a method to handle these specific surfaces. This method constructs the ranges of rectangles that

both contain all points and are δ-covered. The same chapter described how to adapt this method to construct convex shapes with corners with predetermined angles. Both variations compute the ranges of δ-covered shapes in $O(n \log n)$ time, where n is the number of input points.

In many cases, the exact shapes of the surfaces in the scene are not known in advance. However, the prevalence of orthogonal directions in urban scenes causes most surfaces to be rectilinear. Chapter 4 presented a method that constructs a rectilinear shape in the (α, δ)-sleeve of the point set. Specifically, this method searches for the rectilinear shape with the fewest edges. Constructing the (α, δ)-sleeve takes $O(n \log n)$ time and constructing the minimum-link rectilinear shape takes $O(n)$ time per rotation considered, where n is the number of input points.

If no fixed properties of the shapes are known in advance, yet another approach is required. Related work has presented the α-shape (Edelsbrunner et al. 1983), a suitable shape to bound a planar point set. However, these shapes have a piecewise-linear boundary, for which all corners lie on the input points. These shapes are difficult to combine into 3D shapes. For this purpose, we want to use the intersection lines of neighboring surfaces to influence the shapes.

Chapter 5 presented two methods to construct the shape of a set of points and lines. The generalized α-shape considers the input lines as additional infinite collections of points and constructs a shape containing all points. The guided α-shape considers the input lines as recommended locations for the shape boundary and constructs a shape with its boundary on these lines where points are nearby. Both methods construct the shape in $O(n \log n)$ time, where n is the number of points and lines.

7.2 Conquer

Computing a shape per surface makes the reconstruction problem more tractable. However, the resulting geometry does not constitute a complete model of the scene. In many cases, the reconstructed surfaces are disconnected and some surfaces of the scene may be missing.

Chapter 6 presented the conforming constrained Delaunay triangulation (CCDT), a geometric data structure that embeds the surfaces into a space-filling construct. This structure contains connectivity information between the different surfaces while at the same time defining a nice way to subdivide 3-space.

This chapter also presented a method to construct a geometric model of the complete

scene. This method selects which parts of the CCDT are inside and outside of the objects in the scene. This selection is achieved by a graph-cut, guided by lines-of-sight between the sensor and data points and by the general quality of the local surface. Surfaces supported by planar shapes have a higher chance of appearing as separators between inside and outside regions, compared to unsupported surfaces. This produces a concise and visually pleasing geometric model of the scene.

7.3 Future Research

In this thesis we have presented several aspects of shape reconstruction from large urban LiDAR data. We have presented a pipeline that follows a novel divide-and-conquer methodology and we have presented several methods to perform the steps of this pipeline. However, the urban shape reconstruction problem is far from solved and several interesting directions of research remain.

Firstly, the quality of our results greatly depends on the quality of the input. Our methods are applied to data in an advanced stage of processing and there are several parts of preprocessing that may be performed differently. In this dissertation, we have performed limited analysis on the LiDAR data and the particularities of its collection. The results of our methods may improve by incorporating more knowledge of the specific measurement artifacts of LiDAR. For example, a better model for the noise and outliers in the data may improve our results.

Another important part of preprocessing is clustering the data into separate surfaces. We use Efficient RANSAC (Schnabel et al. 2007) to perform this clustering. However, there may be other methods that perform this task better. Similarly, we have chosen the parameter settings of RANSAC empirically, but deeper analysis of the data may provide better settings. It may even be possible to set these parameters automatically by studying the data distribution. Conversely, we may look at different ways to improve the robustness of our methods to clustering errors.

Each of the methods presented in Chapters 3–5 depends on one or two parameters related to the sampling density and accuracy. These parameters have a clear geometric meaning and can be set empirically, but this requires some expert knowledge or several trial-and-error tries. These methods would become more accessible if the parameters could be set automatically. Another reason to automatically set these parameters is to increase the amount of automation in the pipeline. This should enable the methods to process more data in less time, even when various data sets were measured under different conditions. Various related works (Cazals et al. 2005; Mandal and Murthy 1997) have shown that automatic selection of such parameters is a difficult problem.

In a similar vein, we may consider varying these parameters within one surface to better model the local density. For the results in Chapters 3–5 we have used different parameter settings for horizontal and vertical surfaces. The reason was the difference in point density on these surfaces. However, due to various aspects of measurement, the point density within a single surface may vary locally. Related work (Akkiraju et al. 1995) has shown that reconstructing shapes with locally varying parameter settings is a challenging problem.

Our method for reconstructing watertight geometry uses other parameters, which are related to the measurement accuracy and minimum object thickness. Some parameter setting may result in the correct model for large parts of the scene. However, the same setting may produce errors near small details, thin objects, and vegetation. It may be possible to identify the different objects that points were measured on and apply different parameter settings based on the data point types. This approach may also prove beneficial for our boundary reconstruction methods.

It may also be possible to implement some improvements by examining the pipeline as a whole. For example, several of the methods used within the pipeline use the 2D or 3D Delaunay triangulation. RANSAC uses an estimate of the local surface normal, which can be computed using the 3D Delaunay triangulation. Later, our gap-filling method uses the CCDT. Reusing the earlier structure would reduce construction times. Similarly, several methods may benefit from parallel computing. The most straightforward example of this is the surface boundaries, which can be computed independently.

In this dissertation, we have maintained the general assumption that most surfaces in an urban scene are planar. While this assumption may be valid, a scene may also contain surfaces on other simple objects, like cylinders and spheres. We may be able to adjust our methods to correctly reconstruct these surfaces, but this will require careful evaluation of their features and guarantees.

Within shape reconstruction there is an unfortunate lack of benchmark datasets and ground truths. Even when a reconstruction looks good, it is difficult to tell how close it is to the original object that the data was measured from. This makes it very difficult to objectively compare methods. If you cannot quantify how correct your results are, how can you objectively state your method is better than others? While ground truth CAD drawings are available for small scale reconstructions, such as mechanical parts, these are lacking for urban data.

Similarly, there are few metrics for comparing different shape reconstruction results. For 3D meshes, the most used difference quantifier is the Haussdorf distance. However, this distance measure is not symmetric, i.e. the distance from mesh A to mesh B is not

the same as the distance from B to A. Furthermore, the distance is generally expensive to compute, making the use of the symmetric Haussdorf distance restrictive. Finally, the distance depends on the sampling density, because only the distances from points to the other mesh are measured. This means that for lower density models or lower density regions, the distance measure can be inaccurate.

Novel metrics may be found in various directions. We could develop novel metrics specific to shape reconstruction. These metrics could include several aspects of the geometric models. Some models or regions should be smooth, while others should include features such as corners and creases. Similarly, some regions should be simplified or represented in a concise manner, while others should contain as much detail and precision as possible. Different metrics may value these aspects differently. Another direction would be to develop a distance measure that expresses surface-to-surface distance instead of point-to-surface distance.

Finally, as the size of the data sets increases, so does the importance of reducing the memory footprint of the reconstruction methods. If the structures used during processing become too large for the working memory, caching will occur and this may greatly increase the processing time. This may be averted by using streaming methods, which aim to process the data in small chunks while minimizing the memory footprint.

Bibliography

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

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

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

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

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

Amenta, N. and Bern, M. (1998). Surface reconstruction by voronoi filtering. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 39–48, New York, NY, USA. ACM.

Amenta, N., Bern, M., and Eppstein, D. (1998). The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135.

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

Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266.

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359.

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

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

Buchin, K., Meulemans, W., and Speckmann, B. (2011). A new method for subdivision simplification with applications to urban-area generalization. In *Proc. SIGSPATIAL*, pages 261–270.

Carlberg, M., Andrews, J., Gao, P., and Zakhor, A. (2009). Fast surface reconstruction and segmentation with ground-based and airborne LiDAR range data. Technical Report ADA538860, University of California at Berkeley.

Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH*, pages 67–76.

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

Cazals, F., Giesen, J., Pauly, M., and Zomorodian, A. (2005). Conformal alpha shapes. In *PBG*, pages 55–61.

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

Chazelle, Devillers, Hurtado, F., Mora, Sacristan, and Teillaud (2002). Splitting a Delaunay triangulation in linear time. *Algorithmica*, 34:39–46.

Chevallier, N. and Maillot, Y. (2011). Boundary of a non-uniform point cloud for reconstruction: extended abstract. In *Proc. SoCG*, pages 510–518.

Chew, L. P. (1989). Constrained delaunay triangulations. *Algorithmica*, 4(1):97–108.

Cohen-Steiner, D. and Da, F. (2004). A greedy Delaunay-based surface reconstruction algorithm. *The Visual Computer*, 20:4–16.

Computational Geometry Algorithms Library (2013). `http://www.cgal.org/`.

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proc. SIGGRAPH*, pages 303–312.

de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition.

Debevec, P., Taylor, C. J., and Malik, J. (1998). Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *9th Eurographics Rendering Workshop*.

Dey, T. K. and Goswami, S. (2003). Tight cocone: a water-tight surface reconstructor. In *Proc. SM*, pages 127–134.

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

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

Edelsbrunner, H. and Tan, T. S. (1992). An upper bound for conforming Delaunay triangulations. In *Proc. SoCG*, pages 53–62.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Graphics and Image Processing*, 24(6):381–395.

Ford, D. R. and Fulkerson, D. R. (2010). *Flows in Networks*. Princeton University Press, Princeton, NJ, USA.

Fugro EarthData, Inc. (2012). `http://www.fugroearthdata.com/`.

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

Gopi, M., Krishnan, S., and Silva, C. (2000). Surface reconstruction based on lower dimensional localized Delaunay triangulation. *Comp. Graph. Forum*, 19(3):467–478.

Graph Cut optimization library (2012). `http://cbia.fi.muni.cz/user_dirs/gc_doc/`.

Guibas, L., Knuth, D., and Sharir, M. (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413.

Hershberger, J. and Snoeyink, J. (1994). Computing minimum length paths of a given homotopy class. *Computational Geomerty: Theory and Applications*, 4:63–97.

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

Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J. A., Schweitzer, J., and Stuetzle, W. (1994). Piecewise smooth surface reconstruction. In *Proc. SIGGRAPH*, pages 295–302.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. A., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. In *Proc. SIGGRAPH*, pages 71–78.

John Chance Land Surveys and Fugro (2009). Fli-map specifications. `http://www.flimap.com/site47.php`.

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

Lafarge, F., Descombes, X., Zerubia, J., and Pierrot-Deseilligny, M. (2010a). Structural approach for building reconstruction from a single DSM. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):135–147.

Lafarge, F., Keriven, R., Brédif, M., and Vu, H. (2010b). Hybrid multi-view reconstruction by jump-diffusion. In *Proc. CVPR*, pages 350–357.

Lee, S. C. and Nevatia, R. (2004). Extraction and integration of window in a 3D building model from ground view images. In *Proc. CVPR*, pages 113–120.

Ling, R., Wang, W., and Yan, D. (2008). Fitting sharp features with loop subdivision surfaces. In *Proc. SGP*, pages 1383–1391.

Lipman, Y., Cohen-Or, D., and Levin, D. (2007). Data-dependent MLS for faithful surface approximation. In *Proc. SGP*, pages 59–67.

Lorensen, W. and Cline, H. (1987). Marching cubes: a high resolution 3D surface construction algorithm. In Stone, M. C., editor, *Proc. SIGGRAPH*, volume 21, pages 163–169, New York, NY, USA. ACM.

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

Marton, Z. C., Rusu, R. B., and Beetz, M. (2009). On fast surface reconstruction methods for large and noisy point clouds. In *Proc. ICRA*, pages 2829–2834.

Mayer, H. (2005). Scale-spaces for generalization of 3D buildings. *International Journal of Geographical Information Science*, 19:975–997.

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

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

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

Musialski, P., Wonka, P., Aliaga, D. G., Wimmer, M., van Gool, L., and Purgathofer, W. (2012). A survey of urban reconstruction. In *Comp. Graph. Forum*, pages 1–28.

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

Poullis, C. and You, S. (2009). Automatic reconstruction of cities from remote sensor data. In *Proc. CVPR*, pages 2775–2782.

Pu, S. and Vosselman, G. (2009). Knowledge based reconstruction of building models from terrestrial laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(6):575–584.

Regnauld, N., Edwardes, A., and Barrault, M. (1999). Strategies in building generalisation: modelling the sequence, constraining the choice. In *Proc. ICA*.

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

Ruas, A. (1999). *Modèle de généralisation de données géographiques à base de contraintes et d'autonomie*. PhD thesis, Université de Marne la Vallée.

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

Salman, N., Yvinec, M., and Merigot, Q. (2010). Feature preserving mesh generation from 3D point clouds. *Computer Graphics Forum*, 29(5):1623–1632.

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

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

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

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

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

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

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

Shewchuk, J. R. (2002). Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proc. IMR*, pages 193–204.

Shewchuk, J. R. (2003). Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proc. SoCG*, pages 181–190.

Si, H. and Gärtner, K. (2005). Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proc. IMR*, pages 147–163.

Sinha, S. N., Steedly, D., Szeliski, R., Agrawala, M., and Pollefeys, M. (2008). Interactive 3D architectural modeling from unordered photo collections. *ACM Trans. Graph.*, 27(5):1–10.

Swan, J., Anand, S., Ware, M., and Jackson, M. (2007). Automated schematization for web service applications. In *Web and Wireless GISystems*, LNCS 4857, pages 216–226.

Teichmann, M. and Capps, M. (1998). Surface reconstruction with anisotropic density-scaled alpha shapes. In *Proc. VIS*, pages 67–72.

The Stanford 3D Scanning Repository (2010). `http://graphics.stanford.edu/data/3Dscanrep/`.

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

Tseng, Y.-H., Tang, K.-P., and Chou, F.-C. (2007). Surface reconstruction from LiDAR data with extended snake theory. In *Proc. EMMCVPR*, pages 479–492.

van Kreveld, M., van Lankveld, T., and de Rie, M. (2013a). (α,δ)-sleeves for reconstruction of rectilinear building facets. In *Proceedings of the 7th International 3D GeoInfo Conference*, Lecture Notes in Geoinformation and Cartography, pages 231–248. Springer Berlin Heidelberg.

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2009). Identifying well-covered minimal bounding rectangles in 2D point data. In *Proc. EuroCG*, pages 277–280.

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

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2011b). On the shape of a set of points and lines in the plane. Technical Report UU-CS-2011-017, Utrecht University, Department of Information and Computing Sciences.

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2013b). Watertight scenes from urban lidar and planar surfaces. In *Proc. SGP*, number 12, pages 217–228.

van Kreveld, M., van Lankveld, T., and Veltkamp, R. C. (2013c). Watertight scenes from urban lidar and planar surfaces. Technical Report UU-CS-2013-007, Utrecht University, Department of Information and Computing Sciences.

van Lankveld, T., van Kreveld, M., and Veltkamp, R. C. (2011a). Identifying rectangles in laser range data for urban scene reconstruction. *Computers & Graphics*, 35(3):719–725.

van Lankveld, T., van Kreveld, M., and Veltkamp, R. C. (2011b). Identifying rectangles in laser range data for urban scene reconstruction. Technical Report UU-CS-2011-004, Utrecht University, Department of Information and Computing Sciences.

Veltkamp, R. C. (1992). The γ-neighborhood graph. *Computational Geometry Theory and Applications*, 1(4):227–246.

Vosselman, G., Gorte, B. G. H., Sithole, G., and Rabban, T. (2004). Recognising structure in laser scanner point clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 46:33–38.

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

Wolff, A. (2007). Drawing subway maps: a survey. *Informatik Forschung und Entwicklung*, 22(1):23–44.

Yap, C.-K. (1987). An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete Compututational Geometry*, 2:365–393.

Zhou, Q.-Y. and Neumann, U. (2008). Fast and extensible building modeling from airborne LiDAR data. In *Proc. SIGSPATIAL*, pages 1–8.

Notation

Throughout the thesis, we maintain a consistent format for notation. This format determines the symbols and colors used in the text and images. Here, we describe this format, followed by an overview of the semantics for some of the symbols we use multiple times. Finally, we present the abbreviations used throughout the thesis.

Mathematical Notation

p	point
$\overline{pq}$	segment or edge connecting p and q or line through p and q
$\overrightarrow{pq}$	vector from p to q or ray from p through q
$\triangle pqr$	triangle on p, q, and r
S	set of geometric items
$\mathcal{R}$	bounded region, polygon, subspace, or infinite point set
$\partial\mathcal{P}$	bound of a polygon
$\lvert P\rvert$	size or cardinality of P
$\lVert pq\rVert$	distance between p and q
$\lVert s\rVert$	length of s
$\angle pqr$	angle between $\overrightarrow{qp}$ and $\overrightarrow{qr}$
$\mathbb{R}$	scalar number type, e.g. real ($\mathbb{R}$) or integer ($\mathbb{Z}$)

Colors

blue	constructive, e.g. the α-shape
gray	underlying structure, e.g. the DT
green	motion, e.g. the trajectory region
red	restrictive, e.g. the δ-coverage region or a constraint

Prevailing Semantics

α	sampling resolution measure
$\mathcal{A}$	α-shape
δ	data coverage distance
$\mathcal{D}$	disk
$\mathcal{H}$	convex hull
k	integer number
L	line set
m	number of lines or line segments
n	number of points
N	input size
$\mathcal{P}$	general polygon
$\mathcal{R}$	rectangle
θ, ψ	angle
S	point sample
$\mathcal{T}$	trajectory region
$\mathcal{U}_\delta$	δ-coverage region.

Abbreviations

CAD	Computer-Aided Design
CCDT	Conforming Constrained Delaunay Tetrahedralization
CDT	Constrained Delaunay Triangulation, Tetrahedralization
CGAL	Computational Geometry Algorithms Library
DEM	Digital Elevation Model
DT	Delaunay Triangulation, Tetrahedralization
GPS	Global Positioning System
lfs	Local Feature Size
LiDAR	light-radar, Light Detection and Ranging
PCA	Principal Components Analysis
PLC	Piecewise Linear Complex
RANSAC	RANdom SAmple Consensus

Samenvatting

Tegenwoordig kun je virtuele werelden in veel verschillende vormen vinden. Computer spellen spelen zich vaak af in fraai vormgegeven steden of gebieden. Architecten tonen hun ontwerpen in de huidige omgeving op billboards. Als je de weg zoekt naar een onbekende bestemming, gebruik je waarschijnlijk een navigatie systeem. In veel gevallen is het gewenst dat een virtuele wereld 3D en direct herkenbaar is. Het modelleren van deze werelden kost veel tijd en moeite en automatisering kan deze lasten verlichten.

Omdat virtuele werelden in opkomst zijn is er veel onderzoek gedaan naar het automatisch modelleren van deze werelden. Methoden met hun oorsprong in het modelleren van sculpturen proberen vooral oppervlakken met gladde overgangen te gebruiken. Methoden gericht op het modelleren van steden gebruiken vaak een verzameling voorbeeld-gebouwen waarin ze de beste kandidaten zoeken.

In dit proefschrift presenteren we een nieuwe aanpak voor het automatisch modelleren van stedelijke gebieden. Deze nieuwe aanpak komt voort uit de observatie dat stedelijke gebieden grotendeels bestaan uit platte vlakken. We gebruiken een verdeel-en-heers strategie om een gebied effectief en efficient te modelleren. We verdelen de invoer in verschillende vlakken en we modelleren elk vlak grotendeels onafhankelijk van de anderen. Ten slotte voegen we de vlakken weer samen in een model.

We beschrijven verschillende manieren om de vlakken te modelleren. Zo kunnen we ons beperken tot het modelleren van de vlakken die voldoen aan een bapaalde vorm, zoals rechthoeken. Deze methode is beperkt tot convexe vormen, simpel gezegd: vormen zonder deuken. We presenteren een andere methode voor het modelleren van rectilineaire vormen, zoals L-vormen. Deze methode zoekt een vorm die de invoer zo goed mogelijk beschrijft waarbij alleen rechte hoeken gebruikt worden. Ten slotte presenteren we een methode die gebruik maakt van de buur-vlakken om een algemene vorm te modelleren. Deze methode zoekt een vorm die netjes aansluit op zijn buren.

Wanneer de afzonderlijke vlakken eenmaal gemodelleerd zijn, combineren we ze in een compleet model. We beschrijven een methode die tussen de vakken een waterdicht model zoekt. Een waterdicht model is als een vreemd gevormde fles: je kunt hem vullen met water zonder dat hij lekt.

Uiteraard zijn onze nieuwe methoden niet de enige mogelijkheid om stedelijke gebieden te modelleren. Daarom eindigen we het proefschrift met een discussie over onze methoden. Het ontwikkelen van deze methoden heeft verschillende nieuwe vraagstukken naar voren gebracht. Door in te gaan op deze vragen kunnen we ons werk verbeteren om zo mogelijk het onderzoeksveld te vorderen.

Curriculum Vitae

Thijs van Lankveld was born in Veghel, The Netherlands, on July 9th 1983. He attended high school at the Stedelijk Gymnasium Leiden from 1995 – 1998 and the Jeanne d'Arc College, Maastricht from 1998 – 2001, at the VWO level.

Thijs studied Knowledge Engineering at Maastricht University. This college included a five week foreign exchange program at Baylor University, Waco, Texas in September 2003. Thijs also worked as student-assistant for the practical assignments of the Linear Algebra course in 2002. He completed the Artificial Intelligence Master track with his thesis on evolutionary algorithms in robotics, supervised by Guido de Croon and Karl Tuyls. In that thesis, they compared two types of neural networks controlling a Khepara, a small robot equipped only with two wheels and eight infrared sensors. This Master thesis was published in the proceedings of the BNAIC 2007 conference.

This dissertation describes the PhD research Thijs performed at Utrecht University from 2008 until 2012. During this PhD track, Thijs co-supervised the masters project of André van Noorloos together and the experimentation project of Maarten de Rie, both together with Marc van Kreveld. He also supervised the practical assignments for the Imperative Programming (Java and C#) and Three-dimensional Modeling courses for two years. Finally, he followed courses in Pattern Recognition, Visualization and Virtual Reality at the ASCI graduate school.

Acknowledgements

With a great sense of pride and relief I have finally crowned my PhD with this booklet. With the final words of my thesis, I look back at what got me here. For this retrospective, I could start with the years of learning to be a researcher at Utrecht University and all the people highly deserving of praise for all their guidance and support. I feel I should start a bit further back: at the beginning.

Growing up, I have always had a preoccupation with contemplation. I could easily spend hours in thought, either imagining grand fantasy worlds or pondering why something should be done one way or another. This flowed easily into a preference for the exact subjects at school. When time came to choose my direction for college, mathematics was a strong candidate. I chose computer science instead, because at that time I had no idea what I would do with a math degree. When I was young, I had coded a rudimentary program in BASIC that could respond to written questions, so artificial intelligence seemed a promising endeavor.

The next years saw me forming both basic academic knowledge and an active social life. I found that I enjoyed learning and developing creative ways to solve problems. After completing college, I decided I wanted to continue expanding my academic skills. The search for a suited research subject brought me back to more theoretical work in the form of geometry. I seem to have found a point where my regard for theory meets the joy of making it work in practice.

I would like to thank my advisors, Marc van Kreveld and Remco Veltkamp, for guiding me in finding my place in the academic world. For me, they neatly embody the two opposite aspects of my work. Marc provided a theoretical powerhouse, always happy to provide an insightful comment or a productive dialog. Remco grounded my work, with a consistent gaze on my progress and a plethora of applications to put the theory to the test. During my PhD, I have seen both attain the title of Professor, and I feel

honored to have two such great people as my promoters. I would also like to thank the members of the reading committee: Pierre Alliez, Mark de Berg, Massimo Meneti, Monika Sester, and George Vosselman. Without objective judgment by knowledgeable people, scientific work loses all value.

Besides academic growth, I feel I could not have completed my thesis without a social life to find relief in the off hours. I greatly appreciate the conversations, whether academic and not, with Bas de Haas and Ben van Basten. Ioannis Karamouzas, thanks for all the squash sessions. Anne, Frank, Frans, Geert-Jan, Kevin, Maarten, Maike, Robby, Rodrigo, Saskia, and Xinghan: thanks for making the Utrecht University a great place to work. I would also like to thank all my new colleagues at Inria, who made all the difference in making me feel at home in a foreign country. Special thanks go to Ramsay Dyer and Mariette Yvinec for their comments on Chapter 6.

When not at work, Villa Furka was always there to provide distractions. Marnix, Joost, Daan, Dorien, Giselle, Lotte, Mischa, and Sarah, it was awesome to have a great place to rest my feet and an even greater place to move my feet. My heartfelt thanks also go to my gaming group: Bas, Floris, Johan, Matt, Paul, and Rico. Thank you for all the occasions I could indulge in my gaming sweet-tooth. Thank you too, Mickey; whether at Biton or with a game between us, I always had a good time. Special thanks go to Marnix Rijnart and Johan Otten for having my back as *paranimf*.

Although my research kept me busy and at a distance, I owe a lot of thanks to my family. Jacques, thank you for all the comments you gave every time I presented some new work, but even more for the pride I saw for each of my achievements. Francine, thank you for believing that I can do anything I set my mind to. Giel, thank you for surprising me every time by how many interests we share.

Although there are many more I should be thanking, I must end my thesis with the most important one. Nienke, thank you for selflessly being there whenever you feel I need something. Thank you for all the patience and the doubled support while I was laying the finishing touches to my research. Thank you for letting me go all the way to France to chase my future. Thanks to you, I always strive to be the best I can be.