Thomas Holzmann

Image-Based Urban 3D Reconstruction

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Horst Bischof
Institute for Computer Graphics and Vision
Graz University of Technology

Dr. Florent Lafarge
Inria Sophia Antipolis Mediterranee

Graz, Austria, Feb. 2019

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.

Stephen Hawking (1942 - 2018)

Abstract

Having overlapping images from a scene, there exist mature 3D reconstruction techniques to reconstruct the scene geometry. These approaches can use a high amount of high-resolution images and are able to reconstruct point clouds or surface models representing the underlying geometry with high accuracy. However, the resulting reconstructions are very often noisy, which leads to visually unappealing results and non-compact models, which need to be described with a big amount of data.

For several applications like mapping services or construction industry, well regularized and visually appealing reconstructions of urban environments are demanded. These reconstructions should also be representable in a compact way in order to be able to transmit, process and visualize them efficiently. In this thesis, two methods are proposed tackling this problem. These methods are able to create well regularized 3D reconstructions of urban environments, which can be represented in a compact way.

The first method is a slicing-based approach: It partitions the scene into horizontal slices and computes a 2D outline for every slice. Then, using the computed slice outlines, an irregularly shaped volumetric cell decomposition is created covering the whole scene. Finally, an inside/outside labeling for every cell is computed by solving an energy minimization problem with energy terms covering visibility information, vicinity to input data and detected line segments. As a result, this method delivers compact building models consisting of planar surfaces and sharp edges.

The second method is a more generic reconstruction method, which is based on a tetrahedralization of the scene. Planes are detected within the scene and incorporated in the tetrahedralization. For this, a plane detection algorithm using 3D lines is proposed which improves the plane detection results especially for poorly textured urban environments. Then, an energy minimization problem is solved, which labels each tetrahedron as inside or outside. As energy terms, visibility-based energy and semantic class-dependent terms are used, which favor Manhattan-like structures at buildings and a smooth reconstruction of the surroundings. As a result, this method delivers visually appealing reconstructions of urban scenes containing buildings reconstructed by planar surfaces and sharp edges, and surroundings of buildings reconstructed by a smooth surface.

In our experiments, we compare the results of the proposed methods with state-of-the-art methods and show in detail the effects of the individual contributions. We show that the slicing-based approach creates very compact and regularized building models which still cover the most important details. Further, we show that this method is very robust to erroneous input data and can produce regularized results even if the input contains large errors. We show that the proposed tetrahedra-based method generates visually appealing and compact 3D reconstructions of urban environments containing buildings modeled mainly by planes and having sharp outlines while still being able to reconstruct fine details whereas comparable methods either create over-smoothed or over-simplified results.

Kurzfassung

Mit dem aktuellen Stand der Technik entsprechenden 3D Rekonstruktionsmethoden kann ausgehend von überlappenden Bildern einer Szene die Szenengeometrie rekonstruiert werden. Diese Methoden können eine hohe Anzahl von hochauflösenden Bildern verarbeiten und Punktwolken oder Oberflächenmodelle erstellen, die die zugrundeliegende Geometrie mit hoher Genauigkeit widerspiegeln. Jedoch sind die resultierenden Rekonstruktionen oft verrauscht, was zu visuell nicht ansprechenden Ergebnissen und zu nicht kompakten Modellen führt. Diese müssen wiederrum mit einer großen Menge an Daten beschrieben werden.

Für verschiedene Anwendungen (z.B. für Kartierungsdienste oder in der Baubranche) sind gut regularisierte und visuell ansprechende Rekonstruktionen von urbanen Umgebungen gewünscht. Diese Rekonstruktionen sollen auch in einer kompakten Weise repräsentierbar sein, um sie effizient übertragen, prozessieren oder visualisieren zu können. In dieser Dissertation werden zwei Methoden vorgestellt, die dieses Problem behandeln. Diese Methoden können gut regularisierte 3D Rekonstruktionen von urbanen Umgebungen erstellen, welche in einer kompakten Weise repräsentiert werden können.

Die erste Methode ist ein scheibenbasierter Ansatz: Sie unterteilt die Szene in horizontale Scheiben und berechnet den 2D Umriss von jeder Scheibe. Unter Verwendung dieser berechneten Scheibenumrisse wird folgend eine Szenenunterteilung in unregelmäßig geformte volumetrische Zellen, die die ganze Szene beinhaltet, erstellt. Als letzter Schritt wird durch Lösen eines Energieminimierungsproblems, das die Sichtbarkeitsinformation, die Nähe zu den Eingangsdaten und detektierte Liniensegmente berücksichtigt, jede dieser Zellen als innen oder außen markiert. Als Ergebnis liefert diese Methode kompakte Gebäudemodelle bestehend aus planaren Oberflächen und scharfen Kanten.

Die zweite Methode ist eine generischere Rekonstruktionsmethode und basiert auf einer Tetrahedralisierung der Szene. Ebenen werden in der Szene detektiert und in die Tetrahedralisierung eingearbeitet. Da punktbasierte Ebenendetektoren in schlecht texturierten urbanen Umgebungen nicht immer zufriedenstellend funktionieren, verwendet unsere vorgeschlagene Ebenendetektionsmethode 3D Linien, welche in solchen Umgebungen konsistenter rekonstruiert werden können. Als nächster Schritt wird ein Energieminimierungsproblem gelöst, das jeden Tetraeder als innen oder außen markiert. Als Energieterme werden sichtbarkeitsbasierende Energie und semantische klassenabhängige Terme verwendet, die Manhattan-ähnliche Strukturen bei Gebäuden und eine glatte Rekonstruktion der Umgebung der Gebäude bevorzugen. Diese Methode liefert visuell ansprechende Rekonstruktionen von urbanen Szenen als Ergebnis, in denen Gebäude hauptsächlich mit planaren Oberflächen und scharfen Kanten und die Umgebungen von Gebäuden mit glatten Oberflächen repräsentiert sind.

In den Experimenten vergleichen wir Ergebnisse von den vorgestellten Methoden mit anderen, dem neuesten Stand der Technik entsprechenden Methoden und diskutieren die Effekte der individuellen Beiträge dieser Arbeit. Wir illustrieren, dass der scheibenbasierende Ansatz sehr kompakte und regularisierte Gebäudemodelle erstellt, welche noch immer die wichtigsten Details enthalten. Weiters zeigen wir, dass diese Methode sehr robust gegenüber fehlerhaften Eingangsdaten ist und auch regularisierte Ergebnisse erzeugen kann, wenn die Eingangsdaten große Fehler beinhalten. Wir zeigen, dass die vorgeschlagene tetraederbasierende Methode visuell ansprechende und kompakte 3D Rekonstruktionen von urbanen Umgebungen erzeugt. In diesen werden Gebäude hauptsächlich als Ebenen und mit scharfen Kanten repräsentiert und gleichzeitig auch feine Details rekonstruiert, wohingegen vergleichbare Methoden entweder zu stark geglättete oder zu vereinfachte Ergebnisse erzeugen.

Affidavit

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 indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Acknowledgments

First I would like to thank Prof. Horst Bischof for giving me the opportunity to pursue a PhD degree under his supervision. Further, I want to thank Ass.-Prof. Friedrich Fraundorfer for his continuous assistance and many fruitful discussions. In general, it was a pleasure to pursue a PhD in such an inspiring atmosphere at the ICG and having many people with different specializations within the area of computer vision around me.

Then, I also want to thank my second examiner, Florent Lafarge, for taking the time to read my thesis and come the long way to Graz for attending my defense. It is an honor having him as an examiner, as several research ideas presented in this thesis were inspired by his work.

More general, I want to thank my colleagues at ICG (and especially my office colleagues and all aerial vision group members) not only for research discussions but also for the social factor at the institute. It was great talking with you also about random stuff within the coffee breaks or during other social activities.

I also want to thank Martin R. Oswald from ETH Zurich. Together with him, several ideas were developed which led to publications during my studies. Further, I want to thank Stefan Kluckner from Siemens. During the time working together with him at Siemens, he motivated me for pursuing a PhD at the ICG.

Last but not least, I want to thank my family and friends who always supported me during my studies. Especially, I want to thank Gudrun for supporting me all the time and going with me through all the highs and lows of my PhD time while simultaneously founding a family and giving birth to our daughter Lara.

This work has mainly been funded by the Austrian Science Fund (FWF) in the project V-MAV (I-1537). In addition, I further received funding from the following institutions/companies during my time as PhD student: From the Austrian Research Promotion Agency(FFG) within the project FreeLine (Bridge1, 843450), the European Union for working on the 3D-pitoti project (Seventh Framework Programme, 600545) and from Sony for working on the project SemanticSfM.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Contributions . . . 5

1.3 Outline . . . 6

2 Image-Based 3D Reconstruction **7**

2.1 Basics and Notation . . . 7

2.1.1 Camera Model . . . 8

2.1.2 Two-View Geometry . . . 10

2.2 Structure-from-Motion . . . 12

2.2.1 Feature Extraction and Matching . . . 13

2.2.2 Geometric Verification . . . 13

2.2.3 Incremental SfM . . . 14

2.2.4 Global SfM . . . 14

2.2.5 Example Pipelines . . . 15

2.3 Multi-View Stereo Reconstruction . . . 15

2.4 Line Reconstruction . . . 16

2.5 Surface Reconstruction . . . 17

2.5.1 Poisson Surface Reconstruction . . . 17

2.5.2 Tetrahedralization-Based Methods . . . 20

2.5.2.1 Delaunay Triangulation . . . 20

2.5.2.2 Surface Reconstruction . . . 20

2.5.3 Methods Using a Signed Distance Function . . . 23

2.6 Summary . . . 23

3 Related Work **25**
3.1 Reconstruction Using a Scene Hypothesis . . . 25
3.2 Shape Priors Incorporated in Generic Reconstruction Methods . . . 28
3.3 Semantic Scene Reconstruction . . . 30
3.4 Summary . . . 31

4 Slicing for Building Reconstruction **33**
4.1 Method Overview . . . 34
4.2 Input Data . . . 35
4.3 Horizontal Slicing and Cell Decomposition . . . 36
4.3.1 Horizontal Slicing . . . 36
4.3.2 Binary Labeling . . . 37
4.3.3 Slice Combination . . . 38
4.4 Volumetric Cell Labeling as an Energy Minimization Problem . . . 40
4.5 Summary . . . 44

5 Shape Priorization in Tetrahedra-Based Methods **45**
5.1 Method Overview . . . 47
5.2 Plane Detection . . . 48
5.2.1 Point-Based Plane Detection . . . 49
5.2.2 Line-Based Plane Detection . . . 49
5.2.2.1 Line Triple Detection . . . 50
5.2.2.2 Line Triple Clustering . . . 50
5.2.2.3 Inlier Detection and Outline Estimation . . . 51
5.2.2.4 Plane Filtering . . . 51
5.2.3 Plane-Based Denoising . . . 51
5.3 Tetrahedralization of the Scene . . . 51
5.4 Tetrahedra Subdivision . . . 52
5.4.1 Tetrahedra Intersected by Plane . . . 52
5.4.2 Consistency Adoptions . . . 52
5.4.3 Margin Tetrahedra . . . 52
5.5 3D Reconstruction Using Tetrahedral Occupancy Labeling . . . 55
5.6 Visibility-Based Energy Computation . . . 56
5.6.1 Unnormalized Energy Formulation . . . 56
5.6.1.1 Unary Costs . . . 56
5.6.1.2 Pairwise Costs . . . 56
5.6.2 Normalized Energy Formulation . . . 56
5.6.3 Computation of Visibility Information . . . 58
5.6.3.1 Using Pre-Computed Visibility Information . . . 58
5.6.3.2 Visibility Computation Using Depth Maps . . . 59
5.7 Plane-Aware Regularization . . . 60

5.7.1 Manhattan Regularity Term 60
5.7.2 Level of Detail Term 61
5.7.3 Plane Intersection Artifacts Removal 61
5.8 Semantically Aware Urban Reconstruction 62
5.8.1 Semantic Segmentation 62
5.8.2 Input Data Subdivision and Semantic Preprocessing 63
5.8.3 Semantically Varying Smoothness Terms 64
5.9 Summary 65

6 Experiments **67**
6.1 Input Data 67
6.1.1 Block Building 68
6.1.2 House 69
6.1.3 Residential Area 70
6.2 Implementation Details 71
6.3 Default Parameters 71
6.4 Results and Comparisons 72
6.4.1 Runtimes 75
6.5 Method-Specific Detailed Evaluations 78
6.5.1 Slicing-Based Method 78
6.5.1.1 Parameter Selection 78
6.5.1.2 Block Building 79
6.5.1.3 Long Building 82
6.5.1.4 Non-Manhattan Building 83
6.5.1.5 Runtime Discussion 85
6.5.2 Plane-Based Regularization Method 85
6.5.2.1 Block Building 86
6.5.2.2 Building Complex 88
6.5.2.3 House 89
6.5.2.4 Indoor 90
6.5.2.5 Entry-P10 90
6.5.2.6 Runtime Breakdown 91
6.5.3 Semantically Aware Regularization Method 94
6.5.3.1 Plane Detection Using Lines 94
6.5.3.2 Normalized Visibility-Based Energy Term 95
6.5.3.3 Semantic 3D Data 98
6.5.3.4 Semantic Priors 99
6.5.3.5 Simplification as Post-Processing 99
6.6 Summary 100

7 Conclusions **105**
7.1 Summary 105
7.2 Outlook 107

A List of Acronyms **111**

B List of Publications **113**
B.1 2018 113
B.2 2017 114
B.3 2016 115
B.4 2015 116
B.5 2014 117

Bibliography **119**

List of Figures

1.1 Example reconstructions form commercial reconstruction pipelines 2
1.2 Reconstruction in Google Maps 3
1.3 Reconstruction from Matterport 4

2.1 Pinhole camera model 8
2.2 Epipolar geometry 10
2.3 Example SfM result 13
2.4 Reconstruction from PMVS2 16
2.5 Line reconstruction 17
2.6 Poisson surface reconstruction 18
2.7 Oriented points and Poisson indicator function 18
2.8 Voronoi diagram and Delaunay triangulation 21
2.9 Visibility-based energy computation 22
2.10 Delaunay triangulation-based reconstruction 22

3.1 Illustrations from (Schnabel et al., 2007) 26
3.2 Illustrations from (Xiao et al., 2014) 27
3.3 Illustrations from (Lafarge et al., 2013) 29
3.4 Illustrations from (Häne et al., 2013) 30

4.1 Meshed dense and slicing-based regularized model 35
4.2 Method overview of slicing-based method 35
4.3 Slice boundary detection 36
4.4 Slice labeling 37
4.5 Volumetric cell decomposition and slice extrusion 39
4.6 Constrained Delaunay triangulation 39
4.7 Line detections and backprojected facet 42

4.8 Line distance computation 43

5.1 Textured results of the tetrahedra-based method 46
5.2 Overview of the processing pipeline of the tetrahedra-based method 47
5.3 Line-based plane detection 50
5.4 Tetrahedra intersected by plane - overview intersection cases 53
5.5 Tetrahedra intersected by plane - consistency adaptions 3-point intersection 54
5.6 Tetrahedra intersected by plane - consistency adaptions 4-point intersection 54
5.7 Tetrahedra intersected by plane - subdivision of margin tetrahedra 55
5.8 Normalized visibility-based energy computation 57
5.9 Manhattan regularity term 60

6.1 Images of the Block Building 68
6.2 Images of the House dataset 69
6.3 Ground truth point cloud 69
6.4 Point clouds of evaluation scenes 70
6.5 Images of the Residential Area dataset 70
6.6 Results of commercial reconstruction pipelines 72
6.7 Results and comparisons with state-of-the-art 74
6.8 Ground truth point errors 76
6.9 Meshed dense reconstruction and regularized model of Block Building . . . 79
6.10 Results of Block Building with PMVS2 80
6.11 Results of Block Building with SfM 81
6.12 Images of Long Building 82
6.13 Results of Long Building 83
6.14 Back side of Long Building 84
6.15 Images of Non-Manhattan Building 84
6.16 Results of Non-Manhattan Building 85
6.17 Comparison of generic reconstruction to plane-based regularization method 86
6.18 Results with varying level of detail of Block Building 87
6.19 Comparison to others on Block Building 88
6.20 Comparison to others on Building Complex 89
6.21 Plane-based regularization method with different level of detail settings . . 90
6.22 Comparison to others on House dataset 91
6.23 Comparison to others on Indoor dataset 92
6.24 Comparison to others on Entry-P10 dataset 93
6.25 Runtime plane-based regularization method 93
6.26 Results of semantically aware method 94
6.27 Comparison of line-based and point-based plane detection 95
6.28 Comparison of line-based plane detection to others 96
6.29 Error visualization of normalized and unnormalized visibility-based energy . 98

6.30 Evaluation of individual changes of visibility-based energy 101
6.31 Semantically labeled images 102
6.32 Semantically labeled point clouds 102
6.33 Comparison of results with/without semantics 103
6.34 Simplification as post-processing 103

List of Tables

6.1 Error comparisons with state-of-the-art 73
6.2 Runtimes 77
6.3 Quantitative comparison of plane detection 97
6.4 Errors of normalized and unnormalized visibility-based energy 97
6.5 Errors of individual changes of visibility-based energy 99

1 Introduction

Contents

1.1 Motivation 1
1.2 Contributions 5
1.3 Outline 6

1.1 Motivation

The research area of image-based 3D reconstruction has a long history and was originally developed as a sub-discipline of geodesy, namely photogrammetry. As the physicist Dominique Francois Arago (1786 - 1853) presented the invention of photography in 1839 to the Académie Française, he already mentioned the possibilities for measuring using photographic images. 20 years later, first trials were made to use images for topographic purposes. Subsequently, several researchers investigated in the topic of using images for measuring applications during the second half of the 19th century. At the end of the 19th century, photogrammetry was already used for creating measurements from historic monuments or mountains and glaciers. Starting from the beginning of the 20th century, aerial images were used for photogrammetry in order to create topographic maps of cities [2, 20].

In the late 1960s, a new research field, namely computer vision emerged in which experts in the field of artificial intelligence aimed to make a computer see. Initially, this problem was thought to be solved within a student summer project. 52 years later, we now know that this problem cannot be solved that easily. In contrast to the already existing field of digital image processing, computer vision aimed at understanding the whole scene in 3D.

Several algorithms which are still the basis for algorithms used today were developed in the consecutive decades. For example, early work was done on edge extraction, line labeling, stereo correspondence search, optical flow or Structure from Motion (SfM). In

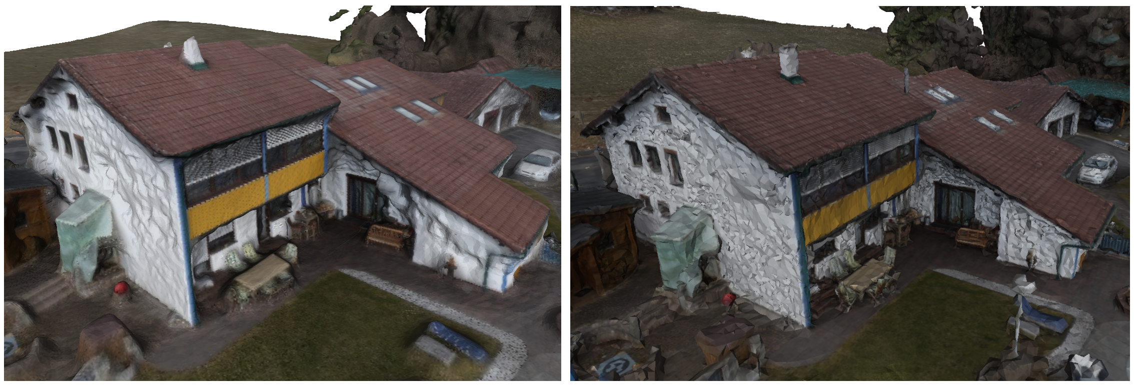


Figure 1.1: *Example reconstruction results from commercial reconstruction software: Agisoft [1] (left) and Pix4D [75] (right).* Using images taken from an Unmanned Aerial Vehicle (UAV) as input, these two commercial 3D reconstruction pipelines can reconstruct the scene accurately and with high detail. However, especially at poorly textured façades, the reconstructed surface gets very noisy. Further, edges of the building are reconstructed very smooth and not as sharp edges, with would be favored for buildings with planar shapes.

the 1990s, the field started using full global optimization for camera calibration, which was later recognized as being the same as the *bundle adjustment* technique used in the area of photogrammetry [88].

With this long history in the field of computer vision and photogrammetry, the research area of image-based 3D reconstruction has reached a high level of maturity. Having this knowledge, reconstruction pipelines evolved, with which it is possible to reconstruct big scenes with high accuracy from hundreds or even thousands of high resolution images. Typical 3D reconstruction pipelines consist of several processing steps: In the first step, which is called *SfM*, the relative camera poses are computed using sparse keypoint matches. Then, a Multi-View Stereo (MVS) approach is applied, which computes dense depth information for every camera. The depth information from all cameras is consecutively projected and fused in 3D to get a dense point cloud representation of the scene. Finally, a surface mesh can be generated using the dense point cloud in order to get a surface representation of the scene.

Nowadays, there exists software for 3D reconstruction which includes all these steps within the processing pipeline: There exist commercial products (e.g., Agisoft [1], Pix4D [75]) and open-source solutions (e.g., Colmap [83]), which take images as input and reconstruct large-scale scenes as dense point clouds or surface models, as can be seen in Figure 1.1.

However, due to errors and clutter in image-based measurements and the lack of information at parts of the scene (e.g., at untextured surfaces like white façades, see Figure 1.1), such reconstruction techniques often produce noisy or incomplete point clouds and surfaces. Further, to describe such a noisy reconstruction with 3D points or a surface, a huge amount of data (i.e., many points or vertices/faces) is needed.

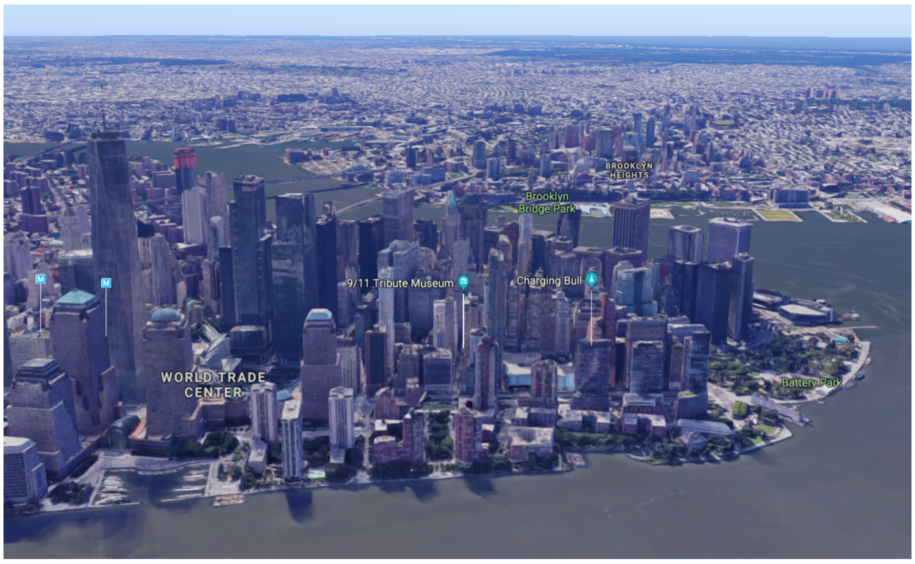


Figure 1.2: *Reconstruction of Manhattan in Google Maps [25].* Within Google Maps it is possible to visualize 3D reconstructions of whole cities. For this, a compact representation of the reconstructed surface is required which is still accurate and, simultaneously, visually appealing. Image taken from [25].

For many applications like, for example, visualization of urban environments, visually appealing urban reconstructions having little noise, planar surfaces and sharp edges are demanded. Additionally, it should be possible to represent the reconstruction with a small amount of data to be able to process, transmit, visualize and store the reconstructions in an efficient way. Further, an adjustable level of detail of the final reconstruction is desired: It should be possible to represent the whole scene just by a small set of geometric primitives, in case a lightweight representation for simple transmission and further processing is wanted. However, in case a more detailed representation of the scene is desired, it should be easily possible to change this level of detail of the reconstruction. Probably the most famous example of large-scale urban 3D reconstruction and visualization of urban environments is Google Maps [25], which includes reconstructions of whole cities and visualizes them (see Figure 1.2) within a web browser. To achieve a good user experience, it needs reconstruction techniques to produce visually appealing and still accurate models which are also compact in order to easily process the data, transmit it via the Internet and visualize it within a browser.

There exist several other application scenarios where similar requirements are defined: For example, with the increasing accuracy and popularity of image-based 3D reconstruction techniques and the availability of low-cost scanning devices, the construction industry includes 3D reconstruction techniques for planning, building and verification purposes into project work cycles. They aim to correct and update original building plans which are often either not available or outdated. Having abstracted and simplified reconstructions of buildings eases this processing step. Further, real estate companies aim to create vi-

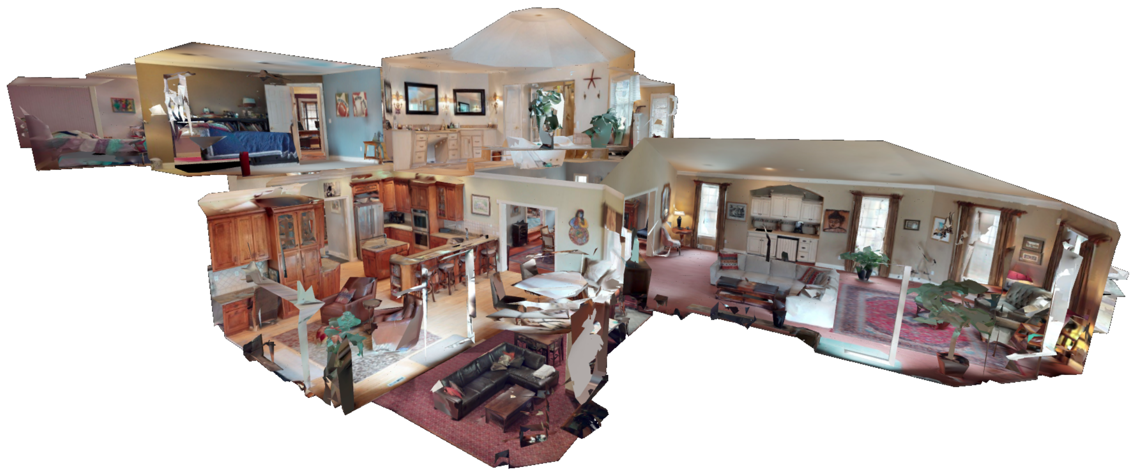


Figure 1.3: *Reconstruction of an indoor scene with Matterport [59].* Matterport creates 3D models from indoor and outdoor scenes using an active 3D scanning device, like a structured light or a laser scanner. After being processed, the user can explore the scene using a *VR* device or in the browser. Image from Pennsylvania Craftsman Home, taken from [59].

sually appealing reconstructions of their offered houses such that potential customers can virtually walk around and into houses. For all these applications, compact models similar to ones created by CAD software is desired.

Another application scenario is the currently emerging field of Virtual Reality (VR) or, more general, computer games: Many companies now aim at creating virtual environments from the real world with the goal to simulate real-world environments or just to ease the process of modeling large scenes. In order to be usable and processable, one cannot just use a detailed reconstruction containing millions of points and faces. A more compact yet still accurate representation is necessary, which can be easily stored and processed.

Usually, state-of-the-art image-based 3D reconstruction pipelines create detailed reconstructions using images as input. However, they are not well suited for creating compact and yet visually appealing reconstructions. Other commercial solutions use active 3D scanners in order to reconstruct indoor and outdoor environments for being visualized later on in the browser or with a *VR* device (see Figure 1.3). However, they also do not solve the problem of creating visually appealing and, simultaneously, compact 3D models even though they are using expensive sensors for capturing.

For such applications, up to our knowledge no available software solution exists which takes only images as input and produces visually appealing and compact 3D reconstructions while still having high reconstruction accuracy. This motivated us to investigate in this topic and elaborate reconstruction techniques to fulfill these requirements.

1.2 Contributions

In this thesis, we contribute with urban 3D reconstruction techniques, which create visually appealing and compact reconstructions of urban environments as result. Such urban environments contain buildings and surroundings of buildings including vegetation, streets and pavement. We define the following criteria for a visually appealing urban 3D reconstruction, which are usually not tackled by current 3D reconstruction approaches:

- *Planar shape prior:* When planar and nearly planar surfaces exist in the scene (e.g., façades, roofs), they should also be reconstructed as planar surfaces.
- *Straight building outlines:* Edges of buildings should be straight and represented by a straight line (i.e., no noisy edges).
- *Detailed buildings and smoothed surroundings:* As for several applications (e.g., real estate companies) detailed building reconstructions are required, details should be kept while regularizing with a plane prior. Simultaneously, the surrounding of buildings does not necessarily be reconstructed with high details but with a smoothed, visually appealing surface.

We contribute with a slicing-based method, which aims to model buildings using the assumption that they can be represented as a set of stacked horizontal slices. It does this by partitioning the scene into horizontal slices, creates irregularly shaped volumetric cells using the slices and computes an inside/outside labeling of the volumetric cells by solving an energy minimization problem using Graph Cuts [10]. Additionally, it uses line cues to improve the reconstruction. As a result, this method delivers compact and geometrically abstracted building models containing mainly planar surfaces and sharp edges. However, the surrounding of buildings is usually not modeled with this method.

Further, we contribute with a method which makes usage of the more general assumption that big parts of the scene can be represented by detected planes in the scene. This method partitions the scene into tetrahedra and computes an inside/outside labeling of every tetrahedron. Detected planes in the scene are integrated within the tetrahedral cell decomposition and additionally semantic information is used to treat different parts of the scene differently. We contribute with a plane detection algorithm, which uses 3D line segments as input, which improves the detection result in poorly textured urban scenes. Additionally, we contribute with a normalized visibility-based energy formulation which eases the combination of multiple energy terms within an energy minimization problem. As a result, the tetrahedra-based method is able to deliver visually appealing reconstructions of buildings and their surroundings, where the building parts are mainly represented by planar surfaces and sharp edges, and the surroundings by smooth surfaces. As a post-processing step, data reduction can be done without loosing accuracy in order to create compact representations of the scene.

1.3 Outline

In Chapter 2 we describe the basic algorithms to compute relative poses of cameras and sparse point clouds, to compute dense point clouds and line reconstructions, and finally how to compute surface models. Next, in Chapter 3 we discuss in detail related work within the area of urban 3D reconstruction. In Chapter 4 and Chapter 5 we describe our proposed methods for creating urban 3D reconstructions and in Chapter 6 a detailed evaluation of all the contributions is presented and results are compared with other state-of-the-art methods. Finally, we discuss conclusions in Chapter 7.

2 Image-Based 3D Reconstruction

Contents

2.1 Basics and Notation 7
2.2 Structure-from-Motion 12
2.3 Multi-View Stereo Reconstruction 15
2.4 Line Reconstruction 16
2.5 Surface Reconstruction 17
2.6 Summary 23

Due to the long history of image-based 3D reconstruction, mature reconstruction methods evolved which are able to reconstruct scenes with different representations like point clouds or surface models. Usually, a reconstruction pipeline contains several processing steps: In the first step, the goal is to compute the camera poses from which input images were taken, and simultaneously reconstruct the scene geometry (which is called Structure from Motion (SfM)). Having this information, it is possible to do further processing steps in order to get a more precise geometry reconstruction: One can reconstruct a dense point cloud using a Multi-View Stereo (MVS) algorithm or a reconstruction consisting of 3D line segments. Using this more precise geometry, it is then possible to create a surface model of the whole scene.

In this chapter, we first discuss some basics and notation conventions used in image-based 3D reconstruction. Then, we review all the above mentioned processing steps used in 3D reconstruction pipelines in more detail.

2.1 Basics and Notation

In this section, we discuss some basic conventions and notations used for image-based 3D reconstruction. We describe a camera model commonly used and its parametrization.

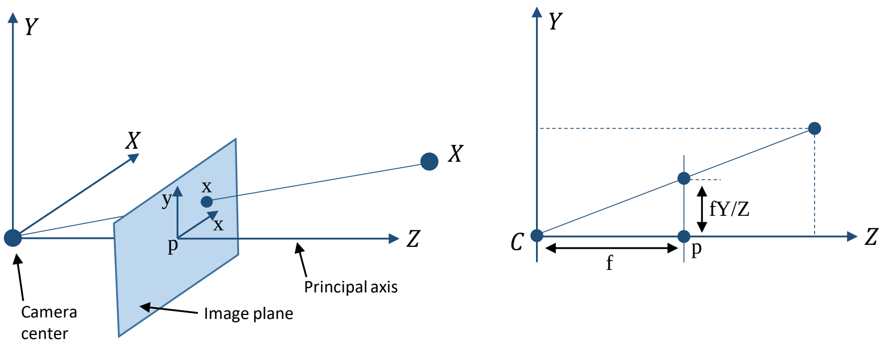


Figure 2.1: Illustration of the pinhole camera model. Left, one can see the camera center placed at the origin with the image plane in front of it. The 3D point X is projected to the 2D point x on the image plane. Right, the mapping using similar triangles is illustrated, with which a mapping from a 3D point to a point on the image plane can be computed.

Then, we discuss the two-view geometry relation, which is used to estimate relative poses between cameras.

2.1.1 Camera Model

One of the camera models frequently used is called the *pinhole camera*, which is a specialization of the general projective camera. According to [31], a mapping of 3D scene points to image coordinates is defined as follows: Considering a central projection and defining as center of projection the origin of an Euclidean coordinate system, a point in space with coordinates $\mathrm{X} = (X, Y, Z)^T$ is mapped to a point on the plane $Z = f$, which is called the *image plane* or *focal plane*. This mapped point on the image plane is computed by intersecting the image plane with a line joining the point X with the center of projection C, which is called *camera center* or *optical center*. The line going perpendicularly through the image plane and originating at the camera center is called *principal axis* or *principal ray*, and the point where the principal axis intersects with the image plane is called *principal point* p. Figure 2.1 illustrates these geometric relations.

Using similar triangles, one can compute a mapping between a 3D point to a 2D point on the image plane (also see Figure 2.1):

$$(X, Y, Z)^T \longmapsto (fX/Z, fY/Z)^T, \tag{2.1}$$

which is a mapping from Euclidean 3-space $\mathbb{R}^3$ to Euclidean 2-space $\mathbb{R}^2$.

By using the projection described in Equation 2.1, it is assumed that the origin of the coordinates in the image plane is at the principal point. However, in practice there is an offset and, hence, a mapping

$$(X, Y, Z)^T \longmapsto (fX/Z + p_x, fY/Z + p_y)^T, \tag{2.2}$$

where $(p_x, p_y)^T$ are the coordinates of the principal point, can be defined. By expressing this equation in homogeneous coordinates, one gets the following equation:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.3}$$

When writing,

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}, \tag{2.4}$$

then Equation 2.3 can be written as

$$\mathbf{x} = K[I \mid 0]X_{cam}, \tag{2.5}$$

where $\mathbf{x}$ is the 2D point on the image plane and X_{cam} is the 3D point expressed in the *camera coordinate frame*. Further, $[I \mid 0]$ defines the 3x3 identity matrix concatenated with a 3x1 zero vector and the matrix K is called the *camera calibration matrix* or *intrinsic calibration matrix*.

Contrary to the 3D point X_{cam}, which is expressed in the camera coordinate frame, points in space are usually expressed in the *world coordinate frame*. This two coordinate frames are related via a rotation and a translation, also known as *extrinsic camera calibration*.

Having $\tilde{X}$ representing an inhomogeneous 3-vector representing the coordinates of a point in the world coordinate frame, and $\tilde{X}_{cam}$ representing the same point in the camera coordinate frame, then there exists a transformation $\tilde{X}_{cam} = R(\tilde{X} - \tilde{C})$, which transforms the point $\tilde{X}$ in the world coordinate frame into the camera coordinate frame. $\tilde{C}$ represents the coordinates of the camera center in the world coordinate frame, and R is a 3×3 rotation matrix representing the orientation of the camera coordinate frame. Writing this equation in homogeneous coordinates, one gets

$$X_{cam} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} X. \tag{2.6}$$

By combining this equation with Equation 2.5, the mapping of a 3D point in the world coordinate frame onto the image plane can be computed:

$$\mathbf{x} = KR[I \mid -\tilde{C}]X, \tag{2.7}$$

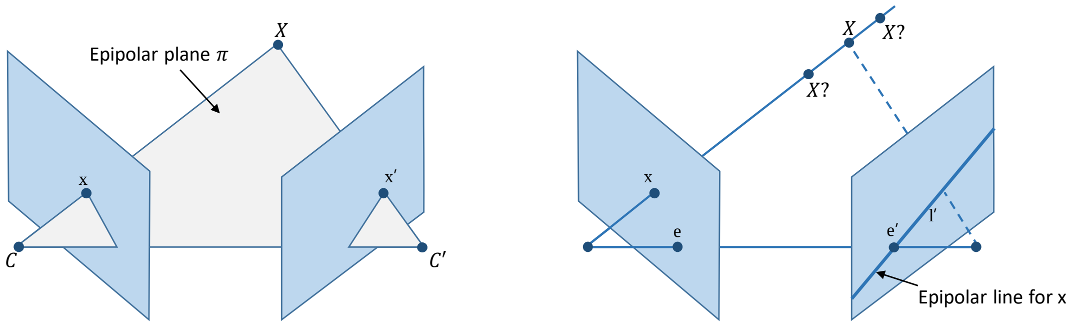


Figure 2.2: Illustration of the epipolar geometry relations. Left, one can see that the points on the image plane **x** and **x'** are coplanar with their corresponding 3D point X and the camera centers C and C'. Right, the relationship between the image point x in the left image and the corresponding epipolar line l' in the right image is illustrated: The image point x backprojected to 3D has to lie on a ray, which projects onto the epipolar line l' going through the epipole e'.

where R and $\tilde{C}$ are called the *external* or *extrinsic* camera parameters.

2.1.2 Two-View Geometry

Having two overlapping views acquired by a camera, it is possible to reconstruct 3D information using the images by triangulating corresponding structure from both images. For this, the geometric relation between both views need to be known. The *epipolar geometry* is the intrinsic projective geometry between two views and describes this relation. It is independent of scene structure and only depends on the cameras' internal parameters and relative pose. In the following, we discuss this epipolar geometry constraint in detail and derive the *fundamental matrix* from it. Finally, we describe algorithms which are used to estimate the fundamental matrix from point correspondences.

Epipolar Geometry. As described in [31], the epipolar geometry between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline, which is the line joining the two camera centers, as axis. This geometry is motivated by considering the search for corresponding points in two images.

Having a 3D point X and corresponding 2D points on the image planes, **x** and **x'**, it can be observed that these points and the camera centers are coplanar (see Figure 2.2, left). This plane is called *epipolar plane*, the *epipole* is the point of intersection of the line joining the camera centers with the image plane, and the *epipolar line* is the intersection of an epipolar plane with the image plane. Hence, a corresponding point for a point in one image in the second image has to lie exactly on the corresponding epipolar line (see Figure 2.2, right).

Fundamental Matrix. The algebraic representation of this geometric relationship is the *fundamental matrix* F. It is a 3×3 matrix of rank 2 with 7 degrees of freedom and satisfies the relation

$$x'^T F x = 0 \tag{2.8}$$

for two points x and x' lying on the first and second image plane. Obviously, having one point fixed, the second one does not have to be on an exact, pre-defined location but has to lie on a point along a line. For example, having a point x, the corresponding point x' in the second image has to be on the epipolar line l'. Hence, the epipolar line can be constructed with the equation

$$l' = Fx, \tag{2.9}$$

where l' is the epipolar line on the second image plane and x a point on the first image plane. Further, the fundamental matrix satisfies the equation

$$det(F) = 0. \tag{2.10}$$

A specialization of the fundamental matrix is the essential matrix E. In comparison to the fundamental matrix, the essential matrix assumes calibrated camera points (i.e., points being multiplied by the intrinsic calibration matrix K).

There exist several algorithms for calculating the fundamental matrix F (or their specialization, the essential matrix E) from point correspondences of two images, from which the relative pose between two cameras can be extracted. Depending on the algorithm, a different number of points is needed. For example, a well known algorithm for estimating the fundamental matrix is the normalized 8-point algorithm as described in [31]. Another method for estimating the essential matrix, assuming a calibrated camera setup, is the 5-point algorithm presented in [68]. Such an algorithm is then usually used within a Random Sample Consesus (RANSAC) [19] scheme, where many hypotheses for the relative orientation are created and scored by a robust statistical measure over all points in two or more views.

5-Point Algorithm. The 5-point algorithm [68] is the currently most frequently used algorithm in order to estimate the relative pose of views within an *SfM* system and, therefore, will be discussed briefly in the following. It is based on computing the coefficients of a tenth degree polynomial in closed form followed by finding its roots. Hence, the algorithm delivers at most 10 solutions.

For this algorithm, an additional property of the essential matrix that the two nonzero singular values are equal, is used. This leads to the following cubic constraints [68]:

$$EE^T E - \frac{1}{2} trace(EE^T)E = 0, \tag{2.11}$$

which an essential matrix has to satisfy.

Having 5 point correspondences, each point leads to a constraint of the form

$$q'^T E q = 0, \tag{2.12}$$

which is the constraint for the essential matrix defined the same way as for the fundamental matrix in Equation 2.8. This constraint can also be written as

$$\tilde{q}^T \tilde{E} = 0, \tag{2.13}$$

where

$$\tilde{q} = [q_1 q'_1 \ q_2 q'_1 \ q_3 q'_1 \ q_1 q'_2 \ q_2 q'_2 \ q_3 q'_2 \ q_1 q'_3 \ q_2 q'_3 \ q_3 q'_3]^T \tag{2.14}$$

$$\tilde{E} = [E_{11} \ E_{12} \ E_{13} \ E_{21} \ E_{22} \ E_{23} \ E_{31} \ E_{32} \ E_{33}]^T. \tag{2.15}$$

When stacking the vectors $\tilde{q}$ for all five points, a 5×9 matrix can be retrieved. Then, four vectors $\tilde{X}$, $\tilde{Y}$, $\tilde{Z}$, $\tilde{W}$ are computed, which span the right nullspace of this matrix. These four vectors correspond directly to four 3×3 matrices X, Y, Z, W and the essential matrix has to be of the form

$$E = xX + yY + zZ + wW \tag{2.16}$$

for the scalar values x, y, z, w. As these four scalars are defined only up to a common scale factor, it is assumed that $w = 1$. By inserting Equation 2.16 into the 10 cubic constraints in Equation 2.11 and performing Gauss-Jordan elimination, an equation system can be obtained, which can be transformed into a 3×3 matrix B containing polynomials in z. The determinant of this matrix B

$$\langle n \rangle \equiv det(B) \tag{2.17}$$

is the tenth degree polynomial $\langle n \rangle$. The real roots of $\langle n \rangle$ deliver the variable z, for which the variables x and y can be found using the equation system B. The essential matrix can finally be computed by using Equation 2.16. Consequently, the relative pose parameters, namely rotation and translation, can be computed. For a detailed description on how to solve this problem in an efficient way, we refer the reader to [68].

2.2 Structure-from-Motion

Having multiple overlapping images from a scene, it is possible to simultaneously estimate the corresponding camera poses and the scene structure. This process is called *SfM*. Usually, local image features are used to register the camera views to each other and, using an optimization called Bundle Adjustment, consistent camera poses with their corresponding sparse scene representation are computed. Generally, there exist two types of *SfM* algorithms: The first one is *incremental SfM*, which is the standard approach. It

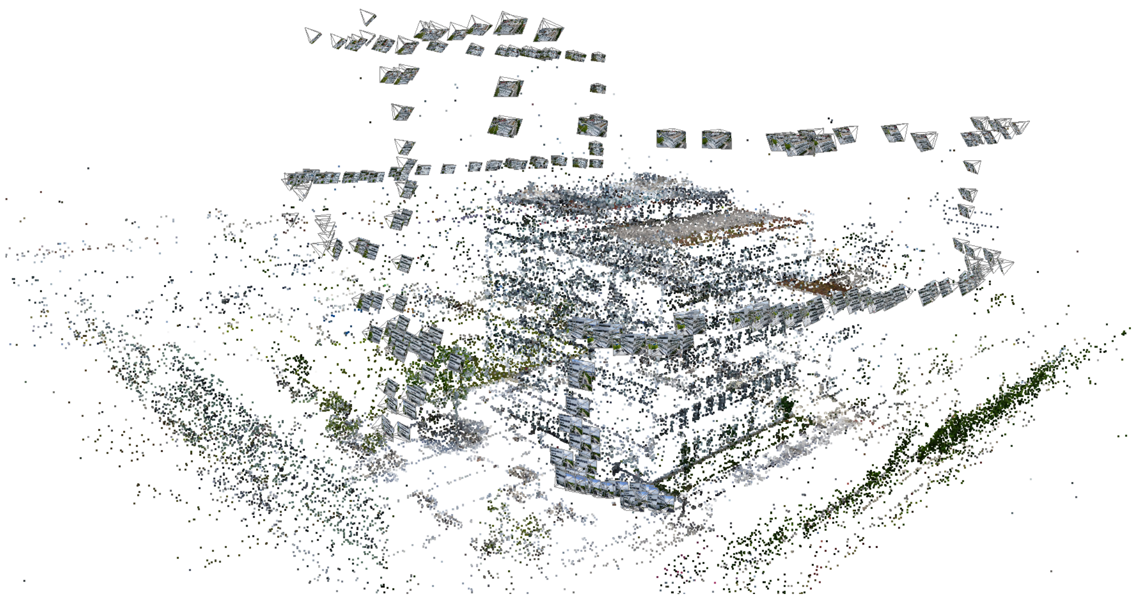


Figure 2.3: Result of a *SfM* system. One can see the sparse point cloud reconstruction of a building and the registered views with which these points were reconstructed. Images were taken from an Unmanned Aerial Vehicle (UAV) equipped with a camera.

starts with two initial images and incrementally adds additional single views to the scene reconstruction. The second one is *global SfM*, which considers all views at the same time. Figure 2.3 shows a result of an *SfM* system. In the following, we will discuss the individual processing steps of an *SfM* pipeline in more detail.

2.2.1 Feature Extraction and Matching

The first step of an *SfM* pipeline is the extraction of local features of every image. Very often, SIFT [57] is used, as it has shown to be very robust to scale and rotation changes. However, it is also possible to use other local feature descriptors like SURF [6] or lightweight ones like ORB [80]. Having detected local features in every image, these features are then matched against features in other images. Assuming an unordered image sequence, this has to be done with every other image in the sequence in order to find overlapping image pairs. As doing this in a brute-force way is very costly in terms of computation time, this procedure can be accelerated by using a vocabulary tree [69], which estimates a smaller set of visually similar images on which the matching can finally be done in a brute-force way.

2.2.2 Geometric Verification

The feature matches retrieved from the previous step might not be consistent in terms of the geometric relation between the corresponding cameras. Hence, in the geometric verification, the relative poses between all camera pairs having matched features are computed. As this is usually done in a robust way using the 5-point algorithm (as described in Section 2.1.2), features classified as outliers at the relative pose estimation step are rejected for further processing steps.

2.2.3 Incremental SfM

Having verified feature matches as input, the camera poses and the corresponding sparse scene representation are computed using *SfM*. Incremental *SfM* first tries to find a good initial image pair having a good geometric relation (i.e., sufficiently big baseline, good triangulation angle, sufficient number of verified matches). Then, further images are added by using an absolute pose estimation algorithm solving the Perspective-n-Point (PnP) problem. In comparison to relative pose estimation algorithms which compute the relative pose between two camera views (as described in Section 2.1.2), an absolute pose estimation algorithm estimates the absolute pose of a camera given a set of 3D scene points. A frequently used absolute pose estimation algorithm is P3P [23], which solves the perspective n-point problem with $n = 3$ i.e., it is able to estimate a camera pose using three 3D points.

Having added several cameras to the scene, geometric inconsistencies may arise for several reasons: First, every triangulated 3D point has a specific uncertainty due to measurement noise and specific geometric properties of the stereo pair used for triangulation (e.g., a small baseline leads to a higher depth uncertainty). Therefore, when having additional observations of this point, the back-projection of the 3D point in the new camera might not be consistent anymore with the triangulated 3D point. Further, the intrinsic camera calibration might not be correct and therefore introduces an error in the system. Consequently, due to this potential error sources, the estimated extrinsic camera calibration (i.e., the estimated camera pose) might not be correct. Therefore, an additional optimization step called bundle adjustment [31] is performed, where the triangulated 3D points and the extrinsic and intrinsic camera parameters are optimized jointly by minimizing the overall reprojection error, which is the Euclidean distance of a 3D point back-projected in an image to the actual image measurement which should represent the 3D point.

After a final bundle adjustment, incremental *SfM* results in optimized extrinsic and intrinsic camera parameters and corresponding optimized 3D points.

2.2.4 Global SfM

In comparison to incremental *SfM*, global *SfM* considers all the camera poses at once. Given the relative poses between cameras, all the global camera parameters and optionally the corresponding 3D structure are optimized in one optimization procedure. Very often this optimization procedure is separated into two steps, where the first step computes the global rotation of each view and the second step computes the camera translations, together with the structure or not.

Global *SfM* has the advantage that the quality of the final reconstruction does not depend on the selection of an initial pair. Further, it does not suffer from drift due to the accumulation of errors caused by sequentially added images and form the difficulty to handle cycle closures of the camera trajectory. However, the space and time requirements for global *SfM* can get very large, as the minimization covers all cameras at once and is based on the structure and the reprojection errors [64].

2.2.5 Example Pipelines

There exist several open-source and commercial implementations of *SfM* pipelines: For example Colmap [83] is an open-source 3D reconstruction system and includes one of the currently best performing *SfM* systems. Further open-source implementations include Theia [87], OpenMVG [65], Bundler [85], and OpenSfM [58]. There also exist mature commercial products including an *SfM* pipeline, for example Agisoft PhotoScan [1] or Pix4D [75].

2.3 Multi-View Stereo Reconstruction

Having the camera parameters (intrinsic and extrinsic) and a sparse scene reconstruction estimated using Structure-from-Motion, it is possible to compute a denser point-based scene reconstruction using a *MVS* algorithm. An *MVS* algorithm usually aims to find correspondences for every point in an image in all overlapping images, which results in an *MVS* depth map for every view or already a dense 3D point cloud of the scene.

A widely used *MVS* algorithm is PMVS2 [22], which reconstructs a semi-dense point cloud given calibrated cameras as input. The main idea of the algorithm is to enforce local photometric consistency and global visibility constraints within an iterative match, expand and filter procedure. There also exists an extension to this approach named CMVS [21], which decomposes the input set of images into a set of image clusters of manageable size as a preprocessing step. Using this extension, it is possible create semi-dense reconstructions of large-scale scenes. An example result can be seen in Figure 2.4.

Sure [79] is another widely used *MVS* approach. In a first step, this algorithm undistorts the images and rectifies them pair-wise. Then, suitable image pairs are selected and matched using a technique similar to Semi-Global Matching (SGM) [32]. The main idea of *SGM* is to do pixelwise matching of mutual information and using a global, 2D smoothness constraint by combining many 1D constraints. Compared to the original implementation, the proposed version in [79] is implemented in a more time and memory efficient way. In a final triangulation step, the disparity information from several stereo views is fused and 3D points or depth images are computed. By exploiting the information of several views, the accuracy of the depth estimates can be improved.

Another technique commonly used in *MVS* is plane sweeping, as it can be done on more than two images within one processing step: For the plane sweeping algorithm, a plane

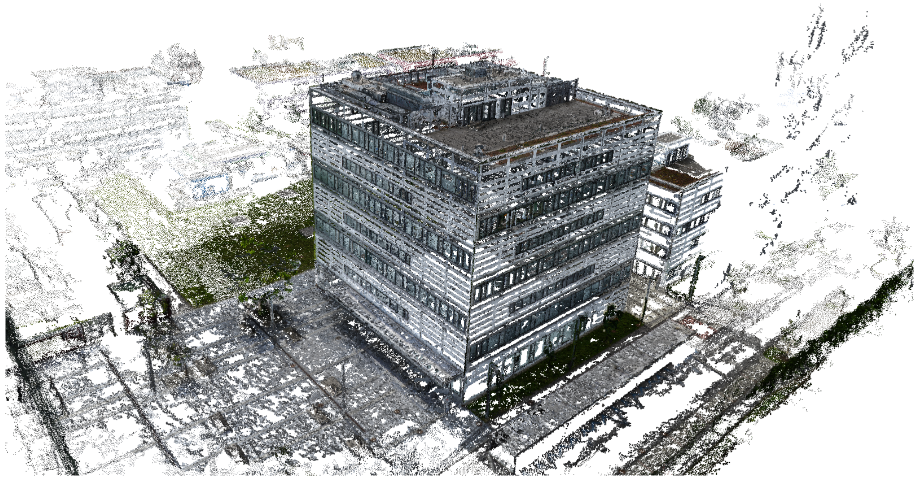


Figure 2.4: Semi-dense point cloud result from PMVS2 [22]. It can be observed that the point cloud is denser as in Figure 2.3. However, especially at poorly textured parts (e.g., at white façades) only few points could be reconstructed.

is swept through space and features from each image are backprojected on the successive positions of the plane [13]. As drawback, this technique does not produce smooth depth maps but just computes depth values on the swept plane. To overcome this issue, there exist several methods in the literature. For example, Häne et al. [28] propose a patch prior modeling the local surface structure. This prior is then used as a regularization term for the depth maps in order to fill holes between planes and at untextured areas.

More recently, PatchMatch-like methods became popular. The basic idea of these methods is to propagate sparse or random depth measurements over the whole image using multiple sweep iterations. Several methods also select corresponding matching views in a pixel-wise manner. Using the potential propagated depth value, a pixel is backprojected into multiple views and an error measure is computed to evaluate the depth [84, 98]. As these methods can be easily parallelized, they can also be implemented efficiently on a GPU.

2.4 Line Reconstruction

Instead of representing the scene with a point cloud, it is also possible to reconstruct line segments representing the scene. Especially at poorly textured environments (like urban scenes, which often contain untextured walls and façades), it is difficult to match point features over multiple images and, hence, it is difficult to reconstruct scene structure using these features. However, very often line segments can be detected in the images and triangulated to represent the scene as 3D line segments.

A recent method for line-based 3D reconstruction is Line3D++ and was presented in [33]. This method assumes given camera poses as input, detects lines in 2D using the

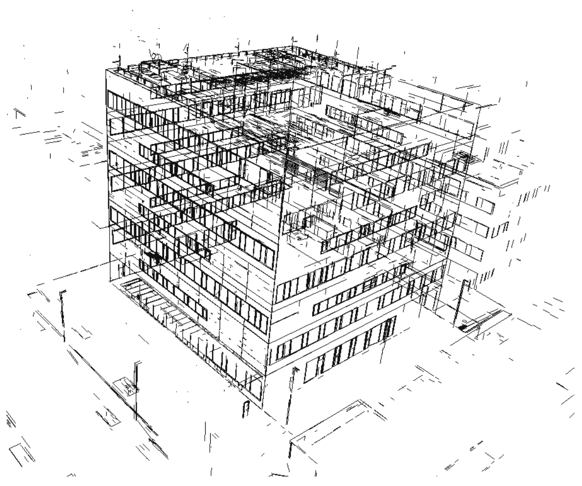


Figure 2.5: Line reconstruction created wit Line3D++ [33]. Compared to a sparse point cloud reconstruction, the scene geometry can be reconstructed much better using line segments.

Line Segment Detector (LSD) [27] and finally uses the 2D lines to reconstruct 3D lines. One has to mention that even though *LSD* contains the word line in its name, it actually detects edges. Hence, also the 3D lines reconstructed by Line3D++ [33] are originating from edges in the images.

An example result of Line3D++ is depicted in Figure 2.5: Compared to a result of an *SfM* pipeline (as depicted in Figure 2.3), the line-based reconstruction can represent the scene with much more details using less primitives. Compared to the *MVS* result depicted in Figure 2.4, the reconstruction has not the same level of density. However, for computing a *MVS* point cloud a higher computational effort is needed and, more importantly, a line set gives more abstracted geometric information of the scene than an unstructured point cloud.

2.5 Surface Reconstruction

Having reconstructed 3D information of the scene in the form of a point set, the next step in a 3D reconstruction pipeline is usually to reconstruct the surface of the scene. In this section, we discuss some surface reconstruction algorithms which are frequently used in current 3D reconstruction systems.

2.5.1 Poisson Surface Reconstruction

The Poisson surface reconstruction algorithm proposed in [43] takes a point set with its corresponding normals as input and produces a watertight triangular mesh as output by approximating a 3D *indicator function* χ and reconstructing the surface by extracting an appropriate isosurface from this function (example result is depicted in Figure 2.6).

The indicator function χ has the value 1 at locations inside the model, and 0 outside the model. In [43] the relationship between oriented points (i.e. points with given normals defined as vector field $\vec{V}$) sampled from the surface of a 3D model and the indicator function of the model is analyzed: The gradient of the indicator function is zero almost everywhere, except near the surface of the model (see Figure 2.7), where it is equal to the inward surface normal.

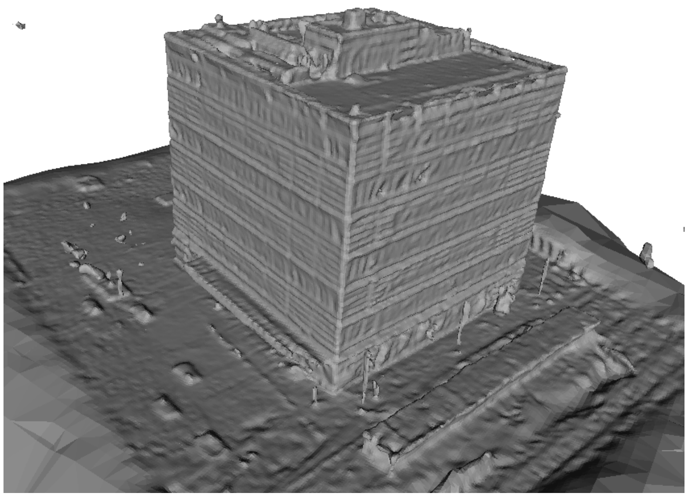


Figure 2.6: Example result of Poisson surface reconstruction [43] using the point cloud from Figure 2.4 as input. The 3D scene is reconstructed as a watertight, smooth surface. However, due to missing input samples at some parts of the façade, spurious artifacts looking like bubbles arise. Further, this approach is not able to reconstruct sharp edges (e.g., at building edges).

Hence, the problem of computing the indicator function can be reduced to the problem of finding the scalar function χ whose gradient best approximates the vector field $\vec{V}$ defined by the input samples.

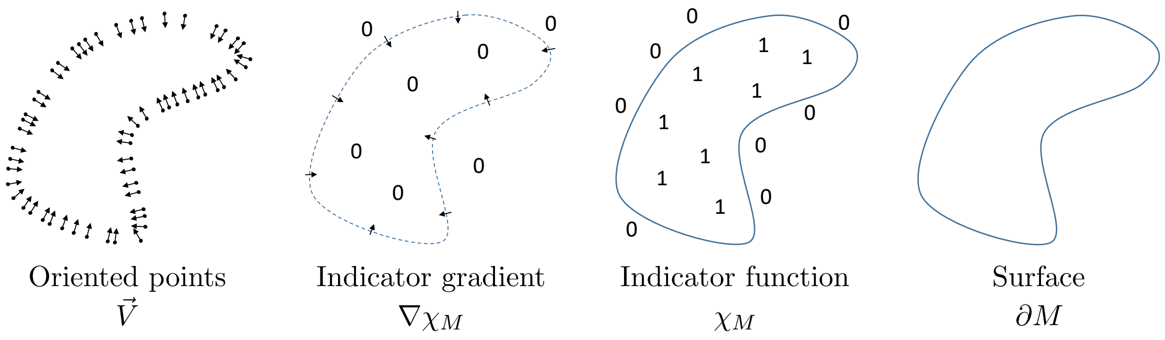

Oriented points $\vec{V}$
Indicator gradient $\nabla\chi_M$
Indicator function χ_M
Surface ∂M

Figure 2.7: 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.

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

$$\nabla(\chi_M * \tilde{F})(q_0) = \int_{\partial M} \tilde{F}_p(q_0)\vec{N}_{\partial M}(p)dp, \tag{2.18}$$

where 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 [43].

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

$$\begin{aligned}\nabla(\chi_M * \tilde{F})(q_0) &= \sum_{s\in S}\int_{\mathcal{P}_s} \tilde{F}_p(q)\vec{N}_{\partial M}(p)dp \\ &\approx \sum_{s\in S}|\mathcal{P}_s|\tilde{F}_{s.p}(q)s.\vec{N} \equiv \vec{V}(q),\end{aligned} \tag{2.19}$$

where $s \in S$ is an input sample. Using the input set S to partition the solid boundary ∂M in patches $\mathcal{P}_s \subset \partial M$, the integral of the patch $\mathcal{P}_s$ can be approximated by scaling the value of the sample $s.p$ to the area of the patch.

Subsequently, we want to solve $\tilde{\chi}$ such that

$$\nabla\tilde{\chi} = \vec{V}. \tag{2.20}$$

As $\vec{V}$ is generally not integrable, an exact solution does not generally exist. However, a least-squares solution can approximate the integral. For this, the divergence operator is applied to form the Poisson equation

$$\Delta\tilde{\chi} = \nabla \cdot \vec{V}. \tag{2.21}$$

For more information on how to solve this Poisson equation, we refer the reader to [43].

Finally, the isosurface can be extracted using, for example, a Marching Cubes method as described in [56].

2.5.2 Tetrahedralization-Based Methods

Another possibility for reconstructing the surface of a scene is to compute a 3D Delaunay triangulation, labeling each tetrahedron as inside or outside and extracting the final surface as the interface of inside and outside labeled cells. A frequently used algorithm following this paradigm was proposed by Labaut et al. [48]: Their proposed approach uses the visibility information of every point as main information, sets up an energy minimization problem for labeling each cell as inside or outside and solves it using Graph Cuts [10]. As input, they use a point set with visibility information (i.e., which point is visible in which camera) and as output they compute a watertight triangular mesh of the scene. In this section, we will explain this approach in more detail.

2.5.2.1 Delaunay Triangulation

When having an input point set, as a first step a Delaunay triangulation of the whole scene is computed. This results in an irregular discretization of space in tetrahedra, where the size of the discretized units is related to the density of the underlying point cloud.

As described in [8], a Delaunay triangulation can be defined by its dual, the Voronoi diagram which is defined as follows: Having 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 space that is closer to p_i than to any other point in $\mathcal{P}$. The Voronoi diagram, denoted by $\mathrm{Vor}(\mathcal{P})$, is the partition of space induced by the Voronoi cells $V(p_i)$.

The Delaunay triangulation $\mathrm{Del}(\mathcal{P})$ 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 2D Delaunay triangulation and a connecting facet in the 3D case. Figure 2.8 illustrates a 2D point set with its corresponding Voronoi diagram and Delaunay triangulation.

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 the application for surface reconstruction [48].

2.5.2.2 Surface Reconstruction

After partitioning the whole scene into tetrahedra, the goal is to label each tetrahedron as inside or outside of the object. Having these labels, a surface mesh can be finally extracted as the interface between inside and outside labeled cells.

A common technique to solve the labeling problem of the volumetric cells is to formulate it as energy minimization problem and solve it using Graph Cuts [10], with which it is possible to compute a globally optimal solution of this binary labeling problem.

There exist several slightly different energy formulations used for this reconstruction problem in the literature (for example: [38, 48, 50, 63]). All of them are using visibility

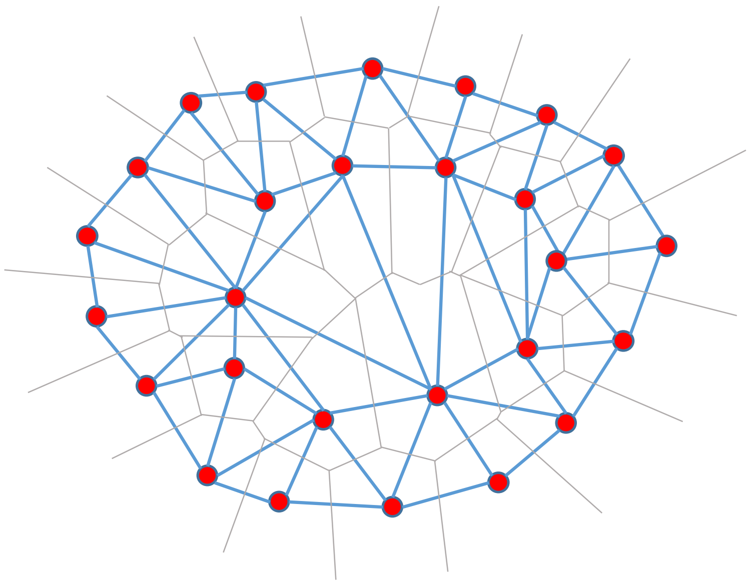


Figure 2.8: The Voronoi diagram (gray edges) of a set of 2D points (red dots) and its dual, the Delaunay triangulation (blue edges).

information as a main source of information (i.e., in which cameras a reconstructed point is visible in) and further add additional energy terms enforcing, for example, photometric consistency and surface smoothness [48] or a specific definition of surface quality [50]. However, as [63] pointed out, these additional energy terms do not lead to a significant improvement on surface reconstruction quality and a small constant pairwise cost added to all facets even performs the best.

In the following, we will describe the energy formulation proposed in [48] in more detail, which contains the same definition of the visibility-based energy than the methods described in [50, 63]. The overall energy is defined as:

$$E(\ell) = E_{vis}(\ell) + \lambda_{photo} E_{photo}(\ell) + \lambda_{area} E_{area}(\ell), \tag{2.22}$$

where ℓ is the labeling of the cell graph, E_{photo} is the photo consistency term, which measures how well the given labeling ℓ matches the different input images in which it is seen, and E_{area} is the area term which encourages surface smoothness. Both have their corresponding positive weights, λ_{photo} and λ_{area} and are pairwise smoothness terms. Most importantly, the visibility term $E_{vis}(\ell)$ defines unary and pairwise terms depending on the visibility information of every point. For computing the visibility-based energy, a ray is cast from every point to every camera it is visible in. Cells behind the visible point get finite weights assigned for being inside, cells containing cameras and infinite cells get infinite weights assigned for being outside. Further, facets intersected by the ray get pairwise terms assigned to penalize ray conflicts for every camera to point correspondence.

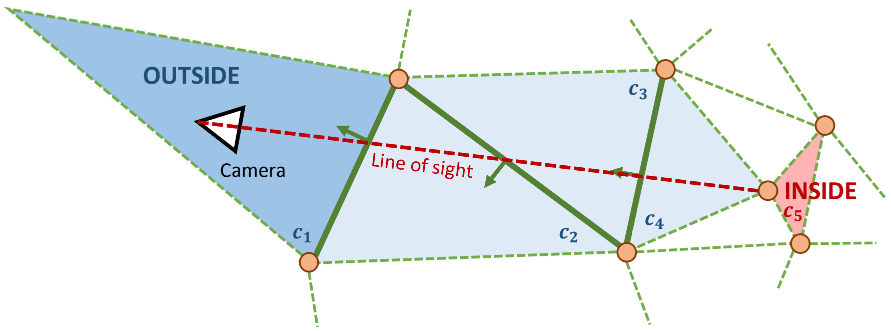


Figure 2.9: *Visibility-based energy computation as originally defined in [48].* Ray casting is used to compute the visibility terms: The cell where the camera is located in (c_1) is labeled as outside by adding infinite weights. Then, every facet (green) which is intersected by the line of sight (red) gets pairwise costs assigned in one direction. Finally, the cell behind the vertex (c_5) is labeled as inside by adding finite weights.

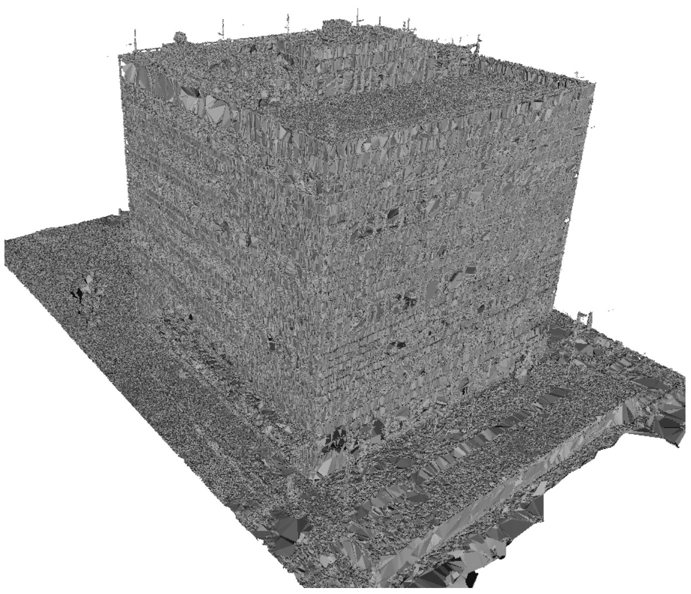


Figure 2.10: *Result of a Delaunay triangulation-based method [63].* Compared to a Poisson surface reconstruction as in Figure 2.6, the mesh contains more details but also more noise and clutter.

The pairwise costs are only added in one direction, since faces cannot exist in front of a measured point. Figure 2.9 illustrates the visibility-based cost computation.

Having solved the energy minimization problem defined in Equation 2.22 by using Graph Cuts, a watertight surface mesh can be extracted as the interface between inside and outside labeled cells (see Figure 2.10).

2.5.3 Methods Using a Signed Distance Function

There exist several methods which use a Truncated Signed Distance Function (TSDF) within a regular discretization of the scene, namely, a voxel grid, in order to reconstruct the surface [15]. A *TSDF* stores the distance to the surface to be reconstructed in a signed way (i.e., positive in front of a surface, negative behind) and truncates this value at a pre-defined distance.

A method using a *TSDF* which became popular simultaneously with cheaply available RGBD sensors is KinectFusion [67]: This method uses the depth map from an RGBD sensor in order to create the *TSDF* in the voxel grid. The zero crossing of the *TSDF* can then be extracted as the final surface. Even though the original paper uses an RGBD sensor for creating input data, this method is not restricted to such sensors. One could also use RGB images, compute their poses using *SfM* (see Section 2.2) and depth maps using an *MVS* algorithm (see Section 2.3) and use this as input.

In comparison to the method discussed in the previous section, this method uses regularly shaped volumetric cells (voxels) as main representation, whereas the method from the previous section uses irregularly shaped tetrahedra. An advantage of using a regular discretization of the scene is that it can be parallelized easily and, hence, can be efficiently implemented on a GPU. However, it has the drawback of not being scalable to big scenes due to the regular discretization.

To overcome this issue, several extensions exist to this method: For example, Infinitam [77] uses voxel block hashing to cope with bigger scenes. The core idea is to drop empty voxels from outside the truncation band and to represent only relevant parts of the volume (i.e., voxels near the surface). This is achieved by using a hash lookup of subblocks of the volume, where all voxels not accessed with the hash lookup table are not stored.

Another possibility to cope with large scenes is to use an octree representation of the scene. In an octree structure, the scene is not represented by regularly sampled voxels. Instead, the volumetric grid has a fine resolution near structure and a coarse resolution where no information exists. Hence, the representation of the whole scene is less memory consuming and bigger scenes can be processed. There exist several approaches which use a *TSDF* within an octree structure [42, 97].

2.6 Summary

In this chapter, we discussed basic methods used in image-based 3D reconstruction and explained commonly used subsequential processing steps within a 3D reconstruction pipeline. We have demonstrated that there exists mature reconstruction methods which produce surface models of scenes. Even though some methods produce visually appealing and smoothed reconstructions, it could be observed, however, that none of these standard generic 3D reconstruction methods fulfill our defined goals for visually appealing urban scene reconstructions containing planar surfaces and smooth edges. In the following chapter we will discuss methods which were specifically designed for reconstructing urban scenes and where some of them target similar goals than our work does.

3 Related Work

Contents

3.1 Reconstruction Using a Scene Hypothesis 25
3.2 Shape Priors Incorporated in Generic Reconstruction Methods 28
3.3 Semantic Scene Reconstruction 30
3.4 Summary 31

In the past years, several works were presented focusing on creating visually appealing and compact 3D reconstructions of urban environments. Many of them follow the idea that the scene to be reconstructed can be approximated by a specific set of primitives. Hence, as long as the scene follows this scene hypothesis defined by the primitive set, it can be correctly reconstructed in a compact way. Others use detected primitives and incorporate them into a generic reconstruction method in order to be able to reconstruct arbitrary scenes while still including shape priors within the reconstruction process. Finally, there exist methods which use semantic information in order to set class-specific shape priors and simultaneously improve the reconstruction and the semantic labeling.

3.1 Reconstruction Using a Scene Hypothesis

Different works focus on reconstructing scenes with a very specific scene prior and, hence, work well for these specific scenes but do not generalize well to others. The most basic approach reconstructing a scene with a scene prior is to represent the whole scene as geometric primitives. Further, an additional optimization step can be done to generate a consistent set of primitives. In this section, we will first discuss basic primitive fitting approaches and will subsequently discuss slightly more generic algorithms which, however, are still restricted to a specific scene hypothesis.

There exist several approaches for detecting basic primitives in the scene. For example, the method of Schnabel et al. [82], is able to detect a set of previously defined primitives

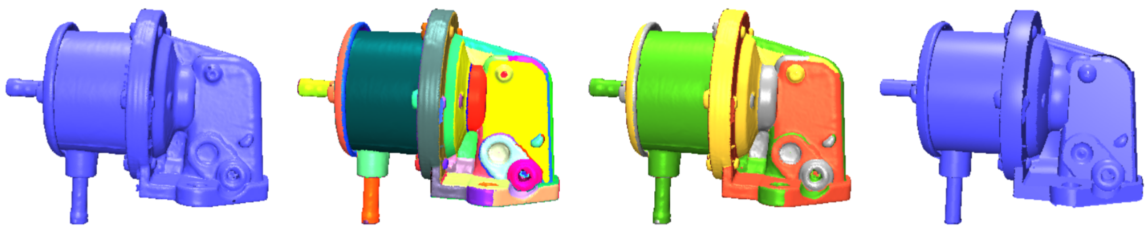

(a) Original **(b)** Random colors **(c)** Colored by type **(d)** Bitmaps

Figure 3.1: *RANSAC-based primitive detection proposed in [82].* In a) one can see the input scan consisting of approx. 500K points. In b), points belonging to detected shapes are grouped and randomly colored and in c) the points are colored according to the type of the shape they have been assigned to: planes are red, cylinders green, spheres yellow, cones purple and tori are grey. In d), a bitmap constructed by using their proposed connected component computation is shown, which provides a rough reconstruction of the object. Figure taken from [82].

within a point cloud by using a RANSAC-based [19] scheme. Having these primitives, it is possible to represent objects or a whole 3D scene (see Figure 3.1). Another work using RANSAC fitting for simple primitives is introduced in Li et al. [54]. This approach does not only fit primitives locally, but also applies a global optimization on the locally detected primitives, which improves the robustness of the algorithm and delivers a more consistent primitive set.

The vast majority of works that search for primitives in point cloud data, however, focus on detecting or fitting planes: Sanchez et al. [81] presented an system for planar 3D modeling of building interiors from point cloud data generated by range scanners. Using a RANSAC-based approach, they can detect and model large-scale structures like ceilings and floors, as well as small-scale structures like staircases. Arikan et al. [5] focus on detecting planes in an input point cloud along with their boundary polygons and then searches for local adjacency relations among parts of the polygons. These adjacency relations are then used to create a polygonal reconstruction which simultaneously fits to the input point cloud. In [18], Dzitsiuk et al. presented a plane detection approach which runs in real-time on robotic devices. In their work, they use RANSAC and least squares for plane candidate generation. Then, they incorporate detected planes in a Signed Distance Function (SDF) in order to create a cleaner and more complete reconstruction of the scene. In the work of Oesau et al. [71], shape detection and regularization is done in tandem. A sparse set of seeds in the input point cloud is first sampled. Then planes are detected by doing region growing.

Nan and Wonka [66] proposed an approach where they fit planes and intersect all the detected planes with each other to generate a possible set of faces for the final reconstruction. The final surface is generated by solving an optimization problem using all these face candidates. For reconstructing buildings from aerial images, Zebedin et al. [96] use sparse line features and dense depth maps. As resulting reconstruction, they represent

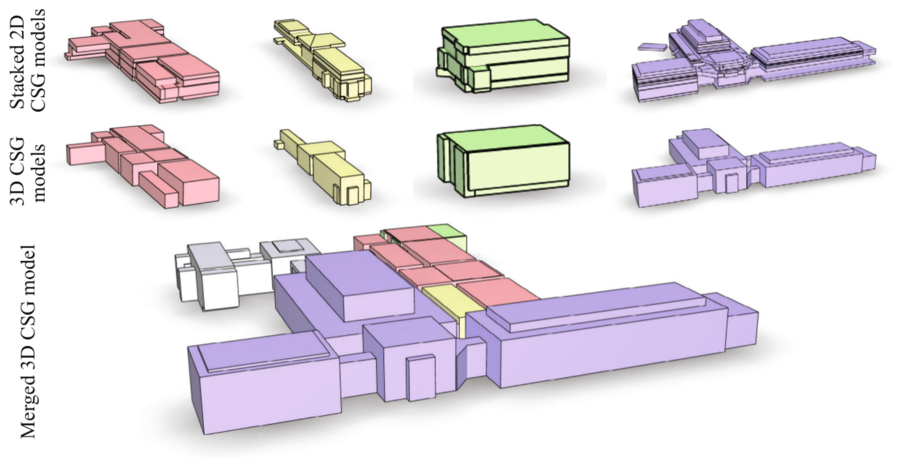


Figure 3.2: Reconstruction of a whole museum (The Frick Collection) using the method of [94] Top: Stacked 2D CSG models for individual slices. Middle: Optimized 3D CSG models (leads to less jagged or misaligned models). Bottom: The final model, where the individual parts are merged. Figure taken from [94] .

whole buildings with planes and surfaces of revolution.

Another widely used assumption in urban scene reconstruction is to only look for axis-aligned piecewise planar structures in a purely orthogonal arrangement, commonly known as the Manhattan-world assumption. Monszpart et al. [61] use this assumption of axis-aligned piecewise planar structures. They focus on the reconstruction of man-made scenes and detect a regular arrangement of planes. Then, they use a selection scheme which balances between data fitting and the simplicity of the arrangement of planes. Another work from Li et al. [53] uses detected planes to create a set of axis-aligned boxes that approximate the geometry of a building. They first detect planes using the method from [82] and restrict the plane set to a Manhattan-like orientation to be able to extract boxes out of the plane set. Finally, they apply a global optimization method on the box set in order to get an optimized selection of boxes representing the building.

Focusing on indoor scenes and using laser scan points as input, Xiao and Furukawa [94] proposed to use an inverse Constructive Solid Geometry (CSG) approach to model scenes as piecewise planar surfaces. They partition rooms or whole buildings into horizontal slices and compute an inverse CSG for every slice. In their paper, they showed whole museums reconstructed with their approach, which were textured afterwards using RGB images. Figure 3.2 shows example geometry results from a museum. Oesau et al. [70] proposed a slicing-based modeling approach for indoor scenes, where the whole scene is partitioned into horizontal slices and vertical structures like walls are detected by using a Hough transform. Using these detected structures, the whole scene gets partitioned into volumetric cells which are labeled as inside or outside using Graph Cuts [10]. Using panoramic RGB-D images as input, Ikehata et al. [40] reconstruct indoor scenes as structured models, which are 3D models containing structure elements like room, wall and objects. Such models are very compact and can also be used to extract floorplans from buildings.

Several of the methods discussed in this section are able to detect a set of planes or other primitives within the scene, as long as the input data is accurate and dense enough. Several, however, might miss primitive detections in case of weakly reconstructed surfaces, which can happen frequently in image-based reconstructions (e.g., at poorly textured façades). Further, such methods usually just aim to detect simple primitives, as detecting higher-order primitives within a RANSAC scheme becomes computationally expensive. Also, applying only primitive detection on a scene cannot create a complete reconstruction of a generic scene, but rather approximates parts of the scene which comply with the primitive definitions. Other methods discussed in this section are able to reconstruct specific, pre-defined scene geometries in a well regularized way. However, they are not intended to generalize to an arbitrary input scene and, hence, will not work well on an arbitrary scene geometry.

3.2 Shape Priors Incorporated in Generic Reconstruction Methods

When modeling the scene geometry with pre-defined primitives or geometric assumptions, it might not be possible to model all scene geometries correctly. Hence, several methods do not aim to explicitly model the input scene with detected primitives but aim to incorporate detected primitives into a global optimization scheme of a generic reconstruction method.

In the works of Labatut et al. [49] and Lafarge et al. [51, 52] primitives are incorporated within a tetrahedral representation of the scene. Similarly as described in Section 2.5.2, an optimal inside/outside labeling of tetrahedra is computed, which is then used to create a watertight surface mesh as result. More specifically, Lafarge et al. [51] use detected planes to create a structured point set, which is used to preserve the structural elements under a Delaunay triangulation. The final reconstruction contains both structured canonical parts and free-form parts. To incorporate detected planes into the Delaunay triangulation, they modify the input point cloud so that the induced planar facets are included in a scene triangulation fulfilling the Delaunay requirement (see Figure 3.3). [52] follows a similar idea: They detect various types of primitives in the scene (planes, spheres, cylinders, cones, and tori) and describe irregular elements with meshes. In a final optimization step, they sample both meshes and 3D-primitives by using a Jump-Diffusion-based algorithm [26], which combines probabilistic and variational mechanisms. As a result, this approach delivers a hybrid model consisting of mesh parts and primitive parts. In Labatut et al. [49], shapes are robustly extracted from a dense point cloud. Then, Binary Space Partitioning (BSP) trees are used to partition the whole scene into volumetric cells using the detected primitives. The *BSP* tree is then approximated by a Delaunay triangulation, which also includes points not being inlier of a detected primitive. Their hybrid surface

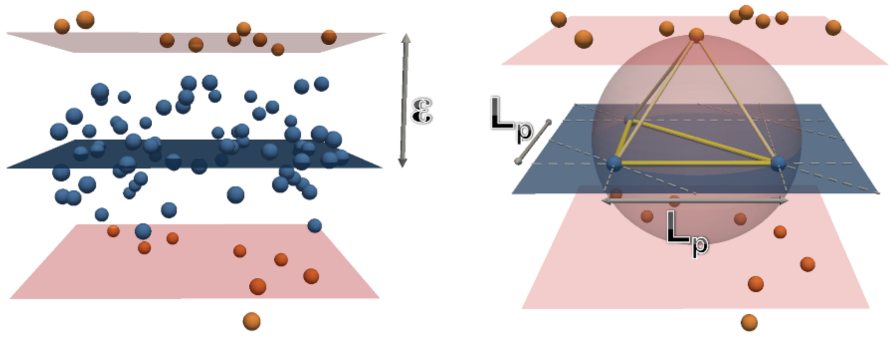


Figure 3.3: After detecting planes, the plane inlier points having a maximum distance of ϵ to the detected plane (blue plane) are removed from the point set and new points on the plane surface are created in order to enforce the Delaunay triangulation to include the detected plane. Image taken from [51].

reconstruction finally outputs a compact segmented model of the scene, where it is also possible to adjust the amount of details in the final reconstruction (i.e., how strong the algorithm should follow the detected primitives).

In [76], Pollefeys et al. presented a real-time 3D reconstruction system for urban environments. The presented system is mostly generic, but plane sweeping directions for stereo and a depth map hole filling leverage urban structure assumptions. Duan and Lafarge [17] present a work on city reconstruction from satellite images. Starting from a super-pixel segmentation, the algorithm assigns each of the super-pixels a height value and a semantic label, resulting in a very efficient algorithm which allows arbitrary building ground shapes reconstructed with a so-called 2.5D representation. This representation does not store the full 3D information for each reconstructed part, but just the position and the elevation on the ground plane, which is sufficient for many applications. However, overhanging structures like bridges or or overhanging roofs cannot be described in 2.5D. Similarly, Zhou and Neumann [100] propose a 2.5D building modeling algorithm, where they incorporate global regularities within the reconstruction process. Their results have high quality in terms of geometry and human judgement.

In Arefi et al. [4], buildings are reconstructed using Lidar data as input. They propose an approach which automatically generates 3D building models based on the definition of Level Of Detail (LOD) in the CityGML standard [44]. In this standard, 5 *LOD*s are defined: *LOD*0 contains the 2.5D terrain model, *LOD*1 the building block model without roof structure, *LOD*2 the building model including the roof structure, *LOD*3 the building model including detailed architecture and *LOD*4 additionally includes the interior model. The approach in [4] models the buildings up to level *LOD*2. In [91], Verdie et al. also present a method to reconstruct different *LOD*s from urban scenes. They use surface meshes generated from multi-view stereo systems as input, then they classify the input, abstract the different scene parts and reconstruct the desired *LOD*.

The methods presented in this section are more generic and several of them are able to

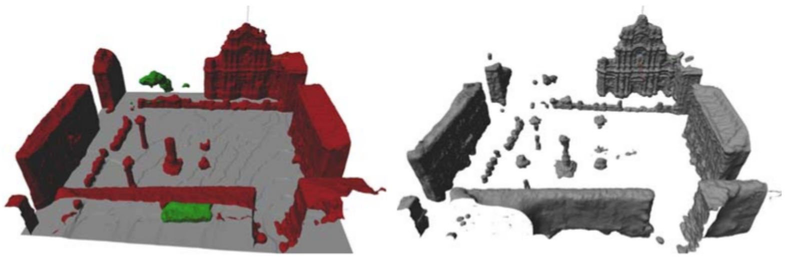


Figure 3.4: Semantic reconstruction of [29] (left) compared to reconstruction without semantics (right). By setting shape priors depending on semantics and by jointly optimizing semantics and scene geometry, the semantic reconstruction delivers a more complete scene geometry and more complete semantic labels. Image taken from [29].

reconstruct arbitrary scenes while still incorporating shape priors within the reconstruction process. However, as these methods do not use semantic information (except some works which use it in a very simple or implicit form), they cannot set class-specific shape priors which could be beneficial for urban scenes containing strongly geometrically varying scene parts like, for example, buildings and trees.

3.3 Semantic Scene Reconstruction

Recently, several methods incorporated semantic information in the 3D reconstruction process in order to simultaneously improve the semantic labeling and the 3D reconstruction.

Häne et al. [29, 30] first learn appearance likelihoods and class-specific geometry priors for surface orientations. Then, using these priors, unary and pairwise potentials are defined and used within a voxel-based volumetric segmentation framework. By jointly optimizing over the geometric and semantic properties, they interact with each other yielding to an improved dense reconstruction and labeling (see Figure 3.4). In Cherabier et al. [11] this approach is extended for being able to handle a higher number of semantic labels. The scene is partitioned into several blocks, where each is then processed separately. In order to be able to process more labels, only labels present in a currently processed block are considered. In [7], an extension to this work for large-scale reconstructions was proposed. Instead of a regular voxelization of the scene, an octree representation was used, which significantly reduced the memory and computation time needed. Consequently, it is possible to reconstruct whole cities with this approach. A similar approach was proposed by Richard et al. [78]: Instead of using an octree representation, they used a tetrahedralization of the scene, which, as described in Section 2.5.2, significantly reduces the memory footprint in comparison to a regular voxelization. In [12], Cherabier et al. presented a 3D reconstruction framework which embeds variational regularization into a neural network. Their approach is able to learn semantic and geometric relationships end-to-end from data with using a lightweight network which can be trained with little data. Compared to similar variational reconstruction approaches like [30] not having incorporated a neural network, it does not need manual and scene-dependent parameter tuning.

All these methods incorporate shape priors into the scene reconstruction using semantic information. However, they are not aiming at creating visually appealing and compact models, but focus more on improving the reconstruction and semantic label accuracy and completeness. In our work, we focused more on reconstructing scenes in a visually appealing and compact way with a sufficiently high accuracy.

3.4 Summary

In this chapter, we discussed related work in the area of urban 3D reconstruction and complementary fields. We discussed approaches using a scene hypothesis, which work well when the scene follows a specific scheme (for example, Manhattan-world assumption), but still might not work well on generic scenes. Several of them directly use a detected primitive set to represent the whole scene geometry. Others use detected primitives as input and apply an additional optimization in order to generate a consistent reconstruction. Next, we discussed methods which incorporate shape priors within a generic reconstruction framework. These methods use a generic reconstruction method but favor specific geometric shapes in order to regularize the resulting reconstruction. Hence, they are generally able to reconstruct generic scenes, but might favor pre-defined geometric properties. Finally, we discussed semantic scene reconstruction methods, which work on generic scenes and favor semantic class-dependent geometric properties.

Even though several of the discussed methods show impressive results, they do not fully aim at fulfilling our goals of creating visually appealing and compact 3D reconstructions of urban scenes originating from images. Such reconstructions should contain a detailed representation of the buildings containing planar surfaces and sharp edges whereas the surrounding should be represented by a smooth surface.

In the next chapters, we will tackle these problems and present two methods for urban scene reconstruction suitable for geometric scene abstraction and for creating visually appealing reconstructions of urban scenes.

4 Slicing for Building Reconstruction

Contents

4.1 Method Overview 34
4.2 Input Data 35
4.3 Horizontal Slicing and Cell Decomposition 36
4.4 Volumetric Cell Labeling as an Energy Minimization Problem 40
4.5 Summary 44

Using standard 3D reconstruction techniques, it is possible nowadays to generate dense point clouds which can be meshed afterwards to create a surface accurately representing the reconstructed scene. For example, one can use open source Structure from Motion (SfM) pipelines (as described in Section 2.2) to estimate the camera poses and compute a sparse point cloud representing the scene. Afterwards, an Multi-View Stereo (MVS) algorithm as described in Section 2.3 can be used to densify the point cloud. Finally, surface reconstruction techniques like the ones described in Section 2.5 produce a detailed mesh representing the reconstructed scene.

However, due to measurement uncertainties in various steps of the reconstruction process, such 3D reconstructions are noisy, contain clutter or oversmoothed surfaces and bubbles. Therefore, they might not appear visually appealing even though they have a high reconstruction accuracy. Additionally, the amount of data used to process, store and transmit such models is quite high, as they can contain millions of points modeling the reconstructed noisy surfaces, which might be problematic for applications using 3D maps which just have limited resources available. Therefore, for many processing and viewing applications, a regularized, more compact representation is desired. This should be a representation excluding the noise and clutter from a dense reconstruction and should be as near as possible to reality if desired, but it should also be possible to generate more abstract models, that don't cover details but represent the geometric structure well.

In this chapter, we present a 3D reconstruction algorithm, which creates regularized 3D building models consisting of geometric primitives from image-based 3D reconstructions. The proposed regularization will remove small details which are likely to be noise and will describe the input mesh with a small set of vertices and faces. Simultaneously to the data reduction, the regularized model should describe the scene in a proper way, i.e. that planar surfaces in the scene are actually planes in the regularized model and not a noisy surface similar to a plane.

Based on the method described in [35], we first divide the dense model into multiple horizontal slices, which are parts of the model bounded by dominant horizontal structures. Then, we compute an inside/outside labeling for each slice using the visibility information of each point in the slice. Using the outlines of the labeling per slice, we compute an irregularly shaped cell decomposition of the whole scene. Finally, we optimize the model by solving an energy minimization problem. In this optimization, the level of regularization can be adjusted. We introduce a smoothness term based on detected line segments in the images, which improves the reconstruction results, especially in areas where the input model contains noisy surfaces (as presented in [34]). An example result can be seen in Figure 4.1.

Due to the nature of the proposed approach using slicing, a mainly Manhattan-like scene containing buildings with flat roofs and vertical façades is assumed as input. Especially sloped surfaces cannot be modeled correctly with this approach, as these structures are just approximated with stairway-like structures resulting from the individual slices.

In the following sections, we will first give a method overview (Section 4.1) and describe the input data used (Section 4.2). Then, we will describe the main algorithm consisting of horizontal slicing and cell decomposition (Section 4.3) and volumetric cell labeling (Section 4.4). Finally, we summarize the chapter in Section 4.5.

4.1 Method Overview

In this section, we give a brief overview of the processing pipeline starting from the input data and enumerating the consecutive processing steps, which deliver a geometrically abstracted 3D model as result. A schematic overview can be seen in Figure 4.2. First, we separate the meshed input point cloud into horizontal slices and compute an inside/outside labeling for every slice. Using the outlines of the inside labeled regions of all slices, we create an irregularly shaped volumetric cell decomposition of the whole scene. Finally, we generate an optimized inside/outside labeling of the volumetric cells by solving an energy minimization problem. For this, visibility information and detected 2D line segments are used. As a result, our system outputs geometrically abstracted 3D models from buildings.

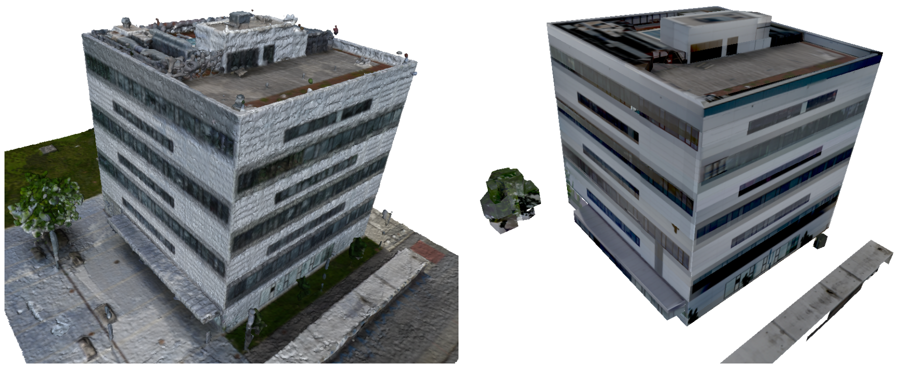


Figure 4.1: *Meshed dense 3D reconstruction and resulting regularized model. Left:* A 3D building reconstruction created from aerial images taken by an Unmanned Aerial Vehicle (UAV). The camera poses and initial 3D structure were computed with *SfM*, then a dense model was computed with Sure [79] which was finally meshed using Poisson surface reconstruction [43] (visualized with vertex coloring). Such a mesh can be used as input for our approach. *Right:* Resulting regularized model from our pipeline, textured with [93]. We deliver clean and smooth surfaces whereas the meshed dense point cloud contains a lot of noise.

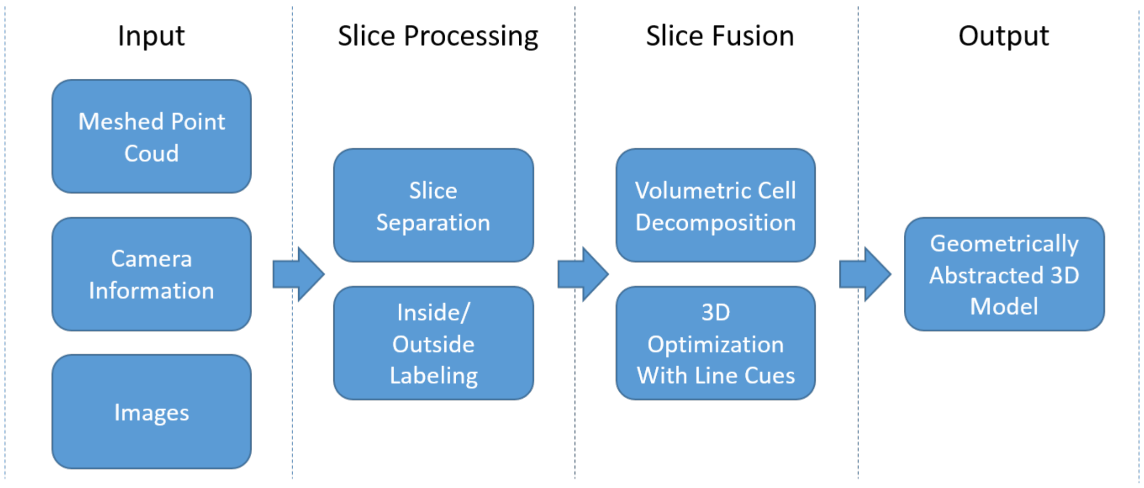


Figure 4.2: *Overview of the processing pipeline.* We take any meshed point cloud with arbitrary density with the corresponding camera positions and image informations as input. We separate the input model into horizontal slices and compute an inside/outside labeling for each slice. Using visibility information and detected lines in images, we compute an optimized regularized 3D model.

4.2 Input Data

As input data, we take a meshed point cloud and its corresponding camera information (i.e., the position of the camera views). Our approach works on meshed dense point clouds, but also on meshed sparse point clouds (e.g., a meshed result from *SfM*). An example dense

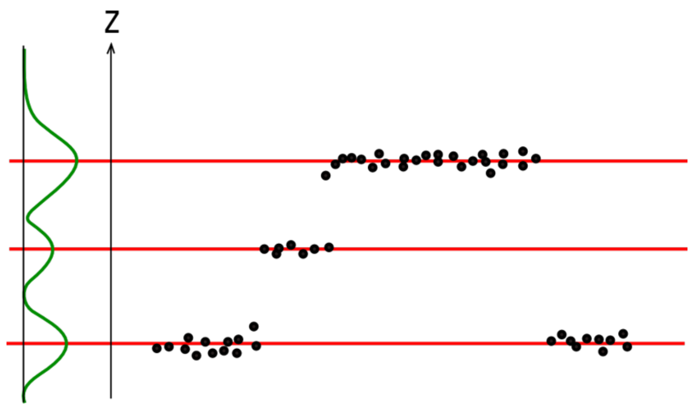


Figure 4.3: *Illustration of the slice boundary detection in the input point cloud.* Using only input points which have a normal similar to the ground plane (black dots), modes are estimated in the z-direction using Mean Shift [14]. The resulting mode centers are used as slice boundaries (red lines).

building model can be seen in Figure 4.1.

4.3 Horizontal Slicing and Cell Decomposition

This section describes the separation of the input model into horizontal slices, the computation of an inside/outside labeling for each slice and the creation of an irregularly shaped cell decomposition of the whole scene.

4.3.1 Horizontal Slicing

As a first step, we detect horizontal structures and separate the model at these structures into horizontal slices. For this, we first need to detect the gravity direction. As we assume that the ground plane is the dominant plane and perpendicular to the gravity direction, we do this by plane fitting: We try to find a plane for which the most points in the scene have a small Euclidean distance to it. However, we could also use different methods to estimate the gravity direction of the scene (e.g., by using inertial measurements). As we usually work with Manhattan-like scenes, we found out that it is beneficial for some processing steps to align the model with the Manhattan world directions. Therefore, we also detect the most dominant plane perpendicular to the ground plane and align the model to these directions. However, one has to mention that our approach works on any scenery and is not limited to Manhattan-like scenes. Though, if the scene geometry assumptions are not fulfilled (e.g., at sloped roofs), more artifacts like stairway-like structures arise.

Having estimated the ground plane, we estimate dominant horizontal structures by applying mode estimation using Mean Shift [14]. For this, we just select points with a normal similar to the normal of the ground plane, which are the points describing

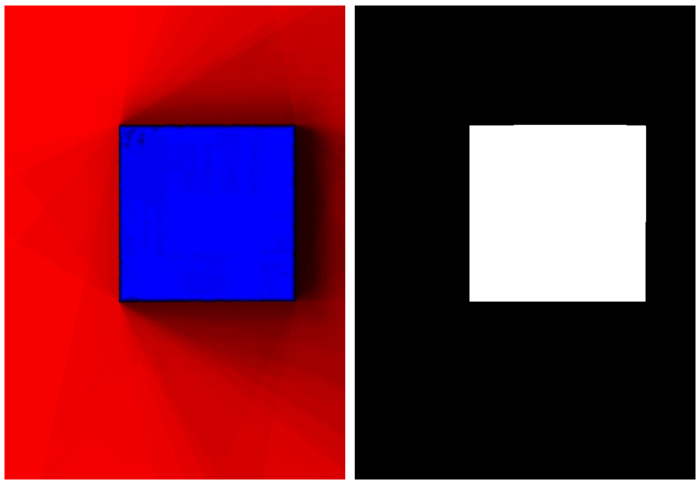


Figure 4.4: *Slice labeling.* Intermediate processing steps of the processing of a slice in the middle of a simple building. *Left*: An illustration of the free space scores of the slice. The more cameras see the specific area, the more intensive the pixel is red. In case of no visibility: The farther away this area is from a visible part, the more intensive the pixel is blue. *Right:* The resulting inside/outside labeling.

horizontal structure, and apply the mode estimation along this dimension, which is the z-direction of the scene. The mean shift bandwidth is defined as:

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

where *height* is the 3D model height and d is an adjustable parameter with which the level of detail in the vertical direction can be adjusted. An illustration of the mode detection in the point cloud is depicted in Figure 4.3.

4.3.2 Binary Labeling

Next, the model is separated into several slices at the detected horizontal structures and for each slice, a 2D inside/outside labeling is computed. For this, we need to compute a free space score for each position in the slice. The free space score is positive in areas which could be seen by cameras and negative in areas which could not be seen.

To compute a 2D free space score for each position in the slice (illustrated in Fig. 4.4), we first compute the free space score for each voxel of a voxel grid spanned over the whole scene. 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 that v_{xyz} is in free space is defined as

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

where $\{max \ \# \ visible \ cameras\}$ is the maximum number of visible cameras for a voxel v_{xyz}.
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. $p(v'_{xyz} = outside|visibility) > 0$:

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

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 [95] which is used in several surface extraction algorithms (see Section 2.5).

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:

$$p(b_{xy} = free|visibility) = \sum_{z} \frac{p(v_{xyz} = free|visibility)}{n}, \tag{4.4}$$

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.

Finally, we obtain an optimal inside/outside labeling of the 2D slice plane by solving an energy minimization problem using Graph Cuts [10], where we directly use the defined free space scores as data term.

For the creation of the geometric 3D model, we just need the outline of the inside-labeled pixels. Though, as the outline is a polygonal line including every pixel as point and we favor simple representations, a simplification of the polygonal outline is applied before continuing with further processing steps. We used the *Ramer-Douglas-Peucker algorithm* [16] for this task, which produces a simplified polygonal line which can be easily extruded to 3D.

Having this polygonal 2D outlines and the slice boundaries, we already have a very compact 3D representation of every slice, which, however, still need to be combined in a reasonable way to get a regularized 3D model of the whole building.

4.3.3 Slice Combination

By extruding the labeling from each slice to 3D (as illustrated in Figure 4.5), we already get an initial 3D model. However, as each slice is just optimized separately and no optimization has been done in the z-direction, the vertical surfaces are not smooth. Therefore, we need an additional 3D optimization step, for which we use irregularly shaped volumetric cells, which represent all important structures from the individual slices.

To get a cell decomposition of the whole scene, we project the outlines of the inside labeled parts of all slices onto the ground plane and compute a Constrained Delaunay

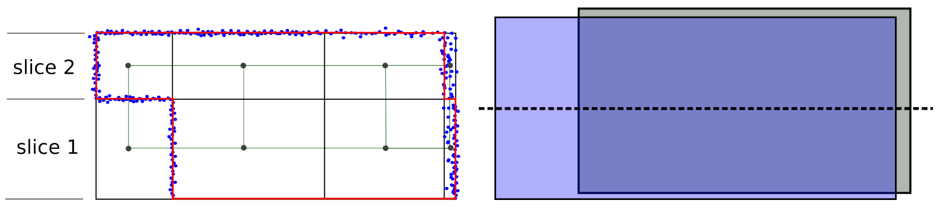


Figure 4.5: *Left: Volumetric Cells.* Vertical cut of a volumetric cell representation of a simple model consisting out of two slices. The black lines are the volumetric cells spanned over the whole scene and the red lines are the outlines of the extruded slices approximating the point cloud (blue dots). A graph is spanned over the whole scene setting cells with a shared facet as neighbors (green lines). *Right: 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. The dashed line represents the vertical cut seen in the left image.

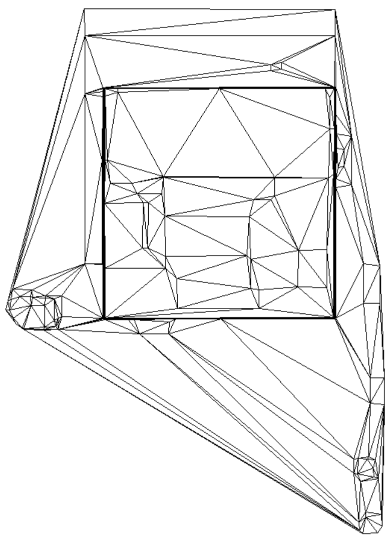


Figure 4.6: *2D Constrained Delaunay Triangulation* of outlines of all slices projected to the ground plane. As one can see, many similar lines are included near the building walls (thicker lines because of several very near lines) due to noise in the input mesh. All triangles are extruded between all slice boundaries to create a volumetric cell decomposition of the whole scene.

Triangulation (CDT) [74] including this lines (see Figure 4.6). The *CDT* guarantees that the projected outlines remain lines in the triangulation. Finally we extrude the computed triangles between all slice boundaries to get an irregularly shaped volumetric cell decomposition of the whole scene (as illustrated as black lines in Figure 4.5, left). This volumetric representation includes all important scene structures (all slice outlines) and is very compact (low number of volumetric cells), which is beneficial for the final optimization procedure.

4.4 Volumetric Cell Labeling as an Energy Minimization Problem

In the final step, we create a regularized labeling of all volumetric cells labeled as inside or outside. As a result, we can create 3D reconstructions consisting of geometric blocks, for which the degree of regularization can be adjusted depending on the smoothness parameters.

We formulate this optimization step as an energy minimization problem. In addition to a smoothness term which depends on the input mesh surface, we introduce a smoothness term which uses lines detected in the input images to create a regularized transition from inside label to outside label.

The energy to minimize is defined as:

$$E(L) = \sum_{p \in \mathcal{I}} E_{data}(L(p)) + \sum_{p,q \in \mathcal{N}} E_{smooth}(L(p), L(q)), \tag{4.5}$$

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 neighborhood relation is defined by the volumetric cell complex: all cells that share a common facet are neighbors. The data terms, $E_{data}(L(p))$, are defined as the summed up free space scores contained in the volumetric cell normalized by the cell size. The smoothness terms, $E_{smooth}(L(p), L(q))$, are a combination of two smoothness terms and defined as:

$$E_{smooth}(L(p), L(q)) = \lambda_{mesh} E_{s_{mesh}}(L(p), L(q)) + \lambda_{lines} E_{s_{lines}}(L(p), L(q)). \tag{4.6}$$

$E_{s_{mesh}}(L(p), L(q))$ is the smoothness term which depends on the mesh surface, i.e. it is likely that a labeling transition happens if the input mesh surface is near the facet between the cells p and q and unlikely otherwise. More precisely, this smoothness term depends on the amount of points near the facet of adjacent cells. Using a constantly upsampled point cloud on the input mesh surface, we count the points which are near this facet. With this approach, it is unlikely that two cells, which have a dominant structure, e.g. a wall, between them, get smoothed into an equally labeled group and it is likely that two cells without structures between them get smoothed into the same group. We calculate the neighbor weights as

$$E_{s_{mesh}}(L(p), L(q)) = \begin{cases} 0 & \text{if } L(p) = L(q) \\ \frac{1}{1 + \frac{\{\# \ points \ near \ facet_{p,q}\}}{area \ of \ facet_{p,q}}} & \text{else} \end{cases}, \tag{4.7}$$

where $\{\# \ points \ near \ facet_{p,q}\}$ is the amount of points which have a smaller Euclidean distance to the facet than a predefined distance which is defined in relation to the model size.

The result has the value

$$0 < E_{s_{mesh}}(L(p), L(q)) \leq 1, \tag{4.8}$$

where $E_{s_{mesh}}(L(p), L(q))$ is near 0 when lots of points are near the adjacent facet, which means there exist scene structures. In this case, no smoothing is wanted and due to $E_{s_{mesh}}(L(p), L(q)) \approx 0$, the smoothness penalty is near 0. $E_{s_{mesh}}(L(p), L(q))$ is 1, when no point is near the adjacent facet, which means that the total smoothness penalty is completely adjusted by λ.

The second smoothness term, $E_{s_{lines}}(L(p), L(q))$, is a term based on lines detected in the input images. Assuming an initial estimate of the model (i.e., having an initial estimate for all cells for being inside or outside), all facets connecting two adjacent cells in the volumetric cell decomposition get backprojected into the images to compute a line-based smoothness score:

$$E_{s_{lines}}(L(p), L(q)) = \begin{cases} 0 & \text{if } L(p) = L(q) \\ facetScore(p, q) & \text{else if visible} \\ 1 & \text{else} \end{cases}, \tag{4.9}$$

where $facetScore(p, q)$ computes a line-based score for the facet connecting cell p and q and visible means that at least one vertex of the facet is on the surface ot the currently estimated model and visible by a camera. Therefore, a score gets also computed for facets which are currently not visible but have a connection to the visible model surface.

To compute the facet score based on lines, first line segments are detected using the Line Segment Detector (LSD) [27]. Then, for every facet, cameras are selected for which the camera centers are nearly coplanar to the facet: The maximum allowed angle between facet and camera center is set to 25 *deg*. This improves the influence of the smoothness term for facets, which belong actually to the model surface but are in the middle of a façade in the images. Figure 4.7 illustrates this problem.

In the selected cameras for a facet, the distance from the detected line segments to each facet edge is computed. Therefore, we search for the nearest line detection with a similar direction than the facet edge: The maximum allowed angle between a detected line and facet edge is set to 10 *deg*. With this restriction, just lines are used that are similar to the structure of the facet.

For all facets with lines with similar direction, we search for the line which has the smallest normal distance to the facet edge. If this distance is above a threshold, it is truncated. The threshold for this maximum line distance is adjusted by the parameter *maxLineDist* (which we set to 70 *pixels*) and is computed as *maxLineDist* normalized by the camera distance to the current facet and the model size:

$$truncVal = \frac{maxLineDist}{normalizedCamDist}. \tag{4.10}$$

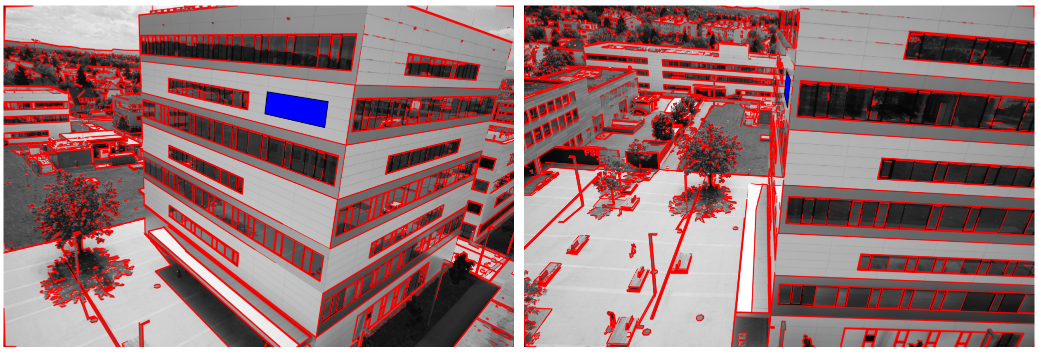


Figure 4.7: *Camera view with detected lines and backprojected facet. Left:* Backprojected facet (blue) is not near detected lines, even though it is on the surface of the building. Therefore, this view is not used for the line criterion, as the camera center is not near to coplanar with the facet. *Right:* Detected facet (blue, hardly visible) and camera are nearly coplanar and the facet is nearer to detected lines, as it is aimed to be for a facet being on the surface of the building. Therefore, this view is used for facet score computation.

$normalizedCamDist$ is defined as:

$$normalizedCamDist = \frac{camDist}{avgModelSize}, \tag{4.11}$$

where $avgModelSize$ is the mean of the maximum x- and y-extension of the input model and $camDist$ is the distance of the camera to the facet in arbitrary scale retrieved by *SfM*. This is necessary, as we don't have a fixed (or metric) scale in our reconstructions and want to enforce a similarly scaled camera distance for all models.

The normalization by $normalizedCamDist$ depicted in Equation 4.10 is beneficial, as the normalized truncation value $truncVal$ now is scaled according to the camera distance, meaning that distances are not only defined in pixel space anymore. This facet-to-line distance truncation significantly improves the results, as it lets the optimization focus on relevant, nearby lines and ignore far away lines.

Consequently, the truncated distance of a facet to a line (illustrated in Figure 4.8) is computed by

$$truncFacetDist = min(facetDist, truncVal)\ normalizedCamDist, \tag{4.12}$$

where $facetDist$ is the normal distance of a backprojected facet to a line, which gets truncated by $truncVal$. Then, the computed truncated facet-to-line distance gets additionally multiplied by $normalizedCamDist$ to transform the distance again from pixel to a normalized scale: If a camera is farther away, one pixel is a bigger distance than in a near camera. To get the total facet-to-line distance for the facet backprojected in one camera, we compute the average of all facet edges.

The final facet score, which is calculated using all cameras which fulfill the above

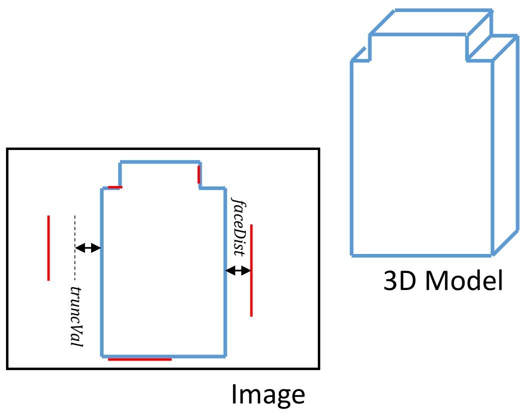


Figure 4.8: *Line distance computation.* All visible facets are projected into the suitable images. If, as illustrated, the projected surface of the model of the current iteration (blue) is near a detected line (red), this surface will likely stay the same also in the next iteration. Contrary, if the current surface has a high distance to a detected line, as can be seen at the right side of the projection, the surface will probably get shifted towards the detected line in the next iteration. If the distance is too high (left side), it gets truncated.

mentioned requirements, is defined as:

$$facetScore = \begin{cases} \frac{avgFacetDist}{maxLineDist} & \text{if } \#validCams > 0 \\ 1 & \text{else} \end{cases}, \tag{4.13}$$

where

$$avgFacetDist = \frac{\sum_{validCams} truncFacetDist}{\#validCams} \tag{4.14}$$

and $maxLineDist$ is the parameter also used in Equation 4.10.

Therefore, the $facetScore$ defined in Equation 4.13 is normalized between 0 and 1, where 0 means the facet is near to structures (lines) in the images, and 1 means that the facet is far from structures.

As we just compute facet scores for facets which have a connection to the currently estimated visible space, we need to do several iterations of the energy minimization step defined in Equation 4.5. In the first iteration, the line smoothness term is disabled (λ_{lines} is set to 0), and an initial model is estimated. Starting from the second iteration, also the line smoothness term is used. We empirically observed that 4 iterations are usually enough to let the model converge to a solution. When doing more iterations, the result does not change significantly.

Finally, we get an optimized inside/outside labeling containing all the irregularly shaped volumetric cells. From this labeling, models consisting of geometric blocks can be created, which contain well regularized vertical surfaces (e.g., façades) even when using erroneous input data.

4.5 Summary

In this chapter, we presented a method for creating regularized building models from image-based 3D reconstructions. By using horizontal slicing, creating irregularly shaped volumetric cells using the detected slices and labeling each cell as inside or outside, this method creates abstracted models of buildings containing planar surfaces and sharp edges. However, as horizontal slicing is an integral part of this method, only shapes fulfilling this geometric constraint can be modeled adequately, whereas shapes not fulfilling this constraint (e.g., sloped surfaces) can only be approximated by staircase-like structures. In Section 6.5.1, a detailed evaluation of this approach will be presented. In the next chapter we will introduce a method which does not have such geometric restrictions and can be applied on any generic scene, while still incorporating shape priors for buildings within a final reconstruction step.

5 Shape Priorization in Tetrahedra-Based Methods

Contents

5.1 Method Overview 47
5.2 Plane Detection 48
5.3 Tetrahedralization of the Scene 51
5.4 Tetrahedra Subdivision 52
5.5 3D Reconstruction Using Tetrahedral Occupancy Labeling . . 55
5.6 Visibility-Based Energy Computation 56
5.7 Plane-Aware Regularization 60
5.8 Semantically Aware Urban Reconstruction 62
5.9 Summary 65

In the previous chapter we presented a building reconstruction method which is able to produce visually appealing and geometrically abstracted models of buildings following specific geometric properties. However, several shapes like, for example, sloped or rounded surfaces can not be reconstructed well with this method. Further, the method does not aim to reconstruct the surroundings of buildings.

In order to overcome these limitations, we present a hybrid method between generic 3D reconstruction and plane-based urban reconstruction in this chapter [36, 37]: In contrast to many works that focus only on urban structure reconstruction, our approach is able to deal with a mixture of urban and generic surface structures. Our method robustly deals with noisy, missing and outlier data by computing a consistent watertight surface of arbitrary topology via volumetric occupancy labeling of tetrahedra via graph-cuts. We present a unified 3D reconstruction framework to jointly favor planar surfaces and orthogonal structures - without explicitly enforcing a Manhattan structure, as well as enforcing user-specified smoothness and level of detail properties.

Similarly as for the method presented in the previous chapter, the following criteria are targeted in order to create a visually appealing urban 3D reconstruction: We aim to model

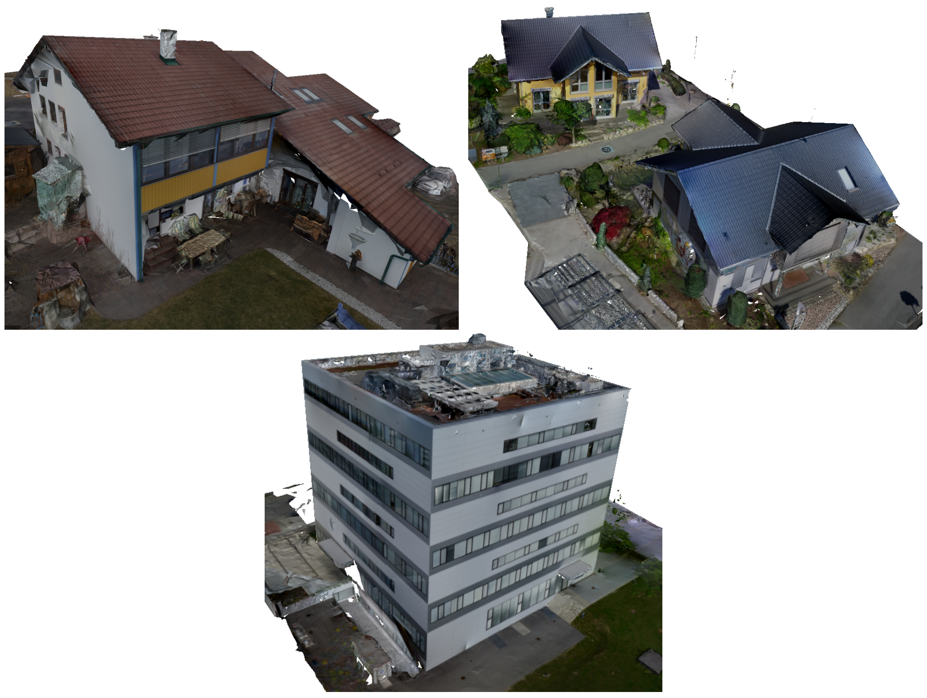


Figure 5.1: *Results of the proposed approach textured with [93].* One can observe that planar parts in the scene (façades, windows, roof) are represented by planar surfaces and building edges where two façades intersect each other are represented by straight lines. Note that holes in the model are due to missing visibility information during texturing.

planar and nearly planar surfaces at building parts (e.g., façades, roofs) as exactly planar surfaces. Doing this, edges of buildings can consecutively be represented as straight lines (i.e., no noisy edges). Simultaneously, the surrounding of buildings does not necessarily be reconstructed with high details but with a smoothed, visually appealing surface.

These criteria do not only define a visually appealing reconstruction, but make it also easier to reduce the amount of data in a post-processing step (i.e., points lying on a planar surface can be diminished without changing the surface).

To apply these criteria, we use semantic priors in the reconstruction process to treat building and surrounding scene parts differently [36]. For building parts, we incorporate plane priors in order to achieve planar surfaces and straight outlines while still keeping important details. For non-building parts, we impose a smooth surface by reestimating a smoother 3D representation of the scene, setting class specific sparsification parameters and smoothness terms. We partition the scene into volumetric cells using a Delaunay triangulation and provide a consistent and minimal approach to account for previously detected planes in the tetrahedral labeling graph by splitting the tetrahedra into smaller ones and adapting the graph and corresponding cost values accordingly. Finally, we perform Graph Cut-based inside/outside labeling and compute a watertight polygonal mesh

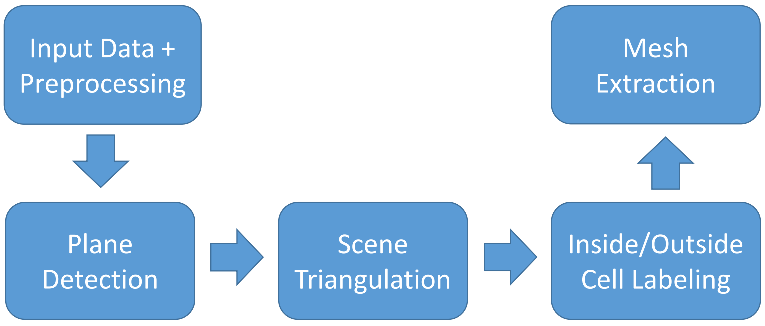


Figure 5.2: *Overview of our processing pipeline* First planes are detected using the input 3D data. Then, a Delaunay triangulation of the whole scene incorporating the detected planes is created. Finally, a mesh is extracted from the triangulation by computing an optimized inside/outside labeling of the tetrahedra.

resulting from the interface of inside and outside labeled cells of the triangulation. The priors of our method are very generic which makes it well suited for reconstructing any mixture of urban and nature scenes as well as both indoor and outdoor scenarios. In order to reduce the amount of data needed, mesh simplification can be applied in a post-processing step without loosing accuracy. Figure 5.1 visualizes textured results of our approach.

In the following, we will first give a method overview (Section 5.1). In Section 5.2 we will discuss tow plane detection algorithms, which either use a point cloud or a set of line segments as input. In Section 5.3 we briefly describe the tetrahedralization of the scene and in Section 5.4, we describe how previously detected planes can be incorporated into the tetrahedralization. In Section 5.5 we describe how to define the energy which is used within the energy minimization to compute the final tetrahedral occupancy labeling and in Sections 5.6 and 5.7 we discuss the individual energy terms in more detail. Finally, in Section 5.8 we describe how to incorporate semantic information within the reconstruction process and in Section 5.9 we give a summary of the chapter.

5.1 Method Overview

In this section, we give a coarse overview of our processing pipeline and subsequently describe each part in detail in the next sections.

In order to be able to reconstruct generic scenes and simultaneously incorporate a plane prior into the reconstruction process, our method is build upon a well-known generic surface reconstruction method: As originally introduced by Labatut et al [48], our method uses tetrahedra for the volumetric partitioning of the scene and has the visibility information of each reconstructed 3D point as the main source of information. To stay generic but introduce the possibility to set a plane prior within the reconstruction process, we incorporate detected planes within the tetrahedralization of the scene and, hence, make them selectable in the final surface reconstruction step. By adding a smoothness prior favoring this planar structure, the amount of smoothing w.r.t. planes can be adjusted. Our proposed method stays generic, as it is still able to reconstruct scenes with arbitrary shapes, but includes an adjustable plane prior within the reconstruction process, which can be also applied just to specific semantic classes.

A schematic overview of our pipeline is depicted in Figure 5.2. As a result, our approach delivers beautiful, visually appealing 3D models from urban scenes where buildings have planar surfaces and straight edges while still containing relevant details embedded in a smoothed surrounding.

Taking images from a scene as input, we first compute the camera poses using Structure from Motion (SfM) and a dense point cloud using a Multi-View Stereo (MVS) algorithm. As further optional preprocessing steps, a line-based 3D reconstruction, dense depth maps for every camera and semantically labeled images can be computed and used in the pipeline.

As a first step, planes are detected in the scene. Having a line-based 3D reconstruction available as input, 3D line segments are used to detect planes. In case no 3D lines are available, planes are detected using a RANSAC plane detection scheme within the 3D point cloud.

Then, a Delaunay triangulation of the scene is computed and planes are incorporated within the triangulation. In order to guarantee that the planes are included in the triangulation, tetrahedra intersected by a plane are split into multiple smaller ones.

Finally, the cells of the triangulation are labeled as inside or outside by solving an energy minimization problem in order to create a watertight mesh surface as result. The proposed shape prior favors Manhattan-like surfaces within the optimization, which leads to planar surfaces and sharp edges while still not being restricted to a Manhattan world or other geometric scene assumptions.

Having semantic labels available, the different scene parts can be handled differently within the reconstruction framework in order to create a smooth reconstruction of the surrounding and a detailed but prior-based smoothing of the buildings.

5.2 Plane Detection

As a first step in our processing pipeline, we detect planes in the scene, which can be incorporated in the reconstruction framework in a subsequent step. In this section, we describe two plane detection algorithms, where the first detects planes within a point cloud and the second one uses 3D line segments as input.

As we could not get complete plane detections in urban environments (i.e., detecting all façades/roofs of a building) using point-based methods due to the lack of reconstructed points at poorly textured surfaces, we investigated in using 3D lines within a plane detection algorithm. Compared to the point-based approach, the proposed line-based approach has the advantage that only few line segments needs to be available to detect a plane. This leads to a more complete plane detection result especially at poorly textured surfaces in urban environments (e.g., at façades).

5.2.1 Point-Based Plane Detection

In the following we describe a plane detection algorithm which uses a point cloud as input and is a RANSAC-based [19] approach, which detects multiple planes until a predefined percentage of the whole point cloud is included as inlier in one of the planes. The algorithm iteratively takes a random set of three points to compute a plane hypothesis. The inlier points supporting the plane hypotheses are defined as points with a maximum distance of d_{inlier} to the plane, which is computed as the median minimum point-to-point distance of the whole point cloud multiplied by 5. If in the current iteration the #inliers is bigger than from all previous iterations, the number of successive iterations needed to guarantee a correct plane estimate by 99% is re-estimated according to [19] and the current estimate is stored as current best estimate. This procedure is iterated until convergence. Given a plane estimate, the supporting inliers are removed from the point cloud. and the algorithm is executed again until 75% of all points are supporting one of the planes in the plane set.

For every plane, point clusters on the computed plane are estimated using Mean Shift [14]. and, for every cluster, an oriented bounding box is computed. Finally, there are several plane segments for every detected plane defined by the bounding boxes around the detected clusters.

In comparison to the line-based approach, the point-based plane detection has the advantage that it is also able to detect planes in generic scenes, where no reconstructed lines set with a specific arrangement is available (e.g., at ground with grass). However, it misses important planes at buildings where surfaces are frequently reconstructed with a low point density due to missing texture.

5.2.2 Line-Based Plane Detection

A very common approach to detect planes in 3D is to use a RANSAC-based algorithm with a point cloud as input (as described in Section 5.2.1 or in [82]). However, especially in urban environments where scene parts like façades might be poorly textured, these approaches fail due to missing reconstructed 3D points. In comparison, 3D lines are more likely to be detected at building façades, as some high-gradient elements like windows or building outlines usually exist. Hence, we are using this 3D line information to improve plane detection in urban environments. As our goal is to reconstruct a well smoothed surrounding of the building, we just use lines labeled as building and ignore all the others in case semantic information is available.

Assuming to have a 3D reconstruction consisting of line segments, we first detect line triples which already describe a plane hypothesis. Then we cluster the triples which are coplanar and in vicinity. Finally, we detect all inlier lines from the plane hypothesis.

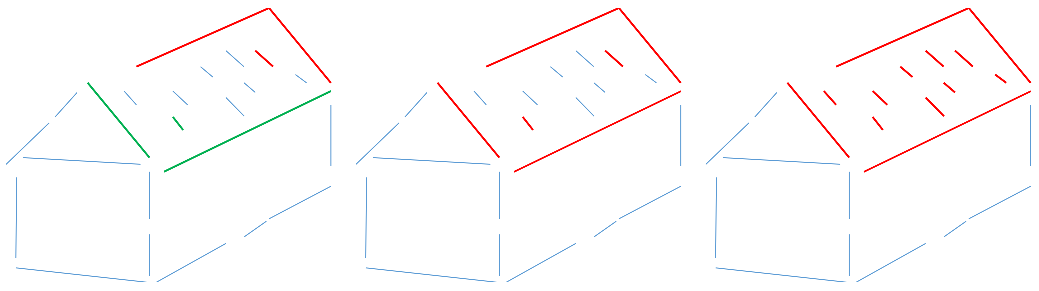


Figure 5.3: *Line triple detection (left), line clustering (middle) and plane inlier detection (right).* Several line triples (left, two depicted in green and red) can be detected on one plane and get clustered to form a plane hypothesis (middle). Finally, all line segments defined as inliers are added to the plane hypothesis and the final plane parameters are estimated.

Figure 5.3 illustrates the subsequent processing steps. Note that we are assuming a metric scale for the described setting of parameters in this section.

5.2.2.1 Line Triple Detection

As we explicitly want to model man-made scenes frequently having rectangular outlines, we search for perpendicular coplanar line pairs which can be used to describe a plane. Additionally, we only accept line pairs which have a small distance between each other. For the coplanarity and perpendicularity tests we accept errors up to $\alpha_{error} = 5\ deg$, and the normal distance from start/end point of the line segment to the computed plane hypothesis must not be bigger than $d_{inlier} = 0.15\ m$. The distance between start/end point of the two line segments must not be bigger than 1.5 m and we ignore line segments shorter than 0.8 m. In order to perform an early removal of spurious planes, we search for a third supporting line which has to be coplanar with the line pair and with small distance to the pair. We just accept the line pair if such a third line exists. Note that line segments can also be part of several line triples, which is beneficial for lines which are exactly, e.g., at corners of a house.

5.2.2.2 Line Triple Clustering

After having estimated plane hypotheses by detecting line triples, several hypotheses can be nearly identical. Therefore, we cluster line triples which represent the same planar surface. We cluster line triples by first checking if the triples are nearly coplanar (i.e., normals of plane hypotheses with enclosing angle smaller α_{error}, normal distance from line triple (i.e., start/end point of its segments) to the current plane hypothesis lower than d_{inlier}). From these coplanar line triples, we greedily add all line triples to a cluster which have a maximum distance of the line projections on the plane of 12 m to the previous line triple. After having clustered the triples, the lines are sampled and the sampled points are used to reestimate the plane using Singular Value Decomposition (SVD).

5.2.2.3 Inlier Detection and Outline Estimation

Finally, we detect all inlier (i.e., lines segments being nearly coplanar in terms of angle α_{error} and distance d_{inlier}) which have a distance of the line projections on the plane smaller than 1.2 m. We estimate an outline of the plane by computing a bounding box around all inlier segments and reestimate the plane parameters with all inliers using *SVD*.

5.2.2.4 Plane Filtering

Having estimated the planes, we filter out plane segments which are included in another plane hypothesis (i.e, having the same plane parameters and the outline is included). Additionally, we filter out planes which don't have sufficient supporting 3D data (i.e., points and sampled line segments, see Section 5.8.2) within d_{inlier} normal distance. These planes are usually erroneous detections due to a specific line segment arrangement.

5.2.3 Plane-Based Denoising

We consider all inlier 3D data within the distance d_{inlier} to the plane as noisy samples of the plane. To remove this small noise, we project all inlier onto the plane surface. Opposed to a dense over-sampling of the plane as proposed in [51] to enforce the plane to be part of the tetrahedralization, this roughly maintains the original density in the 3D data. Further, such a dense over-sampling is not necessary for our approach, as the tetrahedra get cut by the detected planes to for this purpose (see Section 5.4).

5.3 Tetrahedralization of the Scene

In order to get a volumetric decomposition of the whole scene volume, a Delaunay Triangulation is constructed using the input 3D data.

The tetrahedralization T via Delaunay triangulation of a point set is defined as follows [8]: Given a point set $\mathcal{P} = \{p_1, ..., p_n\}$, the Voronoi cell associated to each point p_i is the region surrounding the point p_i in which every point is closer to p_i than to any other point in $\mathcal{P}$. The Delaunay triangulation $Del(\mathcal{P})$ of $\mathcal{P}$ is defined as the geometric dual of the Voronoi diagram. Thus, there is an edge between two points if and only if their corresponding Voronoi cells have a non-empty intersection. Such a Delaunay triangulation leads to a partition of the convex hull of $\mathcal{P}$ into d-dimensional simplices, corresponding to triangles in 2D and to tetrahedra in 3D space. A more detailed description of the Delaunay triangulation can be found in Section 2.5.2.1.

Using a Delaunay triangulation is motivated by its property defined in [3]: Having a sufficient densely sampled surface represented by a point cloud $\mathcal{P}$, a good approximation of the real surface is contained in $Del(\mathcal{P})$. Hence, by selecting an appropriate set of facets from $Del(\mathcal{P})$, a good estimate of the real surface can be computed.

5.4 Tetrahedra Subdivision

After removing the noise of plane inlier data by moving them onto the plane (Section 5.2.3) and applying a Delaunay triangulation afterwards (Section 5.3), the plane surfaces are not necessarily part of the tetrahedralization. We therefore compute intersections of planes and tetrahedra and subdivide them into multiple ones to ensure that all planes are represented as facets in the tetrahedralization. Note that in contrast to [51] that augment the tetrahedralization with a densely sampled representation of every plane, our approach minimizes the amount of added points and tetrahedra. In this section, we discuss in detail how we consistently subdivide tetrahedra in order to ensure that all planes are part of the tetrahedralization.

5.4.1 Tetrahedra Intersected by Plane

For every edge which intersects a plane segment, we subdivide all incident cells of this edge by dividing the cell along the plane. However, a cell division into two parts is not sufficient as cells need to be further subdivided into multiple tetrahedra and kept consistent with their neighbor cells. Therefore, we define subdivision schemes for all possible intersection cases of a cell by a plane (see Figure 5.4). Depending on its neighbors, the correct subdivision orientation is selected and, if necessary, the subdivision is adapted to be consistent with all neighbors. Note that the triangulation might not fulfill the Delaunay property after the tetrahedra subdivision, but we do not require this property in the forthcoming processing steps.

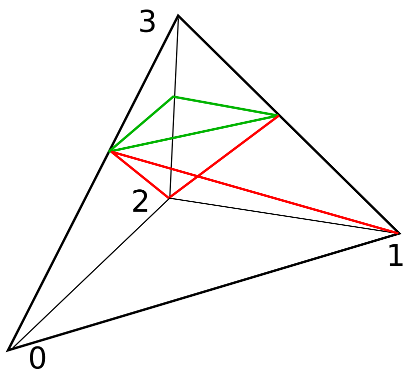
5.4.2 Consistency Adoptions

When an intersected cell has two or more already divided cells as neighbor, the default subdivision scheme from Figure 5.4 may not lead to a consistent triangulation. Therefore, the subdivision scheme needs to be adapted depending on the adjacent cells. In some cases, this may lead just to a differently oriented subdivision with the same amount of resulting cells. However, there exist cases where the subdivision results in more cells (up to 12). These more sophisticated consistency adaptions only happen at 3-point and 4-point intersections (as illustrated in Figure 5.4). A subset of these adaptions can be seen in Figure 5.5 and Figure 5.6. At a vertex intersection, an adaption might also be necessary. However, just a simple one: Only the newly inserted edge needs to be flipped.

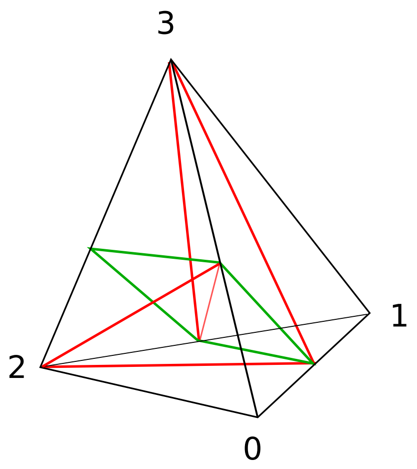
5.4.3 Margin Tetrahedra

Cells not directly intersected by a plane but adjacent to a subdivided cell must also be subdivided in order to keep the whole triangulation consistent.

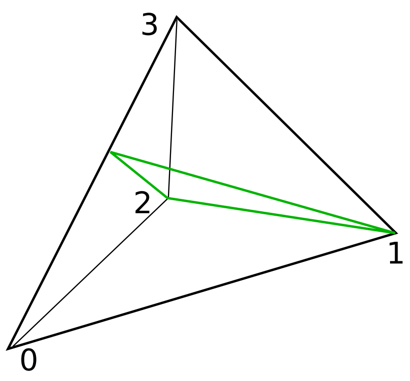
Depending on the adjacent facet of the subdivided neighbor cell (i.e., if the neighboring facet is divided or not), the margin cells need to get divided differently:



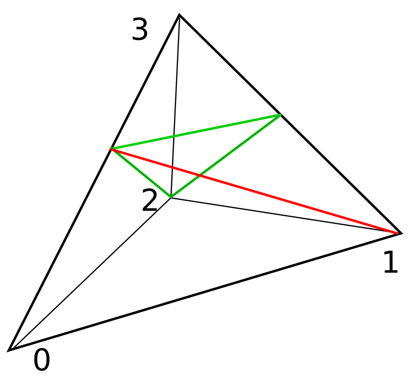
(a) 3-point intersection



(b) 4-point intersection



(c) Edge intersection



(d) Vertex intersection

Figure 5.4: *Intersection cases and subdivision schemes of tetrahedra intersected by a plane.* Depending on the amount of edge intersection points and their locations, the cell needs to be divided differently. The cut by the plane is illustrated with *green* edges, additional edges which need to be inserted are illustrated in *red*. In (a) and (b) there is an edge-plane intersection which results in a new vertex for each intersection. Four new cells are created in (a) and six new cells are created in (b). In (c), the cell-plane intersection follows exactly an edge. Therefore, just one new vertex and two new cells are created. In (d), the cell-plane intersection comprises one cell vertex. The cell gets divided into three new cells by adding two new vertices.

- If a cell has **more than two divided neighboring facets**, this cell also needs to be subdivided as if it would be intersected by the plane.
- If a cell has only **one intersected neighbor with a not divided adjacent facet**, this cell does not need to be subdivided. Only the neighbor relationship needs to be updated.
- If a cell has **one divided neighboring facet**, this cell has to be divided into three new cells, which are consistent to the neighbor. Figure 5.7 (left) illustrates this

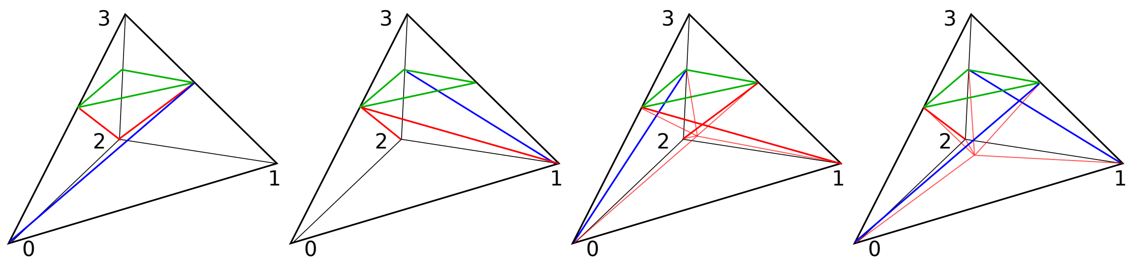


Figure 5.5: Consistency adaptions of cells with 3-point intersection. Edges to be changed in order to stay consistent with neighbors are depicted in *blue*. In the first two subdivision schemes from left, only one edge and its corresponding cells need to be changed. No additional change in the internal structure is necessary. In the third subdivision scheme from left, also only one edge needs to be changed to stay consistent to its neighbors. However, to stay consistent within the new cell structure, a new vertex needs to be inserted in the middle of the cell and finally the cell gets divided into 9 cells (instead of four). In the figure to the right, two edges need to be changed to stay consistent with the neighbors. Also in this case an additional vertex needs to be inserted in the middle of the cell and the cell gets divided into 9 cells. For the other subdivision schemes with two changed edges, the internal structure does not need to be changed.

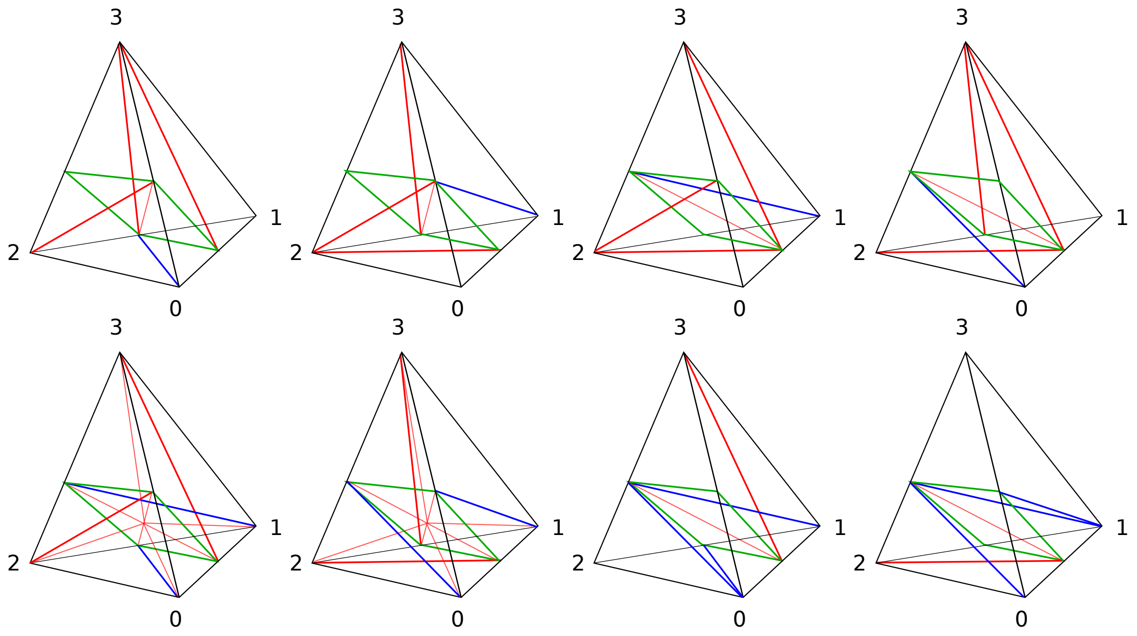


Figure 5.6: Consistency adaptions of cells with 4-point intersection. Edges to be changed in order to stay consistent with neighbors are depicted in *blue*. *Top row:* Subdivision schemes, where only one edge and its corresponding cells need to be changed to stay consistent with the neighbors. In the first two schemes from left, no additional change in the internal structure is necessary. In the first two schemes from right, also the internal (newly created) edge lying on the plane needs to be changed. *Bottom row:* In the first two schemes from left, two outer edges need to be changed. To stay consistent internally, an additional vertex and edge needs to be inserted on the intersection plane. Hence, the cell gets divided into 12 cells (instead of 6). In the first two schemes from right, three outer edges need to be changed. To stay internally consistent, also the internal edge lying on the plane needs to be changed. For the other subdivision schemes with two and three changed edges, the internal structure needs not be changed.

subdivision.

- If a cell has no intersected neighbor, but **contains a newly created vertex** in an edge (i.e., an incident cell of an edge was intersected by the plane), the cell has to be divided into two cells. Figure 5.7 (right) illustrates this subdivision scheme.

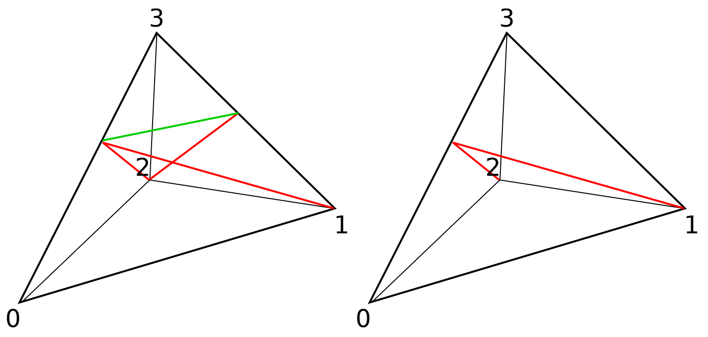


Figure 5.7: Subdivision schemes of margin tetrahedra. *Left:* Margin tetrahedron adjacent to an intersected cell. The cell gets divided into three cells. *Right:* Margin tetrahedron incident to an intersected edge. The cell gets divided into two cells.

5.5 3D Reconstruction Using Tetrahedral Occupancy Labeling

We aim to compute a dense watertight surface as the interface between two disjoint sets labeling every tetrahedron in the scene as either inside or outside. Hence, the surface is fully described by a binary labeling $\ell : T \to \{0, 1\}$. To this end, we formulate the following energy minimization problem which expresses each of our goals with a particular energy term:

$$\underset{\ell}{\text{minimize}} \quad E_{\text{Vis}}(\ell) + \alpha_{\text{Man}} E_{\text{Man}}(\ell) + \alpha_{\text{LoD}} E_{\text{LoD}}(\ell) \ . \tag{5.1}$$

Each of these terms enforces or favors a different property. In particular, E_{Vis} scores the visibility of tetrahedra, E_{Man} favors Manhattan-like solutions and E_{LoD} allows for level of detail adjustment. The corresponding weights $\alpha_{\text{Man}}, \alpha_{\text{LoD}} \in \mathbb{R}_{\geq 0}$ balance the impact of each term. All energy terms are non-negative, submodular and a globally optimal solution can thus be computed via Graph Cuts [10]. The energy terms defined in Equation 5.1 will be discussed in detail in Section 5.6 and Section 5.7: In Section 5.6 we will present two formulations for the visibility-based energy term $E_{\text{Vis}}(\ell)$ and in Section 5.7 we will discuss the Manhattan regularization term $E_{\text{Man}}(\ell)$ and its counterpart, the level of detail term $E_{\text{LoD}}(\ell)$.

5.6 Visibility-Based Energy Computation

In this section, we describe two visibility-based energy formulations used in our approach, which correspond to $E_{\mathrm{Vis}}(\ell)$ in Equation 5.1. First, we describe a commonly used unnormalized energy formulation (Section 5.6.1), then we describe the introduced normalized energy formulation (Section 5.6.2), which eases the combination with other energy terms. Finally, we describe two methods for retrieving the visibility information from the input point cloud.

5.6.1 Unnormalized Energy Formulation

For each cell and for each facet, costs are computed based on available visibility information for every point or vertex in the triangulation. In this section we briefly describe a commonly used formulation of these costs, which are the dominant information in the final optimization. For a detailed description we refer the reader to Section 2.5.2.

These visibility-based costs $E_{\mathrm{Vis}}(\ell)$ are defined as described in [48]. We define unary costs providing a point-wise prior on the cell occupancy, as well as pairwise costs which locally favor or penalize labeling transitions. The costs defined in this section are also used in [37] and are illustrated in Figure 2.9.

5.6.1.1 Unary Costs

The unary costs derived from the visibility information are defined as follows: Every cell containing a camera and every infinite cell is labeled as outside by adding infinite weights. Contrarily, every point which is directly behind a vertex (seen from the camera) is labeled as inside. For this, the cell behind the point gets a finite weight for each camera where the point is visible in. Here, we explicitly avoid using infinite weights, since point measurements are prone to noise and may contain outliers.

5.6.1.2 Pairwise Costs

The pairwise costs are set to penalize ray conflicts for every camera to point correspondence. The pairwise costs are only added in one direction, since faces cannot exist in front of a measured point. Hence, every intersection of a camera-to-point ray with a facet gets a constant penalty. In addition to the visibility-based costs, we add constant pairwise costs as a simple regularization term. Although more complicated regularization terms exist in literature (e.g., the beta skeleton term [50]), adding constant costs has shown to be the simplest and most effective regularization term [63].

5.6.2 Normalized Energy Formulation

Visibility-based energy terms as described in the last section tend to be very hard to normalize, as the magnitude of the energy depends on the density of the point cloud,

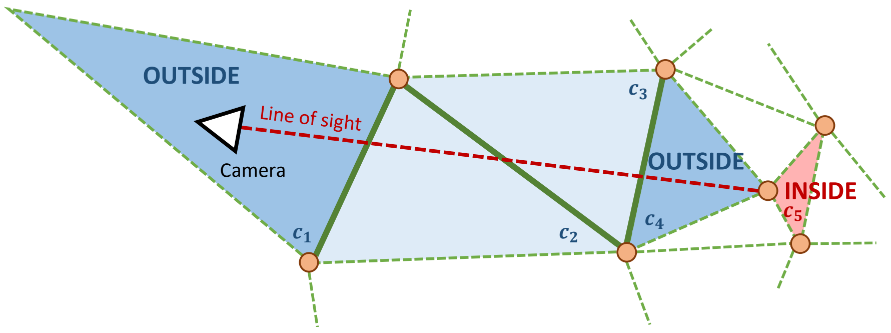


Figure 5.8: *Normalized visibility-based energy computation.* We use ray casting to compute the visibility terms: The cell where the camera is located in (c_1) is labeled as outside by adding infinite weights. Then, every facet (green) which is intersected by the line of sight (red) gets pairwise costs assigned in both directions. Finally, the cell in front of the visible vertex (c_4) is labeled as outside by adding finite weights and the cell behind the vertex (c_5) is labeled as inside by adding finite weights.

the number of cameras the current tetrahedron is visible in and the visibility information of the surrounding. However, having an unnormalized energy is very problematic when trying to combine it with other energy terms.

Therefore, we propose a normalized energy formulation [36], which is slightly better in terms of accuracy and, more importantly, has a normalized magnitude with which a more intuitive combination with additional energy terms is possible.

Inspired by [48] and [38], the energy terms are based on ray casting from every vertex to every camera the vertex is visible in. Unary costs are assigned to cells intersected by a ray adjacent to the visible vertex (i.e., before and behind the vertex), to infinite cells and to cells including a camera, and pairwise costs are assigned to facets intersected by rays (see Figure 5.8). Opposing to [48], we do not only assign pairwise terms in one direction but in both, and we additionally add unary costs in front of a visible vertex. This has shown to significantly improve the result when using normalization afterwards (as demonstrated in Section 6.5.3.2). In order to generate normalized cost terms, our general idea is the following: The visibility-based energies should change significantly when the terms are still low and additional information is added, but the influence of additional visibility information when already sufficient information is available should be decreased. Hence, it should have a significant effect if a point is visible by 1 or 5 cameras, but the effect should be reduced if the visibility is changed from 21 to 25 cameras.

The normalized unary terms for cells directly in front of and behind visible vertices are defined as follows:

$$E_{unary}(t) = (1 - e^{-\frac{\#rays}{limit_u}})limit_u, \tag{5.2}$$

where t defines the current tetrahedron, $\#rays$ define the number of rays intersecting the tetrahedron and $limit_u$ is the energy limit approached asymptotically.

The normalized pairwise terms for every facet are defined as follows:

$$E_{pairwise}(f) = (1 - e^{-\frac{\#rays}{limit_p}})limit_p, \tag{5.3}$$

where f defines the current facet, $\#rays$ define the facet's number of ray intersections and $limit_p$ defines the energy limit. The limits of the unary and pairwise energies need to be set so that additional energy information is not ignored too early. Additionally, the pairwise terms need to be allowed to become stronger, as otherwise at large facets at holes or below roofs the unary term might spuriously dominate and, hence, artifacts might arise. We found out empirically that a setting for $limit_u = 8$ and $limit_p = 24$ is a good choice for most scenes.

These improved visibility-based energy terms are non-negative and submodular. The output is a normalized energy having maximum unary and pairwise terms, which is crucial for combining it with additional energy terms and, hence, makes it possible to easier find scene-independent parameter settings.

5.6.3 Computation of Visibility Information

In order to compute the visibility-based costs for the whole scene, one needs the visibility information for every 3D point, i.e. one has to know for every point in which view it is visible in. In this section, we discuss two methods for computing this information.

5.6.3.1 Using Pre-Computed Visibility Information

Having *MVS* point clouds as input, it is possible to directly use the visibility information delivered by the *MVS* algorithm (as applied in [37]). No visibility computation needs to be done for the input point cloud and, hence also no depth maps are needed as input.

However, when subdividing tetrahedra (see Section 5.4), new vertices are created which have no visibility information assigned. Hence. the visibility-based energy computed for the original, undivided tetrahedra needs to be propagated to the newly created ones.

Energy Propagation at Tetrahedra Subdivision. Using the visibility-based unary and pairwise costs described in Section 5.6.1 or in Section 5.6.2, it is necessary to have visibility information available for each point (i.e. camera-point correspondences). When using a fused *MVS* point cloud with included visibility information as input, there is no known visibility information for the points created by the tetrahedra subdivision. Hence, the visibility-based cost need to be computed using the original tetrahedralization and propagated to the subdivided cells. In the following, we describe the propagation for unary and pairwise energy terms.

For the unary costs we assign the original cost scaled by the volume of the new tetrahedron:

$$E_{\text{unary}}(t) = E_{\text{unary}}(t_{\text{orig}})\frac{v_t}{v_{t_{\text{orig}}}}, \tag{5.4}$$

where t is the new tetrahedron and v_t its corresponding volume, t_{orig} and $v_{t_{\text{orig}}}$ are the original tetrahedron and its corresponding volume, respectively.

For the pairwise costs the weight is scaled according to the area of the facet, if the facet of a new cell is part of a facet of the old cell:

$$E_{\text{pairwise}}(f) = E_{\text{pairwise}}(f_{\text{orig}}) \frac{a_f}{a_{f_{\text{orig}}}}, \tag{5.5}$$

where f, f_{orig} and $a_f, a_{f_{\text{orig}}}$ are the new and original facets with their corresponding areas.

The pairwise costs for all other facets (i.e., facets inside of the old cell), the biggest facet from the original cell with a sufficiently small enclosing angle with the new facet is selected and scaled according to the area.

In order to select a corresponding original facet for new facets not lying on an original one, we retrieve all facets with an enclosing angle smaller than 0.3 *rad* (approx. 17.2 *deg*) and take the biggest one of the retrieved facets. If no facet fulfilling this property exists, we increase the maximum enclosing angle by 0.1 *rad* until an appropriate facet is found.

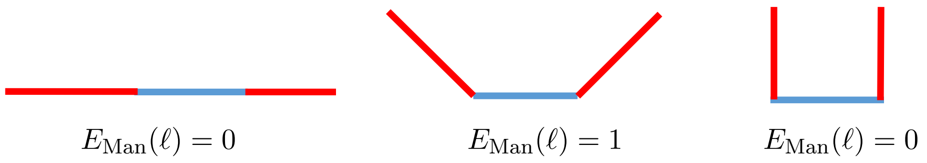
With this propagation scheme, the unary costs overall stay the same and the pairwise costs are propagated from facets which are as similar as possible to the new facets and, by taking the biggest one, carry as much as possible information.

When using depth maps for visibility computation (as described in the next section) this cost propagation step is not necessary, as the visibility-based costs get computed on the final triangulation having already the subdivided tetrahedra included.

5.6.3.2 Visibility Computation Using Depth Maps

To compute visibility-based energy terms, the knowledge of the visibility information of every 3D point (i.e., camera-point correspondences) is necessary. Hence, most of the methods following visibility-based cost computations similar to Labatut et al. [48] assume that the visibility information is known. However, when tetrahedra are subdivided, new points without visibility information are created. Additionally, input 3D information without visibility information cannot be used. Hence, we propose to compute this information for all 3D points using depth maps. Using this method, no visibility cost propagation for subdivided tetrahedra (as proposed in the previous section) is necessary.

Assuming to have dense depth maps for every camera, we project every 3D point into all cameras. If the point is within the image boundaries and in front of the camera, we compare the actual distance of the point to the camera with the depth value in the depth map (using nearest neighbor). If the depth difference is small enough (i.e., smaller than 0.03 m), we assume that the current point is actually visible in this camera and store this camera-point correspondence. Having this estimated visibility information, the visibility-based cost terms can be computed for all tetrahedra including the subdivided ones.



$E_{\text{Man}}(\ell) = 0$
$E_{\text{Man}}(\ell) = 1$
$E_{\text{Man}}(\ell) = 0$

Figure 5.9: *Manhattan regularity term for neighbor facets with different angles.* When a facet (blue) has coplanar neighboring facets (left) or perpendicular neighboring facets (right), E_{Man} is 0. When a facet has neighboring facets with an enclosing angle of 135 deg or 45 deg, E_{Man} is at its maximum.

5.7 Plane-Aware Regularization

In this section, we introduce new regularization terms which allow a continuous choice between generic 3D reconstructions and reconstructions in which the pre-detected planes and Manhattan-like structures are increasingly replacing surface noise and details of the surface structure.

In Section 5.7.1 we will describe the introduced Manhattan regularity term, which enforces the final reconstruction to be aligned with the detected planes. In Section 5.7.2, we will introduce the level of detail term, which is the counterpart of the Manhattan term and brings big smoothed out elements back. Finally, in Section 5.7.3, we will discuss an artifacts removal strategy used.

5.7.1 Manhattan Regularity Term

Similar to [52], we introduce a regularity term which favors orthogonal and parallel scene structures. The following term favors label transitions with Manhattan-like surface structures, i.e., neighboring facets with enclosing angles similar to 0 or multiples of 90 degrees. More exactly, this term penalizes facets which cannot be part of a Manhattan-like surface structure:

$$E_{\text{Man}}(\ell) = \sum_{f \in T} \mathbf{1}_{\{\ell_{t_1} \neq \ell_{t_2}\}} \frac{a_f}{3} \sum_{e \in f} \min_{g \in \mathcal{N}_e} \left\{ \left| \sin\left(2\,\angle(f,g)\right) \right| \right\} \quad , \tag{5.6}$$

where f denotes a facet in the tetrahedralization T, $\mathbf{1}_{\{\cdot\}}$ is the indicator function, ℓ_{t_1} and ℓ_{t_2} are the labels of the adjacent tetrahedra of facet f, a_f the area of f, e are the edges of f and $\mathcal{N}_e$ are all incident facets of edge e. An illustration of the term is depicted in Figure 5.9.

As a major advantage, the term favors Manhattan-like structures, but does not strictly enforce them and is therefore applicable to any kind of surface type. Moreover, the term acts completely local and the surface does not need to be aligned with any world coordinate axis.

5.7.2 Level of Detail Term

In order to control the amount of detail which is removed by the Manhattan term and due to the favoring of pre-detected planes, we introduce another term controlling the amount of removed structure according to its size. For instance, we want to remove the noise on a building roof but keep the chimney. To achieve this, we introduce a new term penalizing the volume deviation of the plane-aware reconstruction including the Manhattan regularity term with respect to the original non-regularized reconstruction. Using a visibility-based energy defined in Section 5.6, the original generic 3D reconstruction without favoring planes is defined as

$$\ell^{\text{Vis}} = \arg\min_{\ell} \, [E_{\text{Vis}}(\ell)]. \tag{5.7}$$

A natural error measure to control structure removal upon plane replacement is the volume difference between the two models. We hence define the level of detail term as follows:

$$E_{\text{LoD}}(\ell) = \sum_{t \in T} v_t \, \mathbf{1}_{\{\ell_t \neq \ell_t^{\text{Vis}}\}} \ , \tag{5.8}$$

where t defines a tetrahedron in the tetrahedralization T, $\mathbf{1}_{\{\cdot\}}$ is the indicator function, ℓ_t is the labeling of t, and v_t denotes the volume of tetrahedron t. This term acts as the counterpart of the Manhattan regularity term: While the Manhattan regularity term removes details not supported by any plane, this term allows to control the amount of details to be removed.

5.7.3 Plane Intersection Artifacts Removal

These terms work well in most cases, but artifacts may arise near plane intersections, typically along sharp edges in the scene like building outlines. Cells which contain a unary term voting for being inside while actually being outside of the object may exist enclosed by two planes due to noise within the 3D reconstruction. However, as some of the cells' facets are lying on the planes and, hence, are not penalized by the Manhattan term, the smoothness term is not strong enough to enforce sharp edges.

To avoid such artifacts, we reduce the influence of the unary and pairwise terms in scene parts around plane intersections. For every tetrahedron for which its centroid point has a normal distance d to the plane intersection smaller than $3d_{\text{inlier}}$ (with d_{inlier} being the plane inlier distance defined in Section 5.2), we update the unary costs by:

$$E(t) = E_{\text{orig}}(t) \left(1 - \exp\left(\frac{-d^2}{3d_{\text{inlier}}^2}\right)\right), \tag{5.9}$$

with t being the tetrahedron to update and $E_{\text{orig}}(t)$ being the initial unary cost before the update.

Similarly, the pairwise costs at facets are updated. For all facets of tetrahedra which are within the distance of $3d_{inlier}$ to the plane intersection and are located on one of the planes, we update the pairwise costs accordingly:

$$E(f) = E_{\text{orig}}(f)\left(1 - \exp\left(\frac{-d^2}{3d_{\text{inlier}}^2}\right)\right), \quad \text{if } f \in \mathit{planes}, \tag{5.10}$$

where $E_{\text{orig}}(f)$ are the initial pairwise costs of the facet f and *planes* is the set of all planes.

Using these cost updates, the influence of inside labeled cells violating the smoothness constraints is reduced and the surface is enforced more intensively to follow the planes in the vicinity of plane intersections.

5.8 Semantically Aware Urban Reconstruction

Using the reconstruction techniques discussed in the previous sections, it is possible to create reconstructions of urban environments consisting of planar surfaces and sharp edges. However, the Manhattan regularity term (introduced in Section 5.7.1) might smooth parts of the scenery not consisting of planar surfaces (like, for example, vegetation) in a wrong way which might lead to artifacts. Further, when focusing on reconstruction of urban environments, one might want to have well smoothed reconstructions of the surroundings of buildings containing less clutter and noise and a smooth surface while simultaneously having buildings reconstructed with planar surfaces and sharp edges while still containing many details.

Using semantic information, it is possible to handle different parts of the scene differently [36]. More precisely, we apply preprocessing steps depending on the semantic classes of the scenery. Then, we use different smoothness terms for different scene parts in order to reconstruct buildings with a Manhattan prior, while the surrounding should be represented by a smooth surface.

5.8.1 Semantic Segmentation

The goal of the semantic segmentation is to get the 3D reconstruction semantically enhanced to be able to perform automated decisions throughout our processing pipeline. To achieve this goal we follow the work of [60] to perform pixel-wise semantical segmentation of the input images. To transfer the labels from 2D to 3D, each 3D point is back projected according to its visibility to 2D and a final majority voting determines the label of the 3D point.

For the semantic segmentation of the input images we use a Fully Convolutional Neural Network (FCN) [55] to get pixel-wise segmentations. The network presented in [55] is adjusted to represent the number of output classes required for our task. We define five output classes, namely: street/pavement, building, vegetation, sky and clutter. As our intermediate aim is to semantically enhance the 3D reconstruction, we need a pixel accurate segmentation of the input images where the segmentation boundaries are aligned with the objects present. Thus, the receptive field of the *FCN* of 32 px and the final up sampling by a factor of eight are too coarse to achieve this goal. We extend the 2D segmentation network by adding a Conditional Random Field as Recurrent Neural Network (CRFasRNN) as presented in [99]. The Conditional Random Field exploits the probabilities of the *FCN* and refines them by taking binary constraints into account. This enforces label changes being aligned with edges.

Having the pixel-wise semantic segmentation of the input images, we propagate this information to 3D: Assuming 3D points with visibility information, we back project every point into every image in which it is visible in and compute a point label by majority voting. For getting labels of the 3D lines, we sample every line with points and compute a label for every point as explained above. The most frequent label within the sampled points defines the line label.

5.8.2 Input Data Subdivision and Semantic Preprocessing

Depending on the semantic label, we initially subdivide the scene into two parts which will be processed differently: For the building part we keep all available 3D information. For the non-building part we sparsify the input data, compute a Poisson surface and use the sampled Poisson surface which results in a smoother, visually appealing reconstruction (e.g., less spurious peaks at vegetation). In the final optimization, the subdivided parts are combined again in order to create a reconstruction of the whole scene.

As we want to have a point cloud representation of the building part which covers all important details, we add all input points and additionally add points from sampled building line segments to the scene (line sampling distance 0.05 m). Adding the sampled line points especially helps at poorly textured scene parts where few reconstructed points are available. This enriched point cloud is subsequently used for triangulation (as described in Section 5.3).

In contrast to the building part, we want a very smooth representation of the surroundings of the building. Hence, we first sparsify these classes: For the clutter class we just keep every fifth point, for street/pavement and vegetation we keep every third point and we remove all sky points. Then, we compute a Poisson surface [43] using these selected points. As Poisson surface reconstruction needs points with its corresponding oriented normals as input, we compute the normals in case no precomputed ones are available. For this, we use Principal Component Analysis (PCA) in order to compute unoriented normals and estimate the orientation using an algorithm similar to [39]. For details about Poisson surface reconstruction we refer the reader to Section 2.5.1. For the triangulation of the scene and subsequent reconstruction steps, we don't use the original street/pavement, vegetation and clutter points but a sampled point representation of the computed Poisson surface. This results in a much smoother surface of this part of the scene in the final reconstruction.

Using this preprocessed point cloud, a Delaunay Triangulation is created (as described in Section 5.3), which naturally combines both representations into one tetrahedral cell complex.

5.8.3 Semantically Varying Smoothness Terms

As has been discussed in Section 5.5, the final 3D reconstruction is computed by energy minimization minimizing visibility-based cost terms, a Manhattan smoothness term and a Level of Detail term. When using semantic information, we rephrase Equation 5.1 for setting class-specific energy terms:

$$\underset{\ell}{\text{minimize}} \quad E_{\text{Vis}}(\ell) + E_{\text{Class}}(\ell) \ , \tag{5.11}$$

where $E_{\text{Vis}}(\ell)$ is the visibility-based energy and $E_{\text{Class}}(\ell)$ are class-specific energy terms, which will be explained in more detail in this section.

Depending on the semantic class facets and cells in the triangulation are assigned to, additional energy terms are added. First, we compute the class dependence for every facet and cell by computing a majority vote using all their corresponding vertices. Then, scene parts get assigned different energy terms depending on their semantic labels.

The energy terms assigned to building parts favor Manhattan-like structures but simultaneously aim to keep important details and were already discussed in detail in Section 5.7.1 and Section 5.7.2: They consist of a Manhattan regularity term E_{Man}, which favors label transitions with Manhattan-like surface structures (i.e., neighboring facets with enclosing angles similar to 0 or multiples of 90 degrees), and a level of detail term E_{LoD}, which punishes volumetric errors with respect to the unregularized model. Hence, E_{LoD} is the counterpart to E_{Man} and brings back smoothed out details which are not supported by planes. Using these energy terms, we strongly favor planar and Manhattan-like structure while still keeping sufficiently big details.

For non-building parts, our goal is to get a reconstruction which is as smooth as possible. Therefore, we just add an area smoothness term E_{area} as defined in [48]. This term should remove spurious artifacts.

Hence, the class-specific energies are defined as follows:

$$E_{\text{Class}}(\ell) = \begin{cases} \alpha_{\text{Man}} E_{\text{Man}}(\ell) + \alpha_{\text{LoD}} E_{\text{LoD}}(\ell) & \text{if building} \\ \alpha_{\text{area}} E_{\text{area}}(\ell) & \text{else} \end{cases} \ , \tag{5.12}$$

where α_{Man}, α_{LoD} and α_{area} define the amount of smoothing.

Having the semantic information incorporated into the reconstruction process, it is possible to reconstruct the surrounding of the buildings more smoothly due to the intermediate Poisson surface representation and the used area term. Further, less spurious artifacts arise at the surroundings, as the Manhattan regularity term is only applied at the building parts. However, the regularization of the buildings stays approximately the same, as the Manhattan regularity term and the level of detail term are applied to these scene parts the same way than without semantics.

5.9 Summary

In this section, we presented a 3D reconstruction method which creates visually appealing reconstructions of urban scenes. First, planes are detected in the scene by a point-based plane detection algorithm or by using a line-based plane detection algorithm which improves the plane detection result especially in poorly textured environments. The planes are incorporated within a tetrahedra-based reconstruction framework where semantic information is used to set shape priors accordingly to get a detailed reconstruction of building parts containing planar surfaces and sharp edges and a smoothed reconstruction of the surroundings. Further, we introduced a normalized energy formulation used within the reconstruction process, which eases the combination of multiple energy terms. In Section 6.5.2 and Section 6.5.3 we present a detailed evaluation of this method containing evaluation with different parameter settings and comparisons to other state-of-the-art methods.

6 Experiments

Contents

6.1 Input Data 67
6.2 Implementation Details 71
6.3 Default Parameters 71
6.4 Results and Comparisons 72
6.5 Method-Specific Detailed Evaluations 78
6.6 Summary 100

In this chapter, we show quantitative and qualitative comparisons of the proposed approaches to other state-of-the-art methods. Further, we evaluate the individual contributions of the proposed approaches and give detailed parameter studies.

First, we describe the input data used for the main experiments. Next, we discuss implementation details in Section 6.2 and define default parameter settings in Section 6.3. In Section 6.4, we show results of the proposed approaches and others and evaluate them quantitatively and qualitatively. In Section 6.5 we discuss method-specific detailed evaluations, which contain parameter studies, additional results and evaluations of individual contributions. Finally, in Section 6.6 we summarize the chapter.

6.1 Input Data

In this section, we describe the three main evaluation datasets used in this thesis in detail. This datasets contain urban scenes including buildings and were acquired using an Unmanned Aerial Vehicle (UAV) equipped with a Sony Alpha 6000 camera capturing images with a resolution of 24.3 MPixel. For two datasets, also ground truth 3D data was acquired using a total station.

For all datasets, we used our own Structure from Motion (SfM) implementation using images only as input to compute the camera poses. To obtain metric input, we scaled the

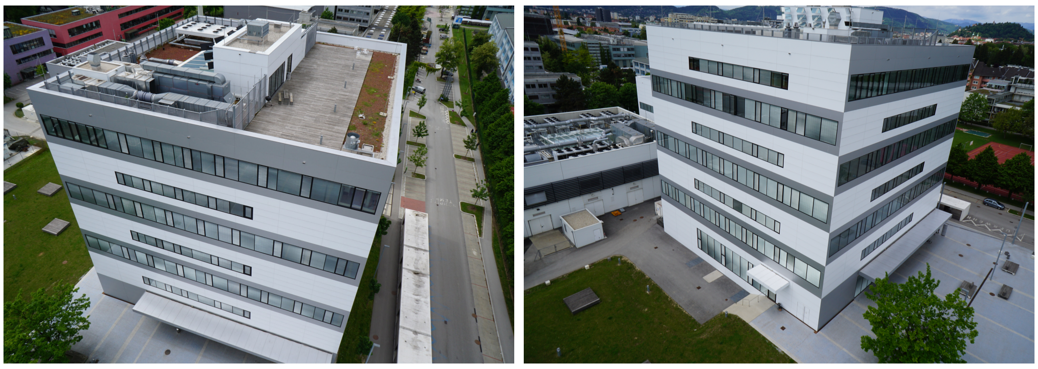


Figure 6.1: *Input images of the Block Building.* As one can see, this simple building consists mainly of planar façades, including several windows and poorly textured surfaces.

SfM reconstruction using measured fiducial markers in the scene. In case no markers were available, we manually scaled it to an approximate metric scale.

Having the *SfM* result as input, we used several densification methods within our experiments: We computed a dense point cloud using PMVS2 [22] or Sure [79] and computed a 3D line reconstruction using Line3D++ [33].

As input meshes for the slicing-based method, we computed a Poisson mesh [39] with Meshlab [92] using the oriented dense points as input.

As the tetrahedra-based method may also need additional input information, we computed dense depth maps using PlaneSweepLib [28] and detected line segments in images using the Line Segment Detector (LSD) [27].

Below, we will give a detailed overview of the three main datasets. Other additional datasets used for method specific evaluations in Section 6.5 will be introduced at the individual experiments.

6.1.1 Block Building

The first dataset, in the following referred as *Block Building*, contains 232 images and shows a scene including a building shaped like a single block and surroundings containing mainly streets and vegetation. On the roof of the building, there are additionally some installations and a smaller block with an exit onto the roof. The façades of the building contain many windows and poorly textured parts. Figure 6.1 shows example input images. The dense point cloud reconstruction used in the main evaluation in Section 6.4 was computed with PMVS2, which resulted in 1.4M points and can be seen in Figure 6.4 (left).

For this dataset, no ground truth 3D data and no measured fiducial markers were available and, hence, we manually scaled it to metric scale.

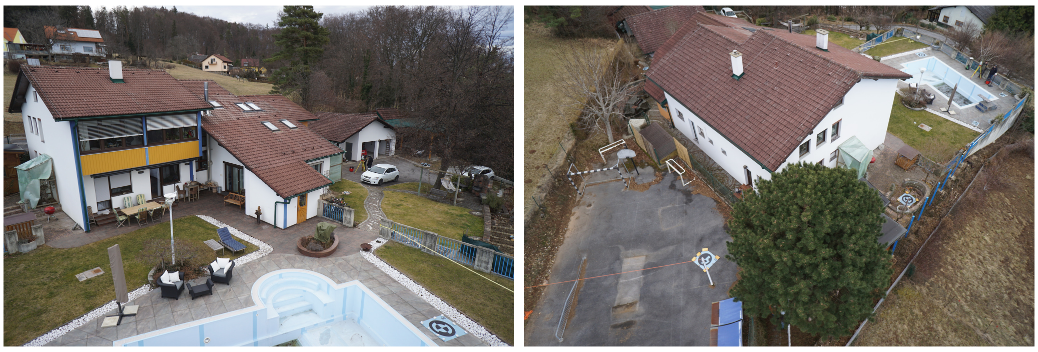


Figure 6.2: *Input images of the House dataset.* The main part of this dataset is a single family house, surrounded mainly by vegetation, pavement and a swimming pool.

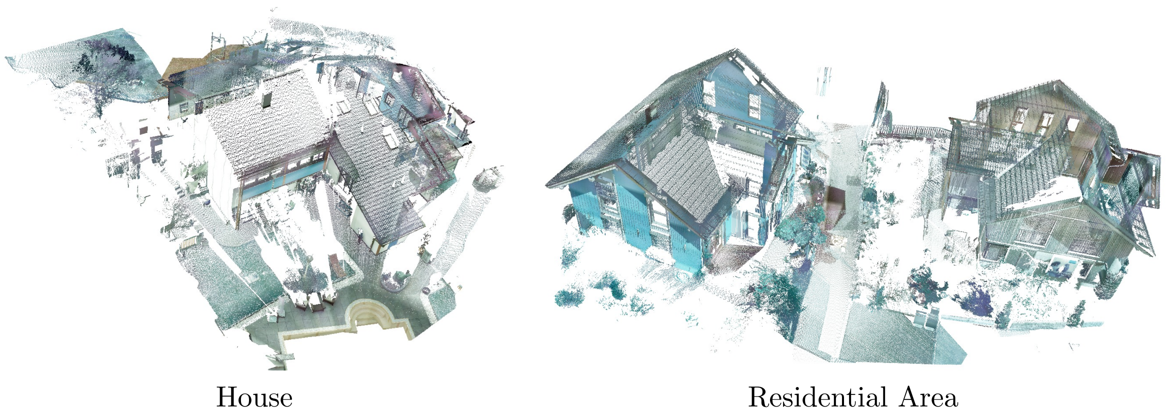


Figure 6.3: *Ground truth point cloud* captured with a total station. Even though the ground truth was captured from several view points, it is not complete at all parts: E.g., big parts of the roof are missing at both datasets and the façade on the left side of the house is missing in the *House* dataset.

6.1.2 House

The next dataset, in the following referred as *House*, contains a scene with a single family house surrounded mainly by vegetation and pavement. From this scene, 233 images were acquired and a dense point cloud was computed with Sure [79] and downsampled to 900K points. This point cloud is visualized in Figure 6.4 (middle) and example input images are depicted in Figure 6.2.

For this dataset, ground truth 3D data was acquired using a total station and measured fiducial markers where used to convert the reconstruction to a metric scale. In Figure 6.3 (left), the ground truth point cloud is visualized.

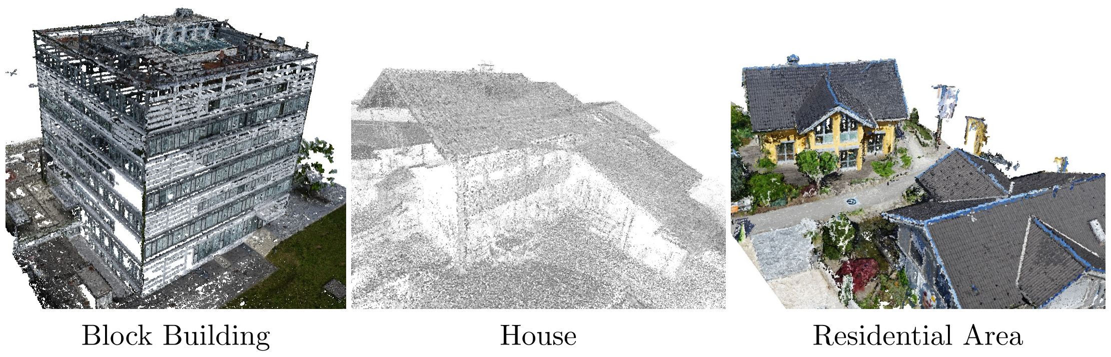


Figure 6.4: *Point clouds of all three evaluation scenes.* At the *House* dataset (middle), one can observe that some parts of the scene like the façade on the left side of the house is reconstructed very sparsely due to missing texture. Also at the *Block Building* (left) dataset, some parts are not reconstructed well due to a white façade and many windows.

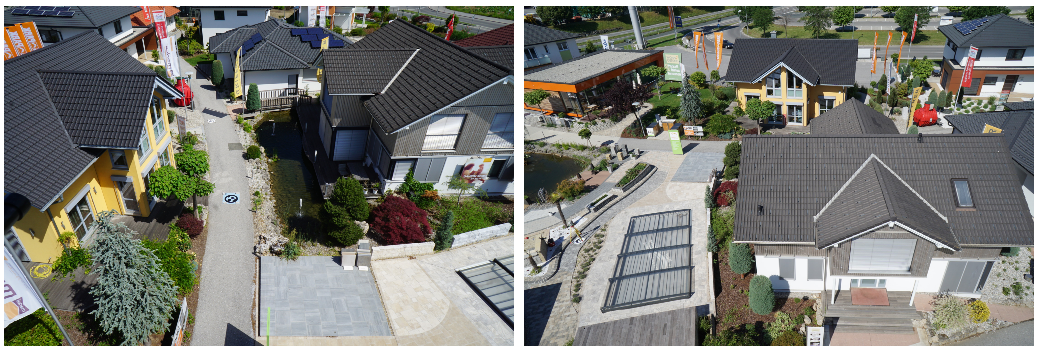


Figure 6.5: *Input images of the Residential Area dataset.* The main part of this dataset contains two houses within a residential area surrounded by vegetation, pavement, street and a small lake in front of a house.

6.1.3 Residential Area

The *Residential Area* dataset contains a scene with several family houses within a residential area, where two of these houses are covered well with imagery. These two houses are surrounded by pavement, a street, a small lake and vegetation. From this scene 446 images were acquired and a dense point cloud was computed using PMVS2 consisting of 3.6M points, which can be seen in Figure 6.4 (right). In Figure 6.5, example input images are depicted.

Also for this dataset, ground truth 3D data was acquired using a total station and measured fiducial markers where used to convert the reconstruction to a metric scale. In Figure 6.3 (right), the ground truth point cloud is depicted.

6.2 Implementation Details

In this section, we list libraries used in our implementation and give some implementation details.

The proposed algorithms are mainly implemented in C++. The implementation uses OpenCV [72] for various image processing steps, the Graph Cut implementation from Olga Veksler [45] [10] [9] and CGAL [89] for the 3D Delaunay triangulation, the 2D constrained Delaunay triangulation and the Poisson meshing (used for the surroundings of the buildings when using semantic information).

The semantic segmentation network is realized in the Caffe framework [41]. For initialization we exploited the weights of the PASCAL-Context network [73] and performed a transfer learning of the network based on 27 labeled training images (16 manually labeled, 11 taken from eTRIMS dataset [46]) that were augmented in scale (0.8, 1.0, 1.2), rotation [deg] (0, 90, 180, 270) and mirroring. Additionally, the augmented images were cropped to patches of 256×256 *px* to easily fit to GPU memory. In total we resulted in a training database of 32,016 image patches. The training itself has been performed in stages to consecutively train the FCN32s, FCN16s, FCN8s and FCN8s with CRFasRNN. Each stage was trained for 400,000 iterations using Stochastic Gradient Descent with a momentum of 0.99, weight-decay of 0.0005 and a learning rate of $1e^{-9}, 1e^{-10}, 1e^{-12}$ and $1e^{-12}$ for each stage respectively.

6.3 Default Parameters

In the following, we define default parameters for the slicing-based method described in Chapter 4 and define two sub-methods and its corresponding parameters for the tetrahedra-based method described in Chapter 5, which are used consecutively in this chapter.

For the **slicing-based method**, we use both the line and mesh smoothness terms and set $\lambda_{lines} = 0.5$ and $\lambda_{mesh} = 0.5$. Further, we set the mean shift parameter d depending on the scene: If a scene contains few vertical structure (like the *Block Building* dataset) we set $d = 50$, if it contains more horizontal/sloped structure (like the *House* and *Residential Area* dataset), we increased this parameter to $d = 80$ in order to generate more slices and to approximate sloped surfaces in a more detailed way.

For the tetrahedra-based method, we define two sub-methods which use different modules of the method and will be evaluated individually:

The first sub-method will be denoted as **plane-based regularization method** in the following sections. It uses point-based plane detection (as described in Section 5.2.1), the unnormalized visibility-based energy formulation as described in Section 5.6.1, precomputed visibility information (as described in Section 5.6.3.1) and no semantic information. This is the same setting as used in [37]. As parameters, we set $\alpha_{Man} = 250K$ for all three datasets. We set $\alpha_{LoD} = 375K$ for the *Block Building* and $\alpha_{LoD} = 250K$ for the

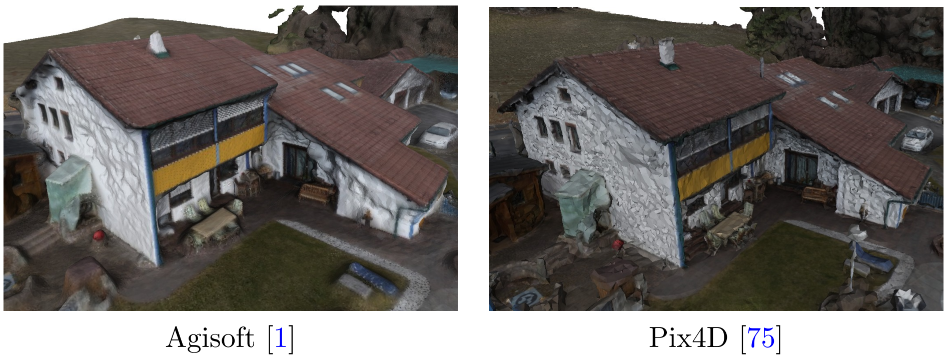


Figure 6.6: *Results from commercial reconstruction pipelines of the House dataset.* Compared to the reconstruction of the proposed approach (see Figure 6.26), Agisoft and Pix4D do not generate planar surfaces and straight edges, as they use no shape or semantic priors. The façades and roofs are noisy and some scene parts are smoothed too much and become rounded (e.g., building edges, chimney).

House and *Residential Area* dataset.

The second sub-method will be denoted as **semantically aware method** in the following sections. It uses line-based plane detection (as described in Section 5.2.2), the normalized visibility-based energy formulation as described in Section 5.6.2, visibility information computed using depth maps as described in Section 5.6.3.2, and semantic information as described in Section 5.8. This setting was also used in [36]. For the three presented datasets the parameters were set as follows: For datasets *House* and *Residential Area*, the parameter set was $\alpha_{Man} = 1000$, $\alpha_{Lod} = 500$. For the dataset *Block Building*, the parameters were set differently to impose a stronger plane prior as this dataset contains more noise near to planar surfaces ($\alpha_{Man} = 2500$, $\alpha_{Lod} = 1250$). α_{area} was set to 0.5 for all datasets.

6.4 Results and Comparisons

In this section, we compare reconstruction results from the proposed methods with generic 3D reconstruction algorithms and with specialized urban reconstruction approaches.

Additionally, we show reconstruction results from two commercial reconstruction pipelines in Figure 6.6. One can observe that these approaches are not explicitly designed for creating visually appealing urban reconstructions. Up to our knowledge there do not exist commercial products which produce results following our goal definitions, which are 3D reconstructions from urban scenes containing planar surfaces and sharp edges at building parts and ideally also a smooth reconstruction of the surroundings.

In Figure 6.7, results from proposed approaches and state-of-the-art reconstruction algorithms are depicted. All of the aim to create visually appealing 3D reconstructions having specific predefined properties: The Poisson surface reconstruction produces smooth surfaces, which results in rounded edges. Additionally, it cannot handle missing data very well and creates spurious artifacts. Polyfit [66] heavily depends on the (point-based) plane detection result and reconstructs non-planar parts not very well, as it just relies on detected planes and uses only the plane surfaces to create an optimized surface model. The slicing-based method lacks the ability to model sloped surfaces. Hence, sloped roofs in the House and Residential Area dataset can only be approximated by stairway-like structures. Further, as the ground in the House dataset is also slightly sloped and, hence, the ground plane is detected wrongly, the horizontal slicing of the scene does also not work ideally. However, for well suited scenes like the Block Building, this method produces well regularized 3D models. The plane-based regularization method incorporates plane priors into a generic reconstruction method and aims to reconstruct planar parts of the scene as perfectly planar surfaces. However, some planar surfaces are not detected correctly and due to the unpredictability of the visibility-based energy it is hard to set the smoothness energy weights correctly. This might lead to artifacts like smoothed out wall parts or whole buildings. As this approach has no semantic class specific smoothing, also the surroundings of buildings are smoothed heavily according to a plane prior and, hence, some parts (like, e.g., vegetation) are smoothed too aggressively and artifacts looking like slices might arise. In comparison, the semantically aware approach uses a more complete plane set as shape prior and imposes planar surfaces just on buildings. Due to the improved visibility-based energy formulation it is easier to set correct smoothness term weights and, hence, to avoid over-smoothing. The representation of the surroundings is smooth while still not over-smoothed.

In Table 6.1 error metrics for two datasets with respect to the ground truth are depicted and in Figure 6.8 these errors are visualized on the ground truth data. It can be

	House		Residential Area	
	μ [m]	σ[m]	μ [m]	σ[m]
Poisson [43]	0.165	0.237	0.101	0.157
Polyfit [66]	0.515	0.352	0.304	0.375
Slicing-based	0.623	0.427	0.323	0.398
Plane-based regularization	0.137	0.237	0.415	0.385
Semantically aware	**0.126**	**0.233**	**0.055**	**0.086**

Table 6.1: *Error statistics compared to the ground truth.* We computed the minimal Euclidean distances of the ground truth points to the surface reconstructions with a maximum distance of 1 m and depicted the mean and standard deviation for all points in this table. On both datasets, the proposed semantically aware method has the lowest error. The slicing-based method has high errors on both datasets, as it in general does not model the ground surrounding the buildings and can only approximate the sloped roofs. Polyfit also has high errors on both datasets, as it does not reconstruct the whole scene but just (parts of) buildings well. The plane-based regularization method has the highest error on the *Residential Area* dataset due to a wrongly smoothed out building. Poisson produces over-smoothed (i.e., no sharp edges) results, but has comparable errors.

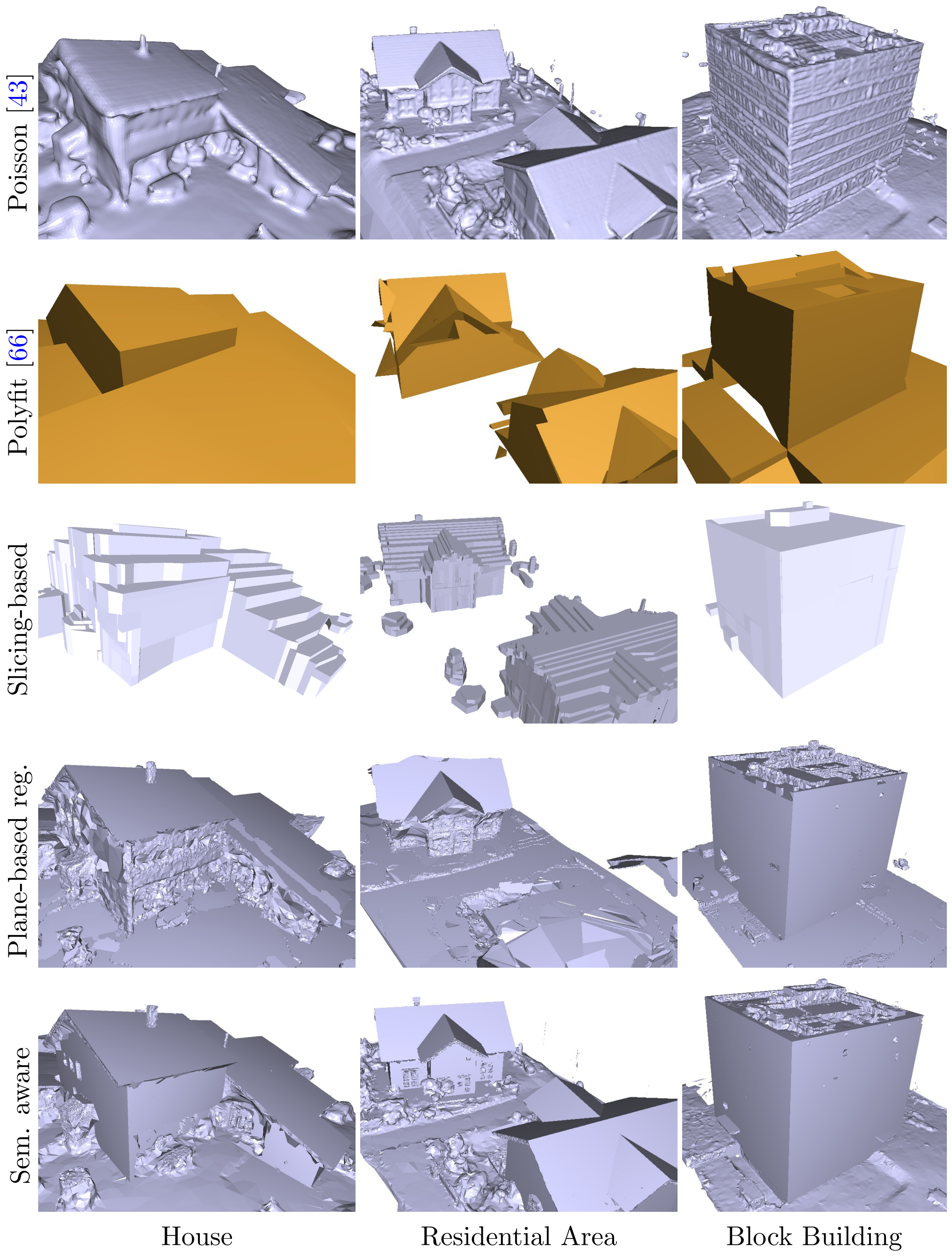


Figure 6.7: *Results and comparison with state-of-the-art methods.* From top to bottom: Poisson meshes [43] computed from the input point clouds, results of Polyfit [66], the proposed slicing-based method, the proposed plane-based regularization method and the proposed semantically aware method. The Poisson mesh looks visually appealing, but often produces rounded edges and spurious surfaces like bubbles at missing data. Polyfit is not able to reconstruct all buildings sufficiently well. As it depends on planes detected from point clouds, undetected surfaces were just not included in the possible solution set and spurious planes lead to erroneous reconstructions. The slicing-based method models the *Block Building* very well. However, at the other two datasets the sloped roof can only be approximated by stairway-like structures. The plane-based regularization method regularizes parts of the scene well with its included plane prior. However, at some parts of the scene not all planes were detected and, hence, planar surfaces remain noisy. Further, due to an unpredictable visibility-based energy term it is difficult to set correct weights for smoothness terms. Hence, some parts of the scene can be smoothed out very quickly. The semantically aware method creates 3D models with planar surfaces at façades/roofs while still keeping the building details like chimneys and reconstructs the surroundings with a smooth surface.

observed that the error metrics for the semantically aware method are the best for both the *House* and *Residential Area* data set. Apparently, the plane prior incorporated within the reconstruction process handles noise and clutter better than, for example, a smooth Poisson surface. Similarly, it can be seen that the proposed semantically aware method has visually low errors on both datasets.

For the methods we compared against, we used the following parameter settings: For Poisson surface reconstruction, which was computed using Meshlab [92], we set octree depth to 9. For Polyfit we used default parameters. We also tried to vary the parameters, but the results did not improve significantly.

6.4.1 Runtimes

In the following, we compare the runtimes of the methods used in the evaluation. For Poisson, Polyfit and the slicing-based method, the computations were performed on an Intel Core i7-4820k @ 3.7 GHz containing 4 CPU cores (8 threads) equipped with 48 GB RAM. The semantically aware and the plane-based regularization method were executed on a server with 2x Intel Xeon E5-2680 v2 running at 2.8 GHz with 10 CPU cores (20 threads) each and 264 GB of RAM. Within our methods, some highly demanding parts in terms of computation time (e.g., the visibility casting) make use of multithreading.

All the below mentioned runtimes are measured assuming precomputed input data (i.e., dense point cloud, 3D line reconstruction, 2D semantic labels, dense depth maps). However, one exception is the slicing-based approach, which runs the 2D line segment detection on demand.

In Table 6.2 the runtimes are depicted. It can be observed, that Poisson meshing has the lowest runtime. Polyfit has comparable high runtimes on two datasets. Especially at the *Residential Area* dataset, it ran longer than a week to produce some results. Further, the plane-based regularization method was significantly slower than the semantically aware method. This is due to the point-based plane detection in the plane-based regularization method, which first of all is slower in detecting planes, but also detects more and more spurious planes. This leads to more cell cutting and more cells for which the energies need to be computed and minimized. However, for both sub-methods most of the computation time was needed for visibility-based energy computation and cell cutting.

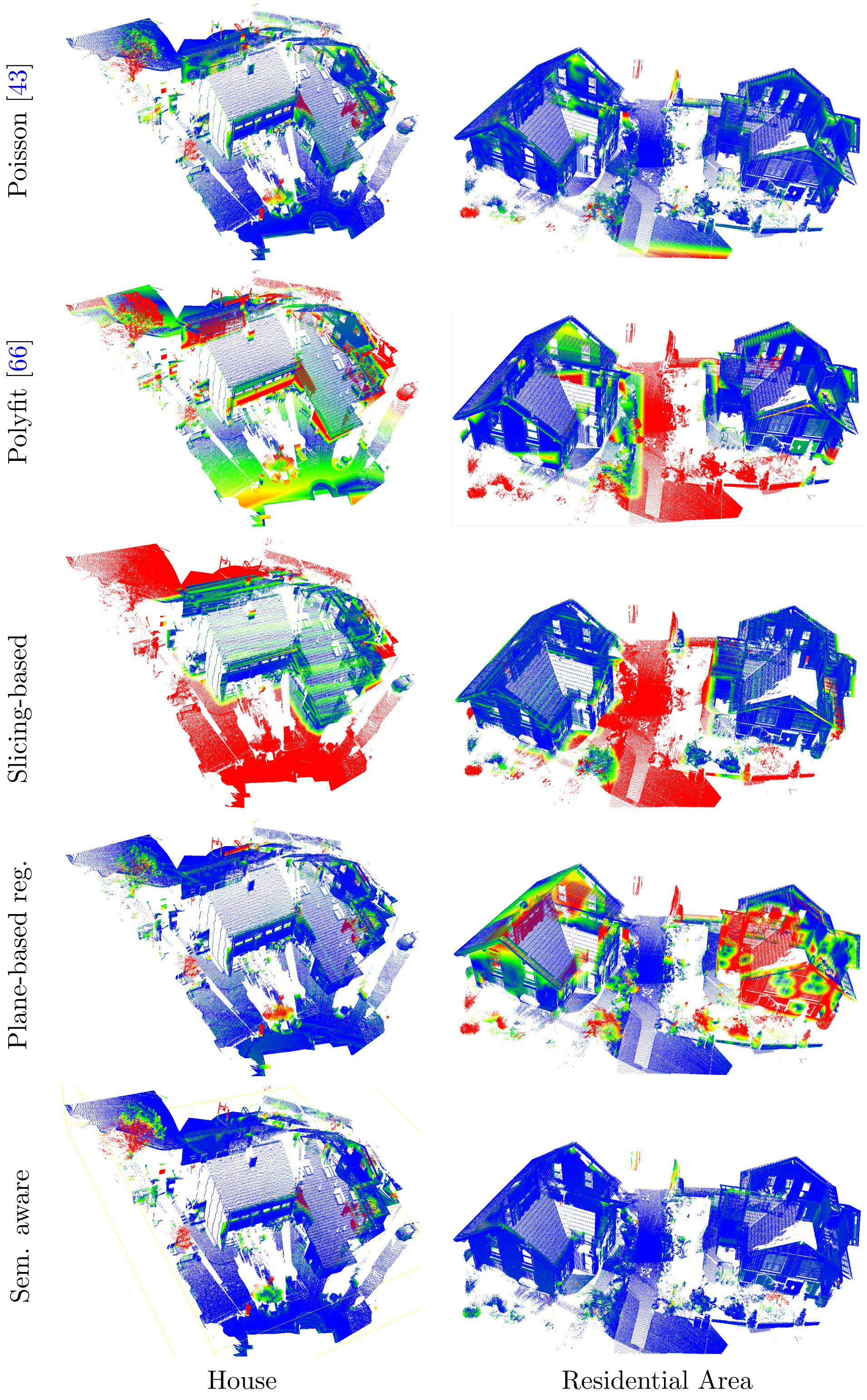


Figure 6.8: *Ground truth point errors* measured as the minimum euclidean distance from ground truth points to the 3D surface mesh of the results. Blue means low error, red means high error. The maximum error is defined as $1m$. For the *House* dataset (left column), it can be observed that the plane-based regularization method has a slightly lower error below the roof and at vegetation compared to Poisson. Polyfit has a much higher error, as it does not succeed in reconstructing the building and the surrounding with an adequate accuracy. For the *Residential Area* dataset (right column), the plane-based regularization method has the highest error, as it smooths out one building nearly completely and also has big errors at the second building. Also Poisson has significantly higher errors compared to the semantically aware approach, especially at walls and roofs. For both datasets, the slicing-based method does not model the surroundings of buildings and can only approximate sloped roofs. Hence, it has high errors.

	Bl. Building	House	Res. Area
Poisson [43]	0.4	0.4	0.6
Polyfit [66]	647.3	1.8	16309.9
Slicing-based	50.4	80.4	924.8
Plane-based regularization	337.1	63.6	903.5
Semantically aware	59.7	13.3	153

Table 6.2: *Runtimes of the evaluated methods in minutes.* Poisson meshing is the fastest on all datasets, while Polyfit is very slow on two datasets. The proposed approaches often have comparable runtimes.

6.5 Method-Specific Detailed Evaluations

In this section, we show individual, detailed evaluations of the three methods used in the previous section. We give detailed parameter studies, evaluate the effects of the individual contributions and show additional results and comparisons.

6.5.1 Slicing-Based Method

The slicing-based method, as presented in Chapter 4, creates geometrically regularized 3D models from buildings which finally consist of smooth and clean surfaces instead of a noisy mesh, as in the input data. It can handle input meshes with different quality and input density, where especially on meshed point clouds including some errors (e.g., a point cloud created from *SfM*) our introduced line-based smoothness term improves the quality of the resulting 3D model compared to using only the mesh-based smoothness term. Finally, we deliver optionally textured output models, which can be variably regularized and contain smooth surfaces where meshed point clouds, in contrast, contain a lot of noise and clutter.

In the following, we will show additional results of the proposed slicing-based method and the effects of different parameter settings. Further, we will demonstrate the benefits of the introduced line-based smoothness term. In addition to a meshed dense point cloud, we also used the resulting sparse point cloud from *SfM* meshed with our implementation of [48] as input for some of the following experiments.

6.5.1.1 Parameter Selection

We changed three parameters during our experiments. First, we set the Mean Shift parameter d (as described in Section 4.3) depending on the desired level of detail in the vertical direction. If only few horizontal elements should be represented by the final model, one can set d lower to produce less slices. If many small structuring elements should be represented in the final model, one should set d higher to produce more slices and cover all horizontal elements. Usually, we set this parameter between 40 and 50, but sometimes it is also beneficial to set it higher (e.g., as in the experiments in Section 6.4, where sloped structures exist in the scene).

The two smoothness parameters λ_{mesh} and λ_{lines} define the amount of regularization. The selection of a good value for λ_{mesh} heavily depends on the quality of the input mesh: If we use a nearly error-free input mesh just including a little bit of noise, it is recommended to set λ_{mesh} higher in comparison to λ_{lines}. Contrary, if the input mesh contains big errors we suggest to set λ_{mesh} lower, as otherwise the regularized output may be influenced too much from the erroneous mesh. λ_{lines} should be set according to the amount of errors which should be corrected with this line-based smoothness term. For an erroneous mesh, one should set λ_{lines} to a higher value, for a mesh containing just few errors, one should set λ_{lines} to a lower value. We usually set λ_{mesh} between 0.2 and 0.8 and λ_{lines} between 0.1 and 0.5.

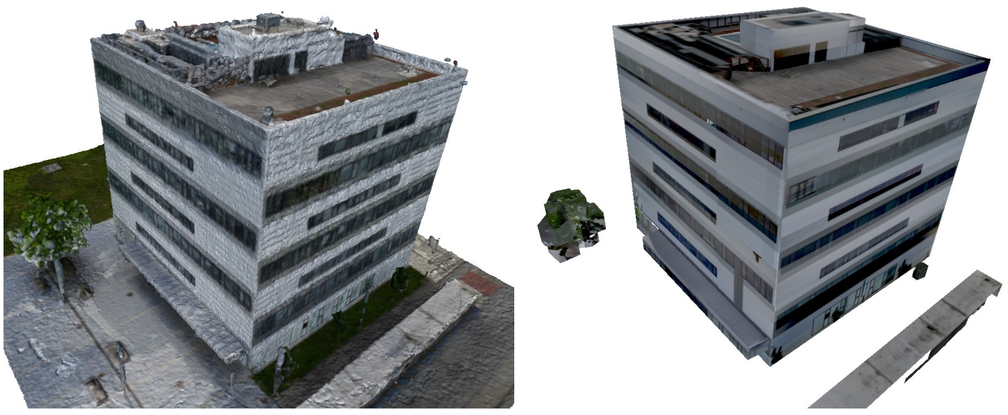


Figure 6.9: *Meshed dense 3D reconstruction and resulting regularized model of the block building. Left:* A 3D building reconstruction, for which a dense point cloud was reconstructed with Sure [79], which was finally meshed using Poisson surface reconstruction [43] (visualized with vertex coloring). Such a mesh can be used as input for our approach. *Right:* Resulting regularized model from our pipeline with line+mesh smoothing, textured with [93]. We deliver clean and smooth surfaces where the meshed dense point cloud contains a lot of noise. The parameters were set to: $\lambda_{mesh} = 0.4$, $\lambda_{lines} = 0.1$ and $d = 50$.

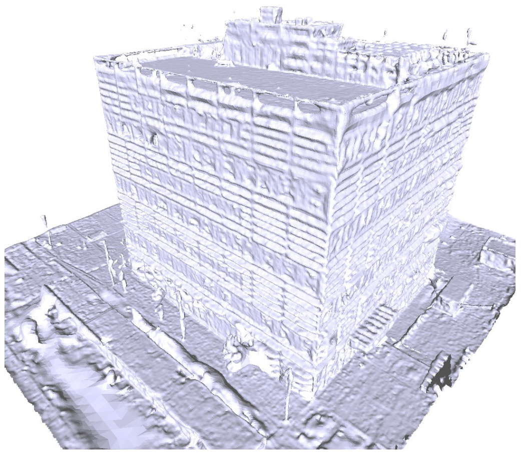
Further, we compared two variants of our approach, which uses different smoothness terms in Equation 4.6: One just uses the mesh-based smoothness term, which is equal to setting $\lambda_{lines} = 0$. This is identical to the method presented in [35] and is depicted as *mesh-only smoothing*. The proposed method, which uses both smoothness terms in Equation 4.6, was originally presented in [34] and is depicted as *line+mesh smoothing*.

When using mesh-only smoothing, we observed that setting $\lambda_{mesh} = 3.5$ is a good compromise for all of our test data: With this setting, important parts of the buildings were still not smoothed out, while the overall result is regularized well.

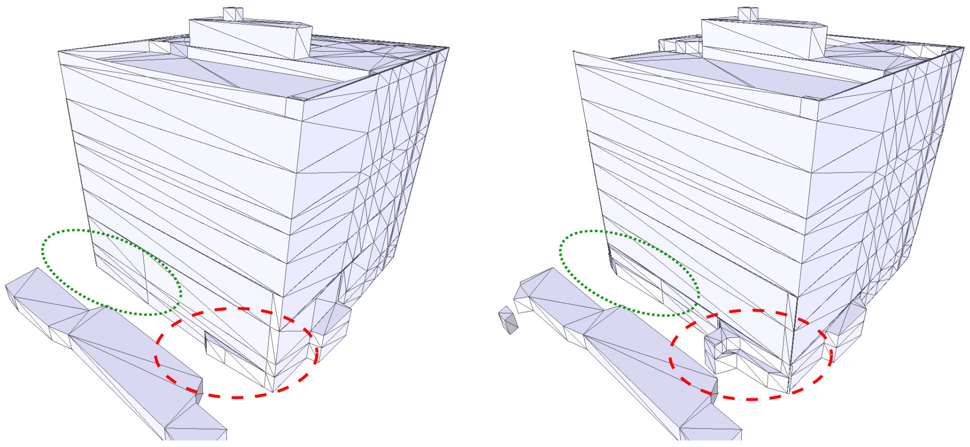
6.5.1.2 Block Building

This simple building, which has already been introduced in Section 6.1, consists out of one block with some additional installations on the roof and some small roofs on the side. Due to this simple, Manhattan-like structure, it is well suited for the proposed approach. For this dataset, we used a meshed sparse *SfM* point cloud in addition to PMVS2 [22] and Sure [79] data as input.

As Sure delivers the most accurate point cloud, also the results from our pipeline using Sure data as input are the most convincing ones. You can see the textured result and the corresponding input data in Figure 6.9. Using dense Sure input data, only a little smoothing was necessary to get smooth surfaces and the line smoothness parameter does not significantly improve the result. Therefore we set especially the line smoothness parameter comparably low, as setting the mesh smoothness term already results in a well



PMVS2 [22] + Poisson [43]



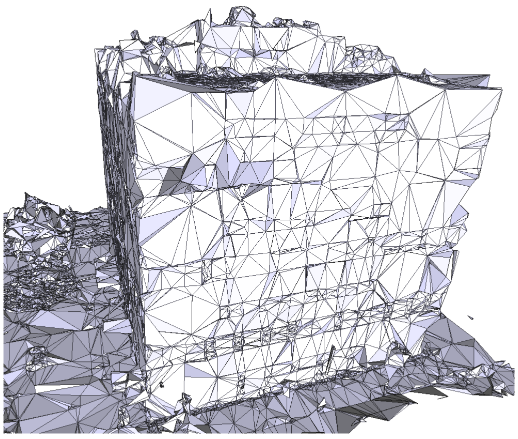
Line+mesh smoothing
$\lambda_{mesh} = 0.5$, $\lambda_{lines} = 0.5$, $d = 50$

Mesh-only smoothing
$\lambda_{mesh} = 1.0$, $d = 50$

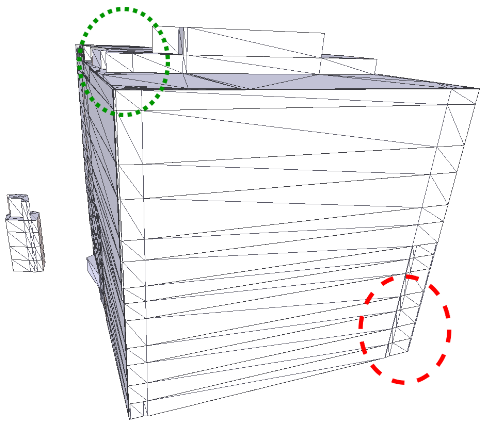
Figure 6.10: *Results for the Block Building with PMVS2. Top:* The input data created with PMVS2 and Poisson surface reconstruction. As one can see, this reconstruction contains more noise and clutter and is incomplete compared to the Sure reconstruction in Figure 6.9. *Bottom left:* The result of line+mesh smoothing, which contains less artifacts (green dotted and red dashed circle) compared to the mesh-only smoothing (*bottom right*). The parameters for both were chosen to create smooth and clean façades while simultaneously keeping most of the structural details.

regularized model. The Mean Shift parameter d was set so that all important horizontal structures were detected (e.g., small roof above entrance, installations on roof).

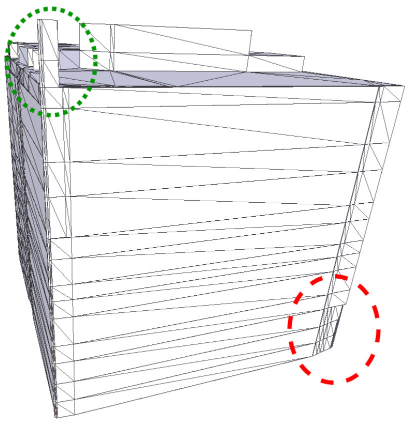
The results from our pipeline using meshed PMVS2 data as input can be seen in Figure 6.10. Line+mesh smoothing (bottom left) has a smoother surface compared to the mesh-only smoothing approach (bottom right) especially in the area in the red dashed and green dotted circle. In the red dashed circle, a tree was fused with the building in the input model. Therefore, using mesh-only smoothing, the optimization found the optimal solution



SfM + Labatut et al. [48] meshing



Line+mesh smoothing
$\lambda_{mesh} = 0.2$, $\lambda_{lines} = 0.5$, $d = 50$



Mesh-only smoothing
$\lambda_{mesh} = 3.5$, $d = 50$

Figure 6.11: *Results for the Block Building with pure SfM. Top:* The input data, which is the *SfM* point cloud meshed by Labatut et al. [48]. This reconstruction is very noisy and also contains quite big wrong parts. *Bottom left:* Nevertheless, our approach with line+mesh smoothing estimates still smooth surfaces. Though, some details are missing compared to Figure 6.10. *Bottom right:* Even though the smoothing is already quite strong, which leads to smoothing artifacts (e.g., in the green dotted circle), the mesh-only smoothing still follows the erroneous mesh strictly (e.g., in the red dashed circle).

by including the tree within the output model. Instead, when using line+mesh smoothing, a line could still be detected in the images at the boundary of the building and therefore the optimization pulled the result towards the correct solution. In the green dotted circle, the mesh-only smoothing also follows the noisy mesh surface, where line+mesh smoothing delivers a cleaner surface. We selected the parameters to keep important parts of the building (e.g., when setting λ_{mesh} for mesh-only smoothing higher, the small low building in front of the Block Building would vanish due to over-smoothing).

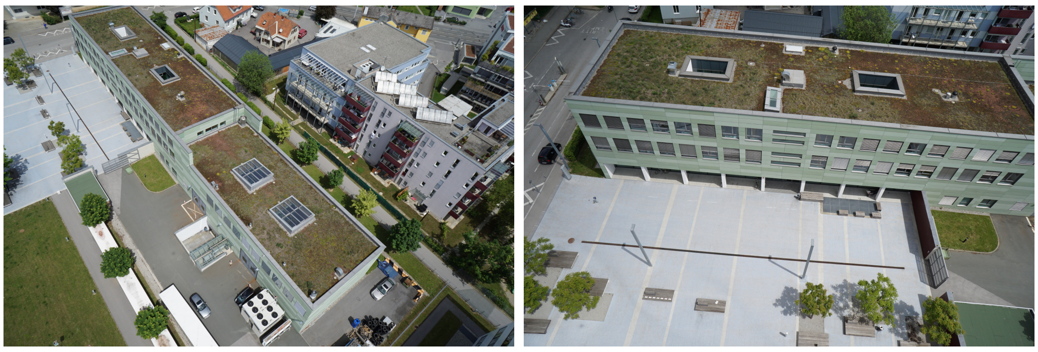


Figure 6.12: *Input images of the Long Building.* The building mainly consists out of 2 block parts with different height.

In order to show that our approach can also handle very noisy meshes created from sparse point clouds, we meshed the resulting point cloud from *SfM* with Labatut et al. [48] and used this data as input for our pipeline. In Figure 6.11, the input mesh and results from our approach are shown. Our approach with line+mesh smoothing (bottom left) delivers smooth surfaces and still contains some small details. In contrast, the mesh-only smoothing (bottom right) still follows the erroneous mesh too strictly at some parts (e.g., in the red dashed circle), even though the smoothing is already at its limits, which leads to some smoothing artifacts (e.g., in the green dotted circle). If we increase the smoothing (i.e. increase the value for λ_{mesh}), more and more artifacts arise while the error in the red dashed circle will stay.

6.5.1.3 Long Building

This building consists of two parts with two different heights. Input images can be seen in Figure 6.12. For this dataset, we just used meshed PMVS2 [22] data as input.

In Figure 6.13, results from our approach with two smoothing variants are printed. As not enough images were captured for the seen side of the building, there are big errors in the reconstruction (top left). The mesh-only smoothing approach (bottom left) cannot correct these errors properly (e.g., at the part in the red dashed circle). When enforcing more smoothing (i.e.using a higher value for λ_{mesh}), the result gets even worse as this approach follows the erroneous mesh. Contrary, the line+mesh smoothing approach (right) approximates a planar surface (which is the geometric structure of the façade) better, as it uses additional information from the original input images. In the top right of the figure, the same result as in bottom right is depicted visible from another viewing direction and textured afterwards with [93]. Due to a wrong geometry in the lower left part, several faces could not be textured and are therefore white in this reconstruction. This happens due to the noisy PMVS2 data, with which we could not estimate a correct geometry for this part. However the other parts look good which indicates that the estimated geometry

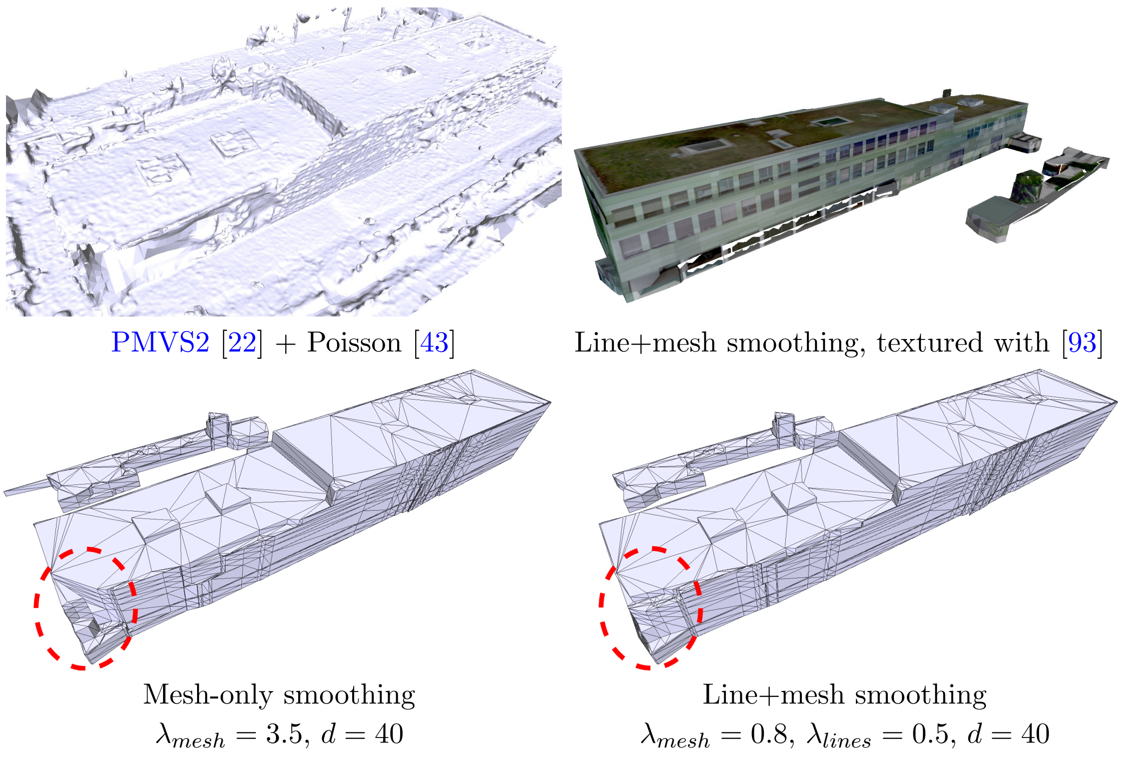

PMVS2 [22] + Poisson [43]
Line+mesh smoothing, textured with [93]
Mesh-only smoothing
$\lambda_{mesh} = 3.5$, $d = 40$
Line+mesh smoothing
$\lambda_{mesh} = 0.8$, $\lambda_{lines} = 0.5$, $d = 40$

Figure 6.13: *Results for the Long Building.* As one can see, the input data (*top left*) has errors in the mesh. This occurred due to insufficient image data of this part of the building. Because of this errors, the mesh-only smoothing (*bottom left*) cannot regularize this part well (red dashed circle), where line+mesh smoothing (*bottom right*) makes a better approximation of the real façade. At *top right*, the same result from line+mesh smoothing from another viewing direction and textured with [93] is depicted.

is correct.

In Figure 6.14, another part of the building of the results presented in Figure 6.13 is visible. The mesh-only smoothing approach (right) has irregularities on the surface (e.g., in the red dashed circle), as it follows the surface of the noisy mesh. Contrary, the line+mesh smoothing (left) delivers a much smoother, planar surface similar to the original façade.

6.5.1.4 Non-Manhattan Building

This dataset contains a more complex building. In difference to the other buildings shown in our experiments, it does not have a Manhattan-like outline and contains sloped surfaces. In Figure 6.15, one can see example input images.

The regularized result and the dense input reconstruction computed with Sure [79] can be seen in Figure 6.16. Non-rectangular outlines of the building are not a problem for the algorithm, as any outline gets approximated by a polygonal line. Also sloped structures

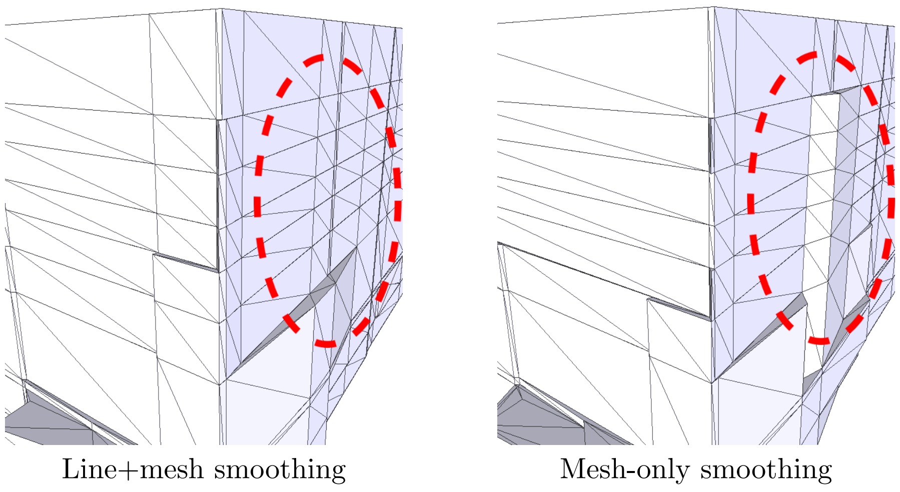


Figure 6.14: *Back side of the Long Building* of the same results as illustrated in Figure 6.13. As one can see, line+mesh smoothing (*left*) delivers a smoother façade compared to mesh-only smoothing (*right*).

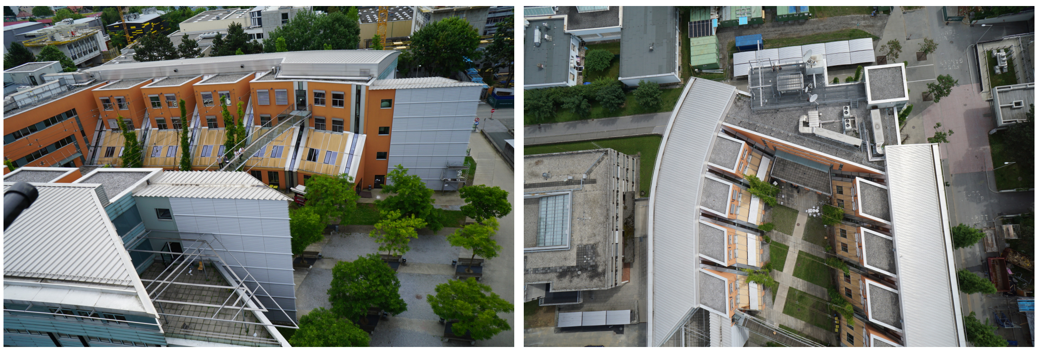


Figure 6.15: *Input images of the Non-Manhattan Building.* This building contains non-Manhattan-like façades and sloped surfaces.

are approximated by stairway-like structures. To get a more detailed approximation one could lower the the Mean Shift bandwidth (i.e. higher the value for d) in order to get more slices and hence a finer stairway approximation. In general, one has to mention that this is a difficult building, where also vegetation is very near or fused with the building, which makes it difficult at some parts to estimate correct building boundaries without additional semantic information. Therefore, some details are missing in the regularized model.

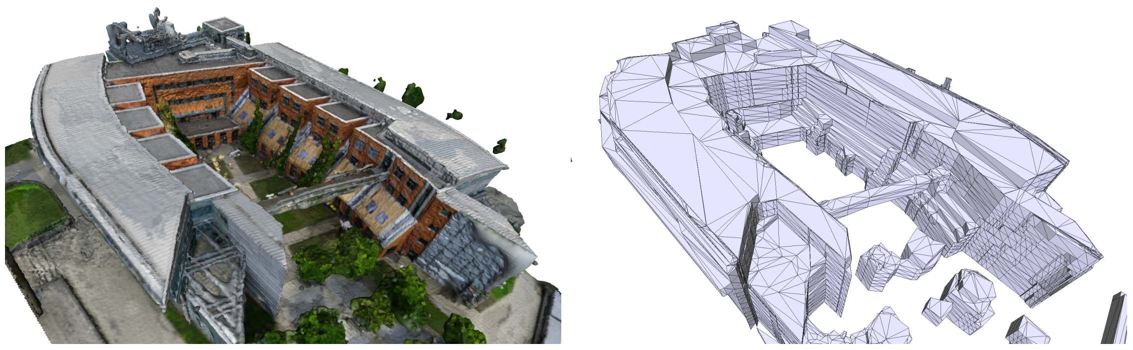


Figure 6.16: *Result from the Non-Manhattan Building Left:* Input mesh created with Sure [79] and Poisson meshing [43] (vertex coloring). *Right:* Resulting regularized model with line+mesh smoothing. Sloped surfaces are approximated with stairway-like structures, non-Manhattan-like outline is approximated by a polygonal line. The parameters were set to: $\lambda_{mesh} = 0.4$, $\lambda_{lines} = 0.2$ and $d = 50$.

6.5.1.5 Runtime Discussion

As for line+mesh smoothing, compared to mesh-only smoothing, additionally line segments needs to be detected and multiple iterations of optimization needs to be done, it has a higher runtime. Though, the runtime highly depends on the structure of the 3D model (e.g., number of slices, ratio free/occupied space voxels, number of images used for line detection) and therefore it can vary significantly between different models (as has already been shown in Section 6.4.1). The runtime at the Block Building model with PMVS2 input data and line+mesh smoothing was 50.4 minutes (included line detection took 4.2 minutes), whereas the runtime with mesh-only smoothing was 34.9 minutes.

6.5.2 Plane-Based Regularization Method

In this section, we show additional results and parameter studies of the plane-based regularization method, which is a sub-method of the tetrahedra-based method described in Chapter 5. This sub-method has already been defined in Section 6.3 and uses point-based plane detection (as described in Section 5.2.1), the unnormalized visibility-based energy formulation as described in Section 5.6.1, pre-computed visibility information (as described in Section 5.6.3.1) and no semantic information. This is the same setting as used in [37].

We additionally evaluated this plane-based regularization method on multiple urban outdoor and indoor datasets and compared it to further state-of-the-art methods. We show that our method yields comparable results while being more flexible than others and that the level of detail of the reconstruction can be adjusted easily with the parameter α_{LoD}.

For all the experiments, we selected $\alpha_{\mathrm{Man}} = 250K$, as with this setting the whole reconstruction just consists of planar surfaces in combination with a low value for α_{LoD} (as

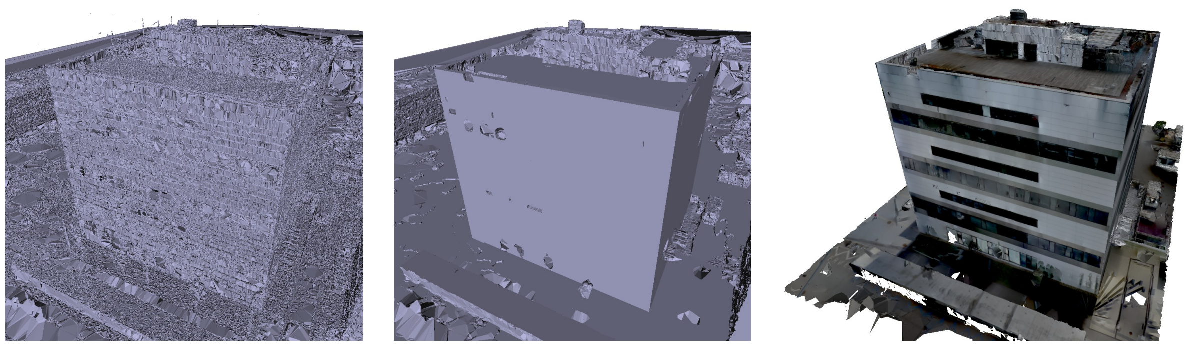

Labatut et al. [48]
Plane-based regularization, $\alpha_{\text{LoD}} = 375K$
Plane-based regularization, textured (using [93])

Figure 6.17: Comparison of a generic 3D reconstruction approach (Labatut et al. [48], *left*) to our plane-based regularization method with plane fitting and level-of-detail adjustment (*middle* and *right* (textured)). Our method produces sharp edges and planar surfaces while simultaneously keeping details like the structures on the roof.

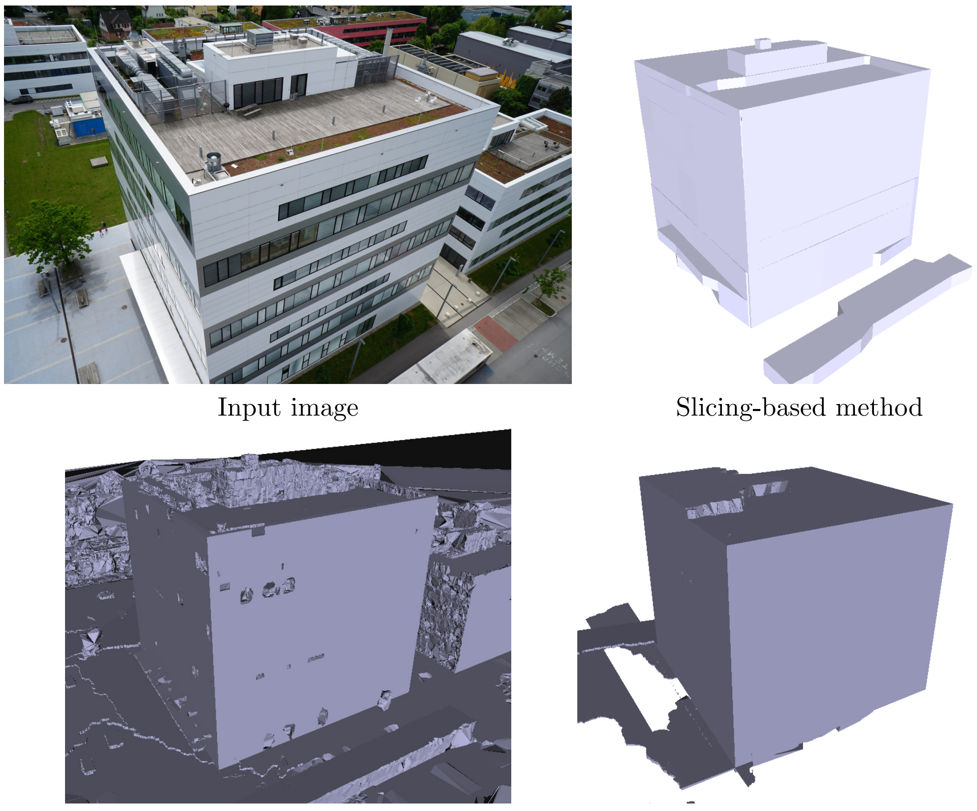
depicted, for example, in Figure 6.18). When comparing against the slicing-based approach from Chapter 4, we used the same parameter settings as in the main evaluation (defined in Section 6.1). When comparing against Labatut et al [48], we were only using the visibility-based energy described in [48] in combination with a constant penalty as described in [63]. This method can be seen as the base method of the plane-based regularization method, as it uses the same visibility-based energy but does not use a plane prior. Further, we compared against Li et al. [53] and Monszpart et al. [61], which both detect planes having a Manhattan-like orientation and aim to model the scene using these planes.

In sum, our method can be seen as a hybrid approach where the user can define the smoothness and level-of-detail continuously between no regularization such as Labatut et al. [48] and very strong regularization and simplification such as Li et al. [53].

6.5.2.1 Block Building

Again we are using the *Block Building* dataset, which consists of a block shaped building with some additional details on the roof. Therefore, it is well suited to show different reconstruction results when adjusting α_{LoD}. In comparison to the main experiments, we used a denser point cloud also created by PMVS2 [22] for this experiments containing approx. 5.2M points. Figure 6.17 shows a (textured) result and a result of Labatut et al. [48]. Our method produces well regularized results and generates models with sharp edges and planar surfaces while still containing details which were not supported by any plane.

In Figure 6.18, we compare results with varying α_{LoD} to the result of the slicing-based method (Chapter 4). For high α_{LoD} (bottom left) more details are kept in the reconstruction, while for low α_{LoD} (bottom right) mostly only plane supported faces are kept. Compared to the slicing-based method described in Chapter 4, the simplification of planar



Input image

Slicing-based method

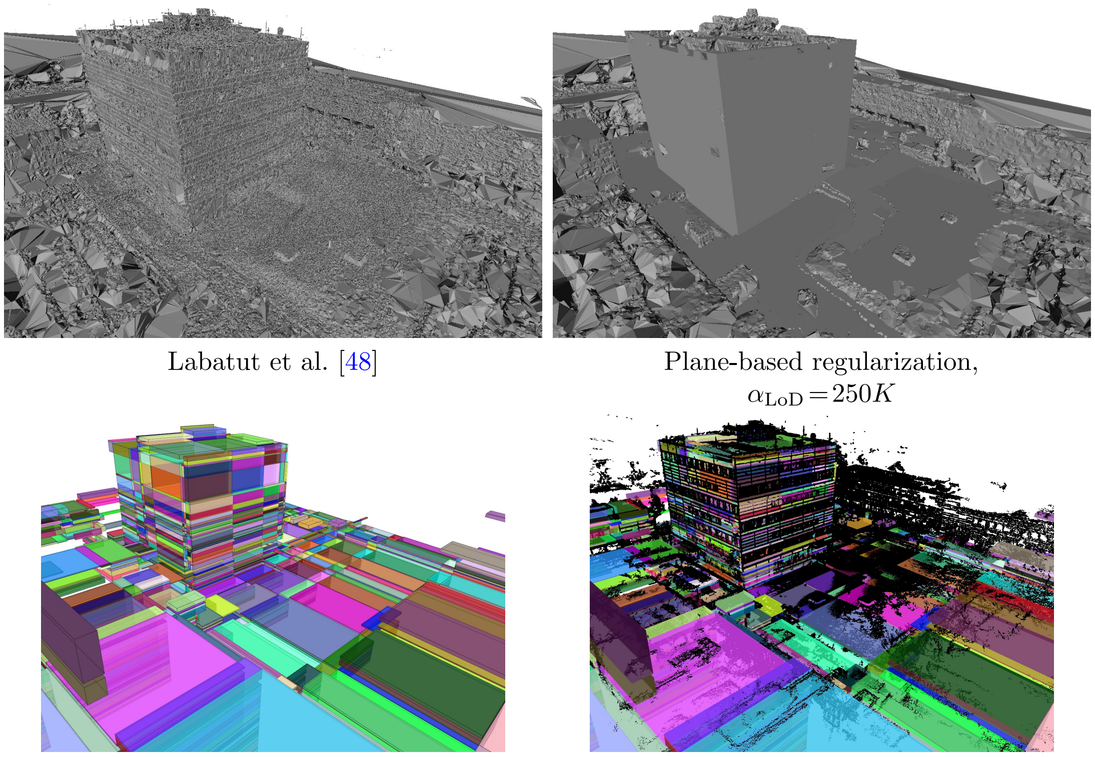
Plane-based regularization, $\alpha_{\text{LoD}} = 375K$

Plane-based regularization, $\alpha_{\text{LoD}} = 25$

Figure 6.18: Results with varying level of detail on the *Block Building* dataset. For high values α_{LoD}, many details of the reconstruction from Labatut et al. [48] which are not supported by a plane are still included in the reconstruction. Contrarily, when setting α_{LoD} low, mostly only plane-supported surfaces are kept. Compared to the proposed plane-based regularization method, the slicing-based method (on the same input) approximates planar surfaces well but misses details on the roof.

surfaces is similar, but structures on the roof are represented with more details by this approach. It is worth mentioning that the slicing-based method cannot deal with slanted roof sections and will also simplify non-building geometry like vegetation or irregular ground structure.

In Figure 6.19, we further qualitatively compare the proposed method with the method of Li et al. [53] and Labatut et al [48]. It can be observed that Li et al. [53] heavily simplifies the geometry while our approach delivers planar surfaces at planar parts and detailed reconstructions at non-planar parts.



Labatut et al. [48]

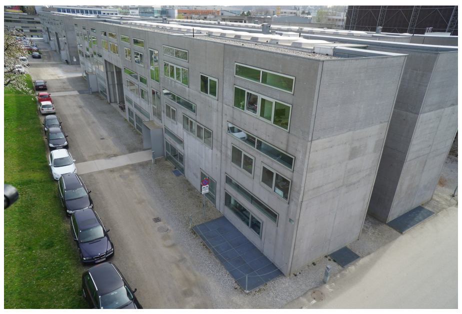
Plane-based regularization, $\alpha_{\text{LoD}} = 250K$

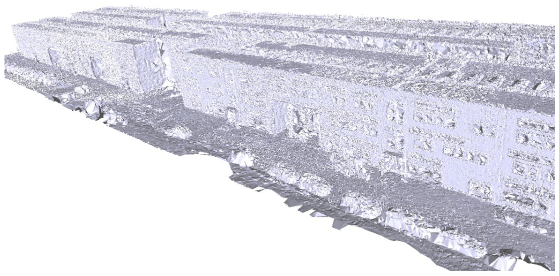
Li et al. [53] without / with point cloud

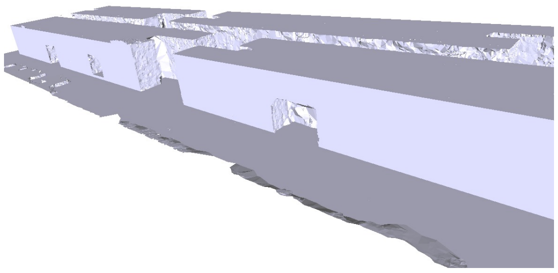
Figure 6.19: *Comparison of results on the Block Building dataset captured from the back side of the building compared to Figure 6.18 and Figure 6.17.* In comparison to Labatut et al. [48] which does not use any shape priors, the proposed plane-based regularization approach simplifies the geometry of surfaces along planar primitives, while keeping any desired level of detail. The strict Manhattan assumption of Li et al. [53] heavily simplifies the scene and leaves little or no control to adjust the level of detail.

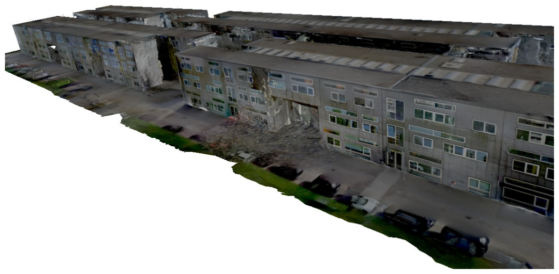
6.5.2.2 Building Complex

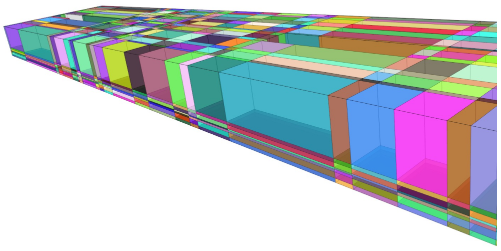
Next, we evaluated on the dataset *Building Complex*. This dataset consists of four similarly shaped buildings mainly composed of planar surfaces. The input point cloud was created by PMVS2 [22] and consists of approx. 3.4M points. In Figure 6.20, an input image and results of our method are illustrated in comparison to others. As one can see, the planar surfaces on the building surface and on the ground floor are reconstructed well with the proposed method. Previously detected planar structures are used to denoise and simplify the scene. The effects are clearly visible in comparison to Labatut et al. [48] which does not aim to simplify surface structures in urban environments. In contrast, Li et al. [53] makes strong (Manhattan) assumptions about the scene and can only deal with orthogonal planar structures.


Input image


Labatut et al. [48]


Plane-based regularization, $\alpha_{\mathrm{LoD}} = 250K$


Plane-based regularization, with texture [93]


Li et al. [53]

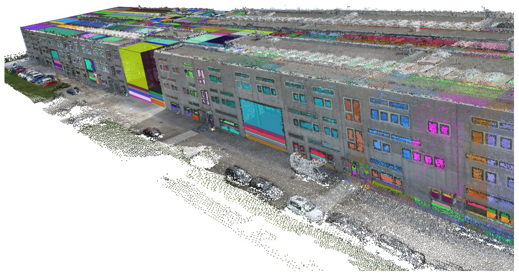

Li et al. [53] + point cloud

Figure 6.20: Comparison of results on the *Building Complex* dataset. While the result without shape priors (Labatut et al. [48]) contains a lot of noise, the result of the proposed plane-based regularization method produces planar surfaces and sharp edges. Due to a lower point density at surfaces in between the buildings, no planes were detected in these parts and they hence remain noisy. The results of Li et al. [53] over-simplify the geometry and several empty parts between the buildings are not correctly identified, because the method is very sensitive to noise.

6.5.2.3 House

For this experiment, we are using the already introduced *House* dataset, which is a family house containing a sloped roof with chimneys. In Figure 6.21, one can observe the varying level of detail, which is especially well observable at the chimneys. Additionally, an example input image and a result from Labatut et al. [48] can be seen in Figure 6.21. In Figure 6.22, further views of this dataset are depicted and shown in comparison to state-of-the-art methods [53, 61].

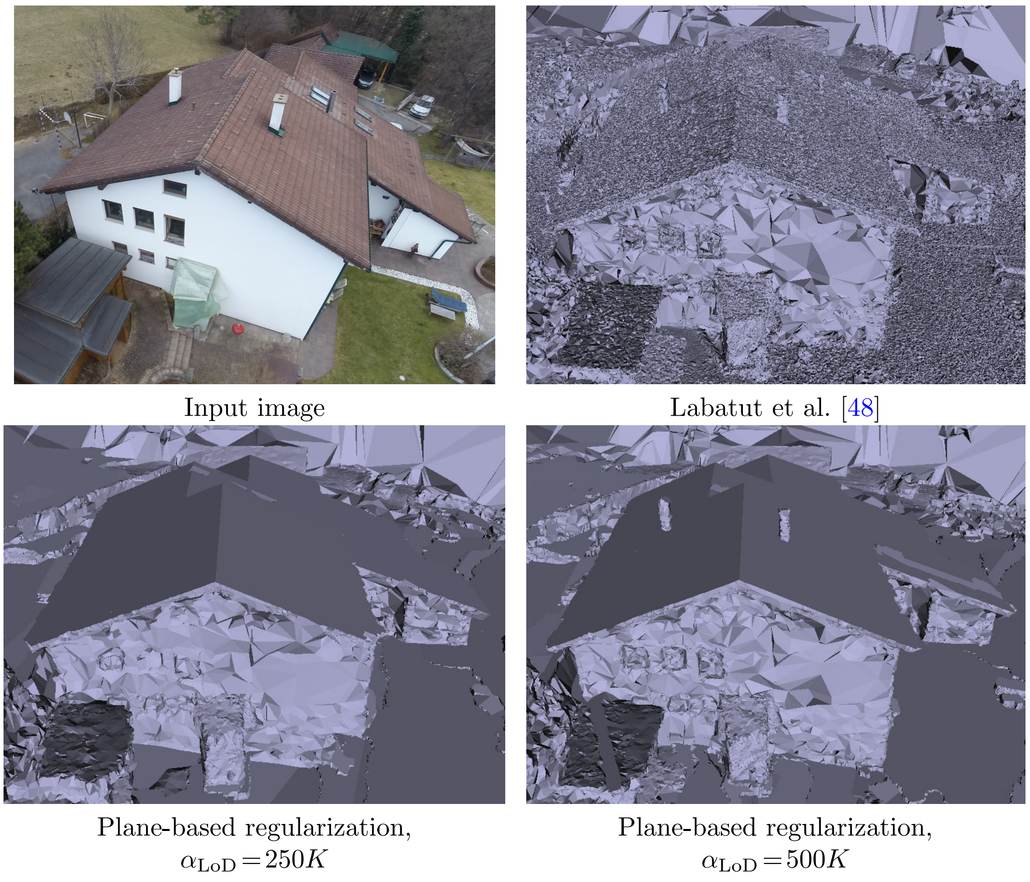

Input image
Labatut et al. [48]
Plane-based regularization, $\alpha_{\text{LoD}} = 250K$
Plane-based regularization, $\alpha_{\text{LoD}} = 500K$

Figure 6.21: The plane-based regularization method with different level of detail settings in comparison to Labatut et al. [48] on the *House* dataset. For a low level of detail (*bottom left*), the chimneys are not reconstructed while for high level of detail (*bottom right*) they are included in the reconstruction.

6.5.2.4 Indoor

We also evaluated on an *Indoor* dataset, which is a reconstruction of a hall with planar walls, some tables and chairs. Again, this is a semi-dense reconstruction created with PMVS2 [22] and consists of approx. 4.6M points. Results and a comparison with other methods can be found in Figure 6.23. While Li et al. [53] over-simplifies the geometry by only fitting boxes, Monszpart et al. [61] did not produce any meaningful results on our datasets.

6.5.2.5 Entry-P10

Finally, we compared our method with the method proposed by Lafarge et al. [52] using the *Entry-P10* dataset [86]. This dataset consists of 10 images captured at ground level. We used the provided camera poses and computed a semi-dense point cloud using PMVS2 [22] consisting of approx. 0.4M points. As can be seen in the results in Figure 6.24, the

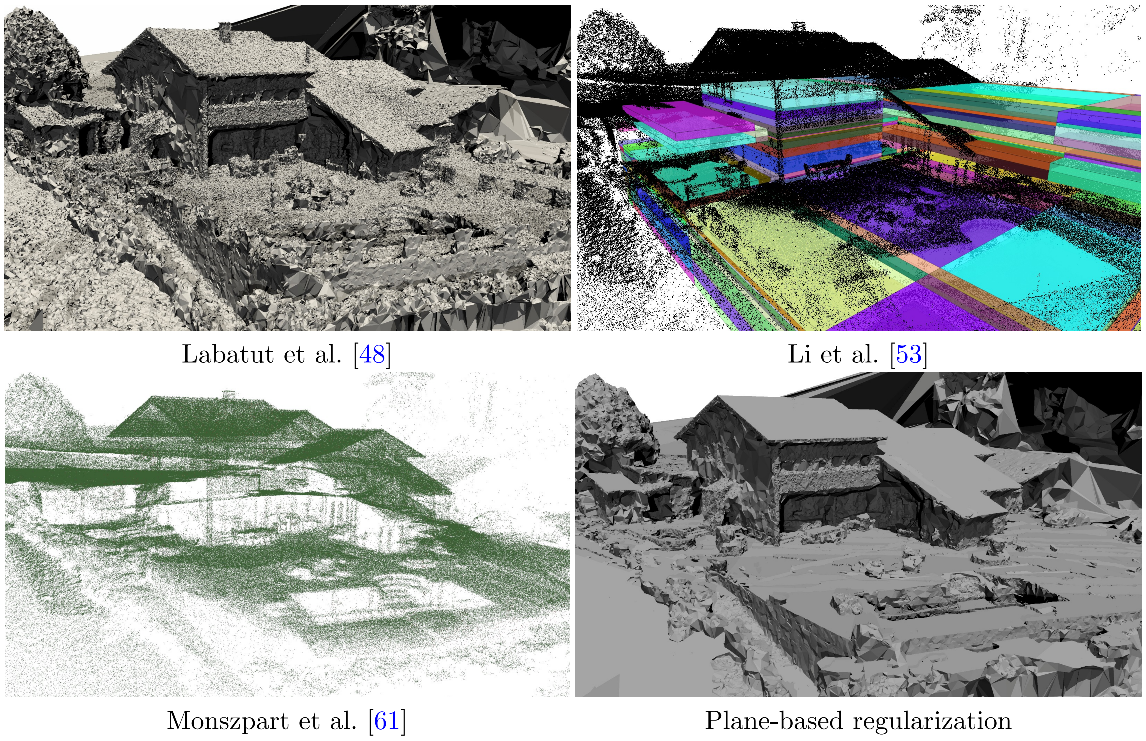
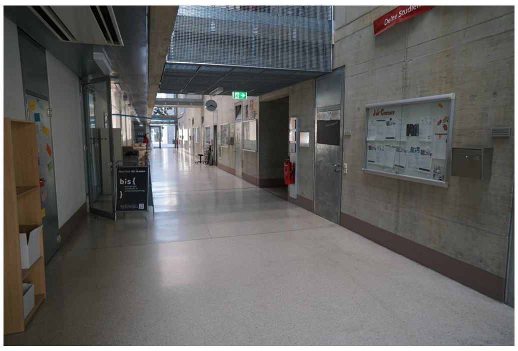


Figure 6.22: Qualitative comparison to other methods on the *House* dataset. While Labatut et al. [48] provides a detailed reconstruction but does not simplify any geometry, Li et al. [53] is not able to represent all scene parts with simple boxes (for example, sloped roofs cannot be modeled). The results of Monszpart et al. [61] did not produce a meaningful result on this dataset. The proposed plane-based regularization method provides a hybrid reconstruction between primitive-fitted planar parts and generic reconstruction for the free-form parts.

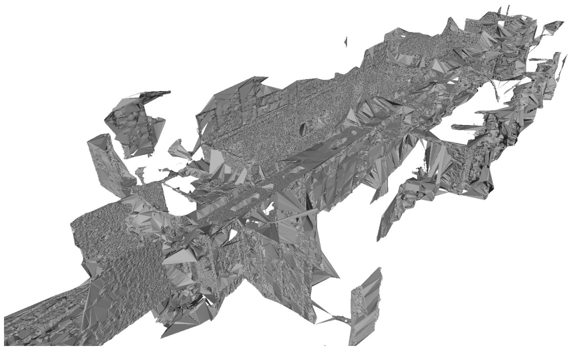
proposed plane-based regularization method smooths the planar surfaces very well while still keeping most of the important details of Labatut et al. [48]. Compared to Lafarge et al. [52], our method delivers a comparable regularization of the planar surfaces.

6.5.2.6 Runtime Breakdown

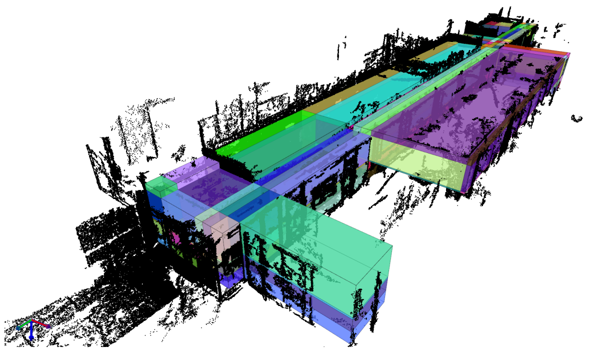
The runtime heavily depends on the amount of input points and cameras and varies from approx. 5 minutes for the *Entry-P10* dataset (0.4M points, 10 cameras) to approx. 9.5 hours for the *Block Building* dataset (5.2M points, 232 cameras). Though, as most of the processing time is needed for preprocessing steps, the final optimization can be rerun with different parameters within less than one minute. A breakdown of the runtime can be found in Figure 6.25. The runtimes for the other methods compared against in this section were as follows: Li et al. [53] was in the range of 10 seconds, and Monszpart et al. [61] more than 16 hours and the runtimes of the slicing-based approach was already mentioned in Section 6.4.1 and was approx. 50 minutes for the *Block Building* dataset.



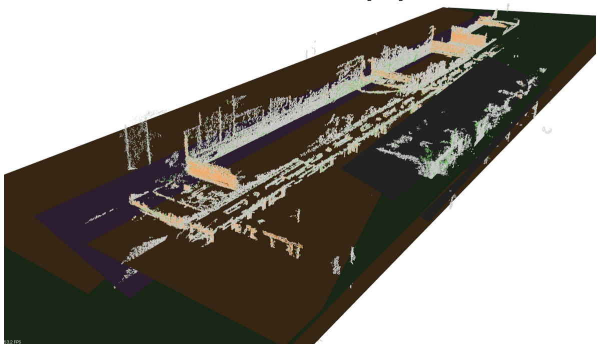
Input image



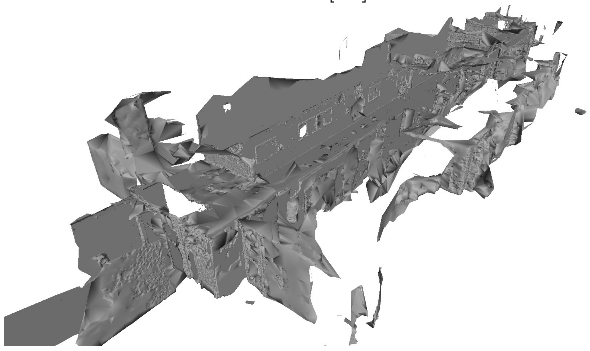
Labatut et al. [48]



Li et al. [53]



Monszpart et al. [61]



Plane-based regularization

Figure 6.23: Qualitative comparison to other methods on the *Indoor* dataset. While Labatut et al. [48] does not simplify any geometry, Li et al. [53] over-simplifies the scene with only very few boxes. The results of Monszpart et al. [61] were not useful on our datasets as detected planes did not well align with the geometry. Our plane-based regularization approach provides a hybrid reconstruction between primitive-fitted planar parts and generic reconstruction for the free-form parts.

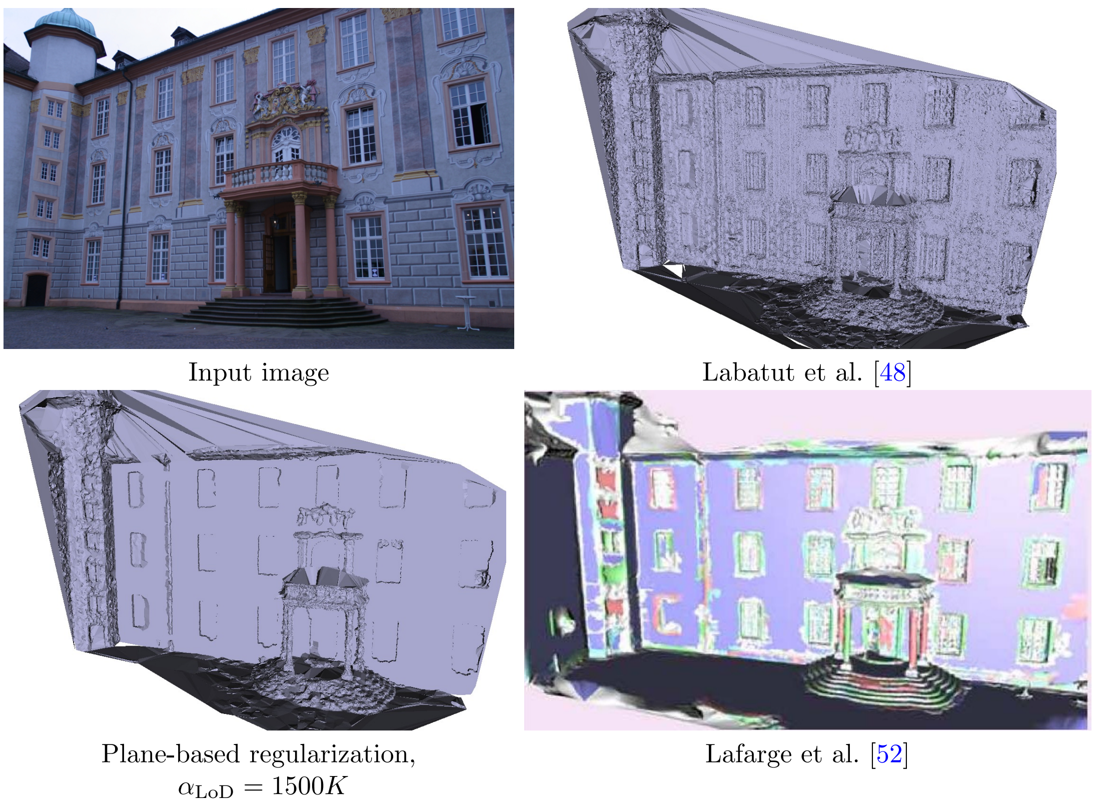


Figure 6.24: Results of the *Entry-P10* dataset [86]. While the result without shape priors (Labatut et al. [48]) contains a very noisy façade, the proposed approach with plane-based regularization reconstructs a completely planar surface and simultaneously keeps most of the significant details. The proposed approach also produces a comparable planar regularization of the façade to [52]. The image for Lafarge et al. is taken from [52].

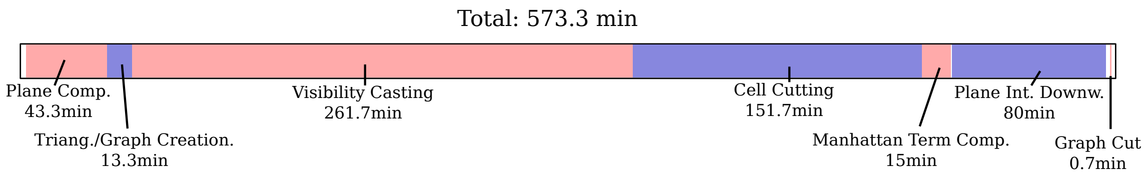


Figure 6.25: Runtime breakdown for an execution on the *Block Building* dataset, which has a total runtime of approx. 9.6 hours (573.3 min). Most of the processing time is needed for visibility casting and cell cutting. Important processing parts are depicted in red and blue, remaining small processing parts (e.g., data loading, LoD weight computation) are depicted in white.

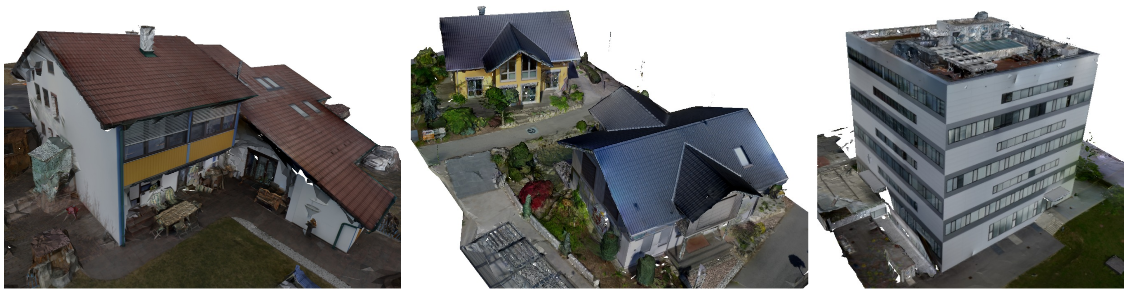


Figure 6.26: *Results of the proposed approach textured with [93].* One can observe that planar parts in the scene (façades, windows, roof) are represented by planar surfaces and building edges where two façades intersect each other are represented by straight lines. Note that holes in the model are due to missing visibility information during texturing.

6.5.3 Semantically Aware Regularization Method

In this section, we show additional results and present parameter studies and evaluations for individual contributions of the semantically aware method, which is a sub-method of our the method described in Chapter 5. This sub-method has already been defined in Section 6.3 and uses line-based plane detection (as described in Section 5.2.2), the normalized visibility-based energy formulation as described in Section 5.6.2, visibility information computed using depth maps as described in Section 5.6.3.2, and semantic information as described in Section 5.8. This setting was also used in [36].

In the following we evaluate the line-based plane detection algorithm, the normalized visibility-based energy and the effect of semantic priors and visualize in detail the semantic 3D data. Finally, we show the effect of mesh simplification as post-processing step.

In Figure 6.26 results from the proposed approach textured with [93] are depicted. The results look visually appealing, contain planar surfaces and sharp edges while still containing details at non-planar building parts whereas the surroundings of the buildings are represented by smooth surfaces.

6.5.3.1 Plane Detection Using Lines

In this experiment, we compare our proposed plane detection algorithm using lines with point-based plane detection algorithms.

Figure 6.27 shows the subsequent processing steps and the result of the line-based approach and a state-of-the-art RANSAC-based plane detection algorithm proposed by Schnabel et al. [82]. For comparison, we used the implementation of [82] in CGAL [89]. We changed the default parameters to make the results comparable to our approach: We set the inlier distance to $0.15m$ as in the line-based approach and reduced the minimum supporting points per plane to 0.5% to generate more plane hypotheses. When reducing the minimum supporting points even more, more spurious planes get detected. Hence, this parameter setting has shown to be a good trade-off between amount of detected planes and

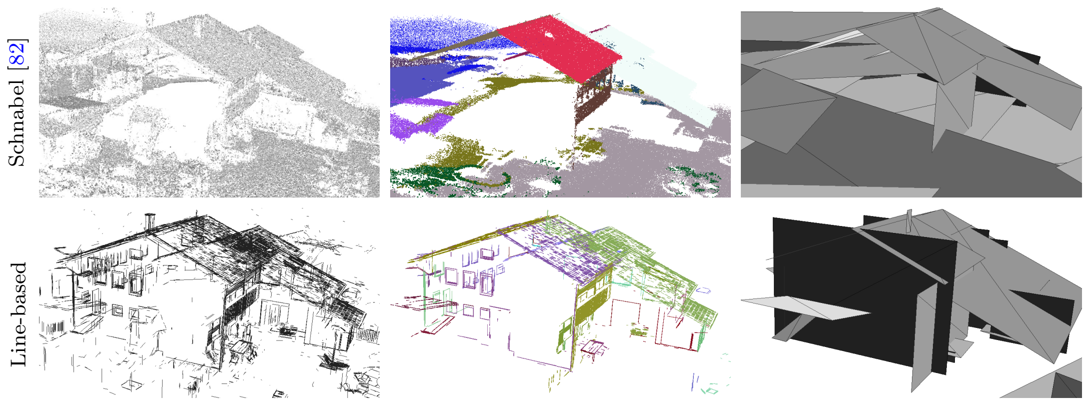


Figure 6.27: *Comparison of the proposed line-based and the point-based plane detection method of Schnabel et al. [82].* In the left column, one can see 3D input data (top: 3D points, bottom: 3D line segments): Especially at the front façade the point cloud has very few points while the line representation still contains some lines (e.g., at windows and at building edges). In the middle column, points and lines corresponding to extracted planes are illustrated (randomly colored). One can observe that the point-based approach detects planes at densely sampled surfaces well while missing sparsely reconstructed surfaces. The proposed line-based approach also detects planes which are just represented with few line segments. In the right column, one can see plane segments created by fitting bounding boxes around the inlier data on the plane surfaces. In the result of the point-based approach, spurious plane segments become visible while the line-based approach mainly contains the planes of the building.

accurately detected planes for our datasets. As can be seen in the results in Figure 6.27, Schnabel et al. [82] misses out important façade planes due to missing 3D points and detects several spurious planes whereas the line-based approach detects a more complete plane set due to the availability of line structure on the façades.

In Figure 6.28, the proposed line-based plane detection algorithm is qualitatively compared against [82] and the point-based approach presented in Section 5.2.1. In comparison to Schnabel et al. [82], the approach from Section 5.2.1 does not use point normals and, hence, results in worse detections. However, both point-based methods miss important planes. In Table 6.3, the plane detection algorithms are quantitatively compared by the number of detected planes and the completeness of detections w.r.t. the ground truth. One can see that the line-based approach has the highest completeness. Even though it detects more planes than the method of Schnabel et al., it detects less spurious ones (see Figure 6.28).

6.5.3.2 Normalized Visibility-Based Energy Term

In this experiment, we show that the normalized visibility-based energy formulation proposed in Section 5.6.2 slightly improves the reconstruction accuracy while simultaneously being easier to handle in combination with other energy terms. We evaluated the proposed energy term on the *Residential Area* dataset and, additionally, on the *Fountain-P11* [86]

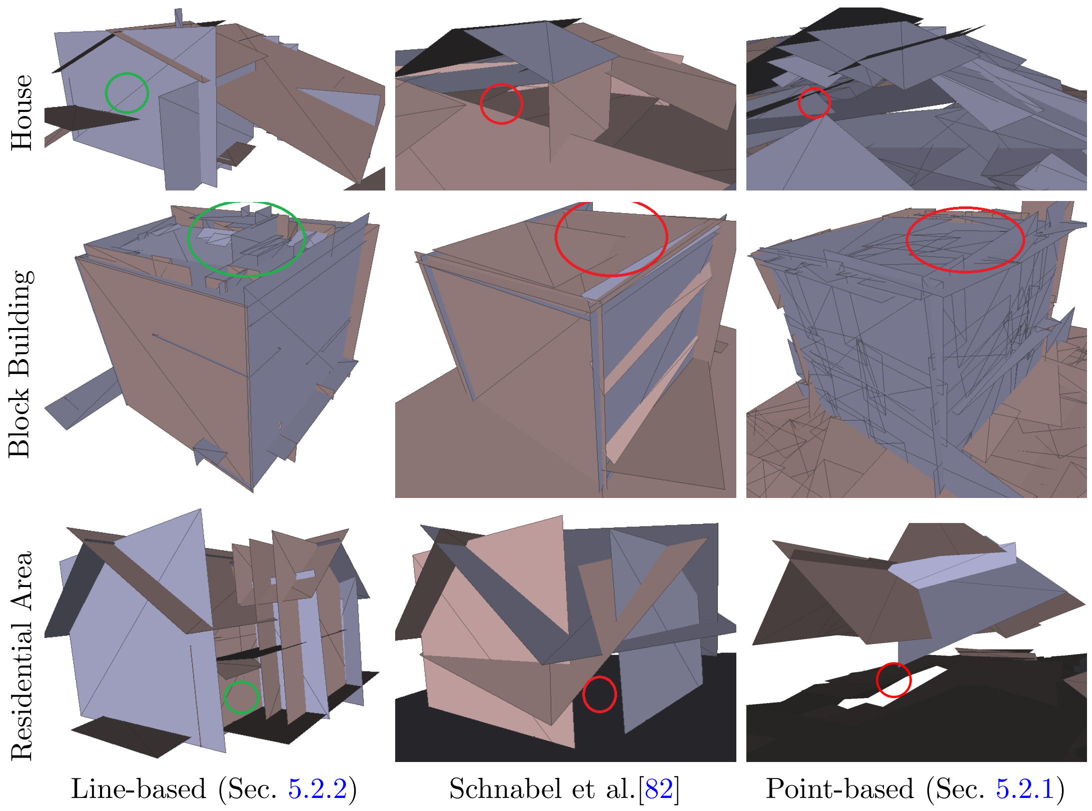


Figure 6.28: *Visual comparison of proposed line-based plane detection with point-based plane detection of [82] and the point-based method described in Section 5.2.1.* Every plane segment is visualized as two triangles. Circles mark planes detected by the proposed line-based approach (green), but not detected by the point-based approaches (red). At the House and the Residential Area dataset several façades were not detected by the point-based approaches which were detected by the line-based approach. At the Block Building, small planes at the constructions on the roof are detected by the proposed line-based approach, where the point-based approaches do not detect any planes. In general, the point-based approaches detect more spurious planes (especially the method from Section 5.2.1) but also more planes at the ground (less lines reconstructed at the ground).

dataset. In Figure 6.29, results and visualized errors are depicted and in Table 6.4 error metrics are printed with just using visibility-based energies (i.e., without additional energy terms). One can observe that the visual results are similar. For both datasets, the error metrics are very similar but slightly better for the proposed, normalized visibility-based energy formulation.

When looking at the result of *Residential Area* using just visibility-based energy, it can be seen that some surfaces are very noisy. Hence, regularizing some parts with plane priors and using a smooth surface approximation for others like vegetation is very beneficial for creating visually appealing models.

In addition to the overall energy evaluation, we evaluate each individual change in the

	House		Res. Area		Bl. Building
	#planes	compl.	#planes	compl.	#planes
Line-based (Sec. 5.2.2)	47	**0.748**	52	**0.864**	56
Schnabel et al.[82]	17	0.428	19	0.598	15
Point-based (Sec. 5.2.1)	14 (110)	0.316	24 (150)	0.434	14 (160)

Table 6.3: *Quantitative comparison of point-based and line-based plane detection approaches.* In this table one can see a comparison in terms of number of detected planes and completeness w.r.t. the ground truth for datasets where ground truth was available. The completeness is defined as the ratio of ground truth points which are within the plane detection inlier distance of $0.15m$ to the detected plane segments. For this, a filtered ground truth is used, which just contains the building parts of the scene. For the point-based plane detection method (last row), not only the detected planes but also the number of plane segments which are created by subdividing the detected planes (see Section 5.2.1) are depicted in brackets. As can be seen, the line-based approach (top row) has the highest completeness for both datasets.

	Residential Area		Fountain-P11 [86]	
	μ [m]	σ[m]	μ [m]	σ[m]
Energy from [48]	0.035	0.071	0.037	0.146
Proposed (Sec. 5.6.2)	**0.034**	**0.069**	**0.034**	**0.135**

Table 6.4: *Errors of the proposed normalized visibility-based energy terms compared to the visibility-based energy in [48] corresponding to Figure 6.29.* In the evaluation on the *Residential Area* dataset, the error is slightly lower (error definition see Table 6.1) while being normalized, which is crucial for combining it with other energies. Also in the evaluation on the Fountain-P11 [86] dataset, the error of the proposed formulation is slightly lower (error definition see [86]).

visibility-based energy term compared to Labatut et al. [48] separately and show its effects on the *Fountain-P11* [86] dataset.

In Figure 6.30 and Table 6.5 visualized errors and corresponding error and completeness values are depicted. We name the results corresponding to the added energy formulation part w.r.t. Labatut et al. [48] described in Section 5.6.2: *unary* defines the additional unary term in front of a visible vertex, $facet$ means that the pairwise energy terms at facets were assigned in both directions (as opposed to one direction in [48]) and for *normalization* the proposed energy normalization from Equation 5.2 and Equation 5.3 was applied. It can be seen visually (Figure 6.30 middle rows right) and on the error metrics in Table 6.5 that the result gets worse when just applying the normalization on the original energy and slightly improves when adding the $facet$ weights in both directions and adding the *unary* term in front of a visible vertex. The combination of all three energy formulation changes, though, delivers the best result in terms of accuracy. Only the completeness is better with normalization only, as then the mesh is just oversmoothed and, hence, the most pixels in the depth map are covered by depth measurements.

To summarize, by adding all three proposed energy formulation changes, we get the

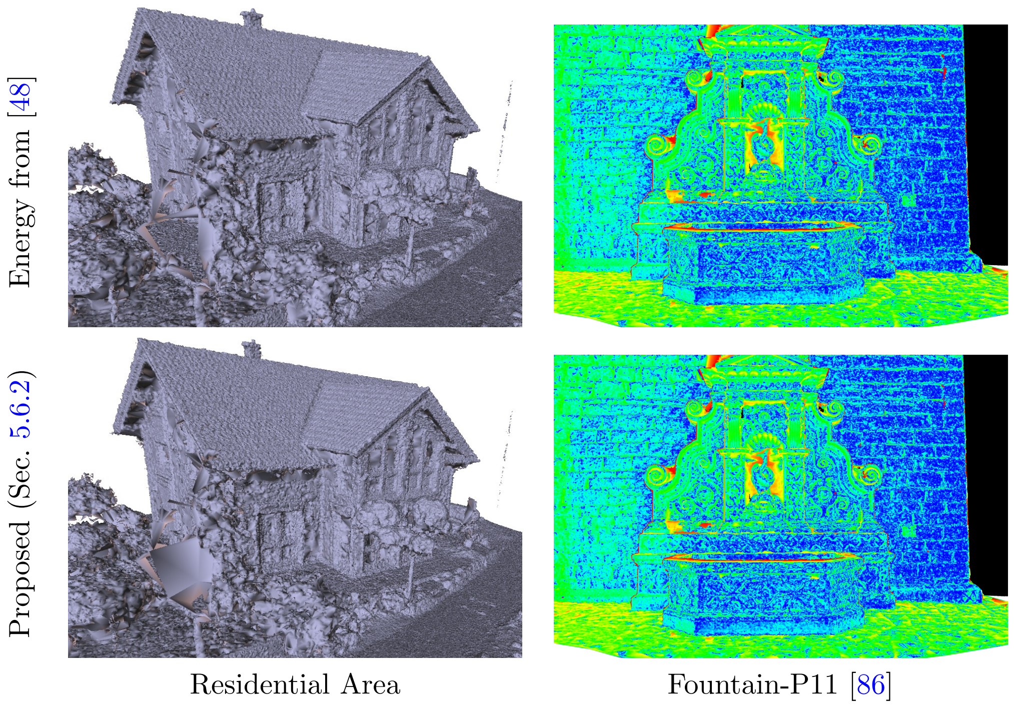


Figure 6.29: *Visualization of results and errors of proposed normalized visibility-based energy terms compared to the visibility-based energy in [48]. Left:* Evaluation on the *Residential Area* dataset. Visually, the result of the proposed energy is very similar. *Right:* Error visualization of the Fountain-P11 [86] dataset. Blue means low error, red high error. Also on this dataset, the visualized errors have no big differences (error definition see [86]). For error measures, we refer to Table 6.4

best result in terms of accuracy and, simultaneously, deliver a normalized energy which has significant advantages when combining it with additional energy terms.

6.5.3.3 Semantic 3D Data

In this section, we give a more detailed insight into the semantic image labelings which we compute and finally fuse into a semantic 3D point cloud.

In Figure 6.31 input images and the corresponding semantically labeled images for all three datasets are illustrated. In Figure 6.32 the semantically labeled input point clouds can be seen. One can observe that the point clouds are generally more consistently labeled due to the redundancy from multiple image labels for one 3D point. However, some artifacts at building edges arise, which can be explained by the depth maps used for visibility computation: We are using depth maps in half resolution and with a strong smoothness prior. Additionally, we are using nearest neighbor when back-projecting a 3D point into the depth maps. Hence, when 3D points are slightly noisy but still have visibility in the depth maps, it may happen that they get back-projected to the background and,

	μ [m]	$\sigma[m]$	*completeness*
Labatut et al. [48]	0.0366	0.1458	0.966
+unary	0.0357	0.1434	0.967
+unary+normalization	0.1076	0.2521	0.967
+unary+facet	0.0340	0.1351	0.967
+facet	0.035	0.1376	0.966
+normalization	2.6905	1.2447	**0.977**
+normalization+facet	2.6901	1.2443	**0.977**
+unary+normalization+facet	**0.0338**	**0.1346**	0.966

Table 6.5: *Error statistics compared to the ground truth.* μ defines the mean and σ the standard deviation of depth values of the result and the ground truth model projected into all cameras. *Completeness* defines the coverage of the projected models in the camera views compared to the ground truth (both defined in [86]). In terms of accuracy, the proposed approach (i.e., adding all three changes in the energy formulation) delivers the best error metrics. Only in terms of completeness other results are slightly better with the tradeoff of lower accuracy and no normalized energy.

hence, get assigned the label of the background (which might be, e.g., sky or vegetation).

6.5.3.4 Semantic Priors

In Figure 6.33 it can be observed that when using no semantics (same as treating everything as building), the surroundings of buildings are noisy and less visually appealing. Also, more data needs to be used to describe the noisy mesh, while with semantics a sparser Poisson reconstruction describes the surroundings. Further, artifacts may arise as the Manhattan regularity term is applied everywhere even though it might not be suited to smooth e.g. vegetation.

6.5.3.5 Simplification as Post-Processing

Due to the planarity of big parts of the resulting mesh of the proposed method, the amount of faces in the mesh can be significantly reduced without changing the surface geometry. In Figure 6.34, results of applying Quadric Edge Collapse [24] as post-processing step are depicted. As the maximum error of Quadric Edge Collapse was set to 10^{-13} (i.e., nearly zero), only edges on planar surfaces were removed. It can be observed that even though the mesh surface did not change at all, the amount of data was extremely reduced. This property of the results of our approach is highly beneficial for further processing steps like texturing, but also visualizing and transmitting the reconstructions becomes easier.

6.6 Summary

In this chapter, we presented a detailed evaluation of the urban reconstruction methods developed in this thesis. We first discussed the input data in Section 6.1 and default parameters in Section 6.3 and then presented results with quantitative and qualitative comparisons to other methods in Section 6.4: We showed that compared to the slicing-based method presented in Chapter 4, the tetrahedra-based method presented in Chapter 5 could also handle buildings containing sloped structures and could reconstruct fine-grained structures with more detail. Additionally using semantic information, the tetrahedra-based method was able to reconstruct a smoothed surface for the surroundings of a building, while the building parts consisted of planar surfaces and sharp edges.

Then we discussed method-specific detailed evaluations in Section 6.5: In Section 6.5.1 we presented additional results for the slicing-based approach. It could be observed, that this approach can create regularized and abstracted reconstructions of buildings fulfilling a specific geometric constraint and that especially the introduced line-based smoothness term improves the reconstruction result when having noisy and erroneous input data. In Section 6.5.2 and Section 6.5.3, we showed additional results and evaluations of two sub-methods of the tetrahedra-based method, where different plane detection algorithms, energy definitions and additionally semantic information were used. Finally, applying a post-processing step, the tetrahedra-based reconstruction result could be extremely simplified without loss of accuracy, which is an important property for further processing steps.

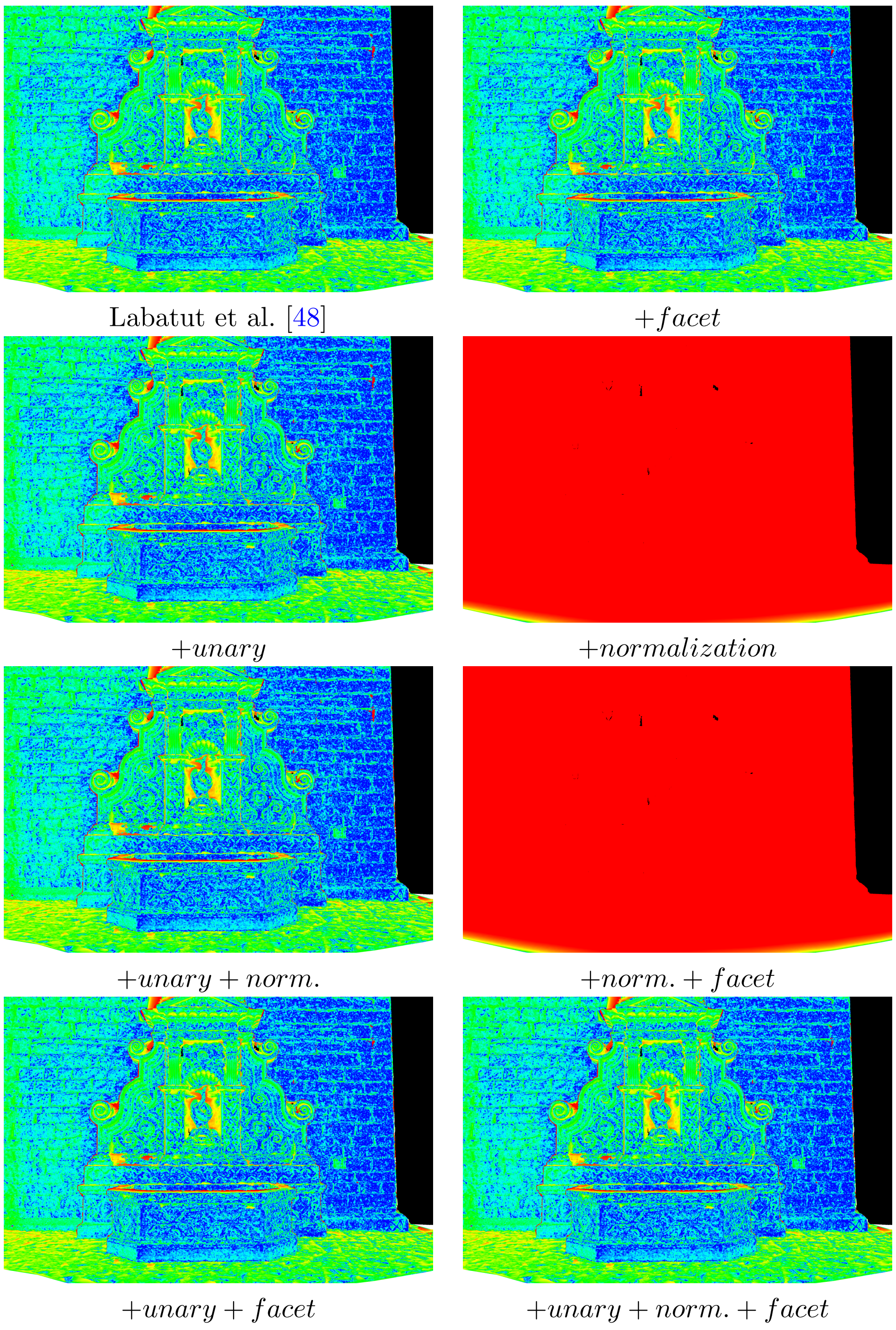


Figure 6.30: *Evaluation of the individual changes compared to [48] in the visibility-based energy on the Fountain-P11 [86] dataset.* As can be seen, when using the normalization without the other two changes, the error is significantly higher due to an oversmoothing of the mesh. All other error images look visually similar.

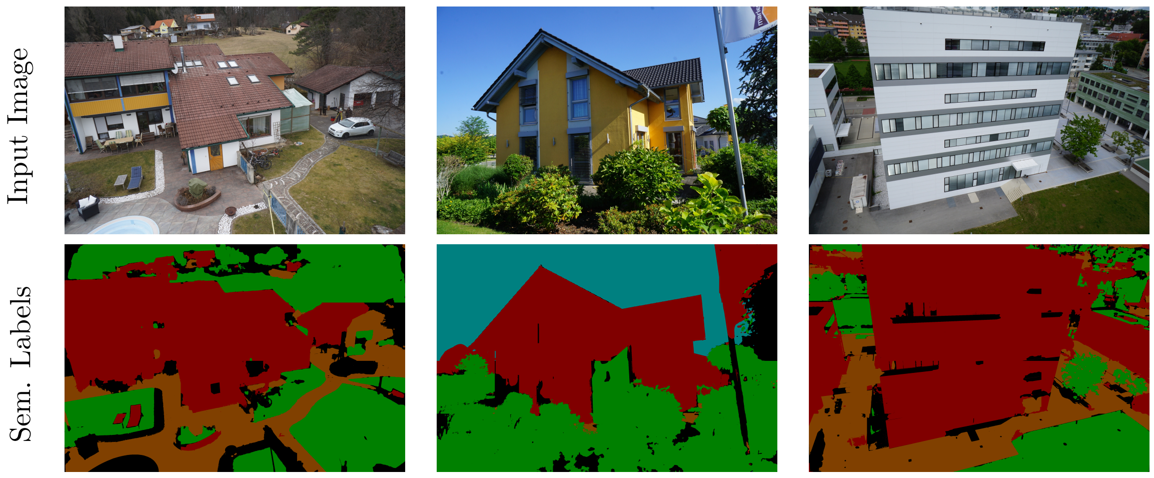


Figure 6.31: *Semantically labeled input images* from the *House* (left), *Residential Area* (middle) and *Block Building* (right) datasets. The pixels are labeled as sky (blue), building (red), vegetation (green), road/pavement (brown) and clutter (black). The labeling is not always perfectly correct. However, due to the redundancy, the 3D points have less labeling errors (see Figure 6.32).

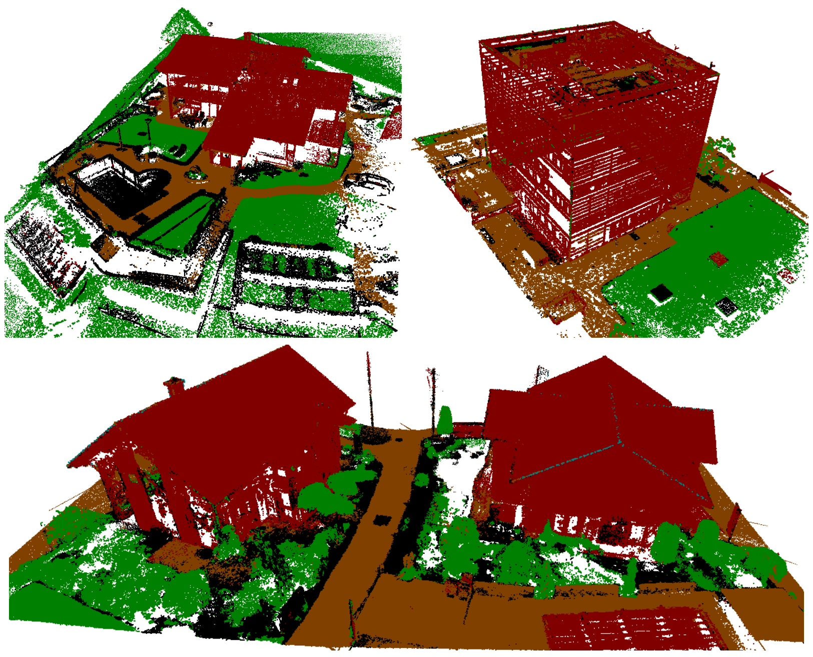


Figure 6.32: *Semantically labeled point clouds* from the *House* (top left), *Block Building* (top right) and *Residential Area* (bottom) datasets. The 3D points are labeled as sky (blue), building (red), vegetation (green), road/pavement (brown) and clutter (black). Due to the redundancy from multiple image labelings, the point cloud labeling is usually more consistent compared to the single image labelings.

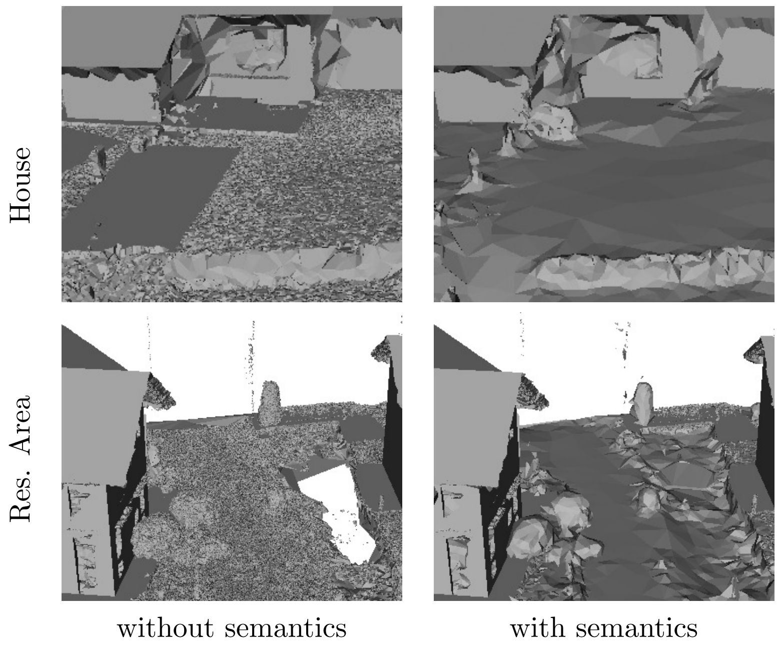


Figure 6.33: *Comparison of results computed with/without semantic information.* As can be seen in the detail examples of two datasets, the reconstructed surface of the surroundings of buildings is more noisy without semantics. Due to the Poisson reconstruction and the different smoothness terms, the surroundings get reconstructed much smoother and with less points (i.e. less data) when using semantics. Also, detected planes are removed in the surroundings, as just building lines are used for plane detection. These changes due to semantics contribute to a more visually appealing final reconstruction.

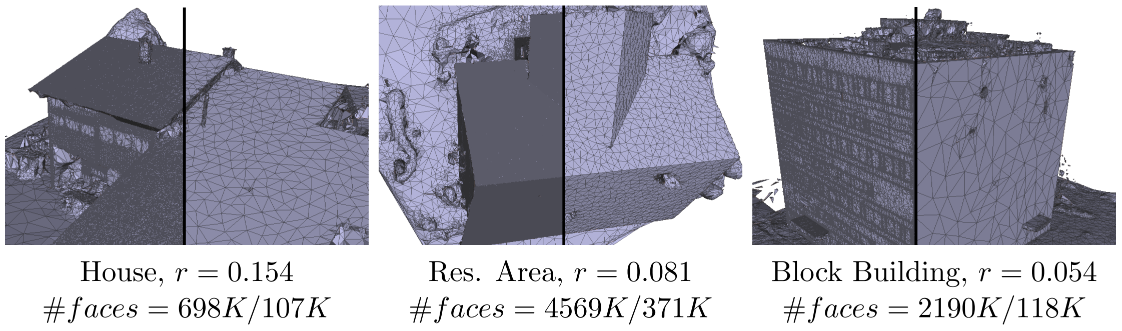


Figure 6.34: *Simplification as post-processing.* Every subfigure consists of the resulting mesh of the proposed approach (*left*) and the mesh simplified by Quadric Edge Collapse [24] in a lossless way (*right*) (i.e., restricting the edge collapse to a quadric error of 10^{-13}). Below, reduction factors and $\#faces$ before/after simplification are depicted. Due to the perfect planarity of the building parts, faces can be merged without changing the surface geometry of the mesh. Though, non-building parts stay untouched as they are not perfectly planar. Very low reduction factors $r = \frac{\#faces_{simpl}}{\#faces_{orig}}$ (i.e., high compression) can be reached for all models. In comparison, when applying Quadric Edge Collapse on Poisson meshes in Figure 6.7, the best reduction factor was 0.929 ($\#faces = 393k/365K$) at House. Quadric Edge Collapse was performed with VCGlib [90].

7 Conclusions

Contents

7.1 Summary 105
7.2 Outlook 107

This thesis has addressed the problem of creating visually appealing 3D reconstructions of urban scenes using images as input. Such visually appealing models should contain buildings modeled with planar surfaces and sharp edges embedded in a smoothly reconstructed surrounding. Further, the level of detail of the reconstruction should be adjustable by changing a parameter. In this thesis, we presented two methods which tackle this problem in different ways.

In the following, a summary and the key findings of our work are presented in Section 7.1 and an outlook for potential future research directions and problems still occurring with the current methods is presented in Section 7.2.

7.1 Summary

In this thesis, we initially stated that even though mature 3D image-based reconstruction techniques exist, they are not capable of creating visually appealing and yet accurate reconstructions of urban scenes which simultaneously can be represented in a compact way. Most of the reconstruction systems either produce compact but oversmoothed reconstructions containing no sharp edges, or reconstructions which contain too many fine-grained details originating from noisy, image-based measurements, which leads to a non-compact representation and noisy surfaces (i.e., no planar surfaces, sharp edges). As such compact and regularized reconstructions, however, are desired for several use cases, this motivated us to investigate more deeply in this topic.

We developed two urban reconstruction techniques whereas the first one is a slicing-based approach which can model arbitrary building scenes with a simple and yet visually appealing representation. Buildings are modeled with planar surfaces and sharp edges even when the input 3D data is very noisy. However, specific scene structure like sloped surfaces cannot be modeled exactly with this approach and, further, it is not possible to model scene parts with fine-grained details. Therefore, we investigated in a second, Delaunay triangulation-based method, which uses a more generic volumetric partitioning of the scene which can represent any structure. We advanced a well-known 3D reconstruction technique and incorporated plane priors within the optimization in order to reconstruct buildings with planar surfaces and sharp edges while still containing details at parts which could not be represented by planes. Further, we incorporated semantic information within the reconstruction process and introduced a line-based plane detection algorithm, which improves the plane detection results at poorly textured urban environments.

The slicing-based approach works by first partitioning the input 3D data into several horizontal slices bounded by dominant horizontal structure. Then, for every slice an inside/outside labeling, which is a 2D labeling of every slice, is created. Having this inside/outside labeling, the slice outlines can be identified as the transition of these labeled parts. By extruding the outlines between the slice boundaries, an initial 3D model can be generated. However, this 3D model is not normalized in the vertical direction and, hence, the vertical surfaces are not planar and the edges are not sharp. To overcome this issue, an additional optimization step is applied: The scene is partitioned into irregularly shaped volumetric cells defined by the slice outlines and slice boundaries and an inside/outside labeling of these cells is computed using Graph Cuts. Additionally, we used lines detected within the input images to formulate an smoothness term, with which it is possible to produce a more smoother building outline even at parts with missing 3D input data. In our experiments, we have shown that this approach delivers compact and clean 3D models of buildings covering the most important geometric details. We have shown that the proposed line-based smoothness term additionally improves the result at poorly reconstructed building parts and that the proposed approach can handle buildings with arbitrary building outlines and is not restricted to a Manhattan world scenery. This approach delivers stacked slices or, more general, 3D blocks as output, which is very similar to models created by CAD software. However, our experiments have also shown that the proposed approach has difficulties to model sloped surfaces, which can only be approximated by staircase-like structures. Further, fine details are not modeled with the presented approach.

In order to be able to reconstruct more generic scenes with arbitrary levels of detail, we developed a second method for urban 3D reconstruction. This method is based on a tetrahedralization of the scene, where every tetrahedron gets assigned a label as inside or outside by solving an energy minimization problem using Graph Cuts and the final surface is extracted as the transition between cells labeled as inside or outside. In order to enforce planarity in the reconstruction, planes are detected in the scene and incorporated within the tetrahedralization to be selectable in the final optimization. Additional energy terms are proposed to favor a planar reconstruction and to adjust the final level of detail of the reconstruction, namely the Manhattan term and the level of detail term. As plane detection using a point cloud as input delivered poor results at weakly textured urban environments, we introduced a line-based plane detection algorithm: We observed that even if no 3D points are reconstructed at some surfaces within urban environments, still sufficient 3D line segments could be reconstructed in order to correctly detect planes within the scene. Using this line-based plane detection resulted in a more complete set of detected planes. Further, we proposed a normalized energy formulation of the visibility-based energy terms, which eases the combination with further energy terms like the proposed additional energy terms. As introduced shape priors should not be the same at every part of the scene in the reconstruction process, we additionally used semantic information in order to enforce shape priors depending on the scene classes. In our experiments, we demonstrated that we can vary the level of detail of the reconstruction by varying the introduced energy terms and that the proposed line-based plane detection algorithm improves the plane detection result at poorly textured urban environments. We have shown that the introduced normalized visibility-based energy slightly improves the reconstruction quality with the advantage of having a normalized magnitude, which is crucial for combining it with other energy terms. Further, we demonstrated that an overall more visually appealing reconstruction result can be achieved when using semantic information. Finally, we have shown that by applying a post-processing step a very compact representation of the resulting 3D reconstructions can be created without any loss of accuracy.

We compared the results of the proposed approaches with state-of-the-art generic reconstruction methods and methods specifically designed for urban reconstruction and have shown that the proposed approach produces more complete, more visually appealing and more accurate results than the methods for comparison.

7.2 Outlook

The proposed methods create visually appealing reconstructions of urban environments containing planar surfaces and sharp edges at building parts and optionally a smooth reconstruction at the surroundings. Even though the proposed methods work well and are sufficient for many scenes, there is, of course, still room for improvement.

A weakness of our methods is that they are currently not scalable to large-scale scenes containing several buildings or bigger city parts: Even though both methods use irregularly shaped volumetric cells adapting to the scene structure, they are both using a global optimization technique which, at a certain size of the scene, will be practically uncomputable. Hence, a partitioning of the scene into several blocks which are processed separately and finally fused together again would be beneficial. One could use semantic information to cluster the input data in order to get processing blocks containing connected building parts and transitions from one processing block to the adjacent one in the surroundings part. However, then artifacts might arise in the transition area of multiple blocks. To overcome such problems at the tetrahedra-based method, one could use a method similar to the one proposed in Mostegel et al. [62]: They apply an additional Graph Cut optimization at the overlapping regions in order to get a consistent mesh of several submeshes. Using such a method, it would be possible to process scenes of unlimited size with an approximately constant memory footprint.

A potential extension for the tetrahedra-based method would be to use additional shape priors for different semantic classes. For example, one could also use a plane prior for streets. Further, a sphere and tube prior for trees could be introduced. This could be done by fitting such primitives within the scene [82], and incorporating them into the triangulation, like it is currently done with planes (see Section 5.4). Then, an additional smoothness term similar to the Manhattan regularization term described in Section 5.7.1 would need to be defined. A simple possibility would be to penalize facets not lying on the detected primitive and being labeled as vegetation (in the case of a tree). Similarly, depending on the architectural type of buildings, one could also introduce additional shape priors for buildings like spheres, tubes, cones or surfaces of revolution. By adjusting the additionally introduced smoothness terms, one could enforce the reconstruction to follow nearly everywhere detected primitives in order to create a very abstracted representation of the whole scene while still being able to reconstruct scene parts which are not represented by a primitive.

Another possibility would be to learn shape priors for individual classes: Inspired by Häne et al. [29], one could aim to learn class-specific favored neighboring relationships of facets, which would result in individual smoothness terms for every class. Further, by using a multi-label optimization technique (like the alpha expansion algorithm [45]), learned class adjacency priors could be incorporated within the optimization process. For example, it should be learned and defined as a smoothness prior that a street should not be on top of a house, but adjacent to grass. By using such class-dependent smoothness terms, the reconstruction should become more accurate and, simultaneously, class-specific shape priors should be enforced. As also the semantic labels would be optimized in combination with the reconstruction process, also the semantic labeling accuracy would be improved.

One could also think about combining the two proposed methods to get the advantages of both: The slicing-based approach delivers very well regularized building models and is also able to cope with low input data density. Though, it does not reconstruct the surroundings of buildings and fine-grained details well. Contrary, the tetrahedra-based method is based on a generic 3D reconstruction method and, hence, is able to reconstruct an arbitrary scene with many details. However, in order to produce well regularized reconstructions, it needs already detected primitives in the scene as input. Hence, the simple representation of the slicing-based approach consisting of geometric primitives could be incorporated within the tetrahedralization. Then, this primitive-based representation is selectable and can be enforced within the tetrahedra-based method while fine grained details are still reconstructible.

A further potential source of geometric primitives would be shape proposals created by a Convolutional Neural Network (CNN): Inspired by [47], one could learn shape proposals from objects in the scene having a point cloud as input. The simplest idea would be to learn a box-like representation for buildings. These geometric primitives could then be incorporated within the tetrahedralization. In the final optimization, these shapes would be selectable and could be enforced by setting a smoothness prior.

To conclude, there is still room for improvement in order to create a generic image-based 3D reconstruction system for urban scenes creating compact and visually appealing reconstructions. In general, we think that this topic will gain importance in the future as more and more applications for 3D data arise where this data should be processable and visualizable on various devices. Hence, compact and yet accurate models are required.

A

List of Acronyms

BSP	Binary Space Partitioning
CAD	Computer Aided Design
CDT	Constrained Delaunay Triangulation
CGAL	Computational Geometry Algorithms Library
CMVS	Clustered Multi-View Stereo
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRFasRNN	Conditional Random Field as Recurrent Neural Network
FCN	Fully Convolutional Neural Network
GB	Gigabyte
GHz	Gigahertz
GPU	Graphics Processing Unit
LOD	Level Of Detail
LSD	Line Segment Detector
MPixel	Megapixel
MVS	Multi-View Stereo
ORB	Oriented FAST and Rotated BRIEF
P3P	Perspective-Three-Point
PCA	Principal Component Analysis
PMVS2	Patch-Based Multi-View Stereo
PnP	Perspective-n-Point
RAM	Random Access Memory
RANSAC	Random Sample Consesus
RGB	Red-Green-Blue
RGBD	Red-Green-Blue-Depth
SDF	Signed Distance Function

SfM	Structure from Motion
SGM	Semi-Global Matching
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
TSDF	Truncated Signed Distance Function
UAV	Unmanned Aerial Vehicle
VR	Virtual Reality

B List of Publications

My work at the Institute for Computer Graphics and Vision led to the following peer-reviewed publications. For the sake of completeness of this Thesis, they are listed in inverse chronological order along with the respective abstracts.

B.1 2018

Deep 2.5D Vehicle Classification with Sparse SfM Depth Prior for Automated Toll Systems

Georg Waltner, Michael Maurer, Thomas Holzmann, Patrick Ruprecht, Michael Opitz, Horst Possegger, Friedrich Fraundorfer, and Horst Bischof
In: *Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems (ITSC)*
November 2018, Maui, USA

Abstract: Automated toll systems rely on proper classification of the passing vehicles. This is especially difficult when the images used for classification only cover parts of the vehicle. To obtain information about the whole vehicle. we reconstruct the vehicle as 3D object and exploit this additional information within a Convolutional Neural Network (CNN). However, when using deep networks for 3D object classification, large amounts of dense 3D models are required for good accuracy, which are often neither available nor feasible to process due to memory requirements. Therefore, in our method we reproject the 3D object onto the image plane using the reconstructed points, lines or both. We utilize this sparse depth prior within an auxiliary network branch that acts as a regularizer during training. We show that this auxiliary regularizer helps to improve accuracy compared to 2D classification on a real-world dataset. Furthermore due to the design of the network, at test time only the 2D camera images are required for classification which enables the

usage in portable computer vision systems.

Related Chapters: -

Semantically Aware Urban 3D Reconstruction with Plane-Based Regularization

Thomas Holzmann, Michael Maurer, Friedrich Fraundorfer, and Horst Bischof
In: *Proceedings of European Conference on Computer Vision (ECCV)*
September 2018, Munich, Germany
(Accepted for poster presentation)

Abstract: We propose a method for urban 3D reconstruction, which incorporates semantic information and plane priors within the reconstruction process in order to generate visually appealing 3D models. We introduce a plane detection algorithm using 3D lines, which detects a more complete and less spurious plane set compared to point-based methods in urban environments. Further, the proposed normalized visibility-based energy formulation eases the combination of several energy terms within a tetrahedra occupancy labeling algorithm and, hence, is well suited for combining it with class specific smoothness terms. As a result, we produce visually appealing and detailed building models (i.e., straight edges and planar surfaces) and a smooth reconstruction of the surroundings.

Related Chapters: 5, 6

B.2 2017

Plane-based Surface Regularization for Urban 3D Reconstruction

Thomas Holzmann, Martin R. Oswald, Marc Pollefeys, Friedrich Fraundorfer and Horst Bischof
In: *Proceedings of the 28th British Machine Vision Conference (BMVC)*
September 2017, London, United Kingdom
(Accepted for poster presentation)

Abstract: We propose a method for urban 3D reconstruction that is a hybrid between a volumetric 3D reconstruction approach and a plane fitting approach in order to obtain a denoised and compact representation of the scene. In our hybrid approach, a single global optimization, using visibility as main information, defines whether the final reconstructed surface should align with a detected plane or rather follow the details of the input data. Our method is based on an established tetrahedral occupancy labeling approach which we taylor for urban reconstruction by adding the possibility to favor an alignment of

the surface with detected planes. We further add novel regularization terms that favor Manhattan-like structures and which allow to control the level of detail of the output model. A variety of experiments demonstrate state-of-the-art performance and show that our approach is suitable for both indoor and outdoor environments.

Related Chapters: 5, 6

A Detailed Description of Direct Stereo Visual Odometry Based on Lines

Thomas Holzmann, Friedrich Fraundorfer and Horst Bischof
In: *Communications in Computer and Information Science (CCIS), vol 693*
August 2017

Abstract: In this paper, we propose a direct stereo visual odometry method which uses vertical lines to estimate consecutive camera poses. Therefore, it is well suited for poorly textured indoor environments where point-based methods may fail. We introduce a fast line segment detector and matcher detecting vertical lines, which occur frequently in man-made environments. We estimate the pose of the camera by directly minimizing the photometric error of the patches around the detected lines. In cases where not sufficient lines could be detected, point features are used as fallback solution. As our algorithm runs in real-time, it is well suited for robotics and augmented reality applications. In our experiments, we show that our algorithm outperforms state-of-the-art methods on poorly textured indoor scenes and delivers comparable results on well textured outdoor scenes.

Related Chapters: -

B.3 2016

Regularized 3D Modeling from Noisy Building Reconstructions

Thomas Holzmann, Friedrich Fraundorfer and Horst Bischof
In: *Proceedings of the 4th International Conference on 3D Vision (3DV)*
October 2016, Stanford, USA
(Accepted for poster presentation)

Abstract: In this paper, we present a method for regularizing noisy 3D reconstructions, which is especially well suited for scenes containing planar structures like buildings. At horizontal structures, the input model is divided into slices and for each slice, an inside/outside labeling is computed. With the outlines of each slice labeling, we create an irregularly shaped volumetric cell decomposition of the whole scene. Then, an optimized

inside/outside labeling of these cells is computed by solving an energy minimization problem. For the cell labeling optimization we introduce a novel smoothness term, where lines in the images are used to improve the regularization result. We show that our approach can take arbitrary dense meshed point clouds as input and delivers well regularized building models, which can be textured afterwards.

Related Chapters: 4, 6

Direct Stereo Visual Odometry Based on Lines

Thomas Holzmann, Friedrich Fraundorfer and Horst Bischof
In: *Proceedings of the 11th International Conference on Computer Vision Theory and Applications (VISAPP)*
February 2016, Rome, Italy
(Accepted for oral presentation)
Best Paper Award

Abstract: We propose a novel stereo visual odometry approach, which is especially suited for poorly textured environments. We introduce a novel, fast line segment detector and matcher, which detects vertical lines supported by an IMU. The patches around lines are then used to directly estimate the pose of consecutive cameras by minimizing the photometric error. Our algorithm outperforms state-of-the-art approaches in challenging environments. Our implementation runs in real-time and is therefore well suited for various robotics and augmented reality applications.

Related Chapters: -

B.4 2015

Performance Evaluation of Vision-Based Algorithms for MAVs

Thomas Holzmann, Rudolf Prettenthaler, Jesus Pestana Puerta, Daniel Muschick, Christian Mostegel, Gottfried Graber, Friedrich Fraundorfer and Horst Bischof
In: *Proceedings of the Workshop of the Austrian Association for Pattern Recognition (ÖAGM/AAPR)*
May 2015, Salzburg, Austria
(Accepted for oral presentation)

Abstract: An important focus of current research in the field of Micro Aerial Vehicles (MAVs) is to increase the safety of their operation in general unstructured environments. Especially indoors, where GPS cannot be used for localization, reliable algorithms for localization and mapping of the environment are necessary in order to keep an MAV

airborne safely. In this paper, we compare vision-based real-time capable methods for localization and mapping and point out their strengths and weaknesses. Additionally, we describe algorithms for state estimation, control and navigation, which use the localization and mapping results of our vision-based algorithms as input.

Related Chapters: -

Graz Griffins' Solution to the European Robotics Challenges 2014

Jesus Pestana Puerta, Rudolf Prettenthaler, Thomas Holzmann, Daniel Muschick, Christian Mostegel, Friedrich Fraundorfer, Horst Bischof
In: *Proceedings of the Austrian Robotics Workshop (ARW)*
May 2015, Klagenfurt, Austria
(Accepted for oral presentation)

Abstract: An important focus of current research in the field of Micro Aerial Vehicles (MAVs) is to increase the safety of their operation in general unstructured environments. An example of a real-world application is visual inspection of industry infrastructure, which can be greatly facilitated by autonomous multicopters. Currently, active research is pursued to improve real-time vision-based localization and navigation algorithms. In this context, the goal of Challenge 3 of the EuRoC 2014 Simulation Contest was a fair comparison of algorithms in a realistic setup which also respected the computational restrictions onboard an MAV. The evaluation separated the problem of autonomous navigation into four tasks: visual-inertial localization, visual-inertial mapping, control and state estimation, and trajectory planning. This EuRoC challenge attracted the participation of 21 important European institutions. This paper describes the solution of our team, the Graz Griffins, to all tasks of the challenge and presents the achieved results.

Related Chapters: -

B.5 2014

Geometric Abstraction from Noisy Image-Based 3D Reconstructions

Thomas Holzmann, Christof Hoppe, Stefan Kluckner, Horst Bischoff
In: *Proceedings of the Workshop of the Austrian Association for Pattern Recognition (ÖAGM/AAPR)*
May 2014, Klosterneuburg, Austria
(Accepted for oral presentation)
Best Paper Prize

Abstract: Creating geometric abstracted models from image-based scene reconstructions is difficult due to noise and irregularities in the reconstructed model. In this paper, we present a geometric modeling method for noisy reconstructions dominated by planar horizontal and orthogonal vertical structures. We partition the scene into horizontal slices and create an 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.

Related Chapters: 4, 6

Bibliography

[1] Agisoft PhotoScan (2018). `http://www.agisoft.com/`. (page 2, 15, 72)

[2] Albertz, J. (2009). 100 jahre deutsche gesellschaft für photogrammetrie, fernerkundung und geoinformation. *Geoinformation*, (6):487–560. (page 1)

[3] Amenta, N. and Bern, M. (1998). Surface reconstruction by voronoi filtering. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 39–48, New York, NY, USA. ACM. (page 51)

[4] Arefi, H., Engels, J., Hahn, M., and Mayer, H. (2008). Levels of detail in 3d building reconstruction from lidar data. In *Proceedings of International Society for Photogrammetry and Remote Sensing (ISPRS)*. (page 29)

[5] Arikan, M., Schwärzler, M., Flöry, S., Wimmer, M., and Maierhofer, S. (2013). O-snap: Optimization-based snapping for modeling architecture. *ACM Trans. Graph.*, 32(1):6:1–6:15. (page 26)

[6] Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 13)

[7] Blaha, M., Vogel, C., Richard, A., Wegner, J., Schindler, K., and Pock, T. (2016). Large-scale semantic 3d reconstruction: an adaptive multi-resolution model for multi-class volumetric labeling. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 30)

[8] Boissonnat, J.-D. and Yvinec, M. (1998). *Algorithmic Geometry*. Cambridge University Press. (page 20, 51)

[9] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*. (page 71)

[10] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 20(12):1222–1239. (page 5, 20, 27, 38, 55, 71)

[11] Cherabier, I., Häne, C., Oswald, M. R., and Pollefeys, M. (2016). Multi-label semantic 3d reconstruction using voxel blocks. In *Proceedings of International Conference on 3D Vision (3DV)*. (page 30)

[12] Cherabier, I., Schönberger, J. L., Oswald, M. R., Pollefeys, M., and Geiger, A. (2018). Learning priors for semantic 3d reconstruction. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 30)

[13] Collins, R. T. (1996). A space-sweep approach to true multi-image matching. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 16)

[14] Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 24(5):603–619. (page 36, 49)

[15] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *ACM Trans. on Graphics (SIGGRAPH)*, pages 303–312. (page 23)

[16] Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*. (page 38)

[17] Duan, L. and Lafarge, F. (2016). Towards large-scale city reconstruction from satellites. In *Proceedings of European Conference on Computer Vision (ECCV)*, pages 89–104. (page 29)

[18] Dzitsiuk, M., Sturm, J., Maier, R., Ma, L., and Cremers, D. (2017). De-noising, stabilizing and completing 3D reconstructions on-the-go using plane priors. In *Proceedings of International Conference on Robotics and Automation (ICRA)*. (page 26)

[19] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395. (page 11, 26, 49)

[20] Freeden, W. and Rummel, R. (2017). *Photogrammetrie und Fernerkundung - Handbuch der Geodäsie*. Springer Spektrum. (page 1)

[21] Furukawa, Y., Curless, B., Seitz, S. M., and Szeliski, R. (2010). Towards internet-scale multi-view stereo. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 15)

[22] Furukawa, Y. and Ponce, J. (2010). Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*. (page 15, 16, 68, 79, 80, 82, 83, 86, 88, 90)

[23] Gao, X.-S., Hou, X.-R., Tang, J., and Cheng, H.-F. (2003). Complete solution classification for the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 25(8):930–943. (page 14)

[24] Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *ACM Trans. on Graphics (SIGGRAPH)*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., New York. (page 99, 103)

[25] Google (2018). Google maps. `http://maps.google.com/`. (page 3)

[26] Grenander, U. and Miller, M. (1994). Representations of knowledge in complex system. *J. Royal Statistical Soc.*, 56(4):549–603. (page 28)

[27] Grompone, R., Jakubowicz, J., Morel, J. M., and Randall, G. (2010). Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(4):722–732. (page 17, 41, 68)

[28] Häne, C., Heng, L., Lee, G. H., Sizov, A., and Pollefeys, M. (2014). Real-time direct dense matching on fisheye images using plane-sweeping stereo. In *International Conference on 3D Vison (3DV)*. (page 16, 68)

[29] Häne, C., Zach, C., Cohen, A., Angst, R., and Pollefeys, M. (2013). Joint 3d scene reconstruction and class segmentation. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 30, 108)

[30] Häne, C., Zach, C., Cohen, A., and Pollefeys, M. (2016). Dense semantic 3d reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pages 1730–1743. (page 30, 31)

[31] Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry In Computer Vision*. Cambridge University Press. (page 8, 10, 11, 14)

[32] Hirschmüller, H. (2008). Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(2):328–341. (page 15)

[33] Hofer, M., Maurer, M., and Bischof, H. (2016). Efficient 3d scene abstraction using line segments. *Computer Vision and Image Understanding (CVIU)*. (page 16, 17, 68)

[34] Holzmann, T., Fraundorfer, F., and Bischof, H. (2016). Regularized 3d modeling from noisy building reconstructions. In *Proceedings of International Conference on 3D Vision (3DV)*, pages 528–536. (page 34, 79)

[35] Holzmann, T., Hoppe, C., Kluckner, S., and Bischof, H. (2014). Geometric abstraction from noisy image-based 3d reconstructions. In *Proceedings of The 38th Annual Workshop of the Austrian Association for Pattern Recognition (ÖAGM)*. (page 34, 79)

[36] Holzmann, T., Maurer, M., Fraundorfer, F., and Bischof, H. (2018). Semantically aware urban reconstruction with plane-based regularization. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 45, 46, 57, 62, 72, 94)

[37] Holzmann, T., Oswald, M. R., Pollefeys, M., Fraundorfer, F., and Bischof, H. (2017). Plane-based surface regularization for urban 3d reconstruction. In *Proceedings of British Machine Vision Conference (BMVC)*, volume 28. (page 45, 56, 58, 71, 85)

[38] Hoppe, C., Klopschitz, M., Donoser, M., and Bischof, H. (2013). Incremental surface extraction from sparse structure-from-motion point clouds. In *Proceedings of British Machine Vision Conference (BMVC)*. (page 20, 57)

[39] Hoppe, H., DeRose, T., McDonald, T. D. J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. In *ACM Trans. on Graphics (SIGGRAPH)*, volume 26, pages 71–77. (page 63, 68)

[40] Ikehata, S., Yang, H., and Furukawa, Y. (2015). Structured indoor modeling. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 1323–1331. (page 27)

[41] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*. (page 71)

[42] Kähler, O., Prisacariu, V. A., Valentin, J. P. C., and Murray, D. W. (2016). Hierarchical voxel block hashing for efficient integration of depth images. *IEEE Robotics and Automation Letters*, 1(1):192–197. (page 23)

[43] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of Eurographics Symposium on Geometry Processing*. (page 17, 18, 19, 35, 63, 73, 74, 76, 77, 79, 80, 83, 85)

[44] Kolbe, T. H., Gröger, G., and Plümer, L. (2005). Citygml: Interoperable access to 3d city models. *Geo-information for Disaster Management*, pages 883–899. (page 29)

[45] Kolmogorov, V. and Zabih, R. (2002). What energy functions can be minimized via graph cuts? In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 71, 108)

[46] Korč, F. and Förstner, W. (2009). eTRIMS Image Database for interpreting images of man-made scenes. Technical Report TR-IGG-P-2009-01. (page 71)

[47] Ku, J., Mozifian, M., Lee, J., Harakeh, A., and Waslander, S. (2018). Joint 3d proposal generation and object detection from view aggregation. In *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*. (page 108)

[48] Labatut, P., Pons, J.-P., and Keriven, R. (2007). Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *Proceedings of International Conference on Computer Vision (ICCV)*. (page 20, 21, 22, 47, 56, 57, 59, 64, 78, 81, 82, 86, 87, 88, 89, 90, 91, 92, 93, 97, 98, 99, 101)

[49] Labatut, P., Pons, J.-P., and Keriven, R. (2009a). Hierarchical shape-based surface reconstruction for dense multi-view stereo. In *International Workshop on 3-D Digital*

Imaging and Modeling (3DIM), ICCV Workshops, pages 1598–1605, Kyoto, Japan. (page 28)

[50] Labatut, P., Pons, J.-P., and Keriven, R. (2009b). Robust and efficient surface reconstruction from range data. *Computer Graphics Forum*, pages 2275–2290. (page 20, 21, 56)

[51] Lafarge, F. and Alliez, P. (2013). Surface reconstruction through point set structuring. *Computer Graphics Forum*, 32(2):225–234. (page 28, 29, 51, 52)

[52] Lafarge, F., Keriven, R., Brédif, M., and Vu, H. (2013). A hybrid multiview stereo algorithm for modeling urban scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(1):5–17. (page 28, 60, 90, 91, 93)

[53] Li, M., Wonka, P., and Nan, L. (2016). Manhattan-world urban reconstruction from point clouds. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 27, 86, 87, 88, 89, 90, 91, 92)

[54] Li, Y., Wu, X., Chrysanthou, Y., Sharf, A., Cohen-Or, D., and Mitra, N. J. (2011). Globfit: consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics*, 30(4):52:1–52:12. (page 26)

[55] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440. (page 62)

[56] Lorensen, W. and Cline, H. (1987). Marching cubes: A high resolution 3d surface reconstruction algorithm. In *ACM Trans. on Graphics (SIGGRAPH)*. (page 19)

[57] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of International Conference on Computer Vision (ICCV)*. (page 13)

[58] Mapillary (2018). Opensfm - open source structure from motion pipeline. `https://github.com/mapillary/OpenSfM`. (page 15)

[59] Matterport (2018). `ttps://matterport.com`. (page 4)

[60] Maurer, M., Hofer, M., Fraundorfer, F., and Bischof, H. (2017). Automated inspection of power line corridors to measure vegetation undercut using uav-based images. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. (page 62)

[61] Monszpart, A., Mellado, N., Brostow, G., and Mitra, N. (2015). RAPter: Rebuilding man-made scenes with regular arrangements of planes. *ACM Trans. on Graphics (SIGGRAPH)*. (page 27, 86, 89, 90, 91, 92)

[62] Mostegel, C., Prettenthaler, R., Fraundorfer, F., and Bischof, H. (2017). Scalable surface reconstruction from point clouds with extreme scale and density diversity. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 108)

[63] Mostegel, C. and Rumpler, M. (2012). Robust Surface Reconstruction from Noisy Point Clouds using Graph Cuts. Technical report, Graz University of Technology, Institute of Computer Graphics and Vision. `https://www.tugraz.at/institute/icg/Media/mostegel_2012_techreport`. (page 20, 21, 22, 56, 86)

[64] Moulon, P., Monasse, P., and Marlet, R. (2013). Global fusion of relative motions for robust, accurate and scalable structure from motion. In *Proceedings of International Conference on Computer Vision (ICCV)*. (page 15)

[65] Moulon, P., Monasse, P., Marlet, R., and Others (2018). Openmvg. an open multiple view geometry library. `https://github.com/openMVG/openMVG`. (page 15)

[66] Nan, L. and Wonka, P. (2017). Polyfit: Polygonal surface reconstruction from point clouds. In *Proceedings of International Conference on Computer Vision (ICCV)*. (page 26, 73, 74, 76, 77)

[67] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '11, pages 127–136, Washington, DC, USA. IEEE Computer Society. (page 23)

[68] Nister, D. (2004). An efficient solution to the five-point relative pose problem. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*. (page 11, 12)

[69] Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 13)

[70] Oesau, S., Lafarge, F., and Alliez, P. (2014). Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. *ISPRS Journal of Photogrammetry and Remote Sensing*, 90:68–82. (page 27)

[71] Oesau, S., Lafarge, F., and Alliez, P. (2016). Planar shape detection and regularization in tandem. *Computer Graphics Forum*, 35(1):203–215. (page 26)

[72] OpenCV (2019). `https://opencv.org/`. (page 71)

[73] PASCAL-Context network (2018). `http://dl.caffe.berkeleyvision.org/pascalcontext-fcn32s-heavy.caffemodel`. (page 71)

[74] Paul Chew, L. (1989). Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108. (page 39)

[75] Pix4DModel (2018). `https://pix4d.com/`. (page 2, 15, 72)

[76] Pollefeys, M., Nistér, D., Frahm, J., Akbarzadeh, A., Mordohai, P., Clipp, B., Engels, C., Gallup, D., Kim, S. J., Merrell, P., Salmi, C., Sinha, S. N., Talton, B., Wang, L., Yang, Q., Stewénius, H., Yang, R., Welch, G., and Towles, H. (2008). Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision (IJCV)*, 78(2-3):143–167. (page 29)

[77] Prisacariu, V. A., Kahler, O., Cheng, M. M., Ren, C. Y., Valentin, J., Torr, P. H. S., Reid, I. D., and Murray, D. W. (2014). A Framework for the Volumetric Integration of Depth Images. *ArXiv e-prints*. (page 23)

[78] Richard, A., Vogel, C., Blaha, M., Pock, T., and Schindler, K. (2017). Semantic 3d reconstruction with finite element bases. In *Proceedings of British Machine Vision Conference (BMVC)*, volume 28. (page 30)

[79] Rothermel, M., Wenzel, K., Fritsch, D., and Haala, N. (2012). Sure: Photogrammetric surface reconstruction from imagery. In *Proceedings LC3D Workshop, Berlin*. (page 15, 35, 68, 69, 79, 83, 85)

[80] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Proceedings of International Conference on Computer Vision (ICCV)*. (page 13)

[81] Sanchez, V. and Zakhor, A. (2012). Planar 3d modeling of building interiors from point cloud data. In *ICIP*, pages 1777–1780. IEEE. (page 26)

[82] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226. (page 25, 26, 27, 49, 94, 95, 96, 97, 108)

[83] Schönberger, J. L. and Frahm, J.-M. (2016). Structure-from-motion revisited. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 2, 15)

[84] Schönberger, J. L., Zheng, E., Pollefeys, M., and Frahm, J.-M. (2016). Pixelwise view selection for unstructured multi-view stereo. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 16)

[85] Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring image collections in 3d. In *ACM Trans. on Graphics (SIGGRAPH)*. (page 15)

[86] Strecha, C., von Hansen, W., Gool, L. V., Fua, P., and Thoennessen, U. (2008). On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 90, 93, 95, 97, 98, 99, 101)

[87] Sweeney, C. (2018). Theia multiview geometry library: Tutorial & reference. `http://theia-sfm.org`. (page 15)

[88] Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer Science and Business Media. (page 2)

[89] Cgal. Computational Geometry Algorithms Library (2018). `http://www.cgal.org`. (page 71, 94)

[90] The VCG Library (2018). `http://vcg.isti.cnr.it/vcglib/`. (page 103)

[91] Verdie, Y., Lafarge, F., and Alliez, P. (2015). LOD generation for urban scenes. In *ACM Transactions on Graphics*. (page 29)

[92] Visual Computing Lab, Istituto di Scienza e Tecnologie dell'Informazione, N. R. C. o. I. (2018). MeshLab, an open source, portable, and extensible system for the processing and editing of unstructured 3D triangular meshes. . `http://www.meshlab.net/`. (page 68, 75)

[93] Waechter, M., Moehrle, N., and Goessele, M. (2014). Let there be color! - large-scale texturing of 3d reconstructions. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 35, 46, 79, 82, 83, 86, 89, 94)

[94] Xiao, J. and Furukawa, Y. (2014). Reconstructing the world's museums. *International Journal of Computer Vision (IJCV)*, 110(3):243–258. (page 27)

[95] Zach, C. (2008). Fast and high quality fusion of depth maps. In *Proceedings of International Symposium on 3D Data Processing, Visualization, and Transmission*. (page 38)

[96] Zebedin, L., Bauer, J., Karner, K. F., and Bischof, H. (2008). Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *Proceedings of European Conference on Computer Vision (ECCV)*. (page 26)

[97] Zeng, M., Zhao, F., Zheng, J., and Liu, X. (2012). A memory-efficient kinectfusion using octree. In Hu, S.-M. and Martin, R. R., editors, *Computational Visual Media*, pages 234–241, Berlin, Heidelberg. Springer Berlin Heidelberg. (page 23)

[98] Zheng, E., Dunn, E., Jojic, V., and Frahm, J.-M. (2014). Patchmatch based joint view selection and depthmap estimation. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. (page 16)

[99] Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. S. (2015). Conditional Random Fields as Recurrent Neural Networks. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 1529–1537. arXiv: 1502.03240. (page 63)

[100] Zhou, Q. and Neumann, U. (2012). 2.5d building modeling by discovering global regularities. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 326–333. (page 29)