Graz University of Technology

Institute for Computer Graphics and Vision

Siemens CT RTC ICV VIA-AT

Master's Thesis

Geometric Abstraction for Noisy Image-Based 3D Reconstructions

Thomas Holzmann

Graz, Austria, April 2014

Thesis supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof

Thesis advisors

Dipl.-Ing. Christof Hoppe

Dipl.-Ing. Dr.techn. Stefan Kluckner

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …………………………… ………………………………………………..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…………………………… ………………………………………………..
date (signature)

Abstract

With state-of-the-art reconstruction methods it is possible to create scene reconstructions resulting in a point cloud representation consisting of millions of points. As such a large amount of data is not processable for many applications, an abstracted representation is needed. However, creating geometrically abstracted models from image-based scene reconstructions is challenging due to noise and irregularities in the underlying reconstructed model. Many state-of-the-art approaches focus on extracting geometric structures from laser scan data which usually contain less noise. Others approximate surfaces using priors like geometric primitives or other parametric representation methods without introducing different levels of detail.

In this thesis, we present a geometric modeling method for noisy, image-based reconstructions dominated by planar horizontal and orthogonal vertical structures. At dominant horizontal structures, we partition the scene into horizontal slices. As a whole slice contains similar vertical scene properties, we create a binary inside/outside labeling represented by a floor plan for each slice by solving an energy minimization problem. Consecutively, we create an irregular discretization of the volume according to the individual floor plans and again label each cell as inside/outside by minimizing an energy function. By adjusting the smoothness parameter, we introduce different levels of detail.

In our experiments, we show results with varying regularization levels using synthetically generated and real-world data. We discuss the level of regularization and the error with respect to geometry for different parameter settings and point out strengths and weaknesses of our approach.

Keywords. geometric scene abstraction, computational geometry, graph cuts, image-based 3D reconstruction

Kurzfassung

Mit aktuellen, bildbasierten Rekonstruktionsmethoden ist es möglich, Rekonstruktionen von Szenen bestehend aus Millionen von Punkten zu erstellen. Für viele Anwendungen ist diese große Datenmenge allerdings schwer prozessierbar und muss deshalb in eine abstrahierte, vereinfachte Form umgewandelt werden. Aufgrund von Rauschen und Unregelmäßigkeiten in bildbasierte 3D-Rekonstruktionen ist es jedoch schwierig, geometrisch abstrahierte Modelle zu erzeugen. Viele aktuelle Ansätze konzentrieren sich nur auf die Extrahierung geometrischer Strukturen von Laser Scan-Daten, welche üblicherweise weniger Stördaten enthalten. Andere Methoden approximieren Oberflächen mithilfe von Annahmen wie geometrischen Primitiven oder anderen parametrischen Repräsentationen, ohne der Möglichkeit der Erzeugung von unterschiedlichen Levels of Detail.

In dieser Arbeit präsentieren wir eine geometrische Abstraktionsmethode für bildbasierte 3D-Rekonstruktionen. Wir spezialiseren uns auf Szenen, welche hauptsächlich aus planaren horizontalen und vertikalen Strukturen bestehen. An dominanten horizontalen Strukturen wird das Modell in horizontale Ebenen unterteilt. Durch Lösen eines Energieminimierungsproblems werden Bereiche der Ebene als Innen oder Außen markiert. Der Übergang zwischen den beiden Bereichen resultiert in einem Grundriss der jeweiligen Ebenen. Unter Verwendung der Grundrisse wird eine irreguläre Diskretisierung für das gesamte Volumen erstellt, wobei jede Zelle durch Minimierung einer Energiefunktion wiederum als Innen oder Außen markiert wird. Mithilfe eines Regularisierungsparameters ist es möglich, unterschiedliche Levels of Detail zu erzeugen.

In unseren Experimenten präsentieren wir Ergebnisse mit verschiedenen Regularisierungsgraden unter Verwendung von synthetisch generierten und realen Daten. Wir diskutieren den Regularisierungsgrad und den Fehler bezüglich der Geometrie für unterschiedliche Parametereinstellungen und zeigen die Stärken und Schwächen unseres Ansatzes auf.

Acknowledgments

This work would not have been possible without the help of my supervisors Prof. Dr. Horst Bischof, Dr. Stefan Kluckner and Christof Hoppe, who supported me continuously during the work of my thesis. Thank you for all the discussions, for carefully proofreading my thesis and for supporting me at further scientific goals in addition to the supervision of my thesis.

I also want to thank the whole team at Siemens in Graz. They supported me with discussions and with the possibility to use all the equipment and data from Siemens for the work at my thesis. Additionally, I gained a lot of experience while working with them besides my studies in the last years.

Last but not least, I want to thank my family and especially my parents. They gave me the possibility to study and supported me during my whole studies.

This work was supported by the Austrian Research Promotion Agency (FFG) and Siemens AG within the CONSTRUCT research project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, 3D reconstruction techniques have reached maturity in quality and performance. For example, using Structure from Motion (SfM), it is possible to reconstruct a sparse point cloud from an unordered set of images. As a result, a point cloud like in Figure 1.1 (left) is reconstructed. There exists freely availably SfM implementations like Bundler [44] [45].

However, for many applications the density of the point cloud is not sufficient and needs to be densified in an additional step. For this, a commonly used method is Patch-based Multi-view Stereo (PMVS) [19], which delivers a dense point cloud like in Figure 1.1 (right). Having a sufficiently dense point cloud, it is possible to approximate the surface of the 3D scene using meshing techniques (for example, Poisson surface reconstruction [27]) to get a watertight mesh of the whole scene (as in Figure 1.2).

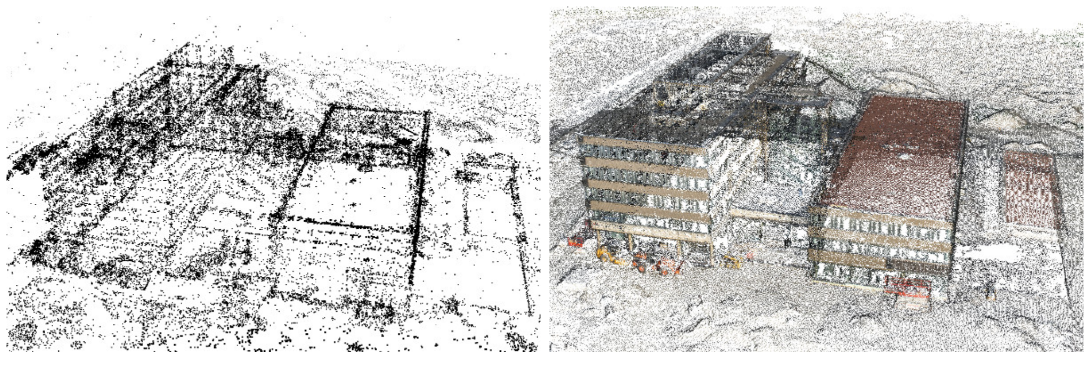


Figure 1.1: *Sparse and dense point cloud.* Left, one can see a sparse point cloud reconstructed with SfM. Many parts of the scene, especially surfaces with less texture, are represented by just a few points. Right, the sparse point cloud densified with PMVS is shown. Many surfaces have a better representation by points. However, windows and the texture-less regions (for example, the left top roof) are still not reconstructed well.

In addition to image-based reconstruction methods, cost effective RGB-D sensors like Microsoft Kinect [4] or Time-of-Flight cameras became available. With these devices, it is possible to get 3D information of a scene even if no texture exists or the scene is insufficiently lightened. However, as these sensors capture their own emitted light, they are limited to a scene in a range of some meters in front of the sensor.

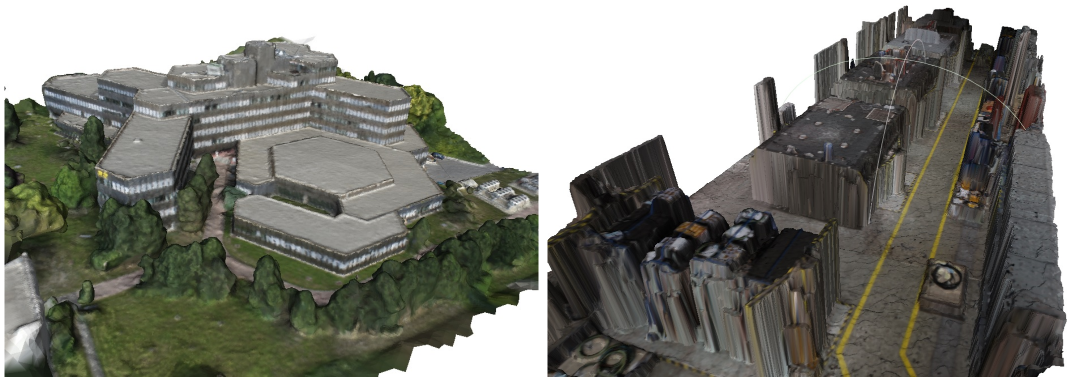


Figure 1.2: *Meshed 3D point clouds.* In this figure, some examples of reconstructions created by Siemens are shown. Left, you can see a 3D reconstruction of a building of the Siemens City in Vienna. Right, you can see a 2.5D reconstruction of a factory hall. In comparison to full 3D, a 2.5D representation just visualizes the depth information in one direction. In this case, for every position of an estimated ground plane the maximum height above this position in the scenery is visualized.

With the availability of these reconstruction methods, it is possible to generate point clouds of a scene consisting of millions of points. However, transmitting, visualizing, processing and analyzing the acquired data is far from practical use within applications. For example, an important use case is to transmit 3D data over the Internet and possibly visualize the data using an Internet browser. Considering this, models must be simplified and data has to be reduced. For example, as building models reconstructed with millions of points usually are dominated by planar structures, they can be modeled easily by a small number of planes which extremely reduces the amount of data. For example, Google Earth [21] visualizes cities using simplified 3D models of buildings mainly consisting of planar structures (see Fig. 1.3). Additionally, as the usage of mobile devices with limited computing power (like smartphones or tablets) increases continuously, a simpler representation of the 3D models has to be used to enable these devices to process and visualize the data in a reliable and usable way. Therefore, it is crucial to transform the 3D data into a compact representation and simultaneously preserve as much scenery and relevant details as possible. Further, to extract semantic information out of a 3D reconstruction, it has to be separated into semantic meaningful parts first. For example, by separating a building model into floors, a floor plan can be extracted for each level as each level contains similar vertical structures. With this information, it is possible to create adjustably regularized geometric 3D models, which can be enriched easily with additional semantic information.

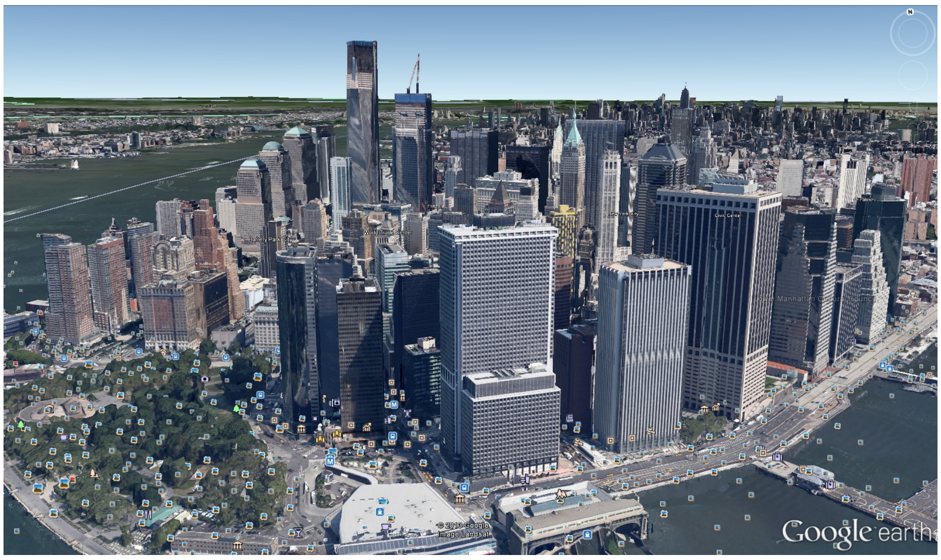


Figure 1.3: *Google Earth 3D Models.* In Google Earth, many 3D models of cities are available. The visualized buildings are a simplified, textured representation of the scenery and therefore can be transmitted smoothly over the Internet. In this figure, you can see parts of Manhattan.

An area, in which compact representations of 3D objects have already been used for a long time is computer-aided design (CAD). In CAD, various approaches exist on how to model 3D object volumes (as described in [3]). One approach is to represent the volumes using wireframes, which are vectors describing the edges of volumes. Another method is to represent the geometric structures using geometric primitives like planes. Thus every volume has to be represented with their object borders as planes. A more complex representation is the solid modeling, where basic three-dimensional geometric forms (e.g. prisms, cylinders, spheres) have solid volumes added or subtracted from them. Moreover, properties for surface or physical interaction can be introduced and assigned to consistent geometric primitives. Another approach is to describe the geometry as a parametric model. In this representation, every entity, such as a solid, a line, an arc or a filtering operation has parameters associated with it. These parameters control the various geometric properties of the entity and also the locations within the model, which enables a compact scene representation.

In our work, we focus on extracting meaningful geometric structures out of man-made environments, which are, in many cases, dominated by planar surfaces. Our approach is mainly inspired by *Reconstructing the World's Museums* by Xiao and Furukawa [48] and *Indoor Scene Reconstruction using Primitive-driven Space Partitioning and Graph-cut* by Oesau et al. [37]. Both are using horizontal slicing in order to extract meaningful horizontal segments of a model originating from laser scanner data and process them separately. Finally, they fuse the slices into a single geometry model using different regularization methods.

In contrary, we develop a geometric abstraction method which takes any meshed point cloud as input. This introduces new difficulties, as image-based reconstructions tend to contain more clutter which has to be handled in an appropriate way by the geometric abstraction method. Following the idea of [48], we first identify horizontal slices that are limited by dominant horizontal planar structures. For each slice, we generate a 2D floor plan by solving an inside/outside labeling problem in a global optimal manner formulating an energy minimization problem. With this approach, we create a labeling of the slices which is robust against clutter and generate an initial semantic interpretation of the scene represented by the individual floor plans. In order to integrate the particular slice informations into a consistent 3D model, we create an irregular discretization of the volume according to the individual floor plans. The obtained volume elements are again labeled as inside/outside by minimizing an energy function. For an individual building, the whole procedure results in a set of floor plans and an adjustably regularized geometric abstraction of the input 3D point cloud.

In our experimental evaluation, we show that our approach massively simplifies the input mesh while the results are geometrically consistent with the input data. We evaluate our method on synthetic models from Trimble Warehouse [32] and on models from buildings reconstructed from images taken by Siemens. We show that the error usually increases when increasing the regularization level and demonstrate that our approach is computationally efficient and delivers accurate results where existing methods fail.

This thesis is structured as follows: In Chapter 2 we discuss state-of-the-art point cloud abstraction methods. Chapter 3 gives background information about the used algorithms in our workflow. In Chapter 4, we describe our processing chain in detail. We define the input data and explain the individual processing steps. Chapter 5 describes the evaluation process and shows results for different configurations. In the last chapter, we discuss our results and possible future improvements.

Chapter 2

Related Work

To transform images or 3D data into geometric structures, numerous approaches exist. They can be divided into two categories depending on the input they use: single-view approaches and 3D data approaches. The following sections discuss these approaches.

2.1 Single-View

Using a single view image, a lot of work has been done in order to reconstruct indoor scenes in a constrained Manhattan world setting. Lee et al. [31] detect line segments and perform geometric reasoning using the segments to generate multiple physically valid structure hypotheses. These hypotheses are evaluated using a so-called orientation map, which is a map that expresses the local belief of region orientations computed from line segments. Similarly, Ramalingam et al. [39] recover the spatial layout of an indoor scene based on line segment junctions. They argument that man-made structures have a big amount of line segments and line junctions. Using this information, they can reconstruct the geometry of an indoor scene. Figure 2.1 shows an example of junctions in a living room.

Targeted at outdoor scenes, Gupta et al. [22] build up a blocks world taking into account physical properties and interactions of objects. Their work is inspired by the "Blocks World" work from Roberts [40] in the 1960's, which was an early attempt to construct a scene understanding system for a closed artificial world with textureless objects by using a generic library of polyhedral block components.

However, as all those approaches rely on single 2D images, they are somehow restricted to a special problem definition and are not able to handle large scenes.

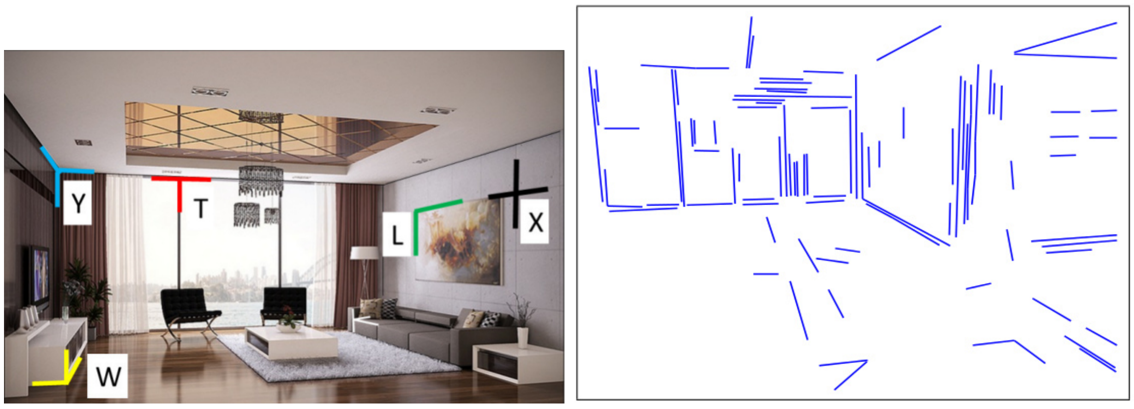


Figure 2.1: *Identifying geometry by line segments.* In [39], according to the appearance of a junction similar to a letter, the junctions are classified in types like **L, T, Y, X**. Using this information, it is possible to reconstruct the geometry of an indoor scene (left). Similarly, in [31] line segments are used to build geometry prototypes of indoor environments (right).

2.2 3D Approaches

To generate an accurate geometric representation of a scene, a 3D approach is necessary. Having multiple 2D images available, one can reconstruct a point cloud using, for example, SfM and possibly PMVS to densify the cloud.

Similarly, using a single view with additional depth information (RGB-D stream, e.g. Microsoft Kinect), restricted 3D information becomes available in form of depth maps. As the resulting data is not a complete 3D point cloud but just points seen from one view, it is called 2.5D. However, with the additional information, further possibilities arise to analyze the geometry of the scene. For example, Singh et al.[41] model indoor environments using superpixel segmentation [7]. Using the additional depth information, the superpixels are agglomerated to fragments which finally form planar structures. However, as the input data is still limited to one view, this approach is also not able to represent large scenes.

Using 3D data, Labatut et al. [30] and Hoppe et al. [26] use a Delaunay Triangulation to extract a mesh out of a (sparse) point cloud. The Delaunay Triangulation decomposes 3D point sets into tetrahedra (3D solids) and finally generates a triangular mesh of the scene. With this approach, even meshing of sparse point clouds is possible. However, it does not simplify the point cloud and does not introduce different levels of detail.

In [42], Sinha et al. contribute with robust plane-fitting of 3D points and lines using strong vanishing point cues to infer their orientation. The proposed algorithm works on sparse point clouds and is not limited to Manhattan scenes. As result, it creates piecewise planar depth maps for each view which can be used to generate a geometric model consisting out of planes.

Kluckner et al. [28] classify buildings on aerial imagery using super-pixel segmentation. A dense 3D model is calculated and a geometric model with plane prototypes is constructed taking the geometry and color information of the superpixels into account.

In Zebedin et al. [50], planes and surfaces of revolution, which are a natural description of domes and spires, are detected and an algorithm for fully automatic building reconstruction from aerial images is proposed. This approach uses a region growing process for detecting planes and models surfaces of revolution by a 3D curve, which moves in space according to an euclidean motion. Using surfaces of revolution, this approach can elegantly describe domes and spires where approaches based only on plane fitting may fail. Further, due to an adjustable amount of regularization, different levels of detail are introduced.

Similar to the moving 3D curve, in [47] a parametric method for reconstructing architectural scenes from sparse point clouds is proposed. Profile curves are swept over a network of transport curves in order to generate swept surfaces. To recover fine details, a displacement map is applied. This approach works on sparse point clouds. Though, for too sparse regions in the reconstruction it may create holes and therefore does not deliver a watertight mesh of the whole scene.

All these methods reduce the amount of data by representing points with geometric representations. However, they do not introduce different levels of detail, for which a semantically enriched geometric model, would be needed.

In particular for indoor scenes, Xiao and Furukawa [48] introduce a system to automatically reconstruct and visualize 3D models for large indoor scenes using ground-level photographs and 3D laser points. A Constructive Solid Geometry (CSG) representation consisting of volumetric primitives (solids), which imposes powerful regularization constraints, is used to model the scene. First, they split up the 3D model into horizontal slices and process each slice individually in the 2D domain. All points of a slice are projected on an image plane and Hough Transform line detection is applied. With the lines as limitations, rectangles are fit in the image and an optimal union of rectangles is calculated, which delivers a 2D CSG model. Finally, having all the rectangles extruded to all slice boundaries as prototypes, an optimal 3D representation consisting out of a union of cuboids (the final 3D CSG model) is created. Figure 2.3 illustrates the processing chain. This approach massively simplifies the input data and creates a semantic description of

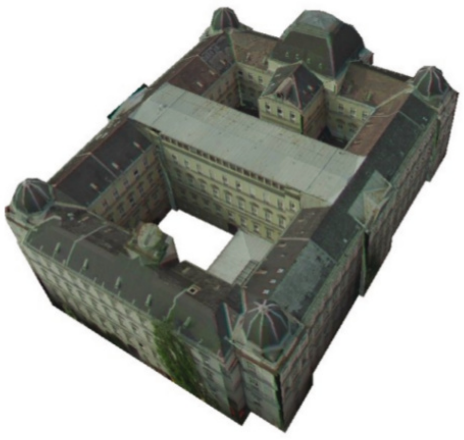


Figure 2.2: *Geometric representations from 3D data.* Zebedin et al. [50] model 3D scenes using planes and surfaces of revolution. In this figure, you can see a reconstruction of the old campus of Graz University of Technology. Image taken from [50].

the scene by creating an inside/outside labeling for each slice and finally merging the slices together while optimizing an objective function. However, as it is based on Hough Transform line detection and primitive fitting, it fails on image-based 3D reconstructions containing more clutter.

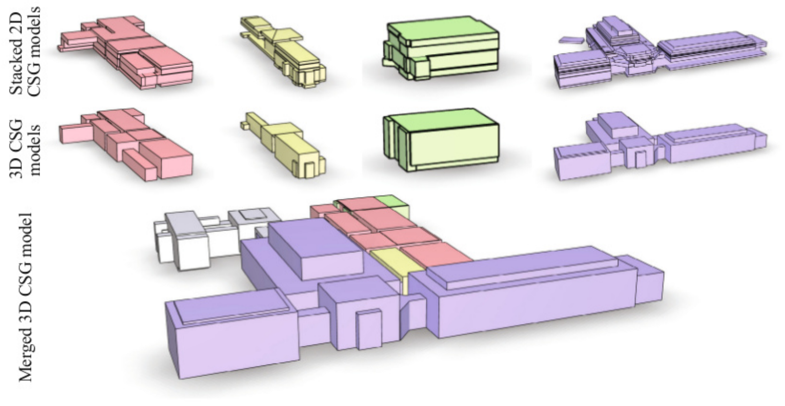


Figure 2.3: *CSG-based geometric representation.* To represent a point cloud with geometric structures, Xiao and Furukawa [48] first construct a 2D CSG model for each slice, which can be extruded to the slice boundaries (first row). Using the 2D information from all slices, they create an optimized 3D CSG model (second row). Finally, also multiple CSG models can be merged (third row). Image taken from [48].

An approach for multi-level indoor scenes (i.e., whole buildings) not limited to orthogonal structures is proposed in [37]. First, permanent structures (walls, floor, ceiling) are detected using horizontal slicing and wall directions are computed using Hough Transform. For every slice, a triangular decomposition is created and extruded to 3D to form so-called volumetric cells, which are right prisms with triangles as base faces. The volumetric cells are labeled as empty or occupied space using ray-casting and graph cut energy minimization. Similarly to [48], this approach creates a semantically enriched geometric representation of a building and introduces different levels of detail using laser scan data as input. However, this approach is also not tailored to image-based 3D reconstructions containing more noise and clutter.

2.3 Summary

In this chapter, we discussed various approaches how to transform scene information acquired by images, laser scanners or RGB-D data into geometric structures.

There exist methods which use a single RGB image for this task, taking into account, for example, line segments, junctions or physical properties. However, all these methods are restricted to a special problem and are not able to handle large scenes.

Other approaches use an RGB image with an additional depth channel. Having this additional information it is easier to find geometric structures. Though, as the information is limited to one view, it is also not possible to model large 3D scenes.

Finally, most of the state-of-the-art work uses a 3D point cloud as input and reconstructs geometry using plane fitting, parametric curves or fitting of pre-defined geometric structures. Having 3D input data, it is possible to compute a geometric representation of a whole scene. However, most of the work does not impose semantic constraints or uses laser scan data as input, which contains less noise and clutter than image-based reconstructions.

Our work is inspired by [48] and [37], where the scene gets partitioned into horizontal slices containing similar vertical structures. [37] introduces the concept of an irregular space partitioning into volumetric cells, which can be used to optimize the resulting geometric model using a Conditional Random Field (CRF). In contrary to these approaches, we use input models reconstructed from image data which contain much more clutter and noise than data obtained from laser scanners. This introduces many new problems for which we find good solutions.

Chapter 3

Theory and Background

Contents

3.1 Reconstruction of Point Clouds

There exist several different techniques to map real world scenes into point clouds. They deliver sparse (e.g. Structure from Motion) or dense (e.g. Laser Scanners) reconstructions, produce semi-dense point clouds out of sparse point clouds (e.g. Patch-based Multi-view Stereo) or deliver 2.5D representation, where an additional depth value is captured with an RGB image (e.g, RGB-D sensors like Microsoft Kinect). In this section, we discuss some of these methods.

3.1.1 Structure from Motion

The Structure from Motion (SfM) approach takes overlapping 2D images as input, calculates the pose of the cameras and reconstructs a sparse 3D point cloud by triangulating corresponding image points. Additionally, bundle adjustment is applied in order to optimize the reconstruction. In this section, we first discuss some preliminary needed knowledge and then explain how a sparse point cloud is reconstructed from an image set of overlapping images using SfM.

3.1.1.1 Camera Calibration

In the following explanations, we assume to have images taken with a camera with a known intrinsic camera calibration. The intrinsic calibration parameters define the internal camera properties and are defined as

$$K = \begin{pmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.1}$$

where f_x and f_y are the focal lengths in the direction of x and y, s is the skew factor and p_x and p_y denote the position of the principal point.

The extrinsic camera calibration defines the relative pose of a camera center. It is defined by a 3×3 rotation matrix R and a 3×1 translation vector t and can be described as

$$C = -R^T t. \tag{3.2}$$

Using the intrinsic and extrinsic camera calibration, it is possible to assemble the camera projection matrix P, which maps 3D points onto the image plane. The matrix P is defined as

$$P = K[R|t] \tag{3.3}$$

This camera representation is valid for a pinhole camera. However, as every camera usually has a lens system which, in practice, is not perfect, lens distortion parameters have to be taken into account. Usually, distortion is described as radial and tangential distortion. Radial distortion defines the "barrel" - effect of lenses, where straight lines are transformed into curves, while tangential distortion results from the lens not being parallel to the image plane.

For details please refer to [23]. Further, there also exist freely available implementations for performing camera calibration (e.g., a Matlab implementation can be found here [10])

3.1.1.2 Image Feature Correspondences

In order to calculate the pose of the cameras, feature correspondences between images have to be computed in advance. Therefore, for every image, salient points are detected and feature descriptors of this points are calculated. As requirement, the detector and descriptor algorithm has to be robust against translation, rotation and scale to a certain degree. Commonly used algorithm for this task are, for example, the SIFT [34] and the SURF [8] algorithm. Next, the feature descriptors of all images can be matched against each other. Figure 3.1 shows the salient points and matches of two images.

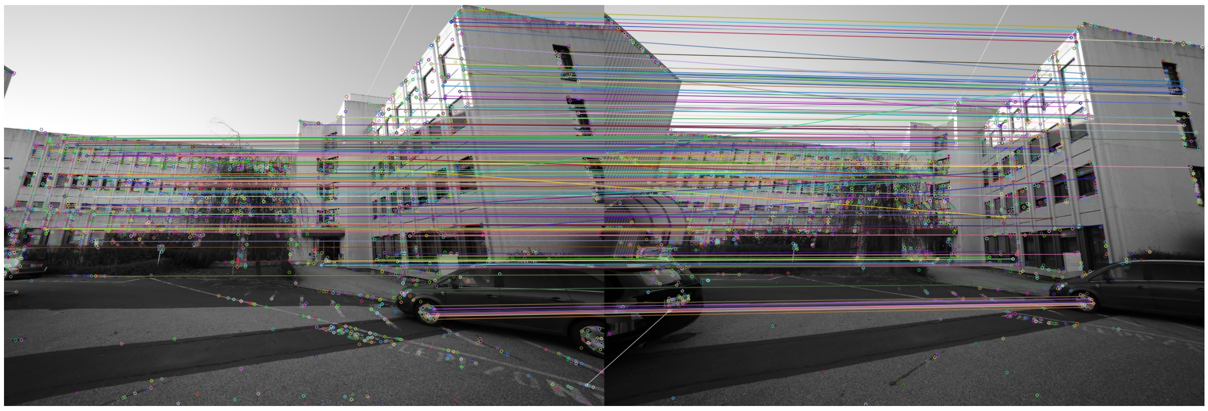


Figure 3.1: *Salient points and matches between two images.* In every image, salient points and feature descriptors are calculated using the SIFT algorithm. The corresponding feature descriptors are matched (connected with lines in this figure). As you can see, there exist some false matches, which can be eliminated using the Random Sample Consensus (RANSAC) algorithm in following steps.

3.1.1.3 Triangulation

Having two images with point matches and the intrinsic camera calibration, it is possible to reconstruct the 3D points of the matches. Epipolar geometry describes the relationship between the two cameras and the reconstructed 3D points (see Fig. 3.2). Knowing that the epipolar plane intersects the camera centers, the corresponding image points and the reconstructed 3D point and simultaneously knowing the intrinsic and extrinsic camera calibration, it is possible to reconstruct the 3D point using triangulation.

As we just know the intrinsic camera calibration, we still have to calculate the extrinsic calibration, which is the relative pose of the cameras. This is achieved by estimating the fundamental matrix F of the stereo setup. The fundamental matrix, which is a 3x3 matrix of rank 2 with 7 degrees of freedom, is the algebraic representation of the epipolar geometry and maps a point x in the first image to the corresponding epipolar line l' in the second image:

$$l' = Fx. \tag{3.4}$$

Further, two corresponding image points x and x' fulfill the following equation:

$$x'^T Fx = 0. \tag{3.5}$$

Though, it is also possible to retrieve the rotation matrix R and the translation vector t (the extrinsic calibration) from the fundamental matrix.

There exist various algorithms for calculating the fundamental matrix F (or their specialization, the essential matrix E) using point correspondences of two images. Depending on the algorithm, a different number of points is needed. For example, a simple algorithm for estimating the fundamental matrix is the normalized 8-point algorithm as described in Hartley and Zisserman [23]. Another method for estimating the essential matrix, assuming a calibrated camera setup, is the 5-point algorithm presented in Nister [36]. In comparison to the 8-point algorithm, this method is able to handle coplanar structures.

Having determined the extrinsic calibration using the fundamental matrix, it is possible to triangulate the image point correspondences and to reconstruct the 3D points.

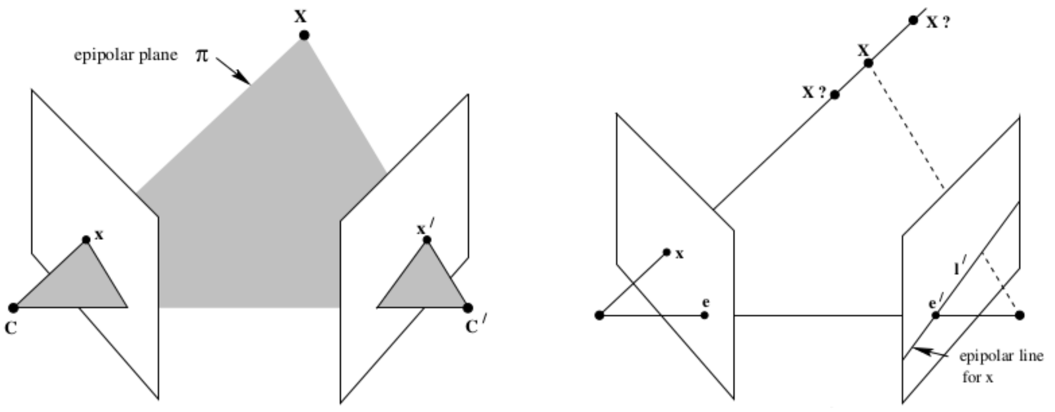


Figure 3.2: *Point correspondence geometry.* The epipolar plane π intersects with the two camera centers c and c', with the 3D point X and the corresponding image points x and x' (left image). For an image point x in the first image, all possible corresponding image points in the second image lie on the epipolar line l' (right image). Image taken from [23].

3.1.1.4 Multiview Reconstruction

Usually, SfM is not applied on just two camera views, but on sets with multiple images. As points can be seen in more than two cameras, we want to find the projection matrices

P^i for all involved cameras i and the 3D points X_i, given a set of image coordinates x^i_j, such that

$$P^i X_j = x^i_j. \tag{3.6}$$

As image measurements can be noisy, the equations $P^i X_j = x^i_j$ will not be satisfied exactly. Therefore, bundle adjustment is usually used to estimate the projection matrices $\hat{P}^i$ and 3D points $\hat{X}_j$. It computes a Maximum Likelihood solution assuming that the measurement noise is Gaussian. More information about bundle adjustment can be found in [23].

There exist several freely available SfM implementations. Bundler [43], for example, is a widely used SfM tool which has been proven to show good performance also on large-scale environments [5].

3.1.2 Dense Reconstruction

In many cases, the density of the point cloud computed with SfM is not sufficient in order to apply further processing (for example, to generate a mesh including the point cloud). Therefore, the generated point cloud usually gets densified. This results in a dense point cloud approximating the same scene as the SfM model.

3.1.2.1 Patch-based Multi-view Stereo

PMVS is a widely used method to generate semi-dense point clouds out of sparse point clouds. The algorithm, introduced by Furukawa and Ponce [19], reconstructs a set of oriented points (i.e. points with their corresponding normals) covering the surface of an object or a scene of interest.

A patch p, which is a set of oriented points, is a local tangent plane approximation of a surface. It is a rectangle defined by its center $c(p)$, its unit normal vector $n(p)$ oriented towards the cameras observing it and a reference image $R(p)$ in which it is visible. More concretely, one edge of the rectangle is parallel to the x-axis of the reference camera (the camera associated with $R(p)$). Figure 3.3 illustrates a patch p.

A core part of the algorithm is the photometric discrepancy function $g(p)$. It is defined as

$$g(p) = \frac{1}{|V(p) \setminus R(p)|} \sum_{I \in V(p) \setminus R(p)} h(p, I, R(p)), \tag{3.7}$$

where $V(p)$ denote a set of images in which p is visible, and $h(p, I_1, I_2)$ denotes a pairwise photometric discrepancy function between images I_1 and I_2, which computes a discrepancy score between the pixel color values of p projected onto I_1 and I_2 using normalized cross correlation. The goal of the algorithm is to recover patches whose discrepancy scores are small.

As the discrepancy function may not work well in the presence of specular highlights or obstacles (e.g., pedestrians in front of buildings), further optimization steps are applied. For a detailed description look at [19].

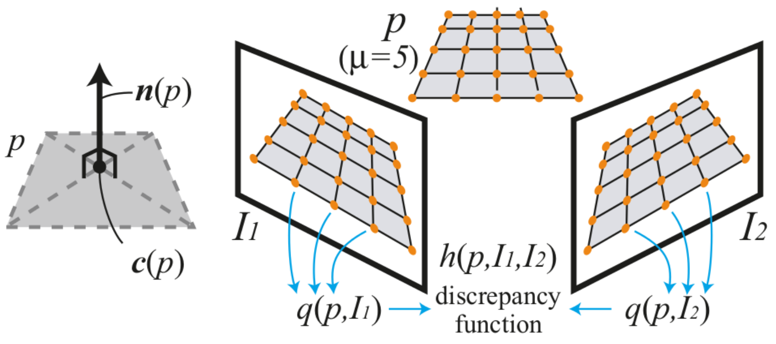


Figure 3.3: *A patch and the photometric discrepancy function.* A patch is a rectangle in 3D space with its center and normal denoted as $n(p)$ and $c(p)$ (left). The photometric discrepancy function $h(p, I_1, I_2)$ takes into account the pixel values of the patch p projected onto the images I_1 and I_2 (right). Image taken from [19].

3.2 From Points to Meshes

Using the generated point cloud, an infinite surface gets generated including the generated points. Such a surface is represented as mesh consisting of triangles. There exist several meshing techniques with different requirements that deliver the surface mesh as result. In this section, we discuss two well-known surface reconstruction approaches in more detail.

3.2.1 Poisson Surface Reconstruction

The Poisson surface reconstruction technique, as proposed by Kazhdan et al. [27], uses a point set with its corresponding inward-facing normals as input data. The points are assumed to be uniformly distributed over the model surface. As a result, it delivers a watertight, triangular mesh.

This approach is based on the spatial Poisson problem. The Poisson equation is a partial differential equation of elliptic type originally used in electrostatics, mechanical

engineering and theoretical physics.
Kazhdan et al. [27] suggest to compute a 3D *indicator function* χ and then reconstruct the surface by extracting an appropriate isosurface. The indicator function χ is defined as 1 at points inside the model, and 0 at points outside. Their key contribution is the relationship between oriented points (i.e. points with given normals defined as vector field $\vec{V}$) sampled from the surface of a model and the indicator function of the model. More precisely, they analyze the gradient of the indicator function which is zero almost everywhere, except near the surface of the model (Figure 3.4).

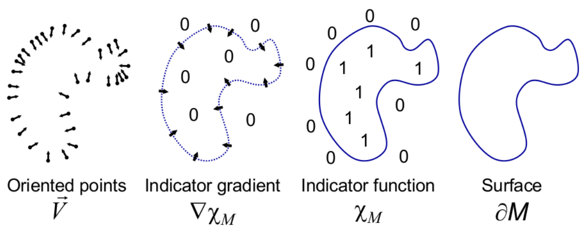


Figure 3.4: Using the oriented points, the indicator gradient and subsequently the indicator function can be calculated. Finally, the surface is reconstructed by extracting an appropriate isosurface. Image taken from [27].

The Poisson reconstruction creates a watertight, triangulated surface by approximating the indicator function. The key challenge, the computation of the indicator function, can be done by utilizing the relationship between the gradient of the indicator function and the integral of the surface normal field.

The gradient field convolved with a smoothing filter $\tilde{F}$ is defined as

$$\nabla(\chi_M * \tilde{F})(q_0) = \int_{\partial M} \tilde{F}_p(q_0)\vec{N}_{\partial M}(p)dp. \tag{3.8}$$

χ_M is the indicator function of a solid M with boundary ∂M. The smoothing filter $\tilde{F}$ is introduced to avoid unbounded values at the surface boundary due to the piecewise constant indicator function. $\vec{N}_{\partial M}(p)$ is the inward surface normal at $p \in \partial M$, and $\tilde{F}(q) = \tilde{F}(q-p)$ is the translation of the smoothing filter $\tilde{F}_p(q_0)$ to the point p. A proof of this relationship can be found in [27].

As the surface geometry is not known until now, we cannot evaluate the surface integral. However, using the information of the input set of oriented points, it is possible to approximate the integral with a discrete summation. This is defined as follows:

$$\begin{aligned} \nabla(\chi_M * \tilde{F})(q_0) &= \sum_{s \in S} \int_{\mathcal{P}_s} \tilde{F}_p(q) \vec{N}_{\partial M}(p) dp \\ &\approx \sum_{s \in S} |\mathcal{P}_s| \tilde{F}_{s.p}(q) s.\vec{N} \equiv \vec{V}(q). \end{aligned} \tag{3.9}$$

Using the point set S to partition the solid boundary ∂M in patches $\mathcal{P}_s \subset \partial M$, we can approximate the integral of the patch $\mathcal{P}_s$ by scaling the value of the point sample $s.p$ to the area of the patch.

Subsequently, we want to solve $\tilde{\chi}$ such that $\nabla \tilde{\chi} = \vec{V}$. As $\vec{V}$ is generally not integrable (i.e. it is not curl free), an exact solution does not generally exist. However, a least-squares solution can approximate the integral. For this, we apply the divergence operator to form the Poisson equation

$$\Delta \tilde{\chi} = \nabla \cdot \vec{V}. \tag{3.10}$$

For more information on how to solve this Poisson equation, look at [27].

Finally, the isosurface can be extracted using, for example, a Marching Cubes method as described in [33].

3.2.2 Surface Reconstruction Using Delaunay Triangulation

The Delaunay triangulation approach, in contrary, requires a point set with its corresponding camera information as input. It decomposes the point cloud into cells in the shape of tetrahedra and calculates the visibility information for each cell using the camera information. There exist approaches for (semi-)dense point clouds (Labatut et al. [30]) and also for sparse point clouds (Hoppe et al. [26]). As a result, these techniques deliver a watertight, triangular mesh.

An irregular discretization of the space is created using a Delaunay Triangulation of the 3D points. With such a discretization, the size of the discretized units is related to the density of the underlying point cloud and can be efficiently adapted to new 3D information [26]. The surface gets approximated as the interface between free and occupied cells of the tetrahedra decomposition.

3.2.2.1 Delaunay Triangulation

Looking at a point set $\mathcal{P} = p_1, ..., p_n$ in $\mathbb{R}^d$, the Voronoi cell associated to a point p_i, denoted by $V(p_i)$, is the the space that is closer to p_i than to any other point in $\mathcal{P}$. The

Voronoi diagram, denoted by Vor($\mathcal{P}$), is the partition of space induced by the Voronoi cells $V(p_i)$.

The Delaunay triangulation Del($\mathcal{P}$) of the point set $\mathcal{P}$ is defined as the counterpart of the Voronoi diagram. All points p and q with a non-empty intersection of their Voronoi cells $V(p)$ and $V(q)$, have a connecting edge in the Delaunay triangulation. Figure 3.5 illustrates a point set with its corresponding Voronoi diagram and Delaunay triangulation. For more details look at [9].

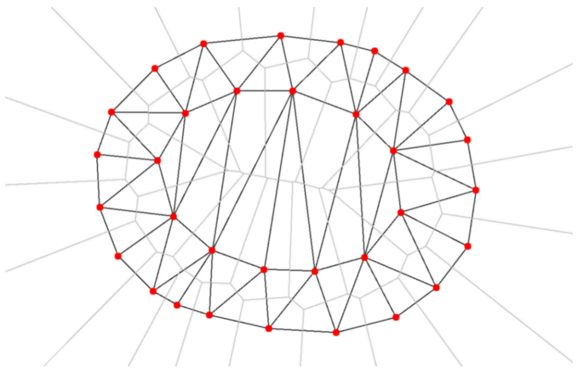


Figure 3.5: The Voronoi diagram (gray edges) of a set of 2D points (red dots) and its associated Delaunay triangulation (black edges). Image taken from [30].

The algorithmic complexity of the Delaunay triangulation of n points is $\mathcal{O}(n \log n)$ in 2D and $\mathcal{O}(n^2)$ in 3D. However, it has been proven that the complexity in 3D drops to $\mathcal{O}(n \log n)$ when the points are distributed on a smooth surface, which is the case in our application for surface reconstruction [30].

3.2.2.2 Reconstruction Method

There exist different approaches on how to reconstruct a surface using a Delaunay triangulation. However, we just focus on the method described in Labatut et al. [30].

Assuming a quasi-dense input cloud, a Delaunay triangulation consisting of tetrahedrons can be incrementally built from the 3D point cloud. Consequently, all the tetrahedrons are labeled either as inside or outside of the scene. The triangular faces between adjacent tetrahedra having different labels build the output triangular mesh.

A global optimal label assignment is efficiently found using graph cuts (an explanation of graph cuts can be found in 3.3.1). [30] constructs an energy function consisting of the following terms:

$$E(S) = E_{vis}(\mathcal{S}) + \lambda_{photo}E_{photo}(\mathcal{S}) + \lambda_{area}E_{area}(\mathcal{S}). \tag{3.11}$$

$\mathcal{S}$ is the surface to be reconstructed, E_{photo} is the photo consistency term, which measures how well the given surface $\mathcal{S}$ matches the different input images in which it is seen, and E_{area} is the term which encourages surface smoothness. Both have their corresponding positive weights, λ_{photo} and λ_{area}. Finally, the visibility term $E_{vis}(\mathcal{S})$ defines every tetrahedron as inside or outside of the scene. The basic idea behind this term can be described as follows: If a vertex belongs to the final surface, it should be visible in the view it comes from. Therefore, all the tetrahedra intersected by a ray going from the vertex to the camera center of one of these views should be labeled as outside. Consequently, the tetrahedron behind the vertex should be labeled as inside. For detailed information about this approach look at [30].

3.3 Energy Minimization via Graph Cuts

As for the surface reconstruction problem in the previous section, graph cuts can be used for many applications in computer vision to efficiently solve energy minimization problems.

A frequent early computer vision problem is the assignment of a label to each pixel of an image. Common constraints for this task are smoothly varying labels with a simultaneously sharp discontinuities preserving labeling. These problems can be expressed in terms of energy minimization [11]. However, as minimizing an arbitrary non-regular function is NP-hard [29], the problem is difficult regarding computation time. Using graph cuts, it is possible to find a global minimum for a binary labeling problem and a local minimum, which is in within a known factor of the global minimum, for multi-label problems [11].

3.3.1 The Energy Minimization Problem

As described in Boykov et al. [11], many early computer vision problems include estimating a spatially varying quantity (such as intensity or disparity) from a noisy measurement. Generally, such quantities are piecewise smooth: they vary smoothly on the surface of an object, but change dramatically at object boundaries. Every pixel $p \in \mathcal{P}$ must be assigned a label. We want to find a labeling f that assigns each pixel $p \in \mathcal{P}$ a label $f_p \in \mathcal{L}$, where f is both piecewise smooth and consistent with the observed data.

These problems can be naturally described as energy minimization problems

$$E(f) = E_{smooth}(f) + E_{data}(f). \tag{3.12}$$

$E_{smooth}(f)$ is the smoothness term, which describes the extend to which f is not piecewise smooth. $E_{data}(f)$ is the data term describing the disagreement between f and the observed data.

The form of $E_{data}(f)$ is typically

$$E_{data}(f) = \sum_{p \in \mathcal{P}} D_p(f_p), \tag{3.13}$$

where D_p measures how appropriate a label is for the pixel p given the observed data.

The selection of $E_{smooth}(f)$ is a critical issue, and many different approaches exist [11]. We just focus on the smoothness term

$$E_{smooth}(f) = \sum_{p,q \in \mathcal{N}} V_{p,q}(f_p, f_q), \tag{3.14}$$

as described in [11] and [29]. $\mathcal{N} \subset \mathcal{P} \times \mathcal{P}$ is a neighborhood system on pixels and $V_{p,q}(f_p, f_q)$ measures the cost of assigning the labels f_p, f_q to the adjacent pixels p, q. On the one hand, this function is used to impose spatial smoothness, on the other hand, it should preserve sharp structures, as pixels at the border of objects should often have very different labels. This requires that V be a non-convex function of $|f_p - f_q|$. Such an energy function is called *discontinuity-preserving* [29].

Energy functions like E are extremely difficult to optimize, as they are non-convex functions in a space with many thousands of dimensions. General-purpose optimization techniques (such as simulated annealing) traditionally have been used to minimize these problems. However, such techniques require exponential time and are extremely slow in practice. Using graph cuts it is possible to solve this problems efficiently [29].

3.3.2 Graph Cuts

Kolmogorov and Zabih [29] describe graph cuts as follows: Looking at a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consisting of vertices ($\mathcal{V}$) and edges ($\mathcal{E}$) with nonnegative edge weights and special vertices (terminals), called the source s and the sink t. The two terminals are vertices with special properties: The source s is a vertex with zero in-degree and the sink t is a vertex with zero out-degree. This graph can be partitioned into two disjoint sets S

and T such that $s \in S$ and $t \in T$. Such a cut $C = S, T$ is called s-t-cut and is a binary labeling of the graph. The cost of the cut is the sum of costs of all edges that go from S to T:

$$c(S,T) = \sum_{u \in S, v \in T, (u,v) \in \mathcal{E}} c(u,v) \tag{3.15}$$

The minimum s-t-cut problem is to find a cut C with the smallest cost. As this problem is equivalent to computing the maximum flow from the source to the sink, it can be solved efficiently by computing the maximum flow between the terminals, according to a theorem due to Ford and Fulkerson [17].

3.3.3 Energy Minimization Using the Expansion Move Algorithm

A very effective algorithm for energy minimization using graph cuts is the expansion move algorithm introduced in [11]. It can be used for several important discontinuity-preserving energy functions where V is a metric on the space of labels.

An expansion move is defined as follows: Consider a labeling f and a label α. A new labeling f' is an α-expansion move from f if $f'_p \neq \alpha$ implies $f'_p = f_p$. This means that the set of pixels assigned the label α has increased when going from f to f' [29]. Figure 3.6 shows an example of an α-expansion move.

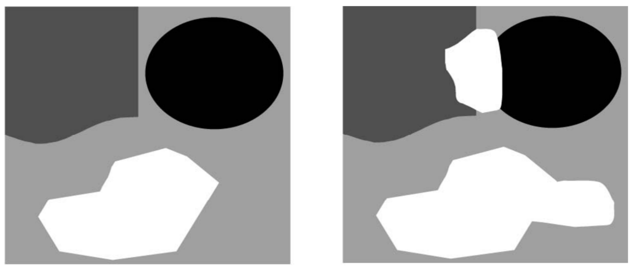


Figure 3.6: An example of an expansion move: The right image is a white expansion move from the left image. Figure taken from [29].

The algorithm finds the lowest energy α-expansion move from the current labeling by cycling through the labels α in a fixed or random order. If the expansion move has lower energy than the current labeling, it becomes the current labeling. The algorithm terminates at the local minimum, which means, that for any label α, there is no α-expansion move from the current labeling that has lower energy than the current one. It can be proven that this minimum is within a multiplicative factor of the global minimum [11] [29].

3.4 Mean Shift

Another commonly used algorithm in computer vision is the Mean Shift algorithm. Mean Shift is a non-parametric mode estimation algorithm whose main application is to cluster data without knowledge about the distribution of the data and the amount of clusters. The algorithm was proposed in 1975 by Fukunaga and Hostetler [18]. Especially in Computer Vision, Mean Shift is commonly used for tasks like segmentation, tracking or discontinuity preserving smoothing, as described by Comaniciu and Meer [12].

3.4.1 Intuitive Explanation

Mean Shift considers the feature space as a probability density function. The local maxima of the function are the modes of the data.

The algorithm is an iterative procedure consisting of following steps:

- For each data point, define a surrounding window with fixed size.
- Find the centroid of all points in the window and recenter the window at the centroid. This shift is defined by the Mean Shift vector $m_{h,G}(x)$.
- Repeat until convergence.

The single parameter is the window size, called bandwidth. Figure 3.7 illustrates the iterative Mean Shift process.

3.4.2 The Mean Shift Procedure

In this section, we discuss the technical details of the algorithm, like described in Comaniciu and Meer[12]. As we want to find local maxima of a probability density function, we have to estimate a density function from the data points. Kernel density estimation (known as the Parzen window technique in pattern recognition literature) is the most popular density estimation method. Given a radially symmetric kernel K, bandwidth parameter h and a set of d-dimensional points, the kernel density estimator is defined as

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right). \tag{3.16}$$

We are interested only in a special class of radially symmetric kernels satisfying

$$K(x) = c_k k\left(\|x\|^2\right), \tag{3.17}$$

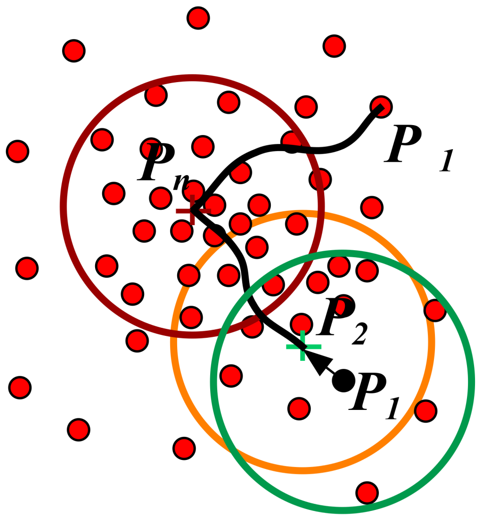


Figure 3.7: Principle of Mean Shift analysis (taken from [13]): To find the cluster center of a point P_1, iteratively find the centroid of the data points within a window around P_1 and recenter the window on the centroid. Repeat this procedure until the window is stationary. This is an adaptive gradient ascent in the space of point densities.

in which case it suffices to define the function $k(x)$ called the profile of the kernel, only for $x \geq 0$. The normalization constant $c_{k,d}$, which makes $K(x)$ integrate to one, is assumed strictly positive.

Employing the profile notation of the Epanechnikov kernel (see [12]), the density gradient estimator is obtained as the gradient of the density estimator. The gradient can be written as

$$\hat{\nabla} f_{h,K}(x) \equiv \nabla \hat{f}_{h,K}(x) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (x - x_i) k' \left(\left\| \frac{x - x_i}{h} \right\|^2 \right). \tag{3.18}$$

We define the function

$$g(x) = -k'(x), \tag{3.19}$$

assuming that the derivative of k exists for all $x \in [0, \inf)$, except for a finite set of points. Using $g(x)$ for profile, we define the kernel $G(x)$

$$G(x) = c_{g,d}\, g\left(\|x\|^2 \right) \tag{3.20}$$

Introducing $g(x)$ into 3.18 leads to

$$\nabla \hat{f}(x) = \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right) \right] \left[\frac{\sum_{i=1}^{n} x_i g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right)}{\sum_{i=1}^{n} g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right)} - x \right]. \tag{3.21}$$

The first term is proportional to the density estimate at x computed with the kernel G

$$\hat{f}_{h,G} = \frac{c_{g,d}}{nh^d} \sum_{i=1}^{n} g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right). \tag{3.22}$$

The second term is the *Mean Shift*. It is the difference between the weighted mean, using the kernel G for weights, and x, the center of the kernel (window)

$$m_{h,G} = \frac{\sum_{i=1}^{n} x_i g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right)}{\sum_{i=1}^{n} g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right)} - x. \tag{3.23}$$

Using 3.22 and 3.23, 3.21 becomes

$$\hat{\nabla} f_{h,K} = \hat{f}_{h,G}(x) \frac{2c_{k,d}}{h^2 c_{g,d}} m_{h,G}(x), \tag{3.24}$$

which yields to

$$m_{h,G}(x) = \frac{1}{2} h^2 c \frac{\hat{\nabla} f_{h,K}(x)}{\hat{f}_{h,G}(x)}. \tag{3.25}$$

The Mean Shift vector $m_{h,G}(x)$ points toward the direction of maximum increase in density and is proportional to the density gradient estimate at point x obtained with kernel K. Iteratively shifting the density estimation window along the Mean Shift vector makes the algorithm converge at local maxima, i.e. where $\nabla f(x_i) = 0$ is satisfied. For a proof of convergence, see Comaniciu and Meer[12].

3.5 Summary

In this chapter, we discussed required background knowledge and methods applied in our approach. We assume to have meshed point clouds reconstructed from image data (as described in Section 3.1 and 3.2) as input for our workflow. For mode estimation used for the partitioning of the model into horizontal slices, we use Mean Shift (as described in Section 3.4). For the optimization in the slice domain and for the final 3D optimization, we use energy minimization via graph cuts (as described in Section 3.3).

Chapter 4

Geometric Abstraction with Horizontal Slicing

Contents

4.1 Overview

Given a meshed point cloud and the corresponding camera poses C as input, we want to extract relevant geometric structures of the scene. As precondition, we assume that the scene is dominated by planar canonical structures. We apply preprocessing steps to transform the model to a new coordinate system aligned with a reference plane and perform a predefined, constant upsampling of the meshed point cloud. At dominant horizontal structures, we partition the model into horizontal slices. A slice represents a part of the model which includes mainly vertical structures and is bounded by horizontal structures. For each slice, we create an inside/outside labeling, based on the visibility information which results in a 2D floor plan. The individual floor plans are used to create an irregular discretization of the volume into cells. Finally, we obtain a regularized 3D model by labeling each cell as inside/outside using a CRF. As result, we generate a 3D model consisting of geometric primitives.

In the following sections, the processing steps will be discussed in detail. Figure 4.1 illustrates the different modules of our processing workflow.

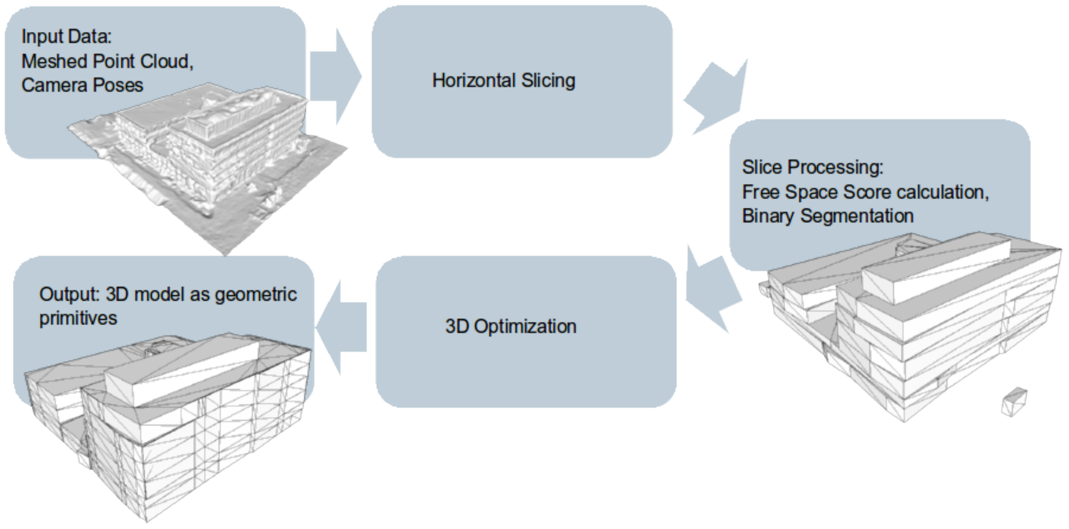


Figure 4.1: *Overview of our processing workflow.* The input mesh gets partitioned into horizontal slices which are parts of the model limited by dominant horizontal structures and containing similar vertical structures. For every slice, an optimal inside/outside segmentation based on the visibility information gets created resulting in a floor plan for each slice. Finally, a 3D optimization is performed by solving an energy minimization problem using an irregular partitioning of the scene based on the individual floor plans.

4.2 Slicing and Boundary Segmentation

In this section, we describe the processing of the point cloud in the slice domain. First, the upsampled point cloud model gets partitioned into horizontal slices. A slice includes similar vertical structures (for example, vertical walls) and is bounded by horizontal structures. For each slice, we calculate an optimal 2D binary segmentation identifying parts inside or outside of the object. The segmentation is performed by using the visibility information that is associated with the input mesh. The polygonal line defined by the transition from one segment to the other results in a floor plan.

4.2.1 Slice Extraction

In our workflow, similar to [48] and [37], slices are defined as parts of the model enclosed by dominant horizontal structures. Such structures might be, for example, a horizontal roof or different levels of a building.

We find horizontal structures by projecting points with normals similar to the ground plane normal (i.e. similar to the z-axis) onto the z-axis. On this one-dimensional data (only the z-coordinate of each point is used), we apply mode estimation using Mean Shift. Figure 4.2 illustrates this process. We use the centers of the modes as boundaries of the slices. By adjusting the bandwidth parameter of Mean Shift, it is possible to generate slice boundaries at all minor horizontal structures or just at the most important dominant structures. The bandwidth parameter is calculated as a fraction of the model height. This leads to independence of the model size and just the denominator d has to be defined:

$$bandwidth = \frac{height}{d}. \tag{4.1}$$

Figure 4.3 shows the slice boundaries with different values of d. The Mean Shift bandwidth parameter d is one of the two most important parameters in our workflow. It defines the level of detail in the vertical direction.

Our approach is inspired by the approach in Oesau et al. [37], where also Mean Shift is used to detect slice boundaries.

We tried different techniques to find slice boundaries. For example, we implemented the method proposed in Xiao and Furukawa [48]. In their work, they create a histogram of the number of 3D points in the gravity direction and convolve the histogram with a Gaussian smoothing operator. The peaks of the histogram are defined as the slice boundaries. Though, as we found out, the Mean Shift approach is much more stable in terms of varying model properties.

4.2.2 Binary Segmentation

For every slice, we calculate a pixelwise binary labeling using the visibility information in order to get a probability for every pixel for inside or outside of the object. Finally, we create an optimized labeling by solving the problem as an energy minimization problem. The discontinuity between differently labeled regions is then the resulting 2D floor plan.

4.2.2.1 Free Space Score

In order to calculate a binary labeling for a slice, we use the visibility information for each position in the slice projected onto a 2D image matrix. We call this information free space score, as it is the evidence for every pixel to be in free space (outside the object) or in occupied space (inside the object).

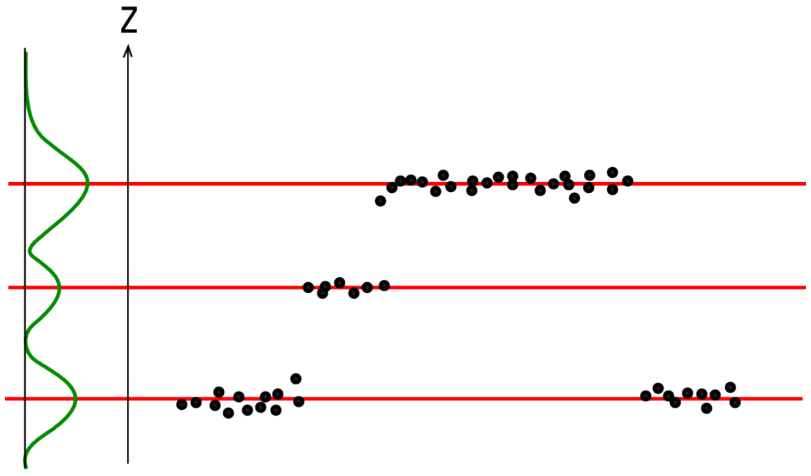
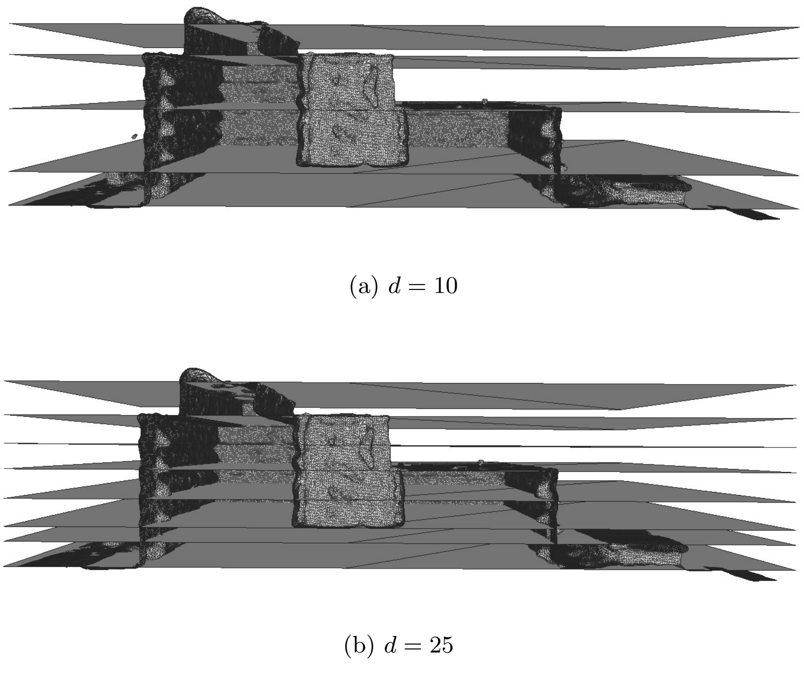


Figure 4.2: Points on faces with normals similar to the ground plane (black points) get projected onto the z-axis. On this one-dimensional data (green line), we perform Mean Shift mode estimation. The mode centers (red lines) are used as slice boundaries.



(a) $d = 10$

(b) $d = 25$

Figure 4.3: *Vertical cut of model and slice boundaries (gray planes) calculated with Mean Shift.* In 4.3(a), the horizontal slicing parameter d is set to 10. The Mean Shift bandwidth is high and slice boundaries are just created at very dominant horizontal structures. In 4.3(b), d is set to 25. The bandwidth is low and slice boundaries are also created at small horizontal structures.

To calculate this information, we create a free space score voxel grid spanned over the whole scene. Every voxel gets assigned a score for free and occupied space calculated from the visibility information. All voxels contained in a slice get projected onto an image matrix which will be used for further processing.

Our free space score computation is based on the approach proposed in [48].

Visibility Testing

In order to get a correct inside/outside labeling of each voxel, we want to find out if it is visible in one or more camera views. Therefore, we need to perform visibility testing for every voxel. Using CGAL's AABB tree structure [6], it is possible to do the visibility tests in a performant way.

Free Space Score Voxel Grid

For each voxel v_{xyz} in the grid V, we calculate a free space score, which is an evidence for inside or outside labeling. The free space score is calculated as follows:

For each voxel $v_{xyz} \in V$, we count the number of cameras a certain voxel is visible in. Therefore, we cast rays from each voxel v_{xyz} to all camera centers C. If a ray from v_{xyz} to a camera $c \in C$ does not intersect the input mesh, v_{xyz} is visible in c. The score for v_{xyz} is in free space is defined as

$$p(v_{xyz} = free|visibility) = \frac{\{\#\ cameras\ v_{xyz}\ is\ visible\ in\}}{\{max\ \#\ visible\ cameras\}}, \tag{4.2}$$

where $\{max\ \#\ visible\ cameras\}$ is the maximum number of visible cameras a voxel $v_{xyz} \in V$ contains.

For all the voxels $v_{xyz} \in V$ which have not been visible in any camera view, we define the score that v_{xyz} is in occupied space by calculating the distance of v_{xyz} to the next voxel v'_{xyz} that is in free space, i.e. $p(v'_{xyz} = free|visibility) > 0$:

$$p(v_{xyz} = occupied|visibility) = \frac{\min(dist(v_{xyz}, v'_{xyz}), maxDist)}{maxDist}, \tag{4.3}$$

where $dist(\cdot)$ calculates the Euclidean distance between the voxel centers and $maxDist$, which is a predefined maximum distance, truncates this distance. Hence, this formula is closely related to the truncated signed distance function [49] which is used for example in surface extraction algorithms.

As we aim to have a free space score between -1 and 1 for each voxel, we set the inside probability as negative score and the outside probability as positive score.

$$score(v) = \begin{cases} p(v_{xyz} = free|visibility) & \text{if } v = visible \\ p(v_{xyz} = occupied|visibility)(-1) & \text{if } v \neq visible \end{cases} \tag{4.4}$$

Figure 4.4 illustrates the ray cast used for free space score calculation.
Further normalization will be executed depending on the application domain.

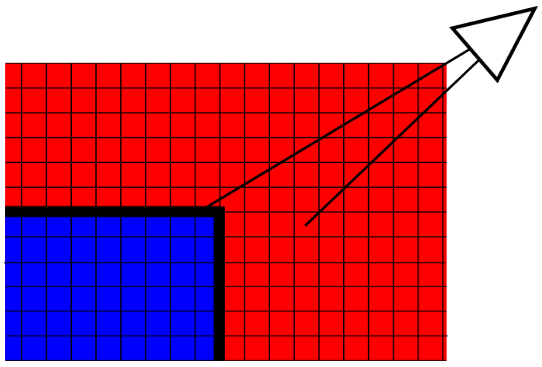


Figure 4.4: *Ray cast from camera to voxels.* The red voxels in front of the surface can be seen in the camera view and get assigned a positive free space score. The blue voxels behind the surface cannot be seen and get assigned a negative free space score.

In the slice domain, given the free space scores for each voxel, we can easily define the scores for each pixel b_{xy} in the 2D slice plane by averaging the scores of the voxels:

$$p(b_{xy} = free|visibility) = \sum_z \frac{p(v_{xyz} = free|visibility)}{n}, \tag{4.5}$$

where n is the voxel dimension in z-direction of the slice. The score that a pixel is occupied is defined in the same way.

To calculate the visibility information of each voxel, we use the Computational Geometry Algorithms Library (CGAL) to perform visibility tests from each camera to each voxel center.

CGAL is an open-source C++library with the goal to provide easy access to efficient and reliable geometric algorithms [1]. With the AABB (axis-aligned bounding box) tree component in CGAL, it is possible to perform efficient intersection and distance queries against sets of finite 3D geometric objects [6]. The AABB tree data structure first converts the geometric input data into primitives, and then creates a hierarchy of axis-aligned bounding boxes out of them. This data structure is used to speed up queries.

Figure 4.5 shows free space scores from one slice projected to 2D. Though, as will be discussed later, the free space score is not just used for every slice separately, but also for the final 3D regularization of our workflow.

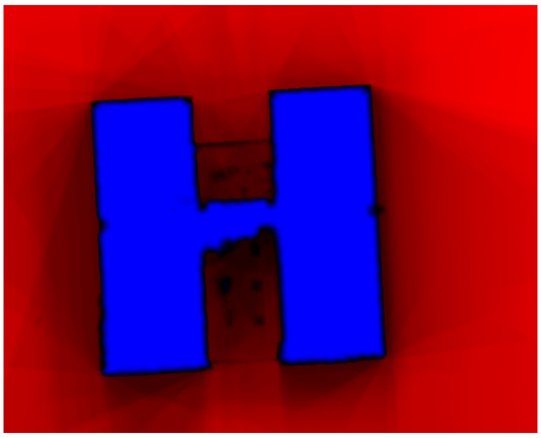


Figure 4.5: *Projected free space score of a slice.* All voxels within a slice are projected onto a 2D matrix. Every pixel gets a score for being inside and outside, depending on the sum of all corresponding voxels (blue means higher inside, red higher outside score). Using this information, a binary segmentation of the slice is calculated.

4.2.2.2 Binary Labeling as an Energy Minimization Problem

With the projected free space score as input, we want to calculate a pixelwise binary labeling for inside and outside. As we want to regularize the geometric structure, we favor a smoothly varying labeling. Simultaneously, as we aim to keep important details, we want to preserve sharp discontinuities. Solving this problem as an energy minimization problem using graph cuts can efficiently fulfill those requirements.

To set up the energy minimization problem, we define the regular pixel grid to be a graph and every pixel to be a graph node. Every node has edges to its adjacent pixel nodes. Hence, neighboring pixels in the image are also neighboring nodes in the graph.

The energy which has to be minimized can be expressed as

$$E(L) = \sum_{p \in \mathcal{I}} E_{data}(L(p)) + \sum_{p,q \in \mathcal{N}} E_{smooth}(L(p), L(q)), \tag{4.6}$$

where $\mathcal{I}$ denotes the set of pixels in the image, $\mathcal{N}$ is the 4-neighborhood of every pixel, and L is the (binary) labeling

The *data terms* E_{data} are set according to the free space scores for each pixel for free (outside) and occupied (inside) space.

$$E_{data}(l_p) = \begin{cases} p(b_{xy} = free|visibility) & \text{if } l_p = outside \\ p(b_{xy} = occupied|visibility) & \text{if } l_p = inside \end{cases} , \tag{4.7}$$

where $p(b_{xy} = free|visibility)$ and $p(b_{xy} = occupied|visibility)$ are the free space scores for each pixel as defined in Equation 4.5 and l_p is the label of the corresponding pixel p.

The *smoothness terms* E_{smooth} for the transitions from one to another label are set to be constant, as we do not want do introduce different smoothness properties in this processing step. Such a configuration is called a Markov Random Field (MRF), in contrast to an Conditional Random Field (CRF), where smoothness values for neighbors are set individually.

E_{smooth} is defined as

$$E_{smooth}(l_p, l_q) = \begin{cases} 0 & \text{if } l_p = l_q \\ 1 & \text{else} \end{cases} . \tag{4.8}$$

To compute a solution for the energy minimization problem in a performant way, we solve it using graph cuts. Figure 4.6(a) shows the result of the binary labeling using graph cut energy minimization.

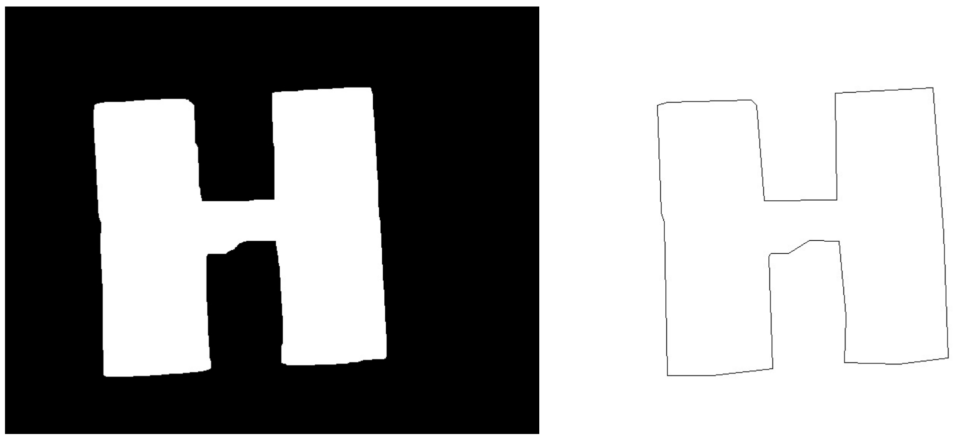
4.2.2.3 Outline Simplification

For the creation of the geometric 3D model, we just need the outline of the inside-labeled pixels. Though, as the previously calculated segmentation delivers a pixel-wise border, we apply an additional regularization step to retrieve a polygonal outline resulting in a floor plan.

We chose to use the *Ramer-Douglas-Peucker algorithm* [15] for this task. However, there is no need to use exactly this algorithm. It can be replaced by more sophisticated methods like proposed in Heber et al. [24].

The goal of the Ramer-Douglas-Peucker algorithm is to reduce the number of points in a curve that is approximated by a series of points. In our case, we have a curve point at every pixel and want to simplify this curve (i.e. reduce the amount of curve points) while simultaneously preserving the geometry of the curve.

The algorithm works globally with a predefined distance dimension $\epsilon > 0$. In the first iteration it creates a line hypothesis including just the beginning and the end point. At



(a) Binary Labeling (b) Simplified Outline

Figure 4.6: *Binary labeling and outline simplification of a slice.* Using graph cut energy minimization and the projected free space scores, a binary labeling for inside and outside is calculated. The result can be seen in 4.6(a). On the binary labeling, outline simplification is applied. As a result, we get a polygonal line as in 4.6(b).

each iteration, it finds the point with the maximum distance to the line hypothesis. If this point has a smaller distance than ϵ, the algorithm aborts and the current line hypothesis is the result. If the point has a bigger distance than ϵ, the point is added to the line hypothesis and the algorithm continues with the next iteration.

The algorithm produces a polygonal line as in fig. 4.6(b), which can be easily extruded to 3D.

4.3 Slice Combination

In comparison to 3D models generated from laser scanner data, 3D models reconstructed from image data tends to contain much more noise and clutter. Therefore, extruding the 2D slice segmentations to 3D does not lead to a sufficiently regularized geometric model. We apply a further regularization step by partitioning the whole possible occupied space into irregular shaped volumetric cells and create an optimal inside/outside labeling using energy minimization.

4.3.1 Extrusion to 3D

We generate an initial geometric representation of the point cloud by extruding the floor plans of every slice to the slice boundaries. An example for an extruded model is shown in Figure 4.7. With this method, most of the inside-labeled space, i.e. the space occupied by objects, is covered by geometric structures. However, as we just regularized within every slice separately, edges can arise due to noise. Further, small irregularities that just occur in one slice are not necessarily wanted to be in the geometric model. Therefore, this simple method of creating a geometric model does not fulfill all of our requirements. We need further regularization surpassing the boundaries of the slices.

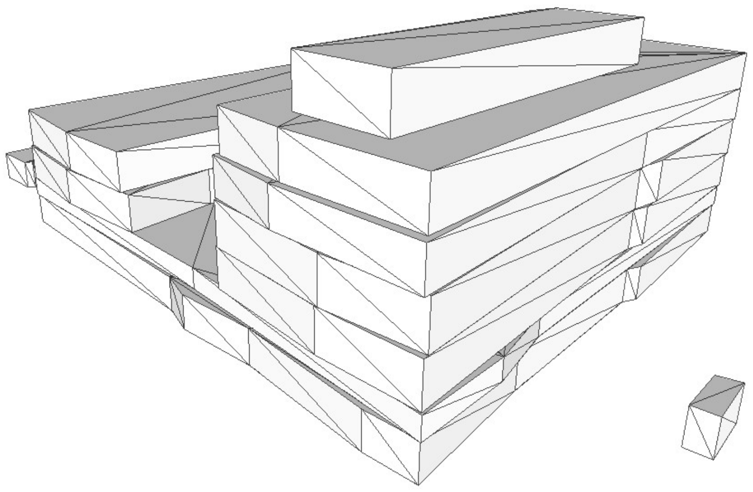
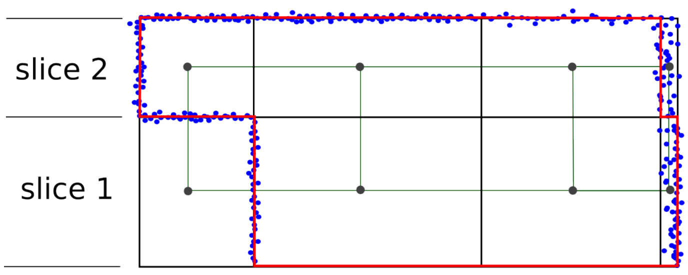


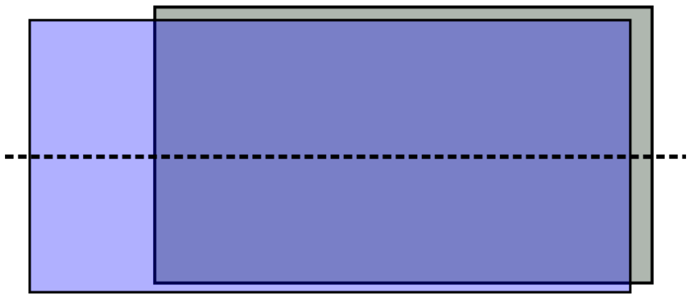
Figure 4.7: *Object outlines extruded to 3D.* For every slice, the floor plan is extruded between the slice boundaries. With this method, we already get a fairly regularized geometric model. However, usually most of the slices have slightly varying floor plans. Therefore, vertical surfaces are not smooth and needs to be regularized in a further regularization step.

4.3.2 Volumetric Cells

As we want to regularize the geometric model using energy minimization via graph cuts, we represent the whole model as a graph. For this, we create a partitioning of the model into irregularly shaped volumetric cells with each cell becoming a graph node. The concept is illustrated in Figure 4.8(a). We define volumetric cells as right prisms with triangles as base faces.



(a) Vertical Cut



(b) Top-View of Two Slices

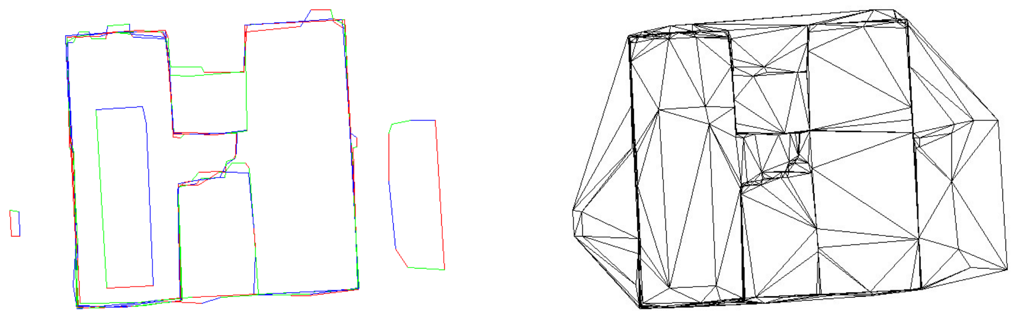
Figure 4.8: *Top: Vertical cut through volumetric cells.* Vertical cut of a volumetric cell representation of a simple model consisting of two slices. The black lines are the volumetric cells spanned over the whole scene and the red lines are the outlines of the extruded slices approximating the point cloud (blue dots). A graph is spanned over the whole scene setting cells with a shared face as neighbors (green lines). *Bottom: Top-view of binary labeling of both slices.* As you can see, a noisy point cloud leads to slightly varying object outlines in each slice. Therefore, the model extruded from the binary labellings does not consist of smooth vertical surfaces.

The creation of the partitioning into volumetric cells works as follows:

- **Projection of floor plans to the ground plane.** We want to keep the possibility that all calculated floor plans in the slices can become outlines in the final, 3D-regularized geometric model. Therefore, as a first step, we project the floor plans of all slices onto the ground plane. If lines intersect, we have to split them as we need a line set with no intersections for the next steps. Figure 4.9(a) shows the projected floor plans on the ground plane.

- **Triangulation.** The volumetric cells are defined to have triangles as base faces. Therefore, we apply a 2D Constrained Delaunay Triangulation (CDT) [38] on the line set of the projected floor plans on the ground plane. Using a CDT, it is guaranteed that specific lines (in our case, the projected floor plans) remain lines in the final triangulation. We use CGAL's CDT component [1] for this step. Figure 4.9(b) shows the triangulated outlines.

- **Extrusion of the Triangles.** Finally, we create volumetric cells with the triangles of the CDT as base faces. We extrude all triangles in all slices, which means, between their slice boundaries. As a result, we get a space decomposition of the whole possible space the geometric model could occupy. Figure 4.11 shows the cell decomposition of the whole scene.



(a) Projected Outlines (b) Volumetric Cells

Figure 4.9: *Outlines and volumetric cells projected to the ground plane.* All floor plans get projected to the ground plane. If lines intersect, they are split into smaller line segments. In 4.9(a), the individual line segments with varying colors are shown. Next, the outlines are triangulated using a Constrained Delaunay Triangulation. Figure 4.9(b) shows the result of the triangulation. As one can see, the irregularities from floor plans of different slices lead to many similar lines near the object borders.

Next, a free space score for every volumetric cell is calculated. For this, the free space score voxel grid that was already calculated for the slice-wise binary segmentation is reused. We calculate an inside and outside score for each cell by summing up the contained free space score voxels. We normalize the cell score with the slice height (which is also the height of the cell) and the size of the base face. Figure 4.10 visualizes the cell scores of one slice.

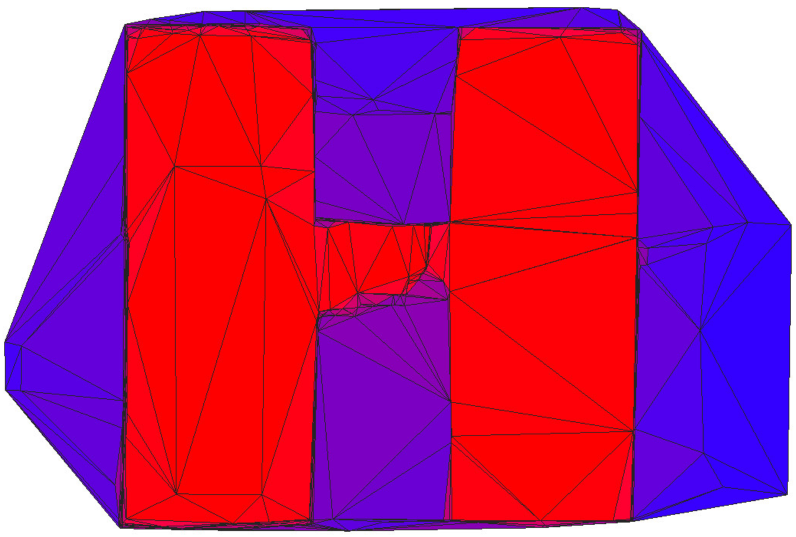


Figure 4.10: *Cell scores of a slice.* This image shows the cell scores of volumetric cells of one slice. Red means a higher inside score than outside score, blue means a higher outside score.

4.3.3 Regularization With Energy Minimization

Having calculated the volumetric cells, it is possible to create a graph which encompasses the whole model. In the graph, every volumetric cell is a node and all cells that share a common face are neighbor nodes. Again, we are using graph cut energy minimization to regularize the geometric model, which is now represented by volumetric cells.

Using the volumetric cell representation representation, we massively reduce the computation complexity compared to a similar computation in a regular voxel grid. Additionally, as we keep the floor plans of the individual slices, we keep important details and enforce smoothing along structures in the input model.

Similar to the slice domain application, for each node $n \in N$, we set two *data terms* E_{data} based on the free space score of the volumetric cells. As we already calculated one score for inside and one for outside for each cell, we can directly use these scores for the data terms.

In difference to the approach in Section 4.2.2.2, where the graph is defined by the regular pixel grid, we have an arbitrarily shaped graph consisting out of volumetric cells in this application. Further, as we want to create a CRF to favor smooth surfaces at the object boundaries, we introduce additional constraints in the regularization process. For this, we set the graph node's *smoothness terms* E_{smooth} individually. E_{smooth} defines an individual smoothing penalty for every neighbor cell pair. Additionally, we weight

E_{smooth} with an adjustable parameter λ, which is the second of the two most important parameters in our workflow (besides the Mean Shift bandwidth parameter d). λ is the parameter which adjusts the amount of the final 3D regularization.

We create the following energy minimization problem:

$$E(L) = \sum_{p \in \mathcal{I}} E_{data}(L(p)) + \lambda \sum_{p,q \in \mathcal{N}} E_{smooth}(L(p), L(q)), \tag{4.9}$$

where $\mathcal{I}$ denotes the set of all volumetric cells, $\mathcal{N}$ is the neighborhood of every cell and L is the (binary) labeling

The data terms, $E_{data}(l_p)$, are defined as

$$E_{data}(l_p) = \begin{cases} insideScore(p) & \text{if } l_p = inside \\ outsideScore(p) & \text{if } l_p = outside \end{cases}, \tag{4.10}$$

where $insideScore(p)$ are the summed up positive free space scores within the volumetric cell and $outsideScore(p)$ are the summed up negative free space scores.

In our approach, the neighbor smoothness penalties from one cell to their neighboring cells, $E_{smooth}(p, q)$, depend on the amount of points near the face of adjacent cells. Using the densified point cloud, we count the points which are near this face reduced by a margin to ignore insignificant data like, e.g., noise near the object boundaries. With this approach, two neighboring cells having dominant structures (for example, a wall) near their adjacent face get penalized while cells without structures near their adjacent face get most likely smoothed into the same group.

$$E_{smooth}(l_p, l_q) = \begin{cases} 0 & \text{if } l_p = l_q \\ \frac{1}{1+\frac{\#\{points\ near\ face_{p,q}\}}{area\ of\ face_{p,q}}} & \text{else} \end{cases}, \tag{4.11}$$

where $\{\#\ points\ near\ face_{p,q}\}$ is the amount of points which have a smaller Euclidean distance to the face than a fraction of the model size.

The result has the value $0 < E_{smooth}(l_p, l_q) \leq 1$, where $E_{smooth}(l_p, l_q)$ is near 0 when lots of points are near the adjacent face, which means there are scene structures. In this case, no smoothing is wanted and due to $E_{smooth}(l_p, l_q) \approx 0$, the smoothness penalty is near 0. $E_{smooth}(l_p, l_q)$ is 1, when no point is near the adjacent face, which means that the total smoothness penalty is completely adjusted by λ.

Figure 4.11 shows the volumetric cells with their corresponding graph and the neighbor weights.

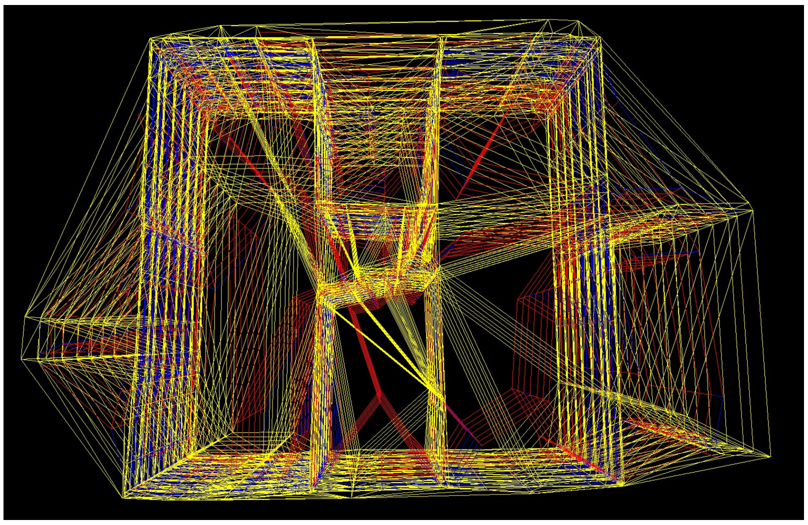


Figure 4.11: *The volumetric cells with their corresponding graph and neighbor weights.* The yellow lines represent the volumetric cells extruded to the slice boundaries. The blue and red lines are the corresponding graph which gets optimized using graph cut energy minimization. The color of the graph's edges represent the neighbor weights $weight_{p,q}$ from vertex p to vertex q. Blue is defined to be a weight near 0, which means that there are many structures near the adjacent face of the two cells. In opposite, red is defined to be a weight near 1, which means not many structures exist near the adjacent face. This volumetric cell representation consists of 4669 cells, while the voxel space used for computations of this model consists of 31360K voxels. Therefore, such a representation massively reduces the computation complexity.

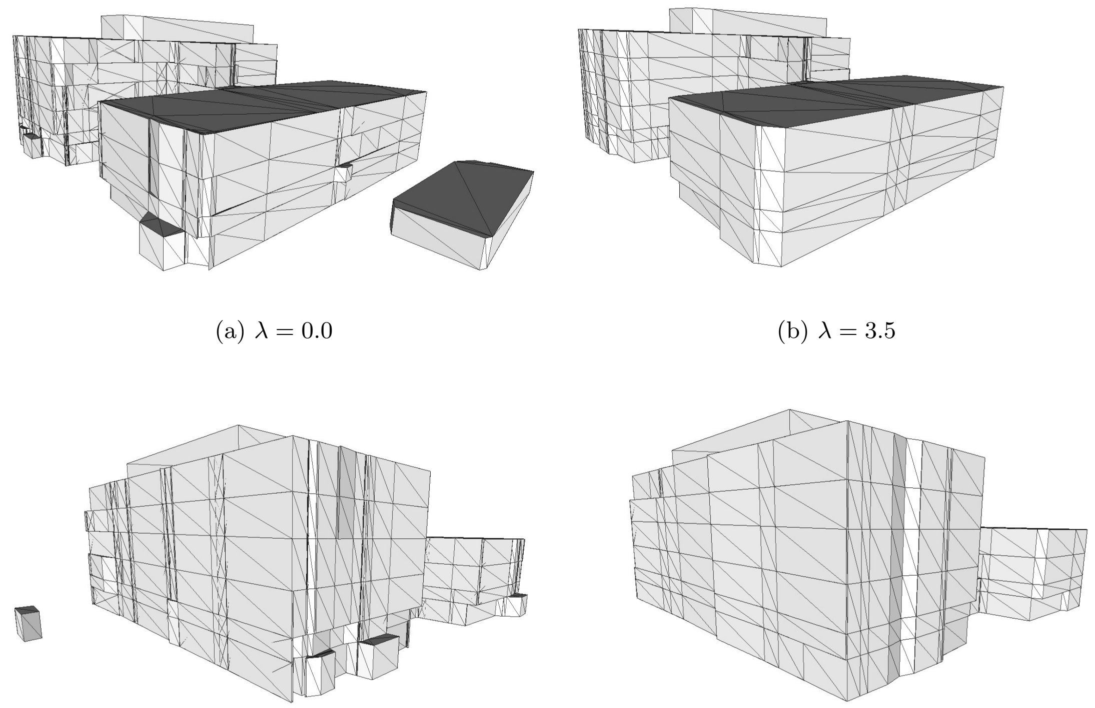
Finally, we get the regularized 3D geometry model with a regularization level (or level of detail) depending on λ. Figure 4.12 shows the results with an appropriate value for λ.

4.4 Input Data and Preprocessing

For our workflow, we set some requirements on the input data and apply preprocessing steps.

In the preprocessing, we transform the model to a canonical coordinate system aligned with a reference plane and perform a predefined, constant sampling of the meshed point cloud.

As input data, our workflow requires a meshed point cloud with the corresponding camera poses. The point cloud may be a sparse or a dense point set, reconstructed, for example, with SfM and optionally densified (for example, with PMVS [19]). Using this point cloud, one can apply meshing techniques like Poisson surface reconstruction [27] or


(a) $\lambda = 0.0$ (b) $\lambda = 3.5$

(c) $\lambda = 0.0$ (d) λ=3.5

Figure 4.12: *Geometries with different values for λ.* As you can see in these examples, when enforcing a high value of λ, outer surfaces of objects get smoother and small irregularities in the model vanish. With $\lambda = 0.0$, smoothing is disabled and just the summed-up free space scores inside the volumetric cells decide about inside or outside. The higher the value of λ is set, the smoother the surfaces get. More results are shown in Chapter 5.

Delaunay triangulation [30] to create a mesh out of the point cloud.

Due to our preprocessing steps, the density of the input points do not significantly affect the quality of the result. However, as we apply horizontal slicing on the whole model, the type of scenery is very important. Our main assumption is that the scene mainly consists out of horizontal planar structures and orthogonal vertical surfaces.

4.4.1 Reference Plane Estimation and Coordinate Transform

To ease the processing of the model, it gets transformed into a new coordinate system aligned with an estimated reference plane. This plane can be the ground plane, but also any other dominant plane structure. As we assume that the scene consists out of horizontal planar structures and orthogonal vertical surfaces, this eases further processing steps like the partitioning into horizontal slices or the neighbor weights calculation used in 3D graph cut optimization.

We create a reference plane hypothesis, where a maximum of points (usually, the area near the ground plane is assumed to have the maximum of points) has a small Euclidean distance to the hypothesis. We iteratively calculate the result using RANSAC (Random Sample Consensus [16]). However, also other approaches for estimating the ground plane are possible. For example, there exists methods to estimate the ground plane using the orientation of the camera views used for the reconstruction [46]. Having the plane information, for all further processing the coordinate system of the point cloud gets transformed to the reference plane.

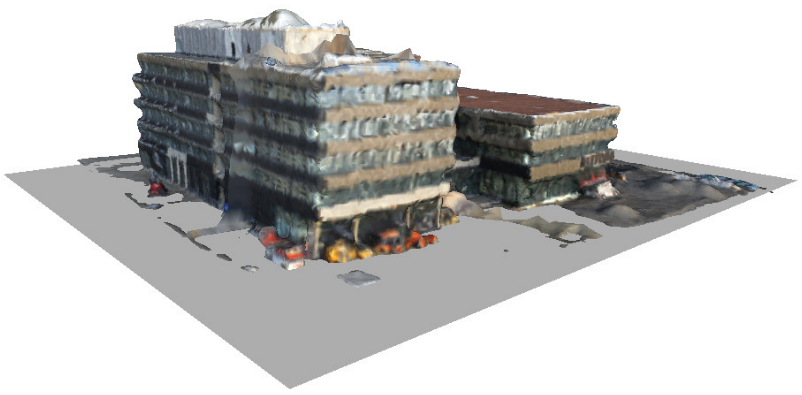


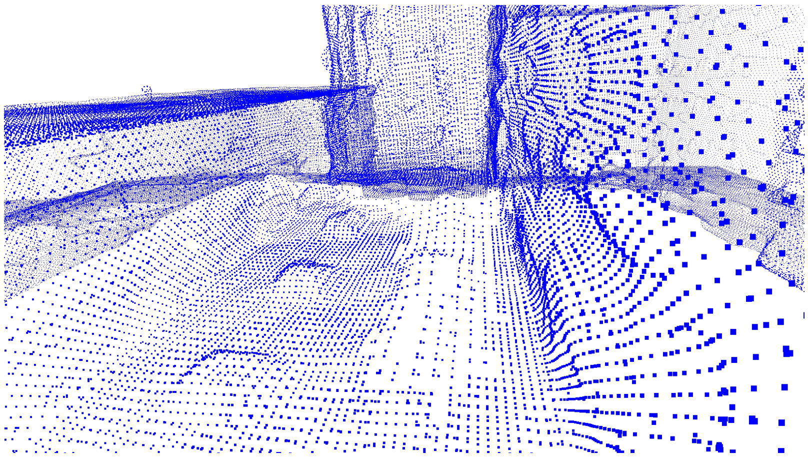
Figure 4.13: A model with the estimated reference plane (gray). In this case, the reference plane is the ground plane. However, our approach also works if the reference plane, for example, is the roof.

4.4.2 Densification of Point Cloud

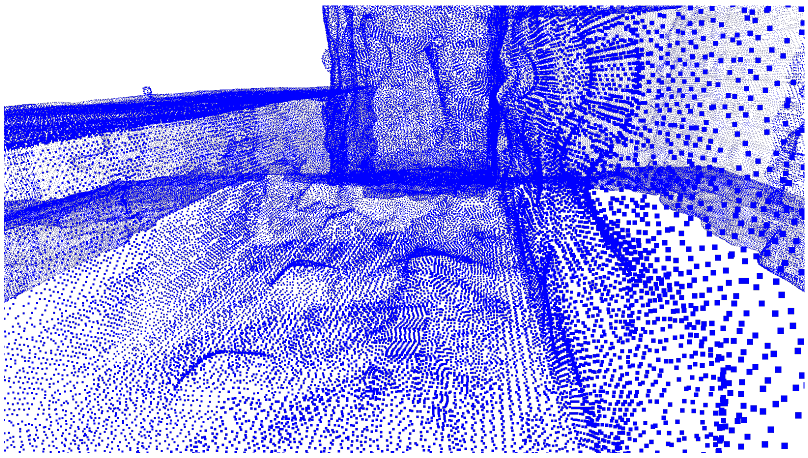
As the input model can have an arbitrary density, we apply a constant upsampling to the point cloud mesh. Using a constantly sampled point cloud, we have several advantages:

To find horizontal structures in the point cloud, we need to detect modes of horizontal-oriented points. Without a constantly sampled cloud, modes cannot be found reliably.

In the last processing step, the 3D optimization, it is crucial to have a constantly sampled point cloud. We calculate cell neighbor weights depending on the amount of points near the shared face of adjacent cells. Figure 4.14 illustrates the upsampling of a point cloud.



(a) Input Point Cloud



(b) Densified Point Cloud

Figure 4.14: In 4.14(a) you can see the input point cloud with spatially varying sampling. After the densification step, we get a constantly dense sampled point cloud (as in 4.14(b)).

Chapter 5

Experiments

Contents

The main goal of our work is to regularize meshed 3D point clouds and simultaneously reduce the amount of data. In our experiments, we show the deviations with respect to geometry of the computed geometric model to a ground truth model and to which amount model simplification can be achieved.

Our algorithm has been implemented in C++ using the Graph-Cut library [11] to solve energy minimization problems. For visibility testing, we use the axis-aligned bounding box (AABB) tree components from CGAL [6]. For distance calculations in 3D space, we use the Fast Library for Approximate Nearest Neighbors (FLANN), as described in [35].

5.1 Evaluation Metrics

In this section, we introduce metrics needed for the evaluation. We discuss the Dice score, which is used to compare backprojected object masks, the Hausdorff distance used to visualize differences between two 3D models and a regularization measure based on projected line segments.

5.1.1 Dice Score

In our evaluation, we make use of an error measure which approximates the perceived error of humans by comparing the computed geometry and the ground truth model. For this, we backproject the geometry and the ground truth into every camera view and compute the backprojection error. As measure, we use the Dice score.

The Dice score, introduced in [14], relates the area of two segments $|E_1|$ and $|E_2|$ with the area of their mutual overlap $|E_1 \cap E_2|$, such that

$$dice(E_1, E_2) = \frac{2|E_1 \cap E_2|}{|E_1| + |E_2|}, \tag{5.1}$$

where $|\cdot|$ denotes the area of a segment. If the two segments are completely identical, the score is 1. Contrary, if there is no overlap, the score is 0. This concept is visualized in Figure 5.1.

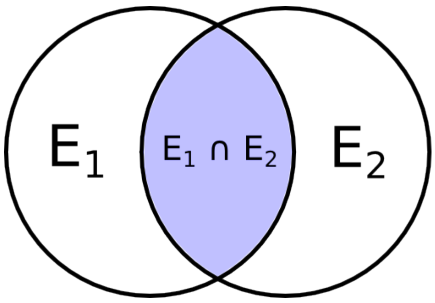
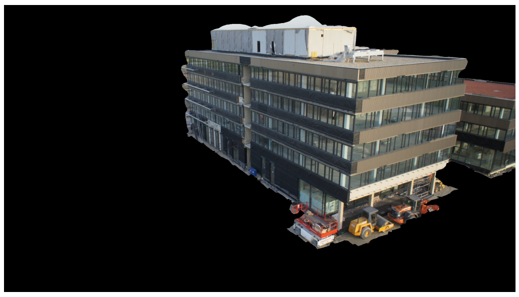
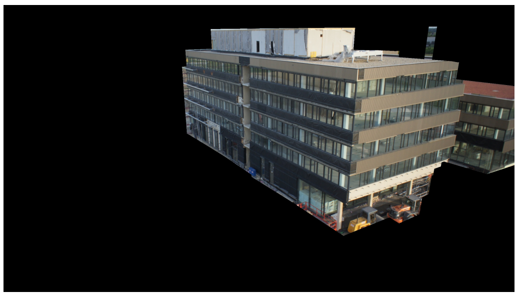


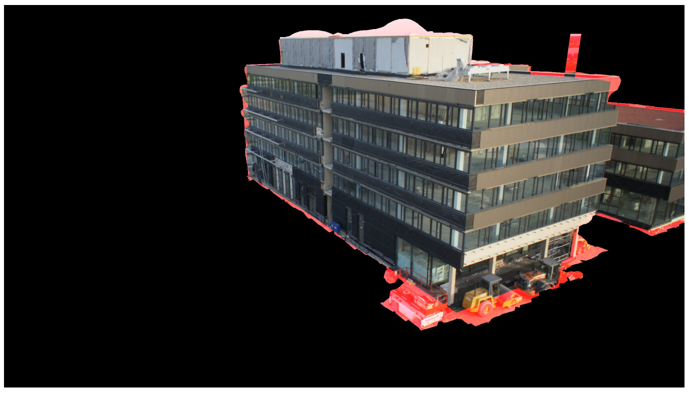
Figure 5.1: *Dice score.* The Dice score is defined as the doubled area of the mutual overlap of E_1 and E_2 divided by the summed area of E_1 and E_2. Identical segments have a score equal 1, segments without overlap a score equal 0.

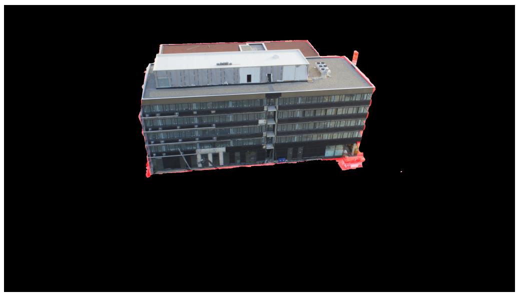
Backprojecting the 3D models into the camera views, we get object masks for each view, represented by an image matrix. With this masks, we have a segmentation for the geometry and for the ground truth and can compare them using the Dice score. Figure 5.2 illustrates the score calculation. We calculate the mean, the median and the variance over all images in order to compare different parameter settings for a data set.

As the dice score drastically fluctuates by changes on images where just a small part of the backprojected model is captured, we skip camera views where the backprojected part of the ground truth model covers less than 5% of the total model surface.


(a) Ground Truth


(b) Reconstructed Geometry


(c) Error between 5.2(a) and 5.2(b), Dice Score = 0.9706


(d) Dice Score = 0.9791

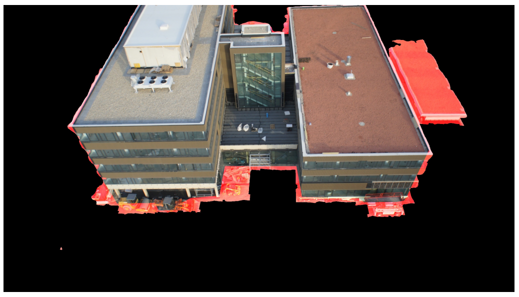

(e) Dice Score = 0.9259

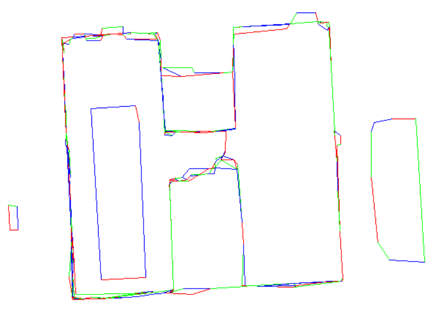
Figure 5.2: *Backprojections used for Dice score.* In 5.2(a) and 5.2(b) one can see the ground truth model and the reconstructed geometry backprojected into a camera view. In 5.2(c), the error between both is marked in red. The error is defined to be the area where the both backprojected masks do not overlap. This information is used for the calculation of the Dice score. In 5.2(d) and 5.2(e) one can see the error masks from different views. Having a big error (as in 5.2(e)), the Dice score decreases.

5.1.2 Regularization Measure

As a main goal is to regularize the input data, we have to define a measure for the degree of regularization. We project the outlines of each slice (i.e., the vertical faces of the geometric model) onto the ground plane and count the number of unique lines as a complexity score. The score is defined as

$$complexity(M) = countUniqueLines(\forall slices \in M : projectOutlinesToGroundPlane()), \tag{5.2}$$

where M is the model and $countUniqueLines(\cdot)$ returns the number of unique line segments The more lines exist, the less regularization could be achieved. A visualization of this concept is shown in Figure 5.3.


(a) $\lambda = 0.0$, $complexity = 477$

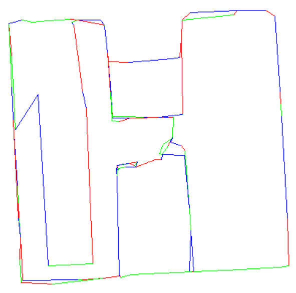

(b) $\lambda = 3.5$, $complexity = 140$

Figure 5.3: *Complexity scores from projected outlines.* In this Figure, one can see the outlines of all slices projected onto the reference plane. The individual line segments are encoded in different colors. Left, the line segments projection of a geometric model with $\lambda = 0.0$ can be seen. It contains many similar line segments due to slightly different outlines of the slices. Right, the projection of a model with $\lambda = 3.5$ is visualized. As you can see, more segments at object boundaries are merged into common segments identical in all slices. Additionally, the small objects vanish completely. As a result, the complexity score decreased from 477 to 140 unique line segments.

5.1.3 Hausdorff Distance

Given two 3D meshes, the Hausdorff distance computes the error between these meshes. As we use synthetic models and reconstructions of these models (see Section 5.2.1), we use this measure to relate the reconstructions to the original synthetic model. However, as we do not want to evaluate the whole 3D reconstruction pipeline, this measure should just be an indicator for how well the synthetic model was reconstructed.

Given two point sets X and Y, the Hausdorff distance $d_{\mathrm{H}}(X, Y)$ measures the longest distance from a point in set X to the nearest point in set Y. It is defined as

$$d_{\mathrm{H}}(X,Y) = \max\{\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y)\}, \tag{5.3}$$

where sup denotes the supremum (also referred to as the least upper bound) and inf denotes the infinum (also referred to as the greatest lower bound). Figure 5.4 illustrates this relationship. For more details, we refer the reader to [25].

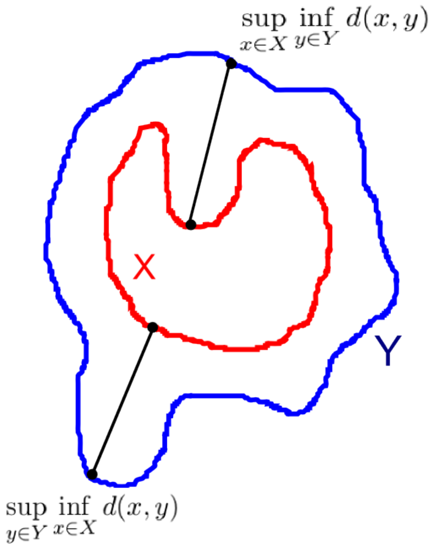

$\sup_{x\in X}\inf_{y\in Y} d(x,y)$
X
Y
$\sup_{y\in Y}\inf_{x\in X} d(x,y)$

Figure 5.4: *Hausdorff distance.* The Hausdorff distance is defined as the maximum of $\sup_{x\in X}\inf_{y\in Y} d(x,y)$ and $\sup_{y\in Y}\inf_{x\in X} d(x,y)$. It is the longest possible distance to directly move from set X to set Y.

The Hausdorff distance between two meshes is computed by sampling one of the two and finding for each sample the closest point over the other mesh. In our experiments, we use the Hausdorff distance computation implemented in Meshlab [2]. The computed Hausdorff distances for our models can be seen in Figure 5.6.

5.2 Evaluation Data

Unfortunately, it does not exist a commonly used evaluation dataset with the possibility to compare our results directly with results of other approaches. However, we choose to evaluate on some freely available synthetic models from the Trimble 3D Warehouse [32] (formerly known as Google Warehouse). Additionally, we evaluate on some real-world 3D reconstructions. For both types, we use models from buildings which meet our requirements (i.e., horizontal planar structures and orthogonal vertical surfaces) up to a certain degree.

5.2.1 Synthetic Models

In order to evaluate our workflow with a correct ground truth, we compute the geometry of synthetically generated models form the Trimble 3D Warehouse. The Trimble 3D Warehouse is a collection of user-generated, synthetic 3D models. It contains objects like chairs or desks, but also models from buildings from all over the world.

For our evaluation, we choose two building models with different complexity. The models can be seen in Figure 5.6. The model in Figure 5.6(a) is the Joanneum building in Steyrergasse, Graz. We chose this one because it is a simple model which fits well to our requirements. It predominantly consists of orthogonal horizontal and vertical planar structures. The second model in Figure 5.6(b), is the Evansville Auditorium and Convention Centre (in the following named Centre). It is more complex because it consists of dominant horizontal planar structures, but containing rounded orthogonal vertical structures. Even though these structures can not be modeled perfectly with our approach, they can be approximated well by a polygonal line.

As input for our workflow, we use reconstructions from the synthetic models. We first texture the model with a random texture and create virtual camera views capturing the whole scene. As result, we get camera views as shown in Figure 5.5. Then, we reconstruct the scene with SfM and PMVS and finally mesh the point cloud using the Poisson surface reconstruction method.

For evaluation, we compare the computed geometry with the ground truth, which is the original synthetic model.

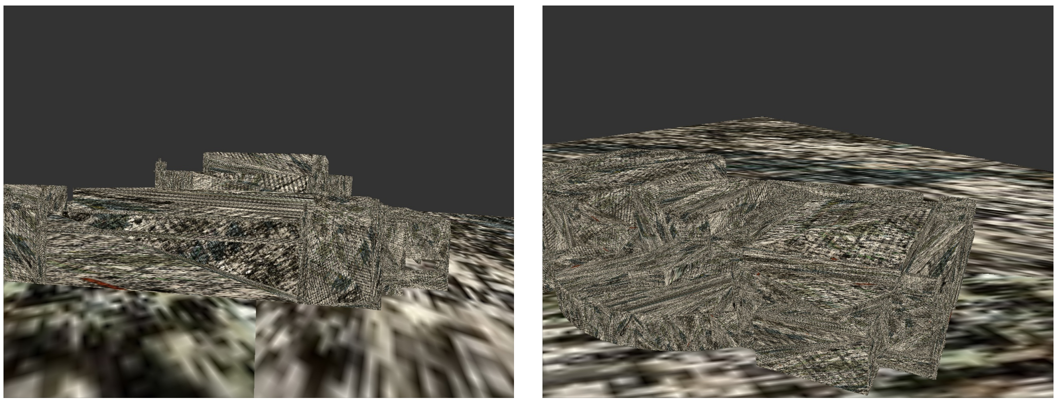


Figure 5.5: *Virtual camera views of synthetic model.* As a synthetic model usually has large regions with homogeneous texture structures, it first gets textured with a random texture. Consecutively, virtual camera views are created to reconstruct the model using SfM.

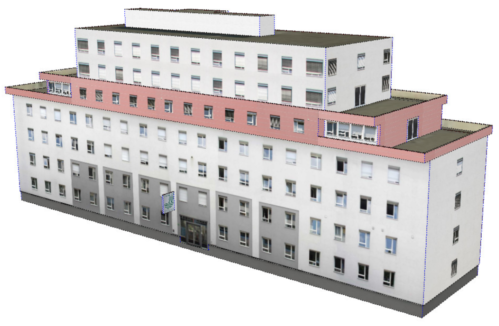
5.2.2 Models from Real-World Image Data

We also evaluate models reconstructed from real-world image data using SfM, PMVS and Poisson triangulation. Again, we use two models from buildings which meet our requirements up to a certain degree.

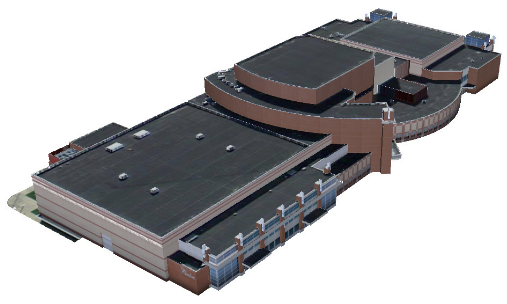
The first model, which can be seen in Figure 5.8(a), is a reconstruction of a building in Aspern, Vienna. It is a rather simple model which suits our requirements well. However, as it is created using real-world image data captured from an unmanned aerial vehicle (UAV), it contains more noise and clutter. This occurs mainly due to untextured parts of the scenery and surfaces consisting out of glass. In Figure 5.7 some images used for the reconstruction process are printed.

The second model is the reconstruction of the Siemenscity in Vienna (Fig. 5.8(b)). It contains very challenging parts, as it contains sloped vertical structures which cannot be represented exactly in our horizontal slicing-based geometric models. However, supposing a sufficiently high level of detail in the horizontal direction (i.e., a sufficiently small Mean Shift bandwidth), they can be approximated by stairway-like structures.

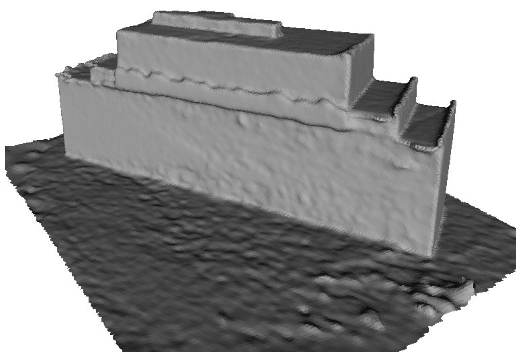
Using real-world models, we do not have a correct ground truth. However, as we need a model to compare our results against it, we use the significant parts of the reconstructed model (i.e, the model reduced by the ground plane) as if it were the ground truth.



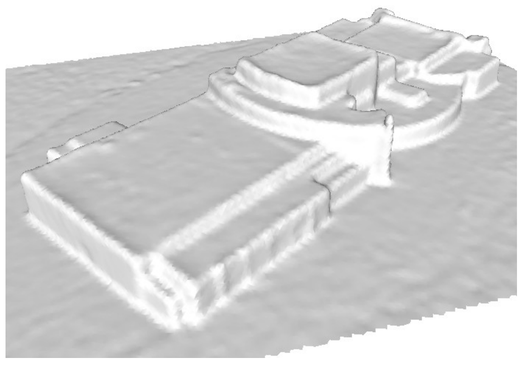
(a) Joanneum - Synthetic Model



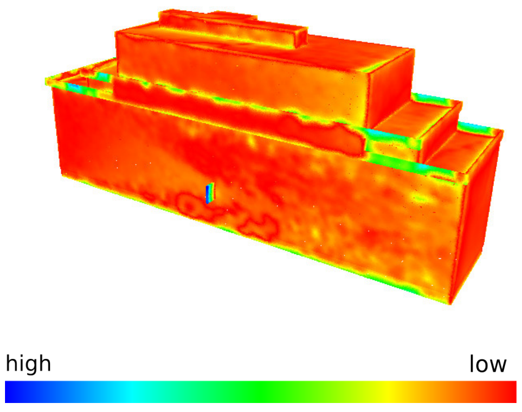
(b) Centre - Synthetic Model



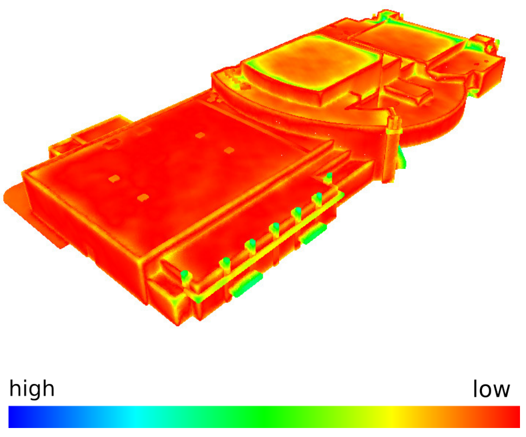
(c) Reconstructed Model



(d) Reconstructed Model



(e) Hausdorff Distance



(f) Hausdorff Distance

Figure 5.6: *Synthetic models used for evaluation.* In the first row, one can see the synthetic models from the Joanneum building and the Evansville Auditorium and Convention Centre taken from Trimble Warehouse and in the second row the corresponding reconstructions. In the third row, the Hausdorff distance between the synthetic and the reconstructed model is visualized. Blue means a high, red a low Hausdorff distance.

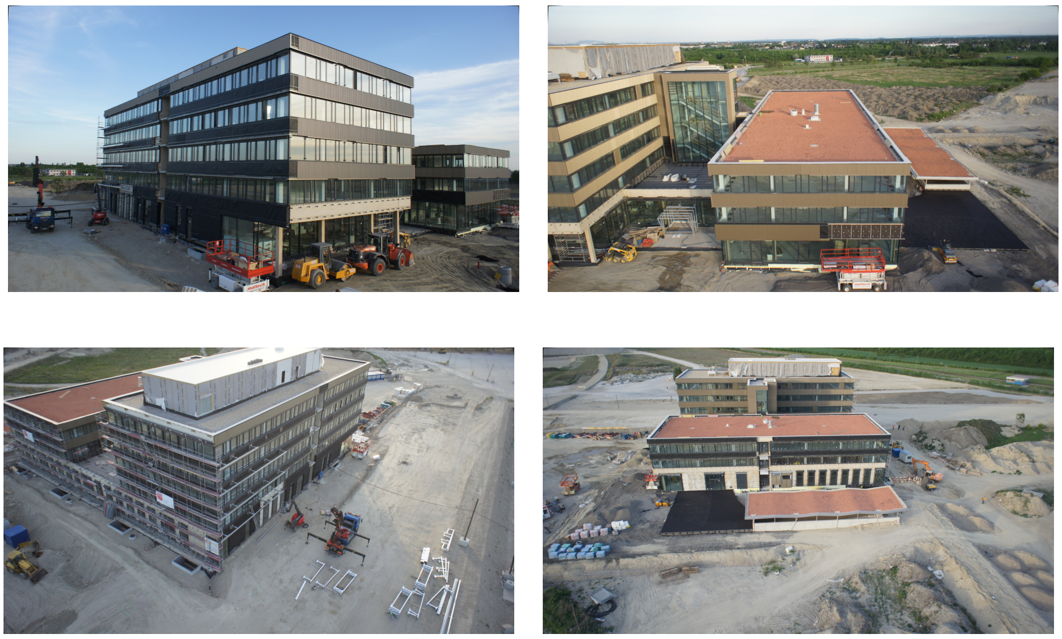
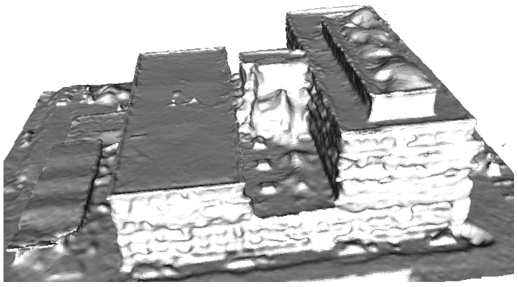
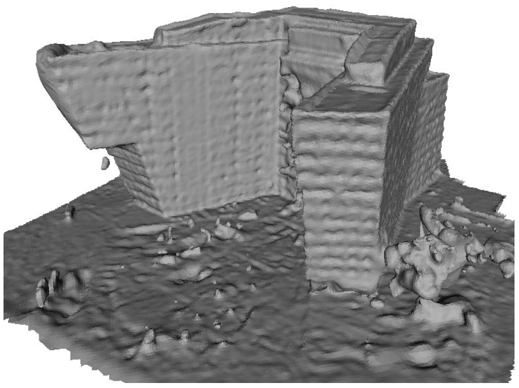


Figure 5.7: *Images from Aspern building taken from an UAV.* As you can see, it contains many surfaces consisting of glass. Additionally, a part of the roof is completely white, which also leads to clutter in the reconstruction (bottom left image).



(a) Aspern



(b) Siemenscity

Figure 5.8: *Real-world data models used for evaluation.*

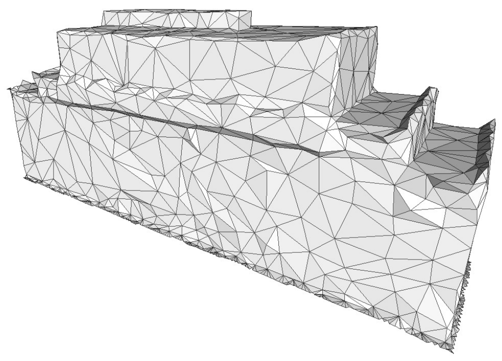
5.3 Results

We evaluate our approach on the four mentioned models using different parameter settings for the smoothing parameter λ and for the Mean Shift bandwidth parameter d. For every model, we choose parameter settings which demonstrate strengths and weaknesses of our approach. The results with their corresponding error and complexity measures are printed below.

5.3.1 Comparison to Commonly Used Mesh Simplification Methods

A commonly used mesh simplification method is Quadric Edge Collapse introduced in [20]. This algorithm iteratively reduces the number of faces and vertices by removing edges from the mesh and simultaneously merging vertices. The best edge to remove is selected using a quadric error metric.

Figure 5.9 shows the result of Quadric Edge Collapse mesh simplification in comparison to a result of our approach. As you can see, our approach delivers more regularized models following concrete assumptions of the scene (piecewise planarity, dominant horizontal structures). Further, in comparison to Quadric Edge Collapse, our approach does not just deliver a simplified mesh but a scene description consisting of slices and their corresponding floor plans. With this information we already deliver a weak semantic description of the scene which can be enriched easily with additional semantic information.


(a) Quadric Edge Collapse

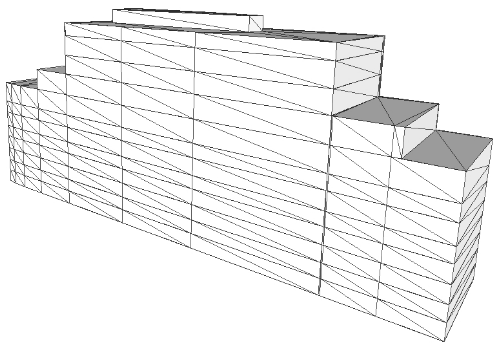

(b) Our Approach

Figure 5.9: *Comparison with Quadric Edge Collapse.* Left, one can see a simplified mesh of the Joanneum model using the Quadric Edge Collapse Simplification method of Meshlab. The target number of faces was set to 2288, which is the number of faces a good resulting model of our approach (right) contains.

5.3.2 Level of Detail in Vertical Direction

As already discussed earlier, a higher Mean Shift bandwidth (i.e., a lower value for the parameter d) leads to lesser slices and therefore to a lower level of detail in the vertical direction. In contrary, a lower Mean Shift bandwidth leads to more slices and a higher level of detail in the vertical direction.

For the Joanneum model (results can be seen in Fig. 5.11), d set to 10 is not a sufficient level of detail in the vertical direction as not all dominant horizontal structures are detected as slice boundaries. As some slice boundaries get positioned between dominant horizontal structures (due to the properties of the Mean Shift), artifacts arise (for example, in Fig. 5.11(a)). With d set to 25, all important horizontal planar structures are detected and good results can be achieved with an appropriate value for λ.

Looking at the results for the Centre model in Fig. 5.14, it seems as if with $d = 10$ the Mean Shift bandwidth is sufficiently small to detect slice boundaries at most of the dominant horizontal structures. However, with $d = 25$ and a higher value for λ, also a similar level of regularization is reached.

In the results of the Siemenscity model in Figure 5.16, one can see that even sloped structures can be approximated by stairway-like structures. Depending on the value for d, the quantization is finer or coarser. With $d = 35$, the sloped structures already get approximated quite well.

Generally, one can observe that a too high value for the bandwidth parameter d (i.e., a too low bandwidth) is not as crucial as a wrong parameter for λ. More smoothing can compensate the incorporated irregularities due to a too low bandwidth up to a certain degree. Contrary, a too high bandwidth may eliminate important details and introduce artifacts. To solve this problem, a generic value for d would be desirable. Though, as the detection of correct slice boundaries depends on the model properties, a generic value for d is hard to find. As the Dice score increases significantly until all dominant horizontal structures are detected and afterwards usually develops with minor increases, one possibility would be to start with a low value for d and increase it until the Dice score does increase less than a given threshold. Having found an optimal d, the level of detail would be adjustable only with the parameter λ.

5.3.3 Smoothing of Volumetric Cells

The second important parameter in our workflow is the smoothing factor λ used in the volumetric cells energy optimization. Even if the level of detail in the vertical direction is too fine, more smoothing can compensate the incorporated irregularities. Generally, the Dice score decrease and the Dice score standard deviation and the level of regularization increases when increasing the value for λ.

In Figure 5.11 one can see the results of the Joanneum model. Even for this simple model that consists mainly of orthogonal horizontal and vertical planar structures, with a low value for λ many small irregularities are incorporated in the resulting geometry. This leads to a high Dice score, but also a high complexity score (which can be seen in Fig. 5.13). Contrary, a too high value for λ leads to vanishing of parts of the model in this case (see Fig. 5.11(f)).

Looking at Figure 5.12 in more detail, one can observe that the Dice scores up to $\lambda = 1$ slightly increase. Usually, the Dice score decreases when increasing λ, as more regularization means a higher error with respect to the ground truth. However, in this case, as too few slices are generated to represent all the important horizontal structures, a slightly higher smoothing leads to an increase of the Dice score. For $\lambda > 1$, the Dice scores and the complexity scores decrease simultaneously as expected. Starting from $\lambda = 3$, the whole model is represented as one cuboid, having the same Dice score and complexity score for many higher values for λ.

The graph for $d = 25$ in Figure 5.13 shows a more continuous development of the Dice score. However, starting at $\lambda = 5$, the model starts to shrink extremely and subsequently the Dice score also decreases a lot. At $\lambda = 7$ (not included in the graph any more) the model completely vanishes due to an over-regularization. Generally, one can see that the mean and the median follow a similar trend and the standard deviation increases when enforcing more smoothing.

For the Centre model (results in Figure 5.14), the Dice scores in Table 5.1 decrease and the standard deviation increases for both values for d when increasing the value for λ. Simultaneously, the complexity score decreases as expected. At certain values for λ (for $d = 10$ at $\lambda = 4.5$ and for $d = 25$ at $\lambda = 7.8$) parts of the model start to vanish (see Fig. 5.14(f)). At a slightly higher λ the model vanishes completely.

In the results for the Aspern model in Figure 5.15 one can see that the small second building at the left side of the reconstruction vanishes already with low smoothing. This can also be observed looking at the Dice scores in Table 5.2, where the score rapidly decreases at $d = 10$ between $\lambda = 0.0$ and $\lambda = 1.0$ and at $d = 25$ between $\lambda = 1.0$ and $\lambda = 1.5$. This happens due to the lack of neighboring volumetric cells with high inside free space scores and the lack of solid structures directly at the vertical cell boundaries, which influence the neighbor weights in the CRF.

A similar effect can be observed at $d = 25$ at the right side of the scene. A tiny part is visible in the geometric model at $\lambda = 0.0$ due to clutter which gets smoothed out quickly. Though, at a high value of λ the model gets over-smoothed and the space between the small clutter object and the parts of the model are all smoothed into the resulting geometric model.

The results for the Siemenscity model can be seen in Figure 5.16. When enforcing low smoothing, an additional object at the right side is included in the geometric model. As this originates from vegetation, it is not included in the ground truth model. Therefore, as one can see in Table 5.3, the Dice score initially rises while increasing the value of λ. However, after the unwanted object vanished due to sufficient regularization, the Dice scores develop as usual and decrease constantly.

Another over-smoothing effect can be observed in Fig. 5.16(e), where the whole space surrounded by structures of the model gets smoothed into the resulting geometric model. As such a free space surrounded by occupied space contains many neighboring volumetric cells which have a higher inside score, it is likely that this space gets smoothed into the geometric model.

5.3.4 Robustness Against Noise and Clutter

As we use models from image-based reconstruction methods, they usually contain error data. In Figure 5.8 one can see that the models reconstructed from real-world image data contains a lot of noise and clutter due to missing texture or errors in the reconstruction process. However, even in reconstructions with images from randomly textured synthetic models (as in Figure 5.6) noise exists as a result of errors in the reconstruction process. In Figure 5.10 one can see points contained within a slice projected on an image matrix. Methods like [48] use Hough line detection to detect the outlines of objects on an image matrix like this. However, as far too many lines get detected in such an image using Hough Transform line detection, this method is computationally inefficient and does not deliver good results for noisy scenes.

The results show that due to the regularization in the slice domain and in the 3D

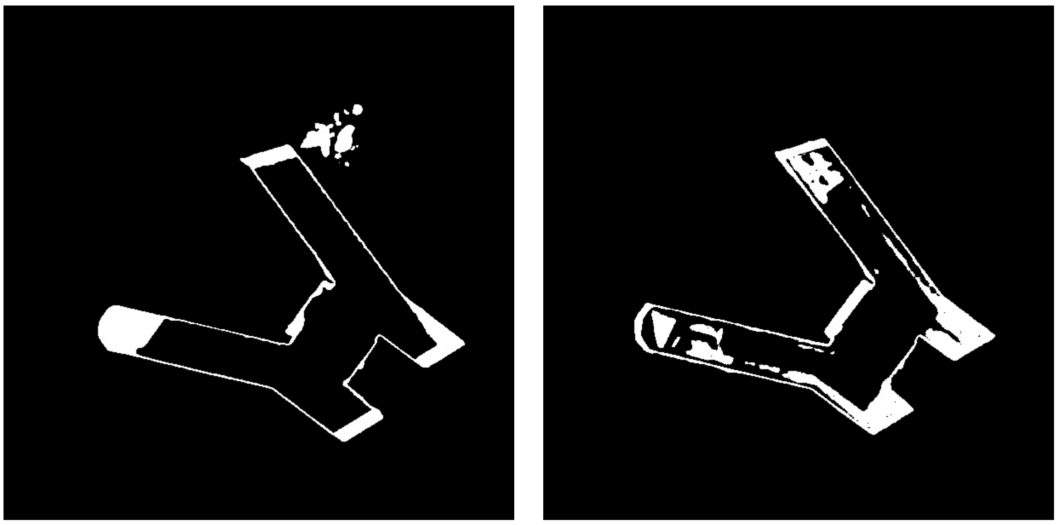


Figure 5.10: *Points contained in a slice.* In this figure, all points within a slice are projected onto an image matrix. The left image is a slice of the Siemenscity model computed with $d = 10$. One can see that errors in the reconstruction and sloped vertical structures lead to many points along the object boundaries. The right image is again taken from Siemenscity, but with $d = 25$. In this example, clutter in horizontal surfaces leads also to clutter in this image. With this difficulties, it is hard to find object outlines using, for example, Hough Transform line detection.

domain using volumetric cells containing the outlines of the slice segmentations, we achieve good results even in presence of noise and clutter. For example, in Fig. 5.11(b) one can see the resulting geometric model from the Joanneum model with $\lambda = 0.0$ which leads to irregularities at the different slices. Contrary, at Fig. 5.11(d), where $\lambda = 4.5$, the model only contains smooth surfaces and simultaneously covers the original model well. For the Aspern model, which is a model created from real-world image data and therefore contains much more clutter, the results (which can be seen in Fig. 5.15) also show smooth surfaces when enforcing enough smoothing.

5.3.5 Limitations

As our approach is based on piecewise planarity in the horizontal (due to horizontal slicing) and in the vertical direction (due to an approximation by a polygonal line), rounded and horizontal sloped structures can not be represented exactly.

However, as one can see in Figure 5.14, vertical rounded structures get reconstructed well with a polygonal line. Even horizontal sloped surfaces can be approximated with

d	λ	Dice Score: *median*	*mean*	*std.deviation*	*complexity*
10	0.0	0.9697	0.9639	0.0219	111
10	2.0	0.9693	0.9634	0.0220	99
10	4.5	0.9099	0.8554	0.1165	67
25	0.0	0.9723	0.9692	0.0186	390
25	4.0	0.9717	0.9678	0.0188	240
25	7.8	0.9478	0.9353	0.0396	175

Table 5.1: *Table of Dice scores and complexity scores of the Centre model.* As one can observe, the complexity values drop with increasing λ and increasing Mean Shift bandwidth.

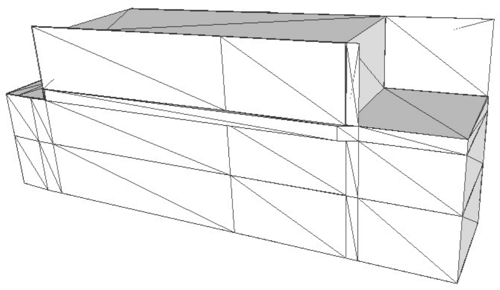
d	λ	Dice Score: *median*	*mean*	*std.deviation*	*complexity*
10	0.0	0.9679	0.9660	0.0148	124
10	1.0	0.9497	0.9369	0.0535	72
10	3.5	0.9400	0.9266	0.0539	51
25	0.0	0.9705	0.9695	0.0133	480
25	1.0	0.9705	0.9685	0.0138	241
25	1.5	0.9501	0.9391	0.0519	198
25	3.5	0.9465	0.9347	0.0540	148
25	4.5	0.8995	0.8960	0.0545	128

Table 5.2: *Table of Dice scores and complexity scores of the Aspern model.*

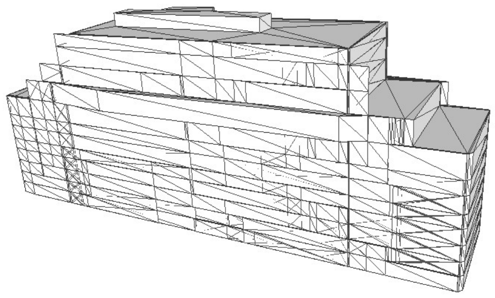
stairway-like structures as can be seen in Figure 5.16. Depending on the Mean Shift bandwidth, the quantization gets coarser or denser. Setting the parameter d to 35, the slope is approximated reasonably well. Again, compared to [48], such structures can be approximated better with our approach than using Hough Transform line detection, as Hough Transform line detection on projected slices (as in Figure 5.10) can just detect the beginning or the end of slopes while our approach finds an optimal object outline depending on the free space score.

5.3.6 Processing Time

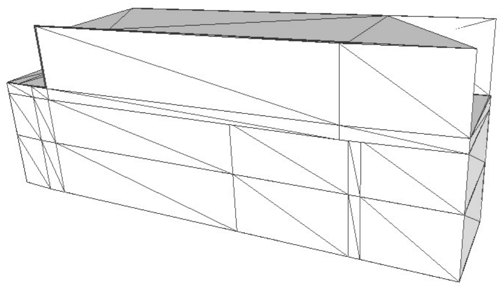
In Table 5.4, the processing times for the different geometry models are listed. Generally, most of the time is needed for the computation of the free space score which includes the visibility tests and the distance calculations for inside-labeled voxels. Therefore, the number of cameras used for visibility testing and especially the size of the voxel space are the most crucial processing time factors. Figure 5.17 visualizes the distribution of the time needed for the different modules. The free space score computation needs 96% of processing time. Additionally, this can be seen at the times needed for the Aspern



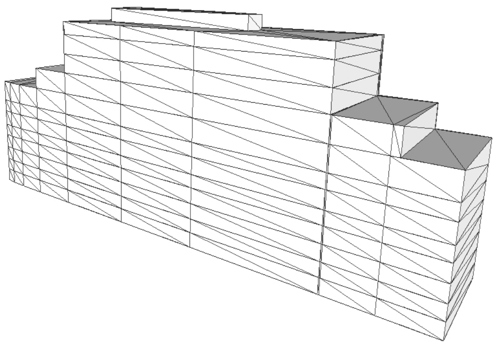
(a) $d = 10$, $\lambda = 0.0$



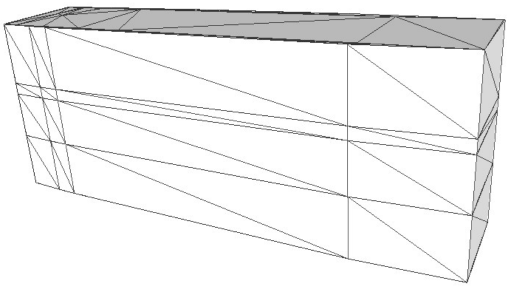
(b) $d = 25$, $\lambda = 0.0$



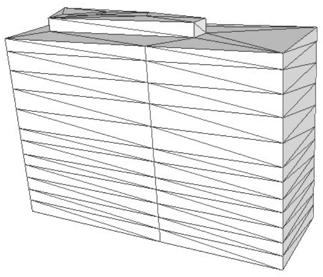
(c) $d = 10$, $\lambda = 2.0$



(d) $d = 25$, $\lambda = 4.5$

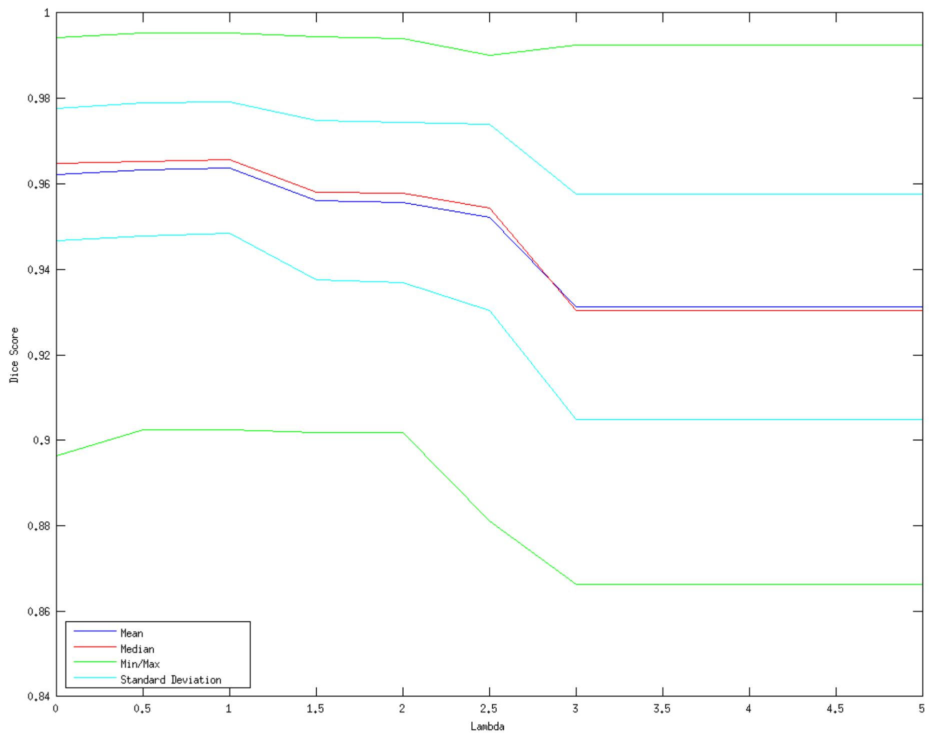


(e) $d = 10$, $\lambda = 4.0$

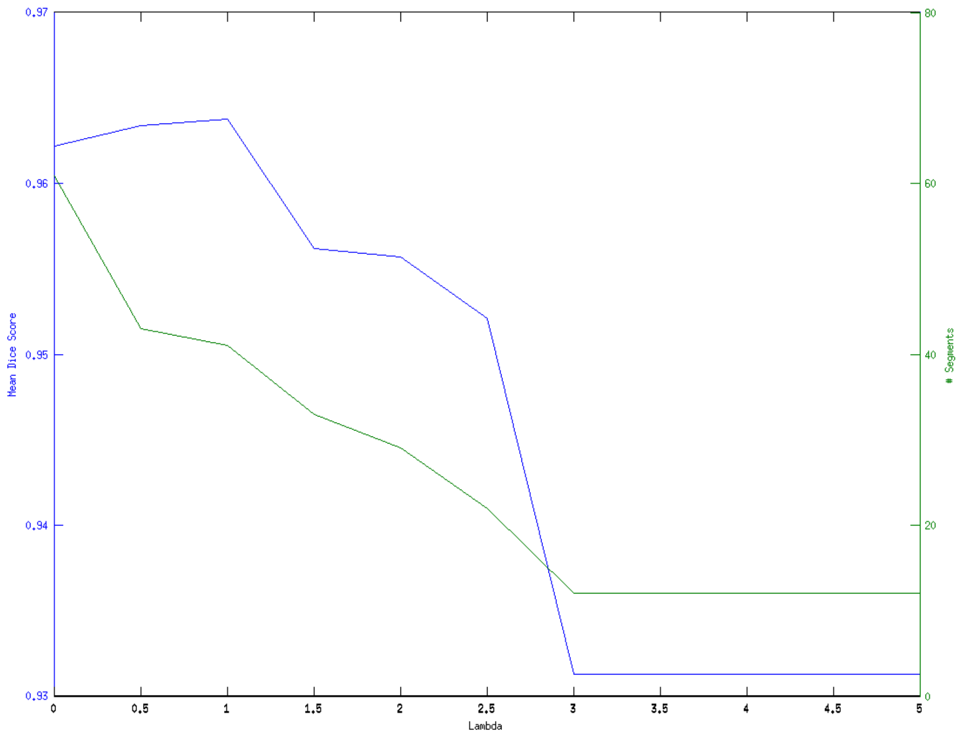


(f) $d = 25$, $\lambda = 6.0$

Figure 5.11: *Results of the Joanneum model.* The Joanneum model is a simple synthetic model which fulfills our requirements well. Therefore, selecting a good d and λ, we get a good geometric representation. With too much smoothing, the whole model gets smoothed into one block or parts of the model start to vanish.

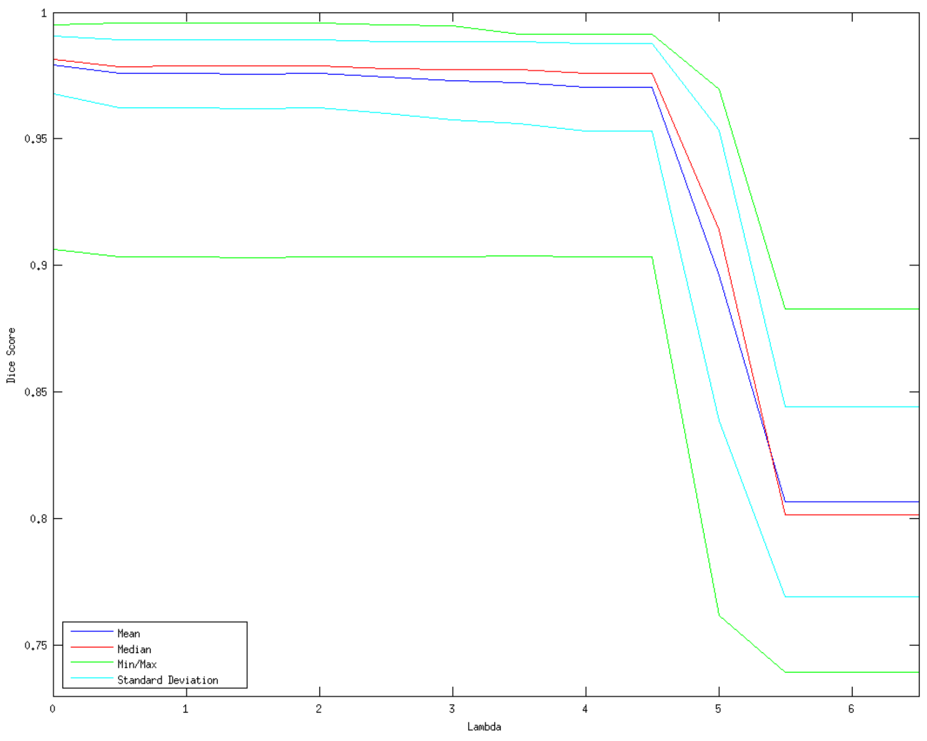


(a) Dice Scores

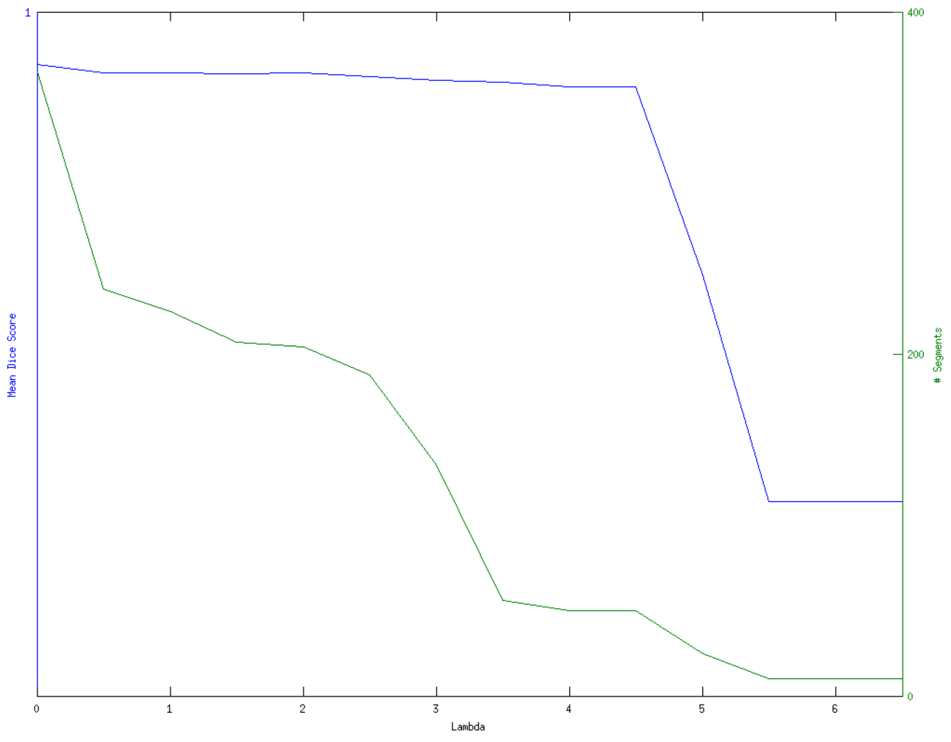


(b) Complexity Scores

Figure 5.12: *Dice scores and complexity scores of the Joanneum model ($d = 10$.)* In the top figure, one can see the mean and median Dice scores over all images, the corresponding standard deviation and the minimum and maximum Dice score. In the bottom Figure, the mean Dice score and the complexity measure is set into relation.

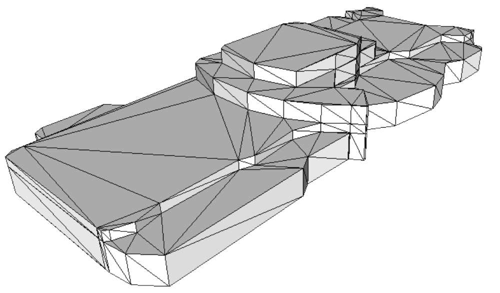


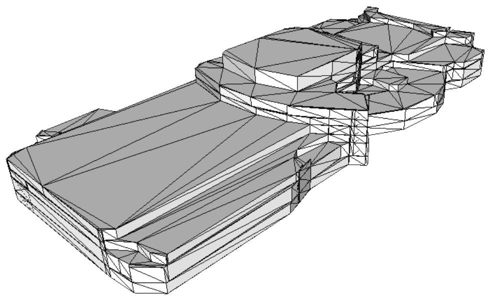
(a) Dice Scores

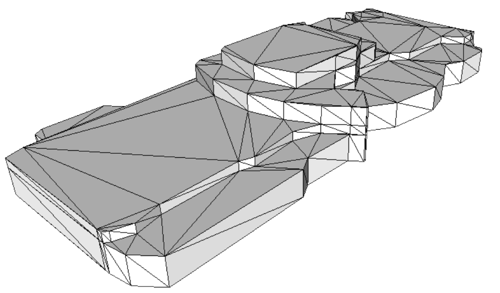


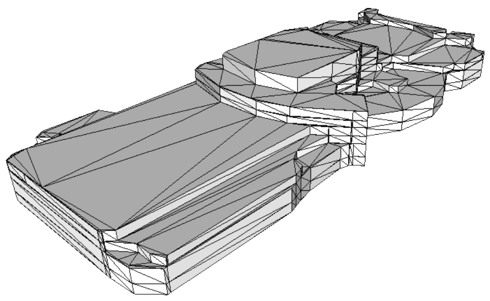
(b) Complexity Scores

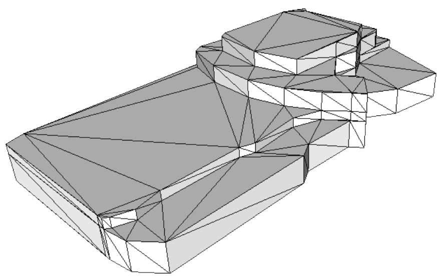
Figure 5.13: *Dice scores and complexity scores of the Joanneum model* ($d = 25$).


(a) $d = 10$, $\lambda = 0.0$


(b) $d = 25$, $\lambda = 0.0$


(c) $d = 10$, $\lambda = 2.0$


(d) $d = 25$, $\lambda = 4.0$


(e) $d = 10$, $\lambda = 4.5$

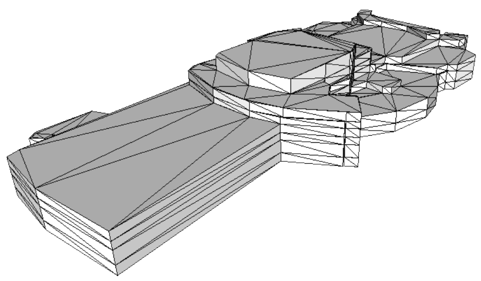
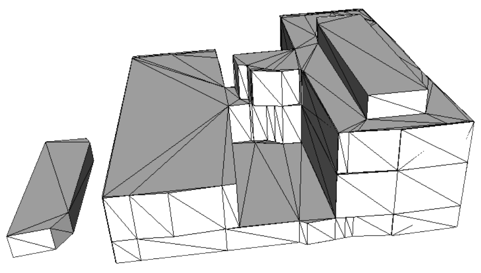
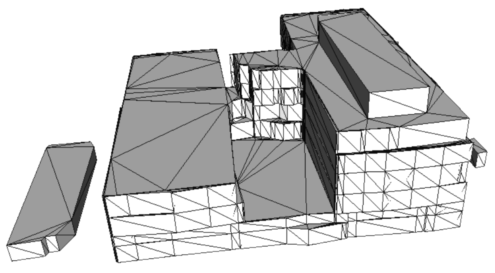
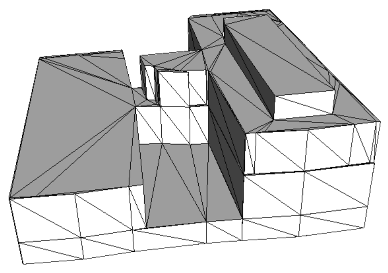
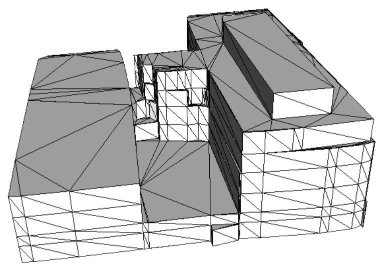

(f) $d = 25$, $\lambda = 7.8$

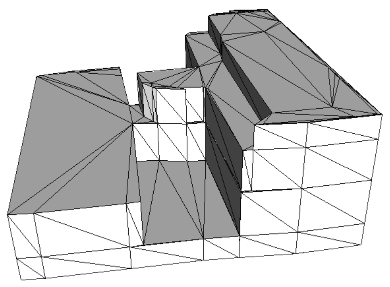
Figure 5.14: *Results of the Centre model.* This model contains rounded orthogonal vertical structures, which just can be approximated by polygonal lines in our approach. However, as you can see, even these structures get approximated well by a polygonal line. In Fig. 5.14(e) and Fig. 5.14(f) parts of the model start to vanish due to over-smoothing.


(a) $d = 10$, $\lambda = 0.0$


(b) $d = 25$, $\lambda = 0.0$


(c) $d = 10$, $\lambda = 1.0$


(d) $d = 25$, $\lambda = 3.5$


(e) $d = 10$, $\lambda = 3.5$

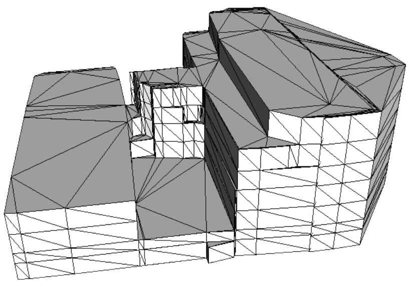
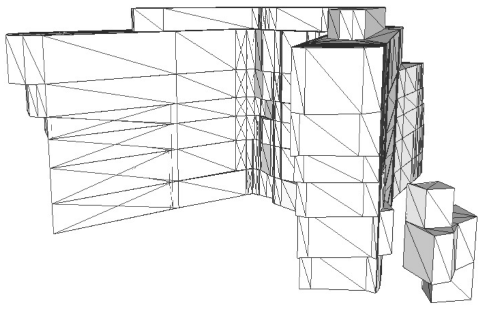
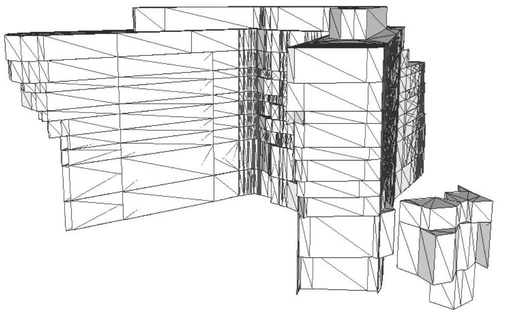
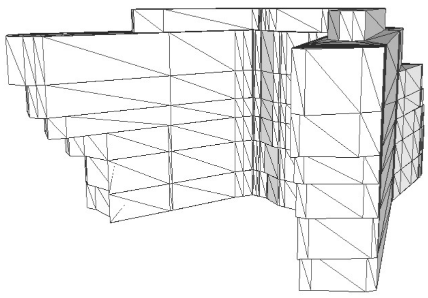
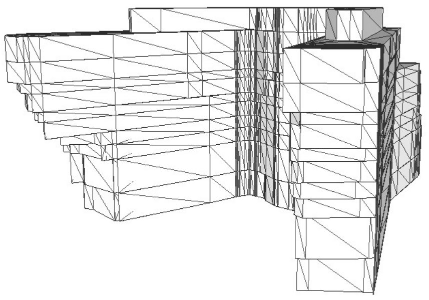

(f) $d = 25$, $\lambda = 4.5$

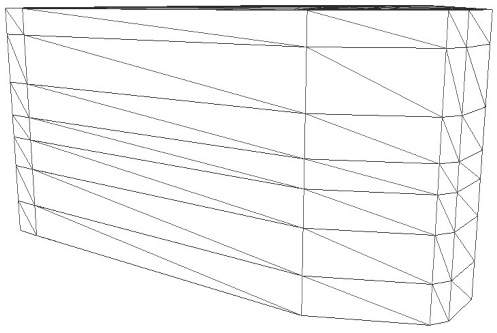
Figure 5.15: *Results of the Aspern model.* As the input model is created using real-world image data, it contains more clutter. However, despite this the reconstructed geometry looks well. With too much smoothing, parts not belonging to the model gets smoothed in the result.


(a) $d = 25$, $\lambda = 0.0$


(b) $d = 35$, $\lambda = 0.0$


(c) $d = 25$, $\lambda = 2.0$


(d) $d = 35$, $\lambda = 2.0$


(e) $d = 25$, $\lambda = 6.0$

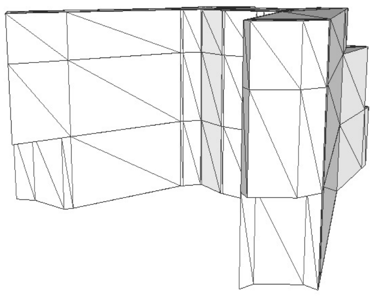

(f) $d = 10$, $\lambda = 2.0$

Figure 5.16: *Results of the Siemenscity model.* This model contains sloped structures, which just are approximated by slicing-based geometric models. However, supposing a sufficiently high level of detail in the vertical direction (i.e., a sufficiently small Mean Shift bandwidth), they can be approximated well by stairway-like structures. When enforcing too much smoothing, the free space between the parts of the model gets smoothed into the model.

d	λ	Dice Score: *median*	*mean*	*std.deviation*	*complexity*
10	0.0	0.9496	0.9507	0.0160	146
10	0.5	0.9538	0.9544	0.0145	112
10	2.0	0.9422	0.9430	0.0200	62
25	0.0	0.9708	0.9677	0.0151	512
25	1.0	0.9791	0.9773	0.0089	282
25	2.0	0.9776	0.9757	0.0091	219
25	6.0	0.8501	0.8224	0.0944	13
35	0.0	0.9743	0.9694	0.0154	1136
35	1.0	0.9827	0.9813	0.0065	657
35	2.0	0.9794	0.9778	0.0091	503

Table 5.3: *Table of Dice scores and complexity scores of the Siemenscity model.* For this model, the Dice score increases with low values for λ due to a small reconstructed object originating from clutter. However, with a higher smoothing the object vanishes and the scores develop as expected.

model, where a higher voxel space size leads to a drastically increased computation time. However, a higher number of slices, which leads to more separate 2D processing steps and more volumetric cells, does not significantly influence the total processing time.

Though, as our main goal was not a perfect processing time performance, there still exist many possibilities to optimize the code. For example, one possibility would be to move the computation of the free space scores, which are computations using a voxel space, to the GPU.

However, having calculated the free space scores it is possible to compute results with varying parameter settings without a big computational cost. For example, when changing the parameter λ, just the 3D Graph Cut part has to be recomputed which can be done in a few seconds (depending on the model). When changing the parameter d more parts have to be recomputed. However, as the free space score computation is by far the most time-consuming part, all the slice computations in 2D and the computations in 3D can be done in about one minute (depending on the model). Therefore, in terms of computational complexity, an algorithm for finding the optimal value for d could be implemented.

Model	# Slices	# Cameras	Voxel Space Size	Computation Time (sec)
Joanneum (d = 10)	4	683	600x542x60	991
Joanneum (d = 25)	13	683	600x542x60	1019
Centre (d = 10)	4	935	600x678x60	1846
Centre (d = 25)	7	935	600x678x60	1869
Aspern (d = 10)	4	151	700x560x80	2907
Aspern (d = 25)	7	151	700x560x80	2978
Siemenscity (d = 10)	3	213	600x604x60	991
Siemenscity (d = 25)	7	213	600x604x60	1049
Siemenscity (d = 35)	11	213	600x604x60	1171

Table 5.4: *Processing time.* All the results are computed on an Intel Xenon X5675 with 16 GBs RAM.

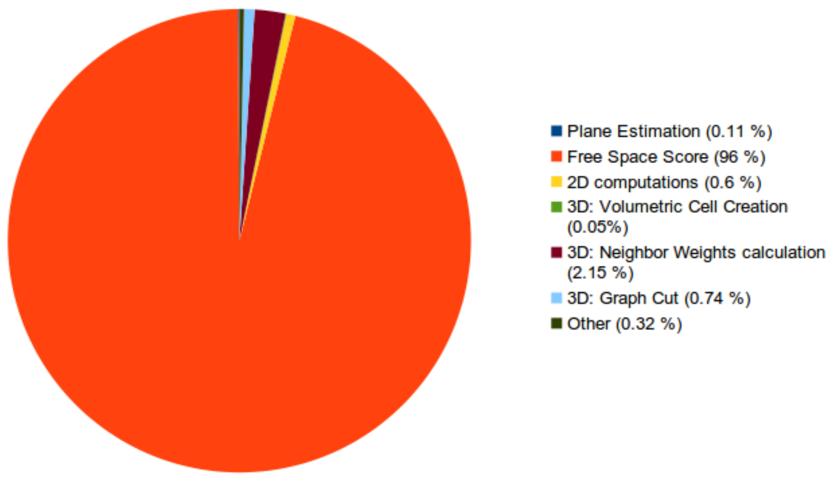


Figure 5.17: *Distribution of processing time for the separate processing steps.* The free space score computation (orange) takes nearly all of the processing time (96%), while the final smoothing (light blue) just needs 0.74% of the total time. The processing time behavior printed in this figure has been observed by processing the Joanneum model with $d = 25$.

Chapter 6

Conclusion and Future Work

In this thesis we have proposed a novel approach for extracting geometric structures from meshed input point clouds dominated by planar horizontal and orthogonal vertical structures. Compared to similar techniques which focus on point clouds acquired by a laser scanner, we have used input models reconstructed from image-based reconstruction methods. Due to a higher amount of clutter and noise in these reconstructions, additional problems arise for which we have found robust solutions. We create geometric abstractions of meshed 3D point clouds by initially partitioning the model into horizontal slices. We perform an optimization-based inside/outside labeling resulting in a floor plan for each slice and finally merge all slices together. For this, we partition the whole scene into volumetric cells based on the floor plans and solve an energy minimization problem. By modifying the smoothness parameter λ and a bandwidth parameter d used for the detection of dominant horizontal structures, the level of detail of the geometric abstraction can be adjusted. In our experiments we have shown that our approach delivers reliable results on noisy input data and that even scenes not fulfilling our requirements completely can be approximated.

In our extensive experimental evaluation we have delivered results based on synthetically generated models and models reconstructed from real-world image data. We have analyzed the complexity and the error with respect to resulting geometric details using different parameter settings. We have shown that our approach delivers reliable results even in the presence of clutter and noise. Even though we do not handle rounded and sloped structures, we have empirically shown that rounded orthogonal vertical structures will be approximated by a polygonal line and even horizontally sloped structures can be approximated by piecewise planar structures.

We further have shown that our approach is computationally efficient and that the most complex part is the free space score calculation, which can be optimized and reused for computations with changed parameter settings.

In future work, one improvement would be to implement an algorithm to find an optimal value for d. As the computational cost of slice extraction and 2D slice computations is relatively small, one possibility would be to increment the value for d until an optimal value has been found. Having automatically calculated d, the level of detail would be adjustable only with one single parameter λ. As recomputing the model with a changed λ just needs some seconds, a level of detail adjustment on-the-fly could be realized. However, the proposed method still needs to be elaborated and evaluated in more detail.

Generally, as the main focus of our work was not to optimize the workflow with respect to processing time, many improvements can still be made in this area. For example, the visibility tests and the distance queries in the voxel space which are needed for the free space score calculation could be implemented efficiently on the GPU or using OpenGL which would massively improve the processing time performance.

For handling large-scale scenes, an additional pre-processing step which partitions the scene into separate objects would be beneficial. Due to varying object properties, this could improve the detection of slice boundaries and consequently the overall reconstruction quality would be improved.

Another promising approach to improve the reconstruction quality and simultaneously add semantic information to the model is to use 2D information propagated to 3D. For example, segmentation methods on the 2D images could be used to detect and classify various segments. This information could be propagated to 3D space leading to a semantically enriched model and to an understanding of the simplified scene representation. Additionally, different levels of detail for differently classified objects could be introduced as well.

Bibliography

[1] CGAL, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[2] MeshLab, an open source, portable, and extensible system for the processing and editing of unstructured 3D triangular meshes. . `http://meshlab.sourceforge.net/`, 2012.

[3] Engineer's Handbook - Types of CAD. `http://engineershandbook.com/Software/cad2.htm`, 2013.

[4] Kinect for Windows - Voice, Movement and Gesture Recognition Technology. `http://www.microsoft.com/en-us/kinectforwindows/`, 2013.

[5] S. Agawal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *Proceedings International Conference on Computer Vision*, 2009.

[6] P. Alliez, S. Tayeb, and C. Wormser. 3d fast intersection and distance computation (aabb tree). In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.3 edition, 2013.

[7] S. Anand, M., V. Singh, and S. Kluckner. Heteroscedastic superpixel segmentation. Siemens internal technical report, 2012.

[8] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *Proceedings European Conference on Computer Vision*, 2006.

[9] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998.

[10] J.-Y. Bouguet. Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[11] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1222–1239, 2001.

[12] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[13] D. Dementhon. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *Center for Automat. Res., U. of Md, College Park*, 2002.

[14] L. R. Dice. *Measures of the amount of ecologic association between species*, volume 26. Ecological Society of America, 1945.

[15] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 1973.

[16] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Communication Association and Computing Machine*, 24(6):381–395, 1981.

[17] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 1962.

[18] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, Jan. 1975.

[19] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[20] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *ACM Trans. on Graphics (SIGGRAPH)*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., New York, 1997.

[21] Google. Google earth, 2014. `http://www.google.com/earth/`.

[22] A. Gupta, A. A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *Proceedings European Conference on Computer Vision*, 2010.

[23] R. Hartley and A. Zisserman. *Multiple View Geometry In Computer Vision*. Cambridge University Press, 2000.

[24] S. Heber, R. Ranftl, and T. Pock. Approximate envelope minimization for curvature regularity. In *Proceedings European Conference on Computer Vision*, 2012.

[25] J. Henrikson. Completeness and total boundedness of the hausdorff metric. *MIT Undergraduate Journal of Mathematics*, 1999.

[26] C. Hoppe, M. Klopschitz, M. Donoser, and H. Bischof. Incremental surface extraction from sparse structure-from-motion point clouds. In *Proceedings British Machine Vision Conference*, 2013.

[27] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, 2006.

[28] S. Kluckner and H. Bischof. Image-based building classification and 3d modeling with super-pixels. In *International Archives of Photogrammetry and Remote Sensing*, 2010.

[29] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? In *Proceedings European Conference on Computer Vision*, 2002.

[30] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *Proceedings International Conference on Computer Vision*, 2007.

[31] D. C. Lee, M. Hebert, and T. Kanade. Geometric reasoning for single image structure recovery. In *Proceedings IEEE Conference Computer Vision and Pattern Recognition*, 2009.

[32] T. N. Limited and Google. Trimble 3d warehouse, 2014. `http://sketchup.google.com/3dwarehouse/`.

[33] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. In *ACM Trans. on Graphics (SIGGRAPH)*, 1987.

[34] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings International Conference on Computer Vision*, 1999.

[35] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, 2009.

[36] D. Nister. An efficient solution to the five-point relative pose problem. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[37] S. Oesau, F. Lafarge, and P. Alliez. Indoor scene reconstruction using primitive-driven space partitioning and graph-cut. In *Eurographics Symposium on Geometry Processing*, 2013.

[38] L. Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.

[39] S. Ramalingam, J. K. Pillai, A. Jain, and Y. Taguchi. Manhattan junction catalogue for spatial reasoning of indoor scenes. In *Proceedings IEEE Conference Computer Vision and Pattern Recognition*, 2013.

[40] L. Roberts. Machine perception of 3-d solids. *PhD. Thesis*, 1965.

[41] M. Singh, V. Singh, S. Anand, and S. Kluckner. Fast statistical approach for semantic 3d modeling of indoor scenes from point cloud data. Siemens internal technical report, 2012.

[42] S. N. Sinha, D. Steedly, and R. Szeliski. Piecewise planar stereo for image-based rendering. In *Proceedings International Conference on Computer Vision*, 2009.

[43] N. Snavely. Bundler: Structure from motion (sfm) for unordered image collections. `http://www.cs.cornell.edu/~snavely/bundler/`.

[44] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring image collections in 3d. In *ACM Trans. on Graphics (SIGGRAPH)*, 2006.

[45] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 2007.

[46] A. Wendel, C. Hoppe, H. Bischof, and F. Leberl. Automatic fusion of partial reconstructions. In *Proceedings International Society for Photogrammetry and Remote Sensing*, 2012.

[47] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Schematic surface reconstruction. In *Proceedings IEEE Conference Computer Vision and Pattern Recognition*, 2012.

[48] J. Xiao and Y. Furukawa. Reconstructing the world's museums. In *Proceedings European Conference on Computer Vision*, 2012.

[49] C. Zach. Fast and high quality fusion of depth maps. In *International Symposium on 3D Data Processing, Visualization, and Transmission*, 2008.

[50] L. Zebedin, J. Bauer, K. Karner, and H. Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *Proceedings European Conference on Computer Vision*, 2008.