CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY

PhD THESIS

ISSN 1213-2365

Large Scale Surface Reconstruction based on Point Visibility

Michal Jančošek

jancom1@cmp.felk.cvut.cz

CTU–CMP–2014–13

August 22, 2014

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/jancosek/Jancosek-TR-2014-13.pdf

Thesis Advisor: Ing. Tomáš Pajdla, Ph.D.

This works has been supported by EU FP7 grant PRoViDe FP7-SPACE-2012-312377, EC project FP7-SPACE-218814 ProVisG, EC project FP7-SPACE-241523 PRoViScout, Czech Government under the research program MSM6840770038, Grant Agency of the CTU under project CTU0908713, TACR grant ATOM TA02011275, and SGS grant No SGS10/186/OHK3/2T/13

Research Reports of CMP, Czech Technical University in Prague, No. 13, 2014

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Large Scale Surface Reconstruction based on Point Visibility

A Dissertation Presented to the Faculty of the Electrical Engineering of the Czech Technical University in Prague in Partial Fulfillment of the Requirements for the Ph.D. Degree in Study Programme No. P2612 - Electrotechnics and Informatics, branch No. 3902V035 - Artificial Intelligence and Biocybernetics, by

Michal Jančošek

Prague, August 2014

Supervisor: **Ing. Tomáš Pajdla, Ph.D.**

Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
fax: +420 224 357 385, phone: +420 224 357 465
http://cmp.felk.cvut.cz

Abstract

The problem of 3D surface reconstruction from calibrated images or laser-scans is well studied in computer vision. There are two main sub-problems that are studied. It is the problem of depth-map computation from images and the problem of depth-maps fusion. In this work, we present several contributions to both sub-problems.

First, we present methods that contribute to the sub-problem of depth-map computation from images. In particular we describe an effective seed construction method for 3D reconstruction which starts with initial estimates of seed position, improves them and computes good estimates normals. Next, to avoid searching for optimal surface position and orientation based on nondiscriminative texture, we (over)segment images into segments of low variation of color and intensity and use each segment to generate a candidate 3D planar patch explaining the underlying 3D surface. We use the effective seed construction method to improve the candidate 3D planar patch. The method further improve, filter and combine the 3D planar patches to produce the resulting 3D surface reconstruction. Finally, we present a scalable multi-view stereo reconstruction method which can deal with a large number of large unorganized images in affordable time and effort. The computational effort of our technique is a linear function of the surface area of the observed scene which is conveniently discretized to represent sufficient but not excessive detail. Our technique works as a filter on a limited number of images at a time and can thus process arbitrarily large data sets using limited memory.

Second, we present methods that contribute to the sub-problem of depth-maps fusion. We compute input points augmented with visibility information from the input depth-maps. We observe that it is even possible to reconstruct a surface that does not contain input points. Instead of modeling the surface from input points, we model free space from visibility information of the input points. The complement of the modeled free space is considered as full space. The surface occurs at interface between the free and the full space. We show that under certain conditions a part of the full space surrounded by the free space must contain a real object also when the real object does not contain any input points, i.e., an occluder reveals itself through occlusion. Our key contribution is the proposal of a new interface classifier that can also detect the presence of an occluder in the scene just from the visibility of input points. To be practical, we assume that the occluder surface contains a reasonable number of input points in order to be able to approximately reconstruct the occluders surface i.e., weakly-supported surface. We use the interface classifier to modify a state-of-the-art surface reconstruction method so that it gains the ability to reconstruct weakly-supported surfaces.

Finally, we present methods that contribute to the problem of hallucinations removal from 3D surface reconstruction and to the problem of globally optimal large-scale 3D surface reconstruction by parts.

Acknowledgements

I would like to express my thanks to my colleagues at CMP who I had the pleasure of working with, especially to Alexander Shekhovtsov, Michal Havlena, Jan Heller, Zuzana Kukelova, and Martin Bujnak for their collaborative efforts. I am greatly indebted to Tomas Pajdla for guiding me throughout my research. His friendly support, patience and theoretical and practical help have been paramount to the successful completion of my PhD study. I also would like to thank to my family and to my friends for all their support that made it possible for me to finish this thesis.

This works has been supported by EU FP7 grant PRoViDe FP7-SPACE-2012-312377, EC project FP7-SPACE-218814 ProVisG, EC project FP7-SPACE-241523 PRoViScout, Czech Government under the research program MSM6840770038, Grant Agency of the CTU under project CTU0908713, TACR grant ATOM TA02011275, and SGS grant No SGS10/186/OHK3/2T/13.

Contents

Keywords: large scale, surface reconstruction, 3D seconstruction, multi-view stereo, weakly-supported surfaces, hallucinations removal, depth-map fusion, plane-sweeping

List of Algorithms

Notation

h	scalar value
$\mathbf{h}$	column vector
$\mathtt{M}$	matrix
$f(.), \mathbf{f}(.), \mathtt{F}(.)$	functions which map domain "." to scalars, vectors or matrices, respectively.
$f : . \rightarrow ..$	function which maps domain "." to domain ".."
$\overrightarrow{(\mathbf{c}, \mathbf{p})}$	oriented vector from the point $\mathbf{c}$ to the point $\mathbf{p}$
H	set
$\lvert H \rvert$	number of elements in set H
$H = \{x \lvert .\}$	set H of elements x such that condition . holds
$\mathtt{M}(i, j)$	$[i, j]$-th element of matrix $\mathtt{M}$
$\mathbf{h}_j$	j-th element of vector $\mathbf{h}$
$\lVert \mathbf{h} \rVert = \sqrt{\sum_i h_i^2}$	Euclidean norm of vector $\mathbf{h}$
$\langle a, b \rangle$	closed interval of values between a and b

Commonly used symbols and abbreviations

SFM	Structure from Motion
MVS	Multi-View Stereo
SSD	Sum of Square Differences
NCC	Normalized Cross Correlation
WSS	Weakly Supported Surface
SSS	Strongly Supported Surface
DT	Delaunay triangulation
GC	Graph cut
DG	Discretization graph
TSDF	Truncated signed distance function
MRF	Markov Random Field

1 Introduction

The problem of surface reconstruction from input 3D points is well studied and different solutions are widely used in many industries. The demand for high quality 3D content is growing rapidly in recent years thanks to growing markets with 3D-printers, UAVs, smartphones, cameras and other gadgets. Image and depth data produced by cameras and depth-sensors can be used to create high quality 3D content.

The input 3D points can be obtained using different techniques. Laser-scanners and structured-light based systems along with a registration method are probably the most widely used as sources of input 3D points. Recently, images processed with a Structure-from-Motion (SFM) method [29, 72, 90, 89] and a Multi-View-Stereo (MVS) method [68, 77, 13, 64, 36, 74, 75, 84, 22, 67] have become another popular source of input 3D points.

Given the input 3D points, different methods [38, 52, 34, 94, 14, 63, 20, 25, 64] are used to create a different output 3D content. It can be filtered and grouped 3D point cloud, triangular 3D mesh, ortho-photo, digital-elevation map etc.

However, producing complete reconstructions of outdoor and complicated scenes is still an open problem. Moreover, the processing of large amount of such data can take long time also on nowadays powerful PC. Therefore, there is still need of algorithms that can both speedup the computation time and increase accuracy and completeness of surface reconstructions.

2 Contribution of the thesis

The thesis deals with the problem of surface 3D reconstruction from images or from a given 3D point cloud augmented with sensors-to-point visibility information. The main contributions of the thesis are:

Effective seed generation for 3D reconstruction [39]. We focus on studying photo-consistency measures. We study criterial functions based on the similarity of reprojection of images on a circular planar patch (seed). We discover that the criterial function is unimodal in a certain domain part and all criterial function values of arguments from this domain part are greater than all criterial function values of arguments outside of this domain part. We propose a new globally optimal method for effective seed generation for 3D reconstruction that is based on the discovered property. The ability to estimate seeds depends on a discriminativity of a surface texture. The method is also able to detect situations when a seed orientation is not possible to detect, e.g., because the texture is not discriminative. See **Chapter 4**.

Segmentation based Multi-View Stereo [40]. The ability to reconstruct a 3D surface from images depends on a texture discriminativity. Therefore, we (over)segment image into discriminative segments. Instead of using circular planar patches (as in Chapter 4), we use nonuniform shapes of the patches given by an image segments. Following this idea, we have proposed a novel multi-view segmentation based method [40]. See **Chapter 5**.

Scalable Multi-View Stereo [45]. The method proposed in the Chapter 5 is able to compute complete depth-maps fast. However, in our previous work [40] (Chapter 5) we used Poisson surface reconstruction [47] to generate the final 3D mesh. Now we develop our own solution to final 3D mesh generation. More specifically, we propose a scalable method that can compute the final 3D mesh in large-scale scenarios. The contribution is that the computational effort of our technique is a linear function of the surface area of the observed scene which is conveniently discretized to represent sufficient but not excessive detail. See **Chapter 6**.

Background. Work presented in Chapter 6 is scalable but has two major disadvantages (i) it does not produce watertight mesh and (ii) it is rather conservative since it reconstructs just surfaces that are strongly supported by input data. Therefore, we have decided to study volumetric-based methods that model observer-to-surface visibility in

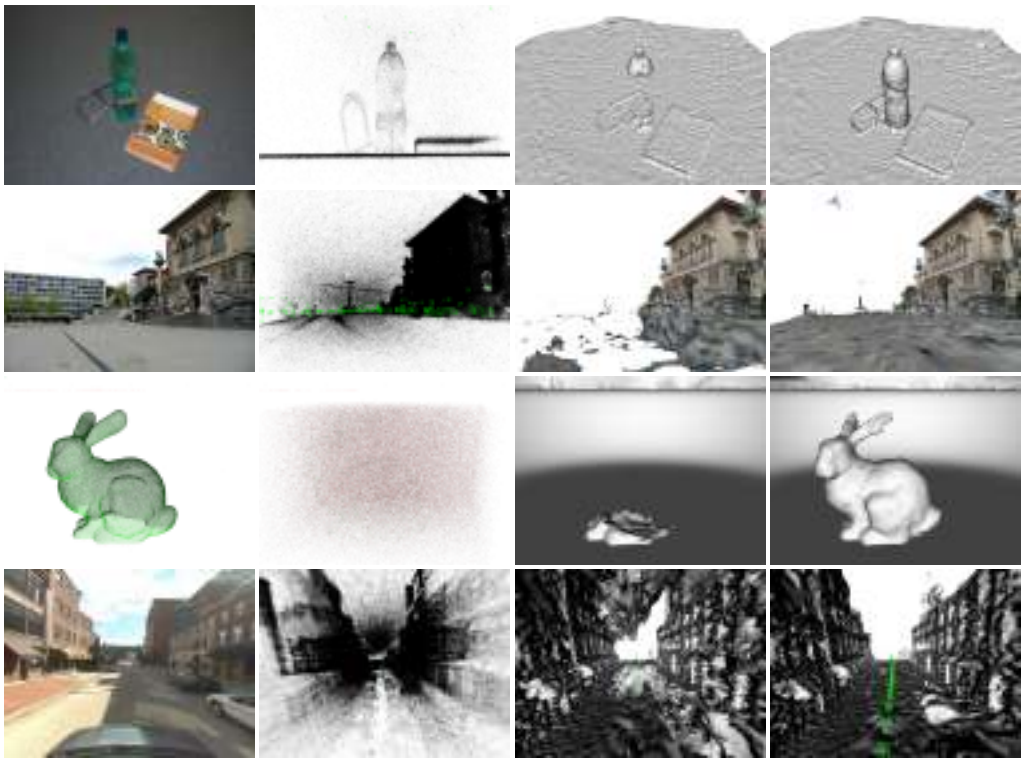


Figure 2.1.: **Weakly supported surfaces.** Columns from left to right: Input images. Input points. Results produced by our implementation of Labatut's method proposed in [52]. Results produced by our method proposed in [41, 38].

order to reconstruct object surface. We have realized that tetrahedral representation of volume is better than voxel based representations. The tetrahedral one better adapts to the distribution of input points and can fit into memory much larger portion of space. We have further focused on volumetric (tetrahedral) approach pioneered by Labatut et.al., proposed in [52]. It is, in our opinion, the best approach to 3D reconstruction because (i) it models observer-to-surface visibility (ii) it solves the problem globally by a global optimization method. Therefore, we have implemented the approach and based on experimental results on many real-world datasets we have figured out that it has the following disadvantages:

- It does not reconstructs weakly-supported surfaces. We have been visually browsing a lot of input point-clouds and output meshes. We consider input points that do not represent a real surface as outliers and points that represent a real surface as inliers. We have observed that besides that the input point cloud contained a surface, the surface was not reconstructed. This happened mainly in situations

when the density and amount of outliers was at least the same as the density and amount of inliers. We call such surfaces weakly-supported surfaces, see Figure 2.1.

- It produces hallucinations. We have also observed that the method produces surfaces that do not represent a real surface i.e. hallucinations, see Figure 2.1 (row 4, column 3) for illustration (the sky is hallucinated).

- It is not scalable. Another big disadvantage is that the method needs all data to fit into the computer memory in order to be able to run the global optimization process. Nowadays computers have tens of gigabytes of RAM memory and the tetrahedral volume representation can fit a relatively large space into the memory. Nevertheless, for city-scale reconstructions the data that need to be processed do not fit into the memory.

Therefore the next main contributions of the thesis are:

Exploiting visibility information in surface reconstruction to preserve weakly-supported surfaces [41, 38]. We present a novel method for 3D surface reconstruction from an input cloud of 3D points augmented with sensor-to-point visibility information. We observe that it is possible to reconstruct surfaces that do not contain input points. Instead of modeling the surface from the input points, we model a free space from the visibility information of the input points. The complement of the modeled free space is considered as full space. The surface occurs at the interface between the free and the full space. We show that under certain conditions the full space surrounded by the free space must contain a real object also when the real object does not contain any input points, i.e., an occluder reveals itself through occlusion. Our key contribution is the proposal of a new interface classifier that can also detect the occluder interface just from the visibility of input points. We use the interface classifier to modify a state-of-the-art surface reconstruction method so that it gains the ability to reconstruct weakly-supported surfaces. See **Chapters 7, 8, and 9**.

Hallucination-free Multi-View Stereo [44]. We present a novel method for 3D reconstruction that does not produce hallucinations. The method is based on the observation that hallucinated surfaces are unstable under small input variations. The method can detect and remove such surfaces. See **Chapter 10, and Section 10.1**.

Hallucination-free Multi-View Stereo using Priors [43]. The method proposed in the Section 10.1 works well. However, it has the following disadvantages (i) it is relative slow since it must run the global optimization several times, (ii) it does not preserve weakly-supported surfaces. It is sometimes impossible to distinguish between hallucinated and weakly-supported surfaces just from input visibility-augmented point cloud. We explain it in Section 7.1. We found that we need to use a prior knowledge in order to be able to distinguish between a hallucinated and weakly-supported surface. Therefore, we propose

a new method for hallucinations removal that is fast and can consider a prior knowledge of the scene. See **Chapter 10, and Section 10.2**.

Scalability [42]. We propose a novel semi-global method for 3D reconstruction by parts. We divide scene into parts. Instead of running the global optimization for all input data, we run it on each part separately. However, we propagate important information from neighboring parts in order to be able to achieve semi-globally optimal solution. See **Chapter 11**.

Experiments: Kinect vs photogrammetry [70]. We have joined all methods proposed in Chapter 9, Section 10.2, and Chapter 11 into one pipeline called CMPMVS as a result of our research. Work presented in [70] contributes with comparison of photogrammetry results with results obtained from input depth-maps produced by Microsoft Kinect depth camera. See **Chapter 12**.

2.1. Authors Contributions not Included in the Thesis

Several contributions were proposed by the author. However they were left out of the thesis to keep the thesis more homogeneous and focused.

- Dynamic 3D Scene Analysis from Omni-Directional Video Data [80].
- AWEAR 2.0 System: Omni-directional Audio-Visual Data Acquisition and Processing [30].
- Stereoscopic Imaging Project Summary Report [37].
- Automatic Reconstruction of Mars Artifacts [33].
- 3D Surface Models from Opportunity MER NavCam [31].
- Digital Elevation Modeling from Aerobot Camera Images [32].

3 State of the Art

In this chapter, we summarize the state-of-the-art of the MVS methods. We focus on the methods that are related to our contributions. Figure 3.1 generalizes structure of most of the methods.

In Section 3.1, we summarize the state-of-the-art methods that are focused on depth-maps computation.

In Section 3.2 we focus on methods that contribute to depth-maps fusion part.

3.1. Depth-Maps computation

The depth-map can be computed from images or can be obtained from a depth-sensor like Microsoft Kinect. To compute depth-map from images the most commonly used approaches are a (two-view) stereo method or a MVS method, for example methods proposed in [20, 67, 36, 13, 64, 75, 74]. Traditional two-view stereo methods [76, 10, 67] first rectify images and then compute depth-map by comparing similarity of small surrounding of points (5x5 pixels) on horizontal epipolar lines. Multi-view plane sweeping based methods [13, 87, 4] sweep a plane to discover the intersections of the plane with a scene. Multi-view growing based methods [96, 27, 21, 93] first detect a seed and then grow it according to photoconsistency. Our contributions falls to the group of multi-view growing based methods.

3.1.1. Growing based methods

Multi-view growing based methods [21, 93, 96, 27] share the same approach. The 3D reconstruction process starts by surface growing from initial seeds represented by a 3D

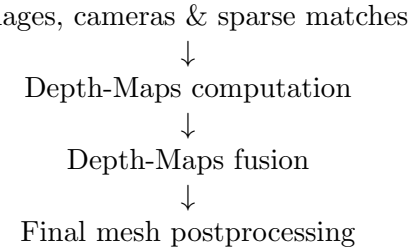


Figure 3.1.: **General reconstruction pipeline.**

point and a normal. The quality of this 3D reconstruction depends on the quality of seeds. Therefore, it is necessary to start with a good estimate of seeds. Initial positions can often be obtained from guided matching [21] or as centers of voxels [92]. Improved positions and normals need to be computed. In Chapter 4, as in [39], we propose method that can compute the seed position and orientation in an efficient way.

A point cloud based approach to seed detection was used in [96]. First, a 3D point cloud is obtained using Lhuillier and Quan's technique proposed in [54]. Next, to compute normals to the surface for every scene point, a plane is fitted to nearest neighboring points in 3D. This approach can fail when the point cloud is not dense enough or when there are many outliers.

In the work proposed in [27], the problem of seed detection is solved by 2D matching approach closely related to motion estimation and image alignment, a research field pioneered by Lucas and Kanade [58]. Unlike [27], our approach to the problem of seed construction (proposed in Chapter 4) is similar to approaches proposed in [21, 93].

In the work proposed in [93], a small planar patches are swept in 3D and exhaustive multiresolution search is used to detect seeds.

In the work proposed in [21], a small planar patches are swept in 3D and optimize their positions and orientations photometrically using conjugate gradient method.

In **Chapter 4**, as in [39], we propose a method that has advantages of the two methods proposed in [93, 21]. In particular, the method proposed in [93] often finds the globally optimal solution, since it uses multiresolution exhaustive search. On the other hand, this method needs to evaluate the objective function many times. In our implementation of the method proposed in [93], we observed good results with 500 evaluations of the objective function (100 evaluations for each of the 5 resolutions).

In the method proposed in [21], a conjugate gradients optimization technique is used to find a local optimum of a photoconsistency based objective function. This method does not need to evaluate the objective function so many times as in the method proposed in [93]. In our implementation of the method proposed in [21], we use gradient descent instead of conjugate gradients. It follows from our experiments, that if it finds globally optimal solution, it does 15 evaluations of the objective function, on average. However the method proposed in [21] often gets stuck in a local optimum.

Our method proposed in **Chapter 4**, as in [39], finds the globally optimal solution as the method proposed in [93], but with substantially smaller number of evaluations of the objective function. We show that the criterial function is unimodal in a certain part of range and additionally all values in this part are greater than the values outside of this part. Thus, we can perform exhaustive search only on the coarsest resolution and then we use the gradient descent optimization to refine the solution. Therefore, our method needs on average only $100 + 15$ evaluations of the objective function in order to be able to find globally optimal solution.

In the method proposed in [27], a variance of intensities of a seed texture is evaluated to decide whether is it possible to determine the surface normal uniquely by comparing image reprojection errors. We show that there also exist nonhomogenous seed textures

for which it is not possible to determine the surface normal uniquely and our method is able to detect them.

3.1.2. Segmentation based methods

In **Chapter 5**, as in [40], we propose a multi-view stereo reconstruction method that can deal with nondiscriminative texture in very homogeneous image areas and that process large images in affordable time.

Our method was motivated by color segmentation based stereo method [78]. This approach was originally designed to achieve the following goals: (1) enforcing global visibility constraints, (2) obtaining reliable depths for depth boundaries and thin structures, (3) obtaining correct depths for textureless regions, and (4) hypothesizing correct depths for unmatched regions. We adapt this approach to multi-view stereo and show that it brings considerable benefits in the quality of results as well as in efficiency.

Many multi-view stereo algorithms cannot find correct depths in low-textured regions. Because of the matching ambiguity involved in these regions, the depth-maps created by picking the best matching score of small windows (usually 5×5 or 7×7 [21]) is usually not well determined. On the other hand, it has been observed in the method proposed in [39] (Chapter 4) that it is possible to obtain the globally optimal position and orientation of a circular 3D patch that approximates the scene surface near some initial 3D position with high success (94% of their test data) when the texture covered by the patch projection is informative. It often calls for large patch sizes in homogeneous image areas.

Work proposed in [78] advocates using image segmentation by assuming that regions with homogeneous color usually do not cover large depth discontinuities. We would rather say that regions with homogeneous intensity and color rarely contain large change in surface normal. Considering lambertian surfaces, the perceived intensity depends on the angle between the surface normal and the line pointing from the surface to the light source. Let us assume that the surface is close to planar, close to lambertian, and the light source is distant. Then, the corresponding textures in different images are related (at least approximately) by affine intensity transforms and thus their correlation coefficient is close to one. These assumptions are valid for large class of real scenes and if they are not valid (or not replaced by other valid assumptions), very little can be done. Therefore, we consider as reasonable to hypothesize that homogeneous image segments, as generated by, e.g., segmentation [17], were generated by 3D planar patches.

The new idea in **Chapter 5** is to generate candidates for 3D planar patches from image segments that are informative, i.e. to use as large image patches as to gather sufficiently informative texture for non-ambiguous evaluation of 3D planar patch consistency. To do so, we segment images [17] into regions with which extend as far as to reach considerable texture variation on region borders.

Method proposed in [61] also reconstructs 3D planar patches of image segments shapes. The method is focused on the reconstruction of urban environment that often consist of

planar surfaces with just several orientations. The method postulates the orientations and formulates the problem of the 3D reconstruction in MRF framework.

3.2. Depth-Maps fusion

We have reviewed the state-of-the-art methods that are focused on depth-map computation, and are related to our contributions in the previous section. In this section, we review the state-of-the-art methods that are focused on depth-map fusion part of the general pipeline, see Figure 3.1. The depth-map fusion takes depth-maps on the input and produces a triangular mesh or a volumetric representation of the scene on the output.

Depth-maps often contain many outliers and are noisy or incomplete, especially when computed from images. Some depth-map fusion methods [52, 34, 94] are designed to be robust against outliers and to produce complete, accurate and visibility-consistent results. Some of them [47] are focused to produce accurate and high quality results. Another depth-map fusion methods [14, 63] can produce high quality result fast however require many and relatively high-quality depth-maps.

3.2.1. Depth-Maps filtering methods

Depth-maps filtering methods [20, 25, 64] alternate the depth-map computation from images with depth-map filtering in order to obtain complete and outlier-less depth-maps. The depth-maps are later processed by an depth-map fusion method, usually the method proposed in [47], to produce the final surface 3D reconstruction.

Depth-map filtering methods proposed in [64, 21, 24, 20, 60] use the same principle to filter out bad depths. The principle is based on the consistency of visibility i.e. the space between the observer and observed surface is free. That means that if two depths from different cameras violate this principle then at least one of them must be removed. The removal is based on various heuristics, usually the more photoconsistent depth stays. The same principle is applied in depth-map filtering methods proposed in [9, 45] but instead of the removal heuristic the problem is formulated as global optimization labeling problem where different labels of a node are represented by different depths from neighboring depth-maps. In general, for depth-map filtering methods there holds that the depth for a pixel is selected from k most probable values obtained from neighboring depth-maps. On the other hand, depth for a pixel is selected from all reasonable space that projects to the pixel in our method.

Depth-map filtering-methods proposed in [24, 20] that deals with problems of scalability, distributability and variation in reconstruction quality has appeared in recent years. Both methods are focused on the reconstruction of large unorganized photo collections. They deal with the problem of optimal grouping of input images according to resolution, color intensity mutual position and orientation etc. The goal is to choose such a minimal subset of images that is best for surface 3D reconstruction, i.e., they remove

redundant images that do not increase the reconstruction quality. This is done on the level of images as well as on the pixel level.

In **Chapter 6**, as in [45], we build on previous works [60, 21, 25, 9]. We follow the reconstruction paradigm of [21] but with important improvements and modifications. In particular, we modify the reconstruction process to be scalable by accumulating partial scene reconstructions and avoiding unnecessary computations and we improve the filtering step by using MRF filtering formulation [9] but with a new optimization scheme that is scalable. We borrowed the 3D scene representation by points on rays of all cameras from [25] (which was also used in [21]) but we work with this representation in a more efficient way avoiding redundant processing of already well reconstructed structures. As in [9], we use the graph cut to recover meshes from 3D points independently reconstructed from image subsets but we developed a new implementation of an approximate graph cut for this problem that can process data locally with acceptable results. Some basic concepts and techniques are also similar to elements from [60, 94, 6].

3.2.2. Volumetric methods

Methods proposed in [14, 24, 63] incrementally accumulate values of a volumetric truncated signed distance function (TSDF) of the input depth-maps in the scene volume. Final surface is considered on the places of volume with zero value. Therefore these methods can produce high quality result fast but require many and relatively high-quality depth-maps.

Methods pioneered by [94] adopted the same approach as previous methods but introduced regularizations and are based on continuous optimization. Methods [52, 34] are based on the discrete s-t cut optimization. Volumetric depth-map fusion methods [14, 94, 34, 52, 41, 38] employ the same visibility consistency principle but in more robust and optimal way.

In Chapters **Chapters 7, 8, and 9**, as in [41, 38], we proposed new volumetric methods that are also able to reconstruct weakly-supported surfaces. We introduce new paradigm of weakly-supported surfaces in the next section and we review state-of-the-art methods with respect to the new paradigm.

Weakly supported surfaces

An input point cloud augmented with a visibility information can be computed from given depth-maps. Depth-maps can be obtained from images using an MVS method or a depth-sensor like Kinect, Laser-Scan etc. By visibility information of an input point we mean that 3D position(s) of sensor(s) that produce the input point are known. We consider input points that are close to the real surface but do not represent the real surface as noise, and input points that are far from the real surface as outliers.

Surfaces that do not have strong support in the input points but represent real surfaces in the scene, i.e, *weakly-supported surfaces*, are essential for achieving complete reconstructions. Such surfaces may be transparent, highly reflective, lacking in texture

or, as in the case of the ground planes, they can be occluded by moving objects such as people and cars. Weakly-supported surfaces are present especially when the input points were obtained from images processed by SFM and MVS methods. Frequent examples are the ground planes in tourist photo collections, which are often blurred, since the cameras are mostly focused on objects of interest (facades and landmarks) and on people above the ground. Additionally, input points can contain relatively large amounts of outliers. The density of outlier points is often similar or greater than the density of points of a weakly-supported surface. We propose a novel method that can reconstruct these difficult weakly-supported surfaces.

Here we review previous work that is most relevant to our methods proposed in Chapters **Chapters 7, 8, and 9**, as in [41, 38].

Silhouette based methods are mostly related to the idea of Visual hull that was introduced by Laurentini in [53]. This technique relies on the ability to clearly separate the scene objects from the background on input images. These objects silhouettes are used to infer shapes. A more robust approach exploiting the same idea is proposed in [19]. This approach introduces occupancy grid concept. It infers where and how likely the matter is present in the scene from given silhouette cues. The method proposed in [26] obtains moving occluder silhouette information from video sequences of static calibrated cameras using a background subtraction technique. A Bayesian sensor fusion formulation is used to process all occlusion cues in the multi-view sequence. The method proposed in [48] is similar to the method proposed in [26] but applies the state-of-the-art natural image matting algorithms for multi-view sequences to obtain occluder silhouette cues. The image matting information is later merged into a 3D coordinate system and a composite model is created. All these methods rely on silhouette cues information and use it to reconstruct occluders in the scene. Our methods build on another principle and do not need any occluder silhouette information.

Local depth-map filtering methods [20, 22, 25, 64, 79, 46] alternate depth-map computation from images with depth-map filtering in order to obtain complete and outlier-less depth-maps. Some of them [20, 22, 25, 64] use a visibility-based filtering approach. Another [46] use bundle adjustment minimization of the camera re-projection error to filter outliers and improve accuracy. The final filtered depth-maps are later processed by a depth-map fusion method, usually the method proposed in [47], to produce the final surface 3D reconstruction. The depth-map filtering methods are rather conservative and tend to reconstruct only those surfaces that are strongly supported by the input data. Unlike our methods, the local depth-map filtering methods leaves the space free where the support is weak.

Volumetric depth-map fusion methods have achieved great success in recent years. Depth-maps often contain many outliers and are noisy or non-complete, especially when computed from images. Some volumetric depth-map fusion methods [52, 34, 94] are designed to be robust angainst outliers and to produce complete, accurate and visibility-consistent results. Some of them [47] are focused to produce smooth and high quality results. Another volumetric depth-map fusion methods [14, 63] can produce high quality

result fast but require many and relatively high-quality depth-maps. Our methods falls to the group of robust volumetric depth-map fusion methods.

Space carving introduced in [50] produces an approximate reconstruction called Photo-Hull, which is, under very strict conditions, guaranteed to subsume all other photoconsistent reconstructions. The assumptions are so strict that the method is useless for reconstruction of outdoor scenes. It also does not reconstruct weakly-supported surfaces well since all non-photo consistent volume is carved out. Photo Flux was introduced in [8]. It is closely related to Photo Hull but allows the recovery of finer shape details without over-smoothing while still handling noise robustly. An approach to multi-view image based 3D reconstruction by statistically inversing the ray-tracing based image generation process was presented in [56]. The method proposed in [34] uses octree partitioning of a 3D space and computes the minimal s-t cut of a graph derived from the octree in order to label each voxel as being occupied (full) or free. The discontinuity cost used in [34] is based on photo consistency (or sensor depth-maps). All previously mentioned methods proposed in [50, 8, 56, 34] rely on 3D reconstruction of a visible, solid, well-textured, and static scenes from images. The photo-consistency of the scene surface plays an important role in all these methods while non-photo consistent parts are usually not reconstructed. These methods do not pay any particular attention to occluders and weakly-supported surfaces. Moreover, the methods using voxel-based volumetric representation of the space. Since the number of voxels depends on the resolution of the volume cubically, the computational and memory costs quickly reach machine limits. For this reason, the voxel-based volumetric methods are suited for reconstruction of small and compact objects only.

The methods proposed in [69, 52] work with Delaunay tetrahedralization of the input points instead of voxel-based volumetric representation. The tetrahedral-based volumetric representation suites better for large real-world reconstructions than voxel-based one. The size of tetrahedra depends just on the spatial distribution and density of input points. Therefore we have adopted the tetrahedral volumetric representation in our methods.

Pioneered by [52], our methods as well as the method proposed in [52] are based on a minimal s-t cut approach. In these methods, an s-t graph is derived from the Delaunay tetrahedralization of the input points in order to reconstruct the final surface. The minimal s-t cut of the graph labels each tetrahedron as "being full" with a full label or "being free" with a free label. The final surface consists of the faces shared by the tetrahedra labeled as being "free" (empty space) and the tetrahedra labeled as being "full" (full space). Therefore, the surface can be seen as an *interface* between the free and the full. The new element in our methods with respect to the method proposed in [52] is that they can reconstruct *weakly-supported surfaces*.

Hallucinations removal

Many MVS methods [22, 52, 41, 38] can produce surfaces that do not represent real surface, i.e. hallucinations.

To our knowledge, there seem to be few works that deal with hallucination removal. In [22, 52] the problem of hallucination removal is solved by removing large triangles. The problem with this approach is that hallucinated surfaces do not necessary contain large triangles, and hence it may not be possible to distinguish between weakly-supported surfaces and hallucinated surfaces using this approach.

In **Section 10.1**, as in [44], a confidence value is assigned to each tetrahedron based on the sensitivity to perturbations in the input 3D points. This approach works reasonably well though on the other hand it is computationally expensive since it needs to reconstruct the scene multiple times in order to compute the confidence value for each tetrahedron. In **Section 10.2**, we focus on a particular type of hallucinations and we propose a fast approach that can effectively remove them.

Large-scale reconstruction

Many volumetric based methods [69, 52, 34, 94, 41, 38] solve a global optimization problem and require therefore to have all data loaded in memory. However, we are working with too huge data to solve the problem globally. In **Chapter 11**, as in [42], we propose new semi-global distributed method that can reconstruct huge scenes by parts without the need of storing all data in the memory.

Part I.

Local filtering

4 Effective seed generation for 3D reconstruction

The ability of estimation a surface normal and 3D position for a given pixel of an image is a crucial for many Multi-View Stereo (MVS) methods. We call the estimated surface normal and 3D position the seed. Given a pair of calibrated cameras, we describe an effective seed construction method in this chapter. It starts with initial estimates of seed positions, improves them and computes good estimates of normals. We formulate the seed construction as an optimization problem with a criterial function based on the similarity of reprojection of images on a hypothetical circular planar patch.

The main contribution of this chapter is that we discover that the criterial function is unimodal in a certain domain part and all criterial function values of arguments from this domain part are greater than all criterial function values of arguments outside of this domain part.

4.1. Texture discriminativity vs. homogeneity

The ability to estimate seeds depends on the surface texture. Some methods evaluate the variance of intensities of seed texture to decide about the possibility of normal detection. We show that there also exist nonhomogenous textures which are nondiscriminative. Our method is able to detect situations when the seed normal is not possible to detect, i.e. the texture is nondiscriminative. Our method founds global optimum in 94% of our test data. We show in experiments, that our approach outperforms other most relevant approaches.

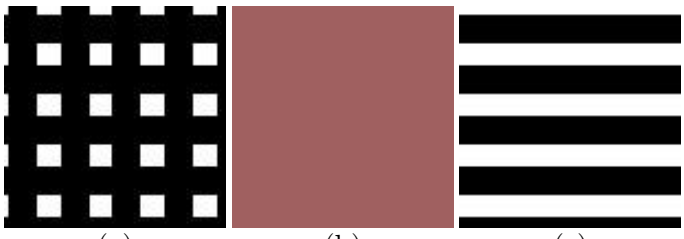


Figure 4.1.: **Three different texture types:** (a) discriminative, (b) nondiscriminative and homogeneous, (c) nondiscriminative and nonhomogenous.

Figure 4.1 shows three typical situations. Table 4.1 shows a comparison of the seed homogeneity and the range of our criterial function for three different textures from

Texture	Homogenity	Range size
Figure 4.1 (a)	24589	0.6
Figure 4.1 (b)	0.0	0.0
Figure 4.1 (c)	79856	0.0001

Table 4.1.: **Comparison of the seed homogeneity and criterial function (Eq. 4.6) range size for three different textures.** Seed homogeneity is the variance of intensities of image reprojection to the seed. We take an minimal homogeneity of the seed from the set of 180^2 seeds with uniformly distributed normals. The criterial function range size is the difference between maximal and minimal value of the criterial function evaluated for the same 180^2 uniformly distributed normals. The criterial function codomain is $\langle 0, 1 \rangle$.

figure 4.1. Seed homogeneity is the variance of intensities of image reprojection to the seed. In [27], they say that if the variance of intensities of seed pixels is bellow the threshold (equals to 4) then the seed is homogeneous. We use the minimal homogeneity of the seed from the set of 180^2 seeds with uniformly distributed normals (to be able to compare with [27]) as the threshold. The range of our criterial function equals to the difference between maximal and minimal value of the criterial function evaluated for the same 180^2 uniformly distributed normals. The criterial function codomain is $\langle 0, 1 \rangle$. If the criterial function range size is very small, all values of the criterial function are very similar, and therefore it is not possible to determine the surface normal uniquely with our approach. As one can see in the first row of the table 4.1, the homogeneity for the texture figure 4.1 (a) is high enough. The range of the criterial function is enough high, too. Therefore, with Habbecke's [27] and our approach too, it is possible to determine the surface normal uniquely. The second row shows that, the homogeneity for the texture figure 4.1 (b) is small. The criterial function range size is small, too. Therefore, with Habbecke's [27] and our approach too, it is not possible to determine the surface normal uniquely. On the other hand, the third row shows that the homogeneity for the texture figure 4.1 (c) is enough high, but the range of the criterial function is small. Therefore, this texture is nonhomogenous but it is not possible to determine the surface normal uniquely. Therefore, our approach is better than Habbecke's [27] because we can distinguish such situations.

4.2. Seed detection

Our approach to seed detection is based on the principle of backprojection. If there is a planar part of the scene surface visible in both images, then the image backprojections to this planar part of the surface will match.

Technically, the patch ring ρ, is the part of the plane between two circles with the same center $\mathbf{p}$ and normal $\mathbf{n}$. See figure 4.2 (b) which illustrates rings for three different normals. The ring is parametrized by point $\mathbf{p}$, normal $\mathbf{n}$, radius r_1 and radius r_2.

$$\rho(\mathbf{p},\mathbf{n}) = \{\mathbf{x} \in \mathbb{R}^3 | \mathbf{n}^\top(\mathbf{x}-\mathbf{p}) = 0 \,\&\, r_1 \leq \|\mathbf{x}-\mathbf{p}\| \leq r_2\} \tag{4.1}$$

We use $r_1 = 1/2\, r_2$. For the ring $\rho(\mathbf{p},\mathbf{n})$ we choose its coordinate system with the center in the point $\mathbf{p}$ and the z axis along the normal $\mathbf{n}$. The points of the ring $x \in \rho(\mathbf{p},\mathbf{n})$ have the coordinates $\mathbf{x_R} = (x_1, x_2, 0)^\top$ in this coordinate system of the ring. The coordinates of these points can be written in a global coordinate system as $\mathbf{x} = R(\mathbf{n})\,\mathbf{x_R} + \mathbf{p}$, where the rotation $R(\mathbf{n})$ is the function of the normal $\mathbf{n}$ of the ring ρ.

We define the grid with the step δ for the ring $\rho(\mathbf{p},\mathbf{n})$ as

$$S(i,j) = R(\mathbf{n})\,(i\delta, j\delta, 0)^\top - \mathbf{p} \tag{4.2}$$

with $i, j \in \mathbb{Z}$.

When the ring $\rho(\mathbf{p},\mathbf{n})$ is on a surface in the scene, then the image I_k is created by projecting the texture of the ring $\rho(\mathbf{p},\mathbf{n})$ to images. It is reasonable to reconstruct this texture $T_k(\mathbf{p},\mathbf{n})$ of the ring $\rho(\mathbf{p},\mathbf{n})$ from the image I_k as follows.

$$\begin{array}{l} T_k(\mathbf{p},\mathbf{n})(i,j) = \\ \begin{cases} f(\mathtt{P_k}\, S(i,j), I_k) & S(i,j) \in \rho(\mathbf{p},\mathbf{n}) \\ 0 & S(i,j) \notin \rho(\mathbf{p},\mathbf{n}) \end{cases} \end{array} \tag{4.3}$$

The matrix $\mathtt{P}_k$ is the projection matrix for image I_k. The function f represents the bilinear interpolation. If the images I_k and I_m were created as projections of the texture of the ring $\rho(\mathbf{p},\mathbf{n})$, then the textures $T_k(\mathbf{p},n)$ and $T_m(\mathbf{p},n)$ will match. Therefore, it is reasonable, for given images I_k, I_m and 3D point $\mathbf{p}$, to search for normals $\mathbf{n}^*$ which maximize the similarity of the textures $T_k(\mathbf{p},n)$ and $T_m(\mathbf{p},n)$

$$\mathbf{n}^* = \arg \max_{\mathbf{n},\|\mathbf{n}\|=1} (sim(T_k(\mathbf{p},\mathbf{n}), T_m(\mathbf{p},\mathbf{n})) \tag{4.4}$$

This is also known as photoconsistency. If the point $\mathbf{p}$ is not known, then we can formulate the problem as

$$\mathbf{n}^*, \mathbf{p}^* = \arg \max_{\mathbf{n},\|\mathbf{n}\|=1,\mathbf{p},\mathbf{p}\in\omega} (sim(T_k(\mathbf{p},\mathbf{n}), T_m(\mathbf{p},\mathbf{n})) \tag{4.5}$$

where the point $\mathbf{p}$ is from some reasonable set ω. Set ω can, for example, consist of uniformly distributed points along a vector $\mathbf{v}$ around the point $\mathbf{p}$. The vector $\mathbf{v}$ equals to $\hat{l}$ such that $\mathbf{l} = \mathbf{p} - (\mathbf{C_k} + \mathbf{C_m})/2$, where C_k, C_m are the k^{th} and m^{th} camera centers.

We have to point out that the high similarity of the textures $T_k(\mathbf{p},\mathbf{n})$ and $T_m(\mathbf{p},\mathbf{n})$ does not imply that the ring $\rho(\mathbf{p},\mathbf{n})$ is a part of a scene surface. It is often happens that textures of the surface generate very similar $T_k(\mathbf{p},\mathbf{n})$ and $T_m(\mathbf{p},\mathbf{n})$ for many different $\mathbf{p}$ near the surface and for many different $\mathbf{n}$. The most common example is a planar surfaces of a constant color. Then, the solution to equation 4.5 is not unique and $\mathbf{p}$ and $\mathbf{n}$ can't be determined.

In all our experiments we used adaptive size of the ring ρ. It means, that we computed r_2 separately for each point $\mathbf{p}$. The size r_2 varies with the distance from the cameras.

The goal is to set the size r_2 so that the $T_k(\mathbf{p}, \mathbf{n})$ and $T_m(\mathbf{p}, \mathbf{n})$ will not be oversampled or undersampled with respect to images I_k, I_m. The size r_2 of the ring is computed so, that, the projection of the ring to the image will cover approximately R^2 pixels (e.g. 14^2).

In all of our experiments we use MNCC [62] as the function *sim* to measure of the similarity of textures.

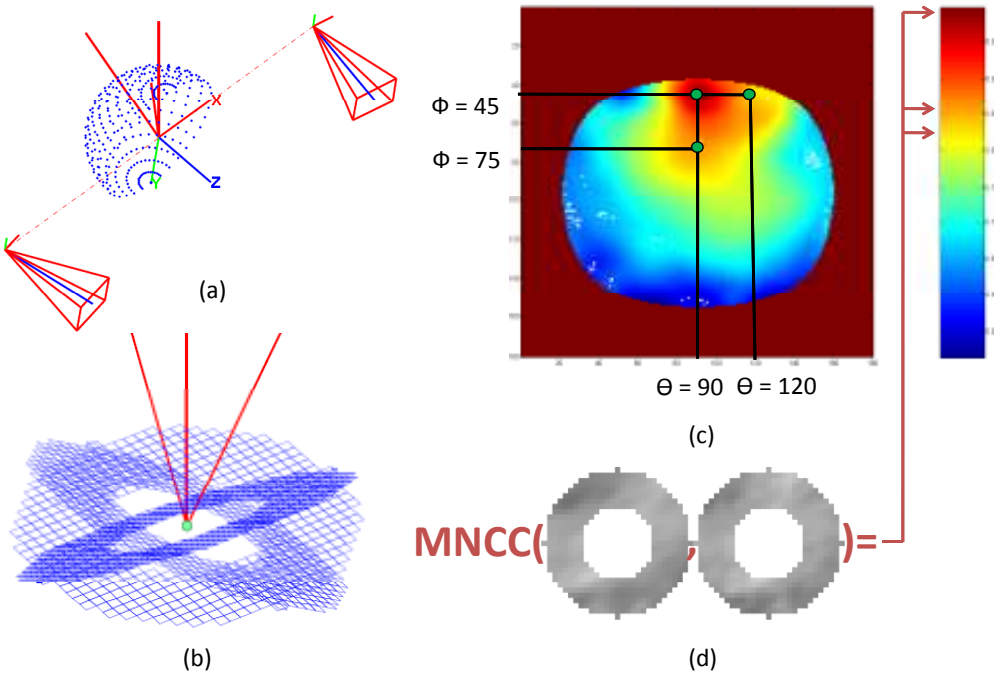


Figure 4.2.: **Criterial function** f_s**.**(a) Cyclopean eye coordinate system, (c) f_s function values. (b) rings for three different normals, (d) $T_k(\mathbf{p}, \phi, \theta), T_m(\mathbf{p}, \phi, \theta)$ for one of the normals.

4.3. Seed normal detection

We assume a given calibrated stereo pair $I_k, I_m, \mathtt{P_k}, \mathtt{P_m}$ (the I means image and $\mathtt{P}$ means the corresponding projection matrix) and a point $\mathbf{p}$ in 3D. In this section, we further assume that the point $\mathbf{p}$ lies on a planar surface. The goal is to find the normal of the surface.

We formulate the problem as searching for the global maximum of the criterial function $f_s : \langle 0, 180\rangle \times \langle 0, 180\rangle \to \mathbb{R}$:

$$f_s(\phi, \theta) = sim(T_k(\mathbf{p}, \phi, \theta), T_m(\mathbf{p}, \phi, \theta)) \tag{4.6}$$

We parametrize normals by two angles (ϕ, θ) in spherical coordinates $\mathbf{n} = (cos(\phi)\, sin(\theta), sin(\phi)\, sin(\theta), cos(\theta))^\top$.

Figure 4.2 illustrates evaluation of criterial function f_s. The cyclopean eye coordinate system shown in figure 4.2 (a). The normal vector (ϕ, θ) is expressed in the cyclopean eye coordinate system. The rings for three different normals are shown in figure 4.2 (b). The textures $T_k(\mathbf{p}, \phi, \theta), T_m(\mathbf{p}, \phi, \theta)$ for one of the normals are shown in figure 4.2 (c). The three green dots in figure 4.2 (c) show the values of the function f_s corresponding to the three normals shown in figure 4.2 (b). The values are computed only for normals which correspond to surface normals that are visible on both cameras.

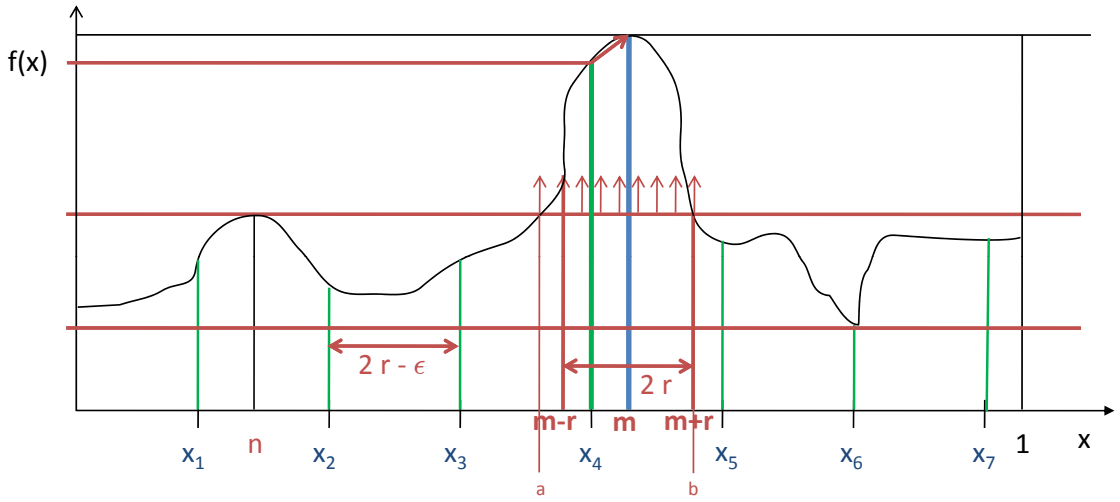


Figure 4.3.: **Global maximum detection for 1D r-unimodal function.** The r-unimodal f is defined on interval $\langle 0, 1\rangle$, with the global maximum in m. The distances between points x_i are $\|x_{i-1} - x_i\| = 2\,r - \epsilon$. The first point is $x_0 = 0$. Point x_4 is the point with the maximal value among all values $f(x_i)$ and therefore lies in $\langle m - r, m + r\rangle$. The interval $\langle a, b\rangle$ is the top-unimodal part of f.

4.3.1. Finding global maximum of r-unimodal function in 1D

In this section we introduce a method for finding global maximum of r-unimodal 1D function $f\colon \langle 0, 1\rangle \to \mathbb{R}$. A function f is unimodal on $\langle a, b\rangle \subseteq \langle 0, 1\rangle$ if for some value $m \in (a, b)$ (the mode), it is monotonically increasing for $x \in \langle a, b\rangle; x \leq m$ and monotonically decreasing for $x \in \langle a, b\rangle; x \geq m$. In that case, the maximum value of f(x) for $x \in \langle a, b\rangle$ is f(m) and there are no other local maxims. We define r-unimodality of the function as follows

Definition 1 ***Top-unimodal part****: The interval $\langle a, b\rangle$ is the top-unimodal part of the function $f : \langle 0, 1\rangle \to \mathbb{R}$ if:*

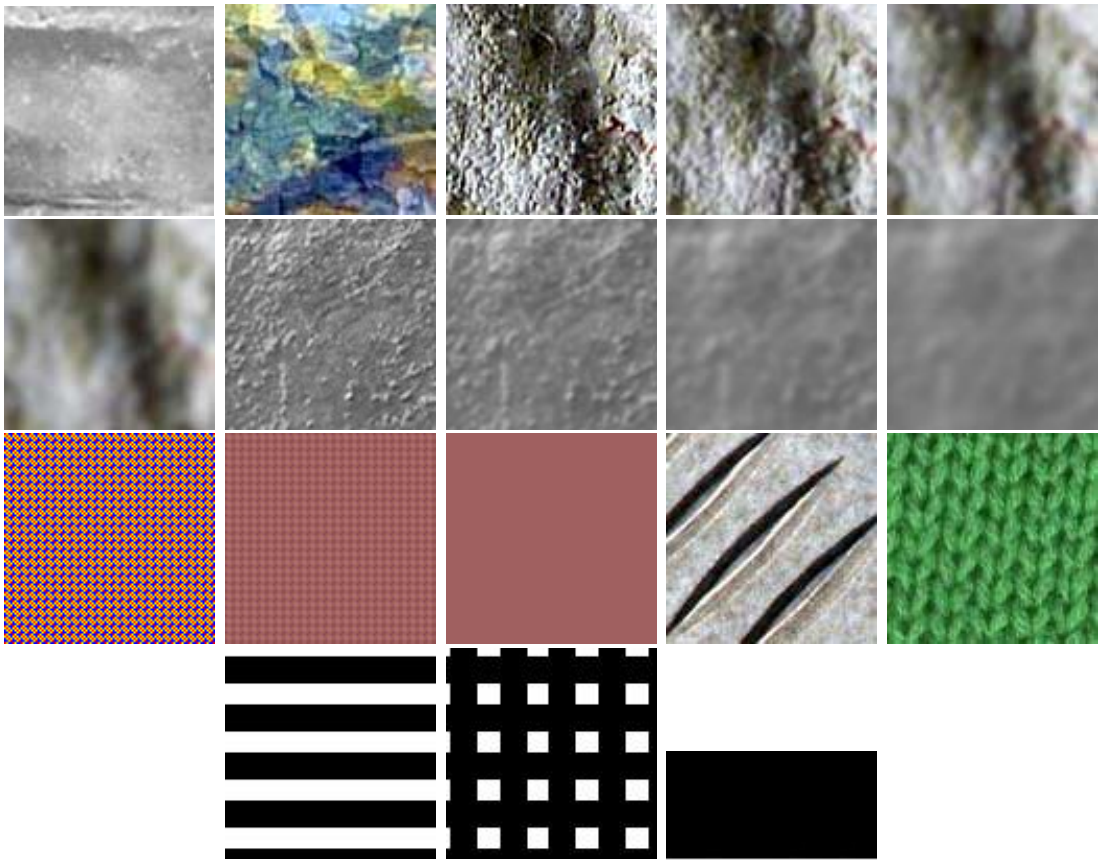


Figure 4.4.: **18 textures used in our experiments.**

- f *is unimodal on* $\langle a\, b\rangle$
- $\forall x \notin \langle a, b\rangle \wedge y \in \langle a, b\rangle; f(x) < f(y)$

Definition 2 ***r-unimodality****: The function* $f : \langle 0, 1\rangle \to \mathbb{R}$ *is r-unimodal for some* r *if* $\langle m - r, m + r\rangle \subseteq \langle a, b\rangle \subseteq \langle 0, 1\rangle$*, where* $\langle a, b\rangle$ *is some top-unimodal part of* f *and* m *is the global maximum of* f*.*

Any local optimization method starting from any point of the interval $\langle a, b\rangle$ will find the global maximum of any unimodal function defined on the interval $\langle a, b\rangle$. Let the function $f : \langle 0, 1\rangle \to \mathbb{R}$ be r-unimodal for some known r. We want to find the global maximum m. Let k be in interval $\langle m - r, m + r\rangle$ then any local optimization method starting from point k will find m. We divide interval $\langle 0, 1\rangle$ by points $x_i; \|x_{i-1} - x_i\| = 2\, r - \epsilon$ for some small ϵ (we call it the sampling step). Then, there is j such that x_j is in the interval $\langle m - r, m + r\rangle$. Next, we know from r-unimodality that $j = \arg\max_i\{f(x_i)\}$. The whole method is illustrated in Figure 4.3.

Instead of a local optimization method we can use the Binary or Fibonacci Search to find the global maximum. Then, for any r-unimodal function $f : \langle 0, 1\rangle \to \mathbb{R}$ we will be able to guarantee the total number of evaluation of f to find the global maximum with a chosen precision.

4.3.2. Properties of the criterial function f_s

Our method is guaranteed to work only when f_s is r-unimodal. Therefore, we need to show properties of the function f_s experimentally. We compute the function values in each point $\phi \in \{0, 1, ..., 179, 180\} \times \theta \in \{0, 1, ..., 179, 180\}$, for 18 different types of texture, see figure 4.4, and for different sizes of $R \in 7, 8, ..., 50$ of patch projections on a simulated scene. So for each texture and each R we have 180^2 values of f_s and we can compute the size of the r-unimodal region around the global maximum. The result of these experiment we suggest that function f_s is r-unimodal and in most of the cases the lower bound of r equals 9.

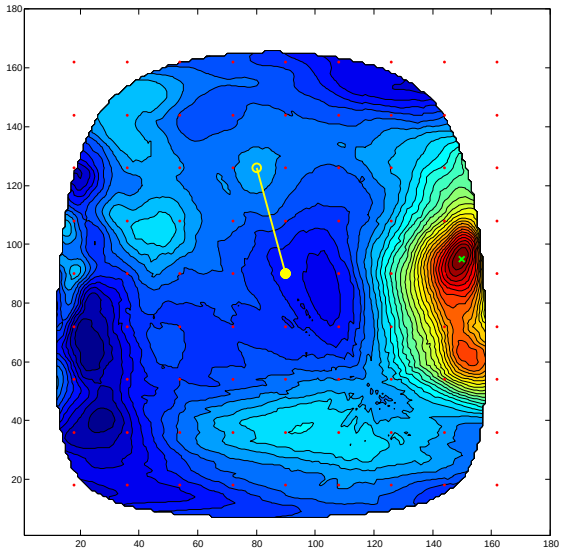


Figure 4.5.: **Criterial function f_s.** The yellow circle represents a local optimum of f_s obtained by a gradient descent starting from $(\phi, \theta) = (90, 90)$ marked by the yellow dot. The green cross represents the global optimum. Red dots represent uniformly distributed points with distance 18 degrees.

We observed that some textures are degenerate in the sense that the global maximum of f_s does not represent the normal to the scene surface. For these textures it is not possible to find the surface normal using our technique. We observed that these textures have the common property, the reconstructed textures $T_k(\mathbf{p}, \phi, \theta), T_m(\mathbf{p}, \phi, \theta)$ are very similar for all ϕ, θ, i.e. f_s, as s function of ϕ, θ is high everywhere, see table 4.1. Therefore, we can incorporate this observation to our algorithm. We simply take the minimal value of f_s over all ϕ, θ. If the difference of the maximum and the minimum of f_s is bellow some

threshold, we say that we are not able to find the normal. Otherwise, we assume that f_s is r-unimodal and the global maximum represents the normal to the scene surface.

Figure 4.5 shows the values of function f_s. The green cross shows the global maximum of f_s which corresponds to the ground truth. The yellow dot represents the normal for $(\phi, \theta) = (90, 90)$ this normal is known as the fronto-parallel normal. The yellow circle represents the nearest local maximum. Therefore the gradient descent and probably most other local optimization methods starting from $(\phi, \theta) = (90, 90)$ would stop in this incorrect point. On the other hand, our method can find the global maximum.

4.3.3. Algorithm

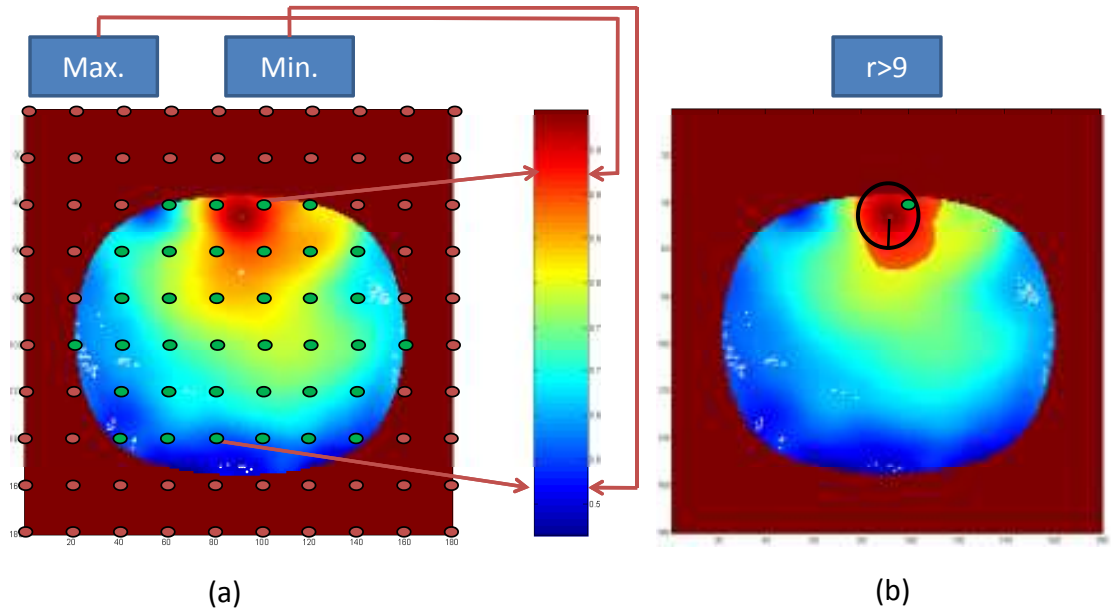


Figure 4.6.: **Global maximum detection of r-unimodal criterial function f_s.** (a) Values sampled with inter distance 18 degrees and minimal and maximal samples. (b) The top-unimodal part of criterial function f_s is highlighted part. The black circle is r-unimodal part of the criterial function f_s. The green dot is the maximal sample.

The method for finding the global maximum of a r-unimodal function defined on some interval (Section 4.3.1) can be easily extended to n-D, since the r-unimodality can be extended to n-D. Assuming that our function f_s is r-unimodal, we can use the method Algorithm 1 to find its global maximum. The figure 4.6 (a) illustrates the first part of Algorithm 1. It first takes the samples marked with red dots. Then, it finds the global maximum by the gradient ascent starting form the sample with the maximal value, see Figure 4.6 (b). There are two thresholds α, β : α for maximal value, β for the difference between maximal and minimal value from the samples. If maximal value of the criterial

Algorithm 1 Normal computation

Require: $I_k, I_m, \mathtt{P_k}, \mathtt{P_m}$ - calibrated stereo pair (I image, $\mathtt{P}$ corresponding projection matrix)

α, β - thresholds for max value of the final similarity, and max-min value from sampling, and the minimal value for homogeneity check

function $\mathbf{n} = findNormal(\mathrm{p})$
  $s_{max} = 0$; $s_{min} = 1$
  for all $\phi = 0$ to 180 step r **do**
    for all $\theta = 0$ to 180 step r **do**
      s $= f_s(\phi, \theta)$
      if $s > s_{max}$ **then**
        $s_{max} = s$
        $\phi_{max} = \phi$
        $\theta_{max} = \theta$
      end if
      if $s < s_{min}$ **then**
        $s_{min} = s$
      end if
    end for
  end for
  $\beta' = s_{max} - s_{min}$
  if $\beta' > \beta$ **then**
    (ϕ, θ, s) = do local optimization on parameters (ϕ, θ) of function $f_s(\phi, \theta)$ starting form $(\phi_{max}, \theta_{max})$ to find a local extrema (we use gradient descent algorithm)
    if $s < \alpha$ **then**
      return failed;
    else
      return ϕ, θ, s
    end if
  else
    return failed;
  end if
end function

function is lower than α, then textures are not similar. If the difference between the maximal and the minimal values from the samples is smaller than β, then the normal is not well determined.

4.4. Improving 3D position

The method explained in the previous section has one strong assumption. The assumption is that point **p** lies on a planar part of a scene. In this section, we relax this strong

assumption and make our algorithm to work in more general and more practical situations. We assume that point $\mathbf{p}$ lies near a planar part of the surface. The goal is to find the orientation and position of this planar part.

We formulate the problem as searching for a global maximum of the criterial function

$$\begin{gathered} f_d : \langle 0, 180\rangle \times \langle 0, 180\rangle \times \langle -K, K\rangle \to \mathbb{R} \\ f_d(\phi, \theta, k) = \\ sim(T_1(\mathbf{p} + \mathbf{v}\,\delta\,k, \phi, \theta), T_2(\mathbf{p} + \mathbf{v}\,\delta\,k, \phi, \theta)), \end{gathered} \tag{4.7}$$

where the feasible range of K is chosen discretization count of steps, K should be greater than R we use $K = 5\,R$, and δ is computed as

$$\delta = (r_2/2)/K \tag{4.8}$$

The step δ between $\mathbf{p} + \mathbf{v}\,\delta\,k$ and $\mathbf{p} + \mathbf{v}\,\delta\,(k+1)$ should be small enough to make the function f_d r-unimodal in r neighborhood of the position. The vector $\mathbf{v}$ equals to $\hat{l}$ such that $\mathbf{l} = \mathbf{p} - (\mathbf{C_k} + \mathbf{C_m})/2$, where $\mathbf{C_k}, \mathbf{C_m}$ are the k^th and the m^th camera centers.

The method based on r-unimodality principle works also for 3D function defined by eq. 4.7.

4.4.1. Properties of the criterial function f_d

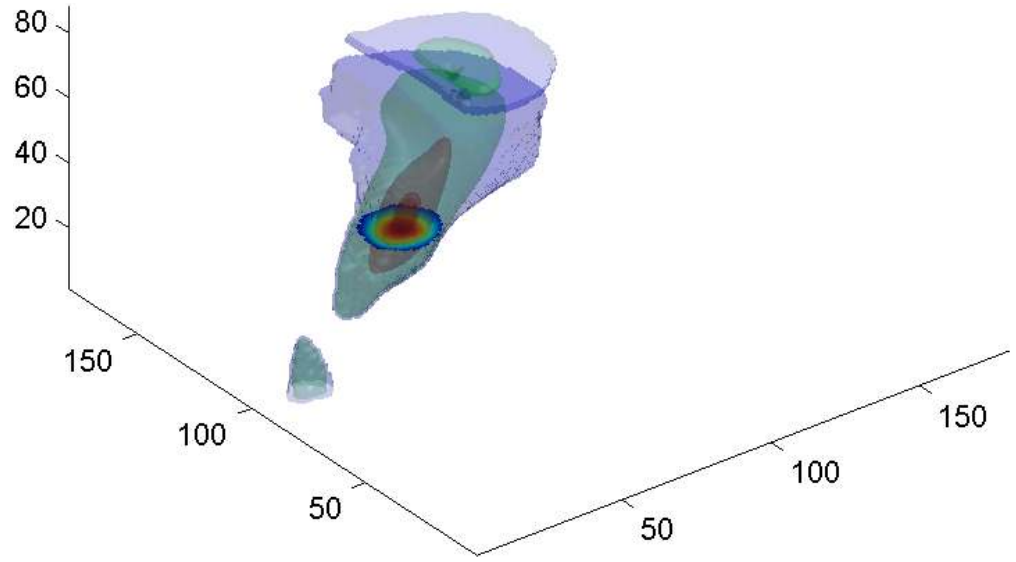


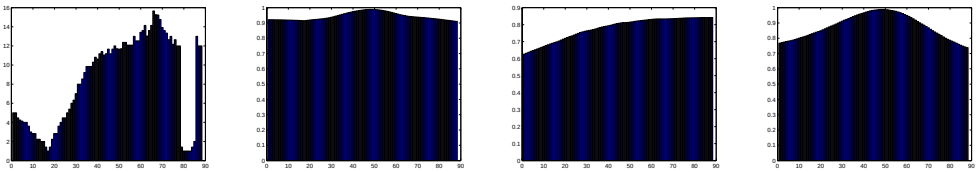
Figure 4.7.: **Values of f_d in the parameter space ϕ, θ, k.** The values are colored using a JET color map. The slice $\phi, \theta, 0$ is top-unimodal part of f_s in the surface point $\mathbf{p}$. The maximal value of f_d is the most red point this slice. Vertical axis z in this figure is scaled such that the interval $\langle -44, 44\rangle$ is mapped to $\langle 0, 88\rangle$.

The values of the unction f_d for some point $k \in \langle -K, K \rangle$ equals f_s for point $\mathbf{p}' = \mathbf{p} + \mathbf{n}\,\delta\,k$. We made a series of simulated experiments to see properties of the function f_d with ground truth point $\mathbf{p}$ and obtained a volume of values of f_d in the parameter space, see figure 4.7.

We observed that the function f_d is r-unimodal. The consequence is that the function f_s is r-unimodal in points near the ground truth point $\mathbf{p}$, too. The question is how does r-unimodality of f_s changes when changing k. We did not do as exhaustive experiments with f_d as with f_s. Nevertheless, we observed that the lower bound of r in the third dimension of f_d is $r = 5$.

The lower bound of r of f_s for points $\mathbf{p}' = \mathbf{p} + \mathbf{n}\,\delta\,k$, where $k \in \langle -5, 5 \rangle$ was $r = 9$. As you can see in figure 4.8 (a) and (b). In this case $K = 49$. Figure 4.8 (a) shows r of the f_s as a function of k. Figure 4.8 (b) shows the global maximum of f_s as a function of k.

Figure 4.8 (c) shows the values of f_d for fixed $(\phi, \theta) = (90, 90)$ as a function of k. The gradient descent optimization of f_d starting from $(\phi, \theta, k) = (90, 90, 0)$ stopped at a local optimum far from the global optimum. Figure 4.8 (d) shows the values of f_d for fixed ground truth $(\phi, \theta) = (90, 45)$ as a function of k. The maximal value corresponds to the ground truth point.



(a) (b) (c) (d)

Figure 4.8.: **Values across depths i.e. across k.** (a) r as a function of k, (b) the global maximum of f_s as a function of k, (c) f_d for fixed $(\phi, \theta) = (90, 90)$ as a function of k, (d) f_d for fixed ground truth $(\phi, \theta) = (90, 45)$ as function of k. Horizontal axis x in these figures is scaled such that the interval $\langle -44, 44 \rangle$ is mapped to $\langle 0, 88 \rangle$.

4.4.2. Algorithm

In this section, we describe Algorithm 2 for improving the position of point $\mathbf{p}$ and finding the normal $\mathbf{n}$. This algorithm first evaluates regularly spaced samples from Algorithm 1. Next, the algorithm starts a local optimization from the maximal sample. First, the algorithm optimizes the value d in function $f_d(\phi, \theta, k)$ with fixed ϕ, θ obtained previously. Then, the algorithm optimizes (ϕ, θ) using the gradient descent on function f_s in previously computed point $\mathbf{p}'$ defined by the value k. These two last steps are iteratively repeated until a convergence.

Algorithm 2 Position improovement

Require: $I_k, I_m, \mathtt{P_k}, \mathtt{P_m}$ - calibrated stereo pair (I image, $\mathtt{P}$ corresponding projection matrix)

α, β - thresholds for max value of the final similarity and max-min value from sampling

function $(\mathbf{n}, \mathbf{p}') = improvePosition(\mathbf{p})$

  $\mathbf{c}$ = vector from the point $\mathbf{p}$ to the middle point between k-th and m-th camera center

  $s_{max} = 0$

  for all d = -15 to 15 step 5 **do**

    $\mathbf{p}' = \mathbf{p} + \mathbf{c}\,d$

    if $(\phi, \theta, s) = findNormal(\mathbf{p}')$ success **then**

      if $s > s_{max}$ **then**

        $(\phi_{max}, \theta_{max}, s_{max}) = (\phi, \theta, s)$

      else

        return failed;

      end if

    end if

  end for

  $\mathbf{p}' = \mathbf{p}$

  $s' = 1.0$

  $\mathbf{p}'_{old} = p'$

  $s'_{old} = s'$

  while not convergence of $s' - s'_{old}$ and $\mathbf{p}' - \mathbf{p}'_{old}$ **do**

    $\mathbf{p}'_{old} = \mathbf{p}'$

    $s'_{old} = s'$

    $(\mathbf{p}', s')$ = do local optimization on parameter d of function $f_d(\phi, \theta, d)$ starting from $(\phi_{max}, \theta_{max}, 0)$ where $d \in \langle -K, K \rangle$

    $(\phi_{max}, \theta_{max})$ = do local optimization on parameters (ϕ, θ) of function $f_s(\phi, \theta)$ for point $\mathbf{p}'$ starting form $(\phi_{max}, \theta_{max})$ to find a local extrema (we use gradient descent algorithm)

  end while

  return $\phi_{max}, \theta_{max}, \mathbf{p}'$

end function

4.5. Experiments

In this section, we show the comparison of results of our method (OUR) with method [93] (ZAB) and with method [21] (FUR). The experiments were carried out on simulated data, see Figure 4.9. We generated 5356 3D points uniformly distributed on each of the two faces of the scene. The size of the ring was computed from $R = 14$. We compare the results of each of three methods (OUR, FUR, ZAB) with the ground truth. For each

(a) (b) (c)

Figure 4.9.: **Simulated scene:** (a) top view, (b) left image, (c) right image

Method	OUR	ZAB [93]	FUR [21]
% of inliers	94.0	94.6	76.0
% of outliers	6.0	5.4	24.0
comp time in section	1264	6139	335

Table 4.2.: **Comparison of methods:** OUR,FUR,ZAB.

Method	OUR: improvePosition	OUR: findNormal
% of inliers	75.4	16.9
% of outliers	24.6	83.1

Table 4.3.: **Comparison of methods improvePosition and findNormal on translated face points in the face normal direction.**

point we computed degree between detected and the ground truth normal. In the first experiment, we detected only normal of ground truth 3D point.

Table 4.2 shows that our method gives the same result as method ZAB in $99,5\%$ of cases but is $4,9$ times faster. On the other hand, our method is $3,8$ times slower than FUR but gives 18% more inliers. The precision threshold was set to angle 5 degrees difference from the ground truth normal.

The second experiment was carried out on the 1250 3D points uniformly distributed on each of the two faces of the same scene and translated in the face normal direction by $10\,\delta$ (that means approximately 2 pixels of reprojection error on the image). The table 4.3 shows that normal detection method (findNormal 1) found correct normal in $16,9\%$ cases on translated points. The improving position and normal detection method (improvePosition 2) found truth normal in 75.4% cases on the same data.

The next experiment was carried out on another simulated scene, see Figure 4.10. The red dots are the seed positions with nondiscriminative texture and the blue dots are the seeds positions with discriminative texture. By texture discriminativity we mean the texture when the function findNormal 1 fails (threshold $\beta = 0.1$).

The last experiment was carried out on a real scene, see Figure 4.11. The red polygons show the seeds with nondiscriminative texture, see 4.11 (b). The blue polygons show

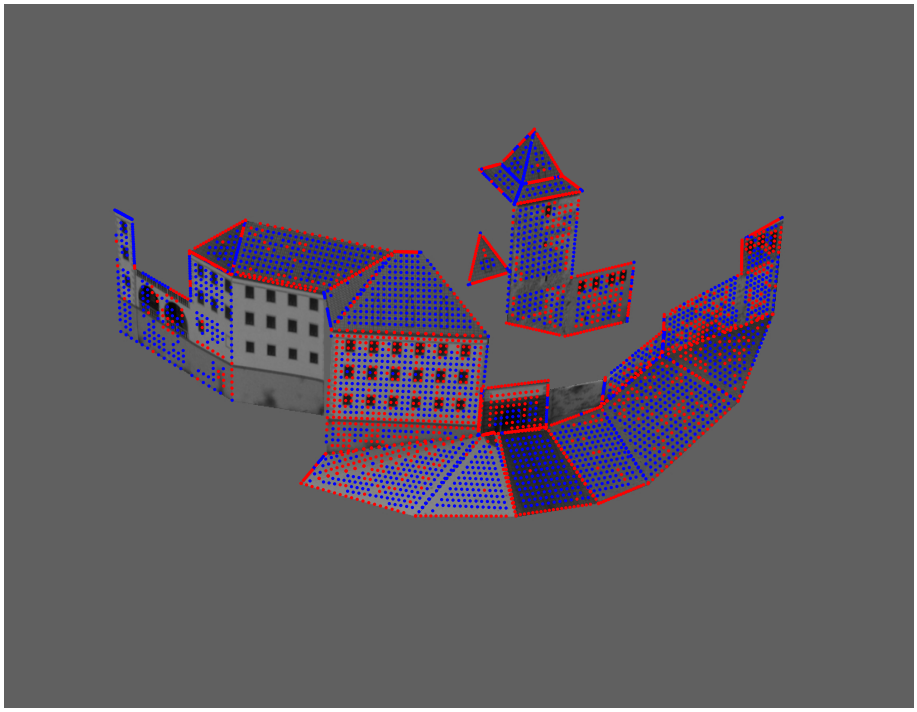
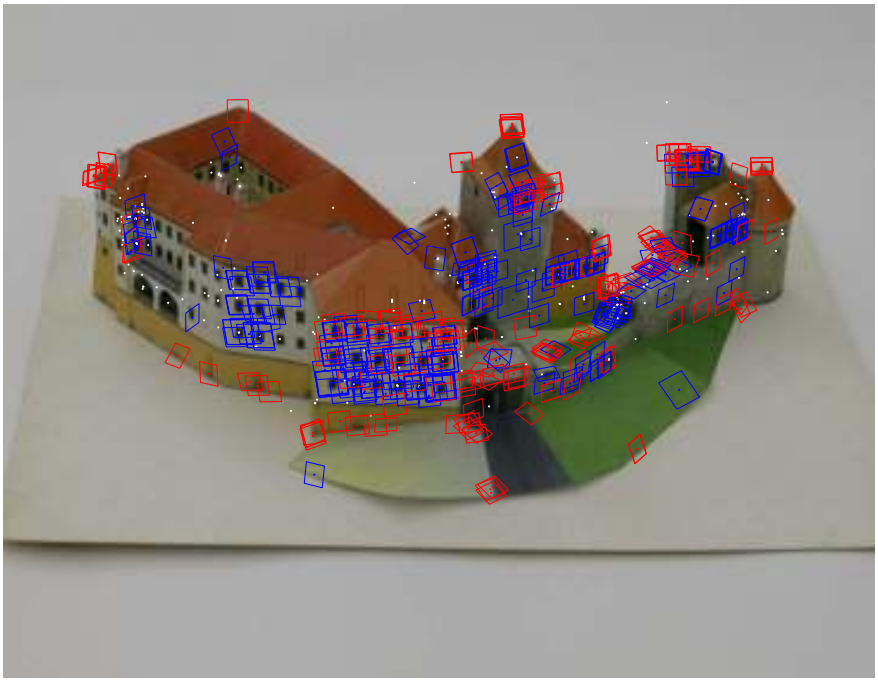


Figure 4.10.: **Simulated scene by Image Modeller [1].** The red dots are the seeds positions with nondiscriminative texture and the blue dots are the seeds positions with discriminative texture. By texture discriminativity we mean the texture when the function findNormal 1 fails (threshold $\beta = 0.1$).

the seeds with discriminative texture, see 4.11 (c). By texture discriminativity we mean the texture when the function improvePosition 2 fails (threshold $\beta = 0.1$).



(a)

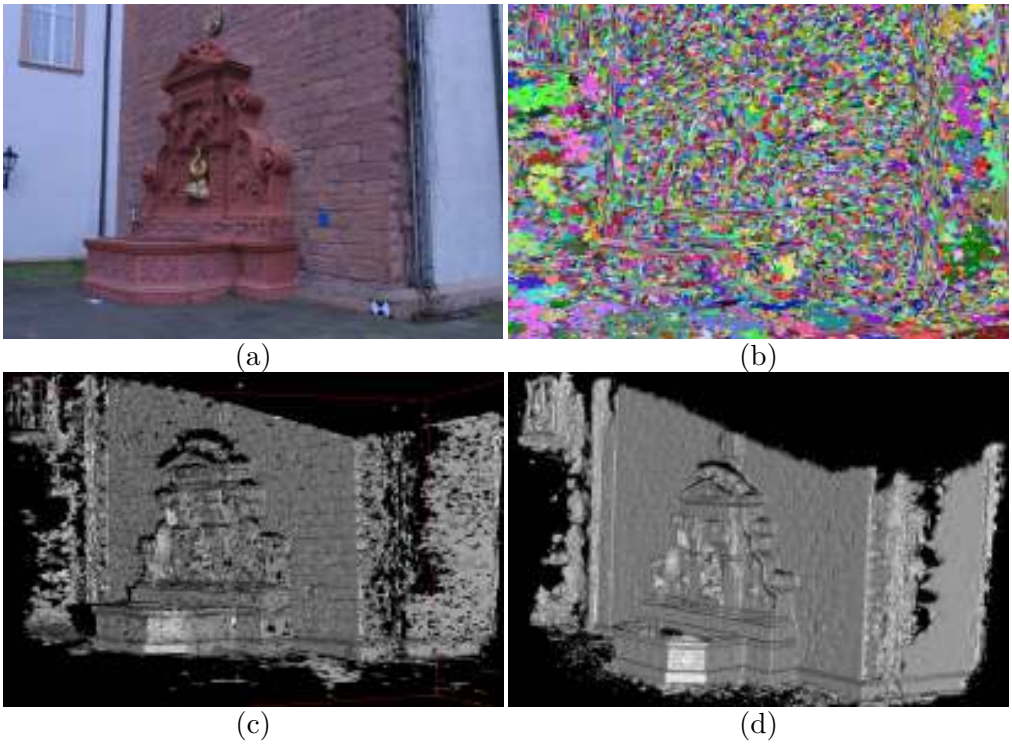
(b)

(c)

Figure 4.11.: **Real scene.** (a) detected seeds using algorithm improvePosition 2. The red polygons shows the seeds with nondiscriminative texture (b). The blue polygons shows the seeds with discriminative texture (c). By texture discriminativity we mean the texture when the function improvePosition 2 fails (threshold $\beta = 0.1$).

5 Segmentation based Multi-View Stereo

This chapter presents a segmentation based multi-view stereo reconstruction method. We address (i) dealing with nondiscriminative texture in very homogeneous image areas and (ii) processing large images in affordable time. To avoid searching for optimal surface position and orientation based on nondiscriminative texture, we (over)segment images into segments of low variation of color and intensity and use each segment to generate a candidate 3D planar patch explaining the underlying 3D surface. Every point of the surface is explained by multiple candidate patches generated from image segments from different images. Observing that the correctly reconstructed surface is consistently generated from different images, the candidates that do not have consistent support by other candidates from other images are rejected. This approach leads to stable and



(a) (b) (c) (d)

Figure 5.1.: **Strecha's Fountain-P11 data set reconstruction.** (a) first image (b) segmentation of the first image (c) 3D segments of first image (d) final mesh

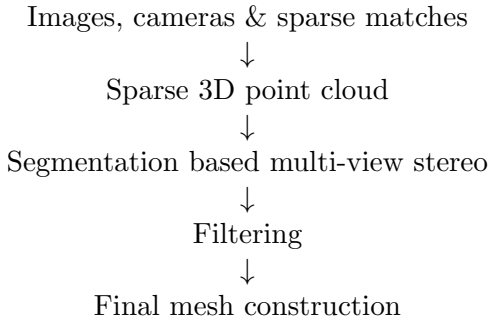


Figure 5.2.: **Our reconstruction pipeline.**

good results since (i) we use larger 3D patches in homogeneous image areas where small patches covered by nondiscriminative texture would lead to ambiguous results, and (ii) we accept only candidates that are consistent across several images. Since the image segmentation used is very fast and it considerably reduces the number of candidates per image on typical scenes, we typically generate and test relatively small number of 3D hypotheses per image and thus can process large images in affordable time. We demonstrate the performance of our algorithm on large images from Strecha's dataset.

Let us first give a gist of our approach. First, we use efficient graph-based image segmentation [17] to obtain image segments of each image, see Figure 5.1(b). We also compute initial sparse 3D point cloud by guided feature matching similarly to [21]. Next we compute a planar approximation of scene surface for each segment and for each feature of the segment, see Figure 5.1(c). We use corresponding 3D point of the feature from the guided matching as initial position of the segment planar approximation.

Next, we use the method described in previous Chapter 4 to compute precise segment planar approximation. Instead of using a circular patch, we use a nonuniform shape of the patch given by a image segment.

The part of the 3D plane that projects to an image segment is further called *3D segment*. The confidence of this 3D segment is the similarity (the normalized cross-correlation) of image reprojections onto this planar 3D segment. We call this *3D segment creation procedure*. If there are more features in one image segment, we then choose the 3D segment with the maximal confidence and accept it if its confidence is above a threshold. Later, we alternate between greedy hypothesizing of neighboring image segments and optimizing them using the method described in previous Chapter 4.

Finally, we perform a filtering step to remove inconsistent 3D segments. Filtering is done by clustering 3D segments generated from the corresponding image segments from different images. We use Poisson surface reconstruction [47] to generate the final 3D mesh, see Figure 5.1(d).

5.1. The processing pipeline

Next we give a detailed description of our reconstruction pipeline. The workflow, Figure 5.2, is described in detail below.

5.1.1. Cameras, and their calibration

We assume that images were obtained by cameras with known internal and external calibration or which was computed by, e.g., feature based structure from motion [71, 59, 55]. For each camera i, we keep its corresponding camera matrices [29, p.163] $\mathtt{K}_i$, $\mathtt{R}_i$, the camera center $\mathbf{C}_i$ and the radial distortion parameter κ_i. We assume that the radial distortion center coincides with the principal point. A point $\mathbf{X}$ from space is projected to the image point $\mathbf{x}_i = \pi_i \left(\mathtt{R}_i(\mathbf{X} - \mathbf{C}_i)/(\mathbf{r}_i^\top(\mathbf{X} - \mathbf{C}_i))\right)$ of the i-th camera with

$$\pi_i((x, y, 1)^\top) = \mathtt{K} \begin{pmatrix} (1 + \kappa_i(x^2 + y^2))\, x \\ (1 + \kappa_i(x^2 + y^2))\, y \\ 1 \end{pmatrix} \tag{5.1}$$

5.1.2. Segmentation

We use the implementation [17] of efficient graph-based image segmentation [17]. We use $\sigma = 0.5, k = 50, minSegSize = 100$ in all our experiments. We have to point out that it is possible to use other segmentation algorithms segmenting image into piecewise constant areas. Over-segmentation of smooth surfaces is tolerated.

5.1.3. Data structures

We shall use the following concepts and data structures:

- I_r stands for the r-th image and C_r stands for the corresponding camera. Vector (x, y) represents an image point and the corresponding ray. S_r stands for the set of all segments of image I_r. The segment $s_i \in S_r$ is a set of all pixels of the i-th segment.

- $I_k(s_i)$ stands for the set of intensities of s_i pixels in k-th image. Our basic 3D element is a *3D segment* $\sigma_i = (\mathbf{X}, \mathbf{n}, r, \Sigma, i)$, which is a part of the plane $\pi = (\mathbf{X}, \mathbf{n})$ that projects to a segment s_i of the image I_r. The set Σ is the set of images which see the patch p.

We also work with *patch* $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$, which is a planar circular disk with center in $\mathbf{X}$, normal $\mathbf{n}$, inner, resp. outer, radius $R/2$, resp. R, and the set Σ of images which see the patch p. R depends on the smallest image detail γ_r and the distance of $\mathbf{X}$ from C_r, $R = d_r(\mathbf{X})\, s$, ($s = 0$ in all of our experiments) so that p covers approximately $(2\, s + 1)^2$ pixels in on images $\{I_i | i \in \{r \cup \Sigma\}\}$ (d_r is defined in Section 6.1.3).

5.1.4. Level of detail

The level of detail $d_r(\mathbf{X})$ of a point $\mathbf{X}$ is a function of camera index r and point $\mathbf{X}$ in space. It is set to the radius of the ball centered in $\mathbf{X}$ that projects to the image detail $\gamma_r = 1$. Given C_r and projection matrix $\mathtt{P}_r = [\mathtt{M}_r | -\mathtt{M}_r \mathbf{C}_r]$, $d_r(\mathbf{X}) = \arg\min_{\alpha \in \mathbb{R}} \|R_r\,(\mathbf{X} - \mathbf{C}_r) - \alpha\,\pi_r^{-1}(\mathbf{x} + (\gamma_r/2, 0, 0)^\top)\|$ is the distance of $\mathbf{X}$ from the ray of sight emanating from the camera center $\mathbf{C}_r$, $\gamma_r/2$ pixels away from the projection $\mathbf{x}$ of $\mathbf{X}$. The detail $d_r(\mathbf{X})$ depends on the corresponding camera detail and on the distance from the camera and is set to avoid working with too fine 3D detail not well captured in images.

5.1.5. Sparse 3D point cloud

Inspired by previous works, e.g. [21], we reconstruct *sparse 3D point cloud* from sparse matches by guided matching [29] in image pairs along epipolar lines. We do it as follows.

We partition image r into rectangular tiles of size $D \times D$ ($D = 16$ in all of our experiments). For every *target* image $I_t, t \in \{1, 2, ..., n\} \wedge t \neq r$ do. If there are any sparse matches between I_r and I_t available from the feature based structure from motion, assign them to the tiles where they project. Then, in the tiles which have $m < K$ ($K = 4$ in all of our experiments) matches, compute Harris feature points [28] and keep the $K - m$ strongest ones. This, according to our experience, generates sufficient but not excessive amount of candidate feature points.

For every feature point $\mathbf{x}$ in I_r, search in the neighborhood of $\pm\delta$ ($\delta = 2$ in all of our experiments) pixels around the corresponding epipolar line l_{rt} in I_t for the most similar Harris feature point $\mathbf{y}$. Evaluate the similarity as the normalized cross-correlation (NCC) of $l \times l$ ($l = 5$ in all of our experiments) image patches centered at $\mathbf{x}$ and $\mathbf{y}$. Points $\mathbf{x}$, $\mathbf{y}$ form a new match if $\mathbf{x}$ is also the most similar point to $\mathbf{y}$ among all candidates in the $\pm\delta$ vicinity of the epipolar line l_{tr} generated by $\mathbf{y}$ in image I_r, i.e. they are mutually best matches.

Next, for each match $(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} = (x, y)$, triangulate the match $(\mathbf{x}, \mathbf{y})$ into point $\mathbf{X}$ and check if the (apical [82]) angle contained by rays ($\mathbf{x}$ and $\mathbf{y}$) is larger than a predefined threshold $\alpha_S = 5$. If yes, then we construct patch $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$ with $\mathbf{n}$ equal the normal of the reference image plane and $\Sigma = \{t\}$. This reconstructs a non-ambiguous (large apical angle) patch that is strongly supported by image matches.

After processing all target images I_t, we cluster according to their centers $\mathbf{X}_p$ by the QL clustering [35] with the diameter equal $3\,d_r(\mathbf{X}_p)$. We replace clusters by patches $p = (\mathbf{X}_s, \mathbf{n}, r, \Sigma_s)$ with the centroid of the cluster $\mathbf{X}_s$, the normal of the reference camera plane $\mathbf{n}$, and the Σ_s containing the union of all target cameras of the cluster.

5.1.6. Segmentation based multi-view stereo

We take each image as reference image I_r exactly once. To obtain at most one 3D segment for each image segment of reference image we perform following two steps which are described in detail below.

Optimal 3D segment creation. We create the optimal 3D segment for each patch $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$ of I_r. Let us consider a patch $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$ and a segment $s_i \in S_r; P_r\,\mathbf{X} \in s_i$ of the reference image I_r. The goal is to maximize the similarity of images (reference I_r and all target $I_t; t \in \Sigma$) re-projections to a plane $\pi = (\mathbf{X}, \mathbf{n})$ constrained by a segment s_i. We optimize criterial function

$$f_s(p, s_i) = \frac{\sum_{t\in\Sigma} c(I_r(s_i), I_t(H\,s_i))}{|\Sigma|}, \tag{5.2}$$

where H is the homography inducted by the plane π and consistent with epipolar geometry between cameras r and t.

We want to find such a $p^* = (\mathbf{X}^*, \mathbf{n}^*, r, \Sigma)$ that

$$f_s(p^*, s_i) = \arg \max_{\mathbf{X}\in R^3; \mathbf{n}\in\langle 0..180\rangle^2} f_s(p, s_i) \tag{5.3}$$

We are searching for an optimum as in Chapter 4.

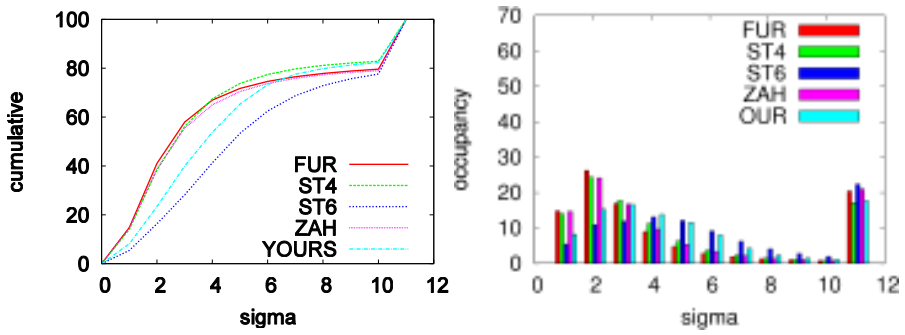
We parametrize normals by two angles (ϕ, θ) in spherical coordinates $\vec{n} = (cos(\phi)\,sin(\theta), sin(\phi)\,sin(\theta), cos(\theta))^T$. First, we compute the samples of f_s in the patches uniformly distributed in orientation space only by the distance of 18 degrees in a $L \times L$ grid ($L = 7$ in all of our experiment) around the $\mathbf{n}$ of the input patch point p. Then we run the gradient descent optimization from the best sample. Then we optimize the patch orientation and position, too. The patch position is optimized on the ray connecting $\mathbf{p}$ and the center of the reference camera with the step of $d_r(\mathbf{X})$.

After processing of all segments we accept only those segments s_i which satisfy the following: (1) if there are more features (patches projections) inside the segment s_i, then we choose the 3D segment with the maximal confidence and (2) the maximal confidence has to be greater than a threshold μ ($\mu = 0.6$ in all of our experiments).

Greedy searching of unexplored segments. We do a greedy searching of unexplored segments to explore more segments. We iterate following steps. For each unexplored segment s_u do. First, for each neighboring explored segment s_e create a new patch $p_{u_e} = (\mathbf{X_{u_e}}, \mathbf{n_e}, r, \Sigma_e, u)$ with the position at the intersection $\mathbf{X_{u_e}}$ of the ray defined by the center of gravity of the s_u in the reference camera r and the plane π_{s_e}. Finally we find optimal 3D segment for it using abovementioned approach. Next we choose the best 3D segment according to $f_s(p_{u_e})$. We do 5 iterations in all of our experiments.

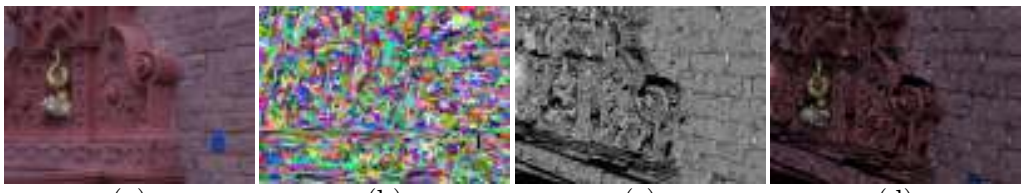
5.1.7. Filtering

We perform a filtering step to remove inconsistent 3D segments. Filtering is done by clustering segments from different images for the same 3D planar part. For each image we create a patch p_{e_i} from each pixel $i \in s_e$ of each explored segment $s_e = (\mathbf{X}, \mathbf{n}, r, \Sigma, e)$. The $p_{e_i} = (\mathbf{X_{e_i}}, \mathbf{n}, r, \Sigma)$, where $\mathbf{X_{e_i}}$ is the intersection of the corresponding ray to the pixel i with the plane $\pi = (\mathbf{X}, \mathbf{n})$.



(a) (b)

Figure 5.3.: **Strecha's evaluation [77] for the Fountain-P11 data set.** OUR - proposed method, FUR [21], ST4 [74], ST6 [75], ZAH [95].



(a) (b) (c) (d)

Figure 5.4.: **Strecha's Fountain-P11 data set reconstruction cutout.** (a) first image (b) segmentation of the first image (c) untextured 3D segments (d) textured 3D segments

First, we filter out patches $p_i = (\mathbf{X}, \mathbf{n}, i, \Sigma)$ which do not have at least δ ($\delta = 1$ or 2 in all of our experiments) neighboring patches from at least δ other images in $3\,d_i(\mathbf{X})$ neighborhood of the center of p_i.

Next, we filter out all patches from images $I_j \neq I_i$ contained in this neighborhood fulfilling $d_i(\mathbf{X}) < d_j(\mathbf{X})$ to preserve the best level of detail.

5.1.8. Final mesh construction

We use Poisson surface reconstruction (PSR) [47] for final mesh construction. The input to PSR is an oriented point cloud with all filtered patches.

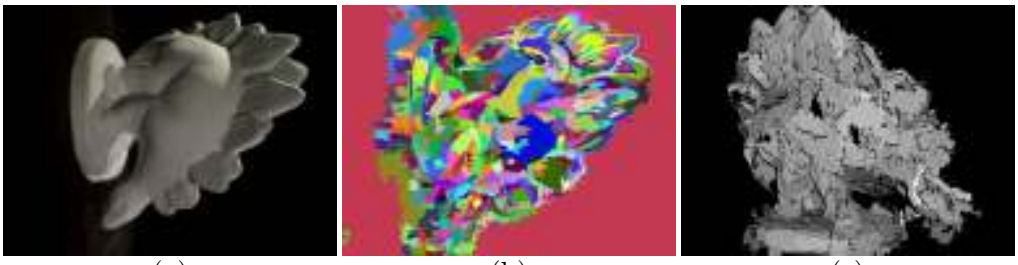
5.2. Results

To evaluate the quality of reconstructions, we present results on data sets from the standard evaluation Middlebury [68] and Strecha's [77] databases.

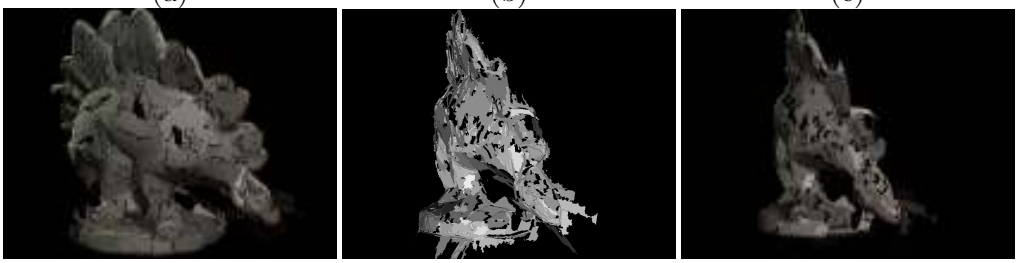
Figure 5.3 shows the evaluation on the Strecha's Fountain-P11 data set. The histograms (a), (b) show that our method is comparable to other methods on the Fountain-P11 set.

(a) (b) (c) (d)

Figure 5.5.: **Final mesh.** Four different views of final mesh of Strecha's Fountain-P11 data set reconstruction.



(a) (b) (c)



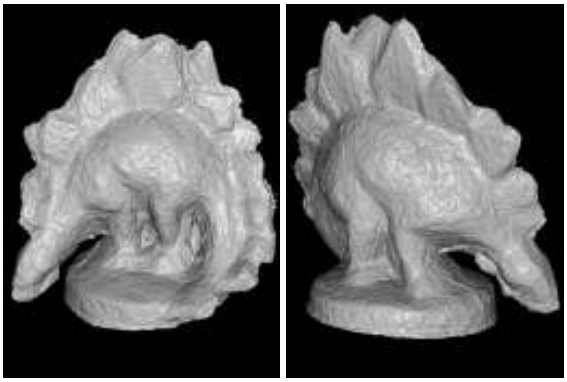
(d) (e) (f)

Figure 5.6.: **Midlebury's dinoRing data set reconstruction.** (a) first image (b) segmentation of the first image (c,e) two views of untextured 3D segments of the first image (d,f) two views of textured 3D segments of the first image

Figure 5.4 shows the detailed view on the 3D segments of first image of Strechas Fountain-P11 data set. The dataset consists of 3072×2048 resolution images. Therefore, the segmentation of such big images can handle all important details (see Figure 5.4(a)(b)).

Figure 5.5 shows PSR reconstruction of filtered patches from different views.

To demonstrate the ability of dealing with nondiscriminative texture in very homogeneous image areas, we show reconstructions on the dinoRing dataset from Middlebury [68] standard evaluation database. Figure 5.6 shows the detailed view on the 3D segments of the first image of Midleburys's dinoRing data set. The large homogenous segments on the dino's neck are problematic in patch based approaches [25, 21]. Even thought PSR usually interpolate missing ungrowed parts well, Figure 5.7 shows that our method can deal with such areas without any posterior interpolation.



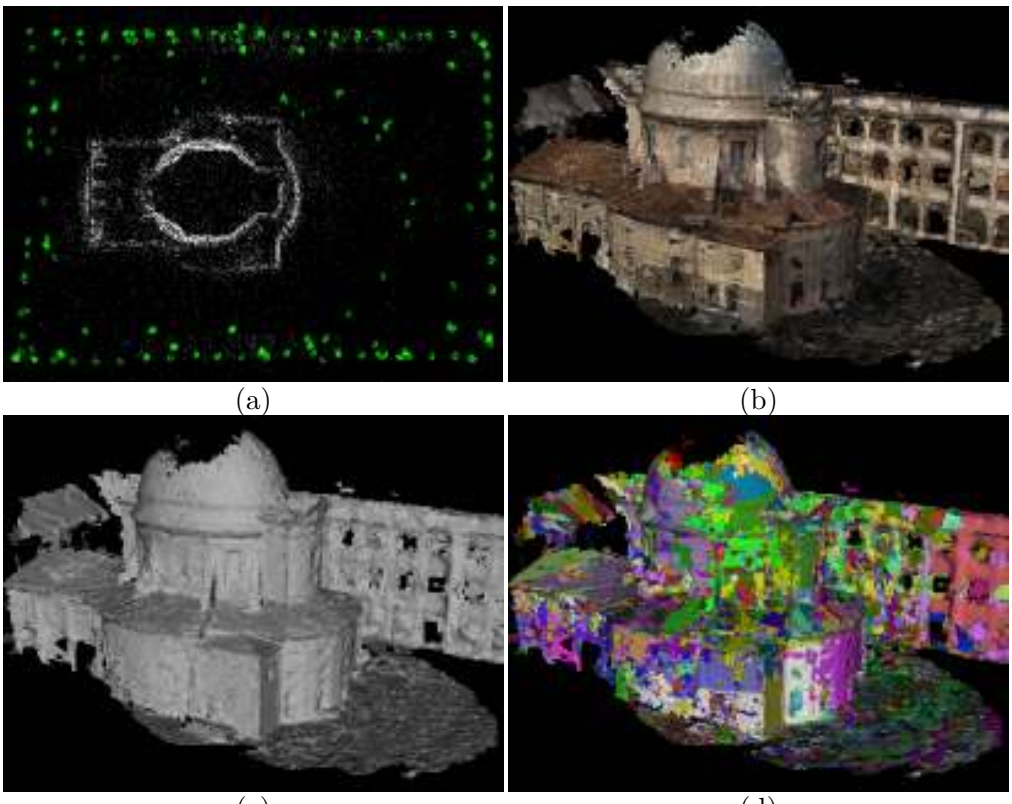
(a) (b)

Figure 5.7.: **Two different views of Midlebury's dinoRing data set reconstruction.**

Figure 5.7 shows reconstructions for the dinoRing data set containing 47 640 × 480 images of the Middlebury Dino. We achieved 0.79 mm accuracy and 95.9% completeness on this dataset. See the Middlebury evaluation page for JancosekCVWW results and their comparison. We conclude that our results are comparable to other state of the art techniques. We have to point out that we do not use silhouettes in our method.

6 Scalable Multi-View Stereo

This chapter presents a scalable multi-view stereo reconstruction method which can deal with a large number of large unorganized images in affordable time and effort. The main contribution is that the computational effort of our technique is a linear function of the surface area of the observed scene which is conveniently discretized to represent a sufficient but not excessive detail.



(a) (b) (c) (d)

Figure 6.1.: **Reconstruction of an outdoor scene from 294 (**1296×972 **pixels) images from the Marseille Data Set, computed in 92 minutes.** (a) 3D seeds and cameras , (b,c) textured and untextured reconstruction, (d) reconstruction I_r-colored to show central cameras used (see the text).

By "scalable" we mean that we can process a very large image data with computational effort growing not more than is necessary for obtaining a required accuracy. When dealing with very large data, we are more interested in an acceptable result in a limited time rather than in "the optimal" result in time which is not acceptable. Clearly, we often can't afford equal treatment of all data since this "linear" approach would eventually become infeasible. We consider large data in three particular situations when (1) a very large number of images covers a very large scene, e.g. a complete city, (2) an object of interest is covered repeatedly by a very large number of images, e.g. the dense ring of the Middlebury temple, and (3) a scene is captured by high resolution images, e.g. 3072×2048.

Dealing with (1) calls for fast processing that does not need to load all data into memory at the same time. The case (2) calls for remembering already solved parts of the scene and avoiding unnecessary processing of redundant data which brings a little improvement. Situation (3) calls for controlling the level of detail in space as well as in images.

We present a scalable multi-view stereo computation technique with the following properties: (1) the computational effort of our technique is a linear function of the surface area of the observed scene which is conveniently discretized to represent sufficient but not excessive detail. Our technique works as a filter on a limited number of images at a time and can process arbitrarily large data sets in limited memory; (2) scene reconstruction is built gradually and new image data are processed only if they noticeably improve the current reconstruction; (3) scene reconstruction uses a variable level of detail, which is not greater than what can be reconstructed from images.

For very large scenes economically covered by images, the effort of our technique is *proportional* to the number of images (thus also to the total number of captured pixels). For scenes covered redundantly by many overlapping images, the effort is *considerably smaller* than the effort needed to process all pixels of all the images and is proportional to the scene surface area. For scenes captured in excessive resolution, the effort is *limited* by the resolution sufficient for reconstructing the scene on a chosen level of detail.

We demonstrate in experiments with Middlebury and Strecha's databases [68, 77] that we achieve the quality comparable to other state-of-the-art techniques. It is clear from our experiments that we can efficiently process redundant data sets, e.g. we used only 7% of pixels of the 311 images of the Middlebury temple and computed the reconstruction in 49 minutes with maximum 1 GB of memory.

A large scale experiment in which we processed 1000 images from the Google Street View Pittsburgh Experimental Data Set [2] in 183 minutes with maximum 1 GB of memory demonstrates that we can process very large data sets. Although the data set has been acquired as a sequence, we were considering it as unorganized. In another large scale experiment, we processed 294 unorganized images (calibrated by [71]) in 92 minutes, Figure 6.1, with maximum 1 GB of memory. These experiments demonstrate that we can process very large unorganized data sets with large images.

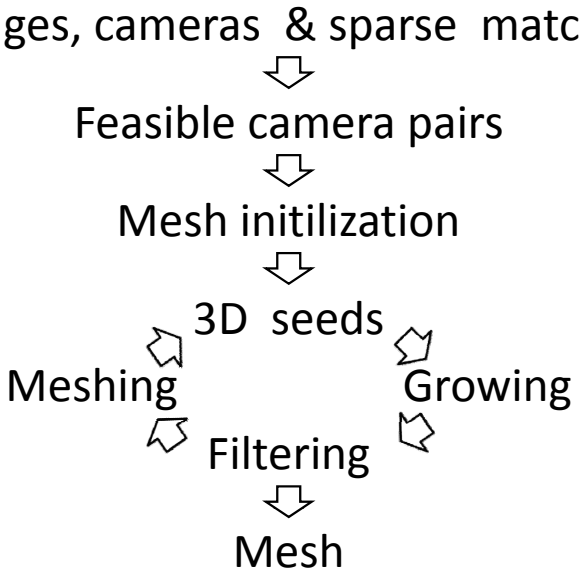


Figure 6.2.: **Our reconstruction pipeline.**

6.1. The processing pipeline

Next we give the description of our reconstruction pipeline. We assume that images were obtained by cameras with known internal and external calibration or which was computed by a feature based SFM method [71, 59, 55]. The workflow, Figure 6.2, is described below in detail and related to the previous work and computational and memory requirements.

6.1.1. Data structures

We shall use the following concepts and data structures. I_r stands for the r-th image and c_r stands for the corresponding camera. Vector (x, y) represents an image point and the corresponding ray. Our basic 3D element is a *patch* $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$, which is a planar circular disk with center in $\mathbf{X}$, normal $\mathbf{n}$, reference image index r, and the set Σ of target image indexes which see the patch p.

The radius of the patch R is a function of camera index r and point $\mathbf{X}$ in space. It is set to the radius of the ball centered in $\mathbf{X}$ that projects to the image detail with radius s ($s = 2or3$ pixels in all of our experiments) so that p covers approximately $(2\,s + 1)^2$ pixels in image I_r.

To avoid processing the space that is occluded in c_r by already accepted (Section 6.1.7) reconstruction (visible in c_r), we use buffer V_r which holds the closest known patch to

c_r (visible in c_r) on the ray (x, y). To facilitate efficient 3D reconstruction by growing, we use buffer G_r that holds up to 10 patches on a ray (x, y). The *final mesh* is assumed to be consisting of a set of contiguous triangulated meshes. Each of these meshes M is represented per parts in buffers M_r associated with *reference images* I_r concurrently. This is possible since only one mesh can be seen by one pixel. Each point of M is represented in exactly one M_r, i.e. M_r partitions M.

6.1.2. Feasible camera pairs

To avoid unnecessary matching of camera pairs, which see only a few points in space generating good matches, we construct *feasible camera pairs*. We say that two cameras form a feasible pair if there are points in space "whose projections do not change scale more than by 80%". We check it as follows.

We find the shortest transversal of the optical axes of cameras say c_i, c_j . Let points A, B, resp. C, be the endpoints, resp. the center, of the transversal. Let α_{ij} be the angle contained by the rays projecting C into the cameras. Let the unit ball centered in C projects to circles with diameters d_i, d_j[1]. The camera pair is feasible if A, B, C is in front of both cameras, projects to both images, $5^\circ \leq \alpha_{ij} \leq 50^\circ$, and $\min(d_i, d_j)/\max(d_i, d_j) \geq 0.8$.

This computation is done only if matches from SFM are not present. Otherwise we use the matches to compute the feasibility. We consider two cameras as feasible if there are some matches (at least 10 in all our experiments). We store the feasibility information in *camera incidence matrix* $\mathtt{C}_I$ with rows and columns corresponding to cameras and $\mathtt{C}_I(i, j) = 1$ for a feasible pair i, j and $\mathtt{C}_I(i, j) = 0$ otherwise.

Using $\mathtt{C}_I$ we construct the *feasibility neighborhood* O_r of camera r to consist of all cameras forming feasible pairs with r. It is determined by non-zero bits in the i-th row of $\mathtt{C}_I$. The matrix $\mathtt{C}_I$ will be used in the further processing to keep track of unprocessed cameras by zeroing element of $\mathtt{C}_I$ corresponding to already processed cameras.

Computational effort & memory requirements Computing feasible camera pairs of N cameras naively calls for computing with $N(N-1)/2$ camera pairs. This is feasible for thousands of cameras. If many more cameras are to be processed, subquadratic algorithms are available in Computer graphics [12], assuming that $\mathtt{C}_I$ is sparse. To store (uncompressed) $\mathtt{C}_I$, we need only $N(N-1)/2$ bits.

6.1.3. Level of detail

To exploit the information efficiently, we control the level of detail in images and in space. The level of detail $d_r(\mathbf{X})$ in space is a function of camera index r and point $\mathbf{X}$ in space. It is set to the radius of the ball centered in $\mathbf{X}$ that projects to the image detail γ_r. The detail $d_r(\mathbf{X})$ depends on the corresponding camera detail and on the distance from the camera to avoid working with too fine 3D detail not well captured in

[1] For ellipses, d_i, d_j are the lengths of their major axes.

images. Parameter γ_r controls overall level of detail and is determined by memory and time constraints. We used $\gamma_r = 1$ for Middlebury and $\gamma_r = 2$ for Strecha's and Marseille data.

Computational effort & memory requirements Values $d_r(\mathbf{X})$ are evaluated online when processing 3D points.

6.1.4. Reference image selection

Take the next reference image I_r by finding the row r of $\mathtt{C}_I$ with the maximal sum, i.e. select the camera with high potential of producing large and reliable 3D structure.

Compute buffer $V_r(x, y)$ using already accepted meshes (Section 6.1.7). Later, $V_r(x, y)$ will contain the closest visible accepted patch to c_i. Buffers V_r are used for efficient processing of redundant data. Everything behind the accepted patch is not considered further.

Computational effort & memory requirements For a fixed r we need to keep in memory buffers V_r, G_r, M_r, which altogether consist of approximately 12 times the number of image pixels divided by γ_r^2. They fit in memory even for large images.

6.1.5. 3D seeds

We create 3D seeds similarly to [21] with the following modifications. We use Harris feature points [28] and the seeds from SFM if available. We are searching for matches in all pairs formed by the reference camera r and all cameras in O_r. We work with seeds that are not occluded by closest already accepted surface (stored in $V_r(x, y)$) and whose apical angle [82] is larger than a predefined threshold α_S (we use $\alpha_S = 3$ in all our experiments). We also update the camera incidence matrix $\mathtt{C}_I$ by removing the camera pairs r, t which have less than m_S ($m_S = 10$ in all our experiments) seeds.

Computational effort & memory requirements Computing Harris points is feasible even for large images since it can be implemented as a filter [5]. The computational effort is linear in the number of image pixels. The typical number of seed patches per ray is 1-3.

6.1.6. Growing

Patches grow in space. The growing starts from seed patches and is guided by patch quality, a function of the patch pose in 3D, the reference image and its feasibility neighborhood. The goal is to obtain reliable proposals on 3D structure efficiently. This step has been inspired by technique [10] used for fast stereo reconstruction from image pairs and [21] used in multi-view stereo.

To evaluate the quality $q(p)$ of a patch $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$, we set

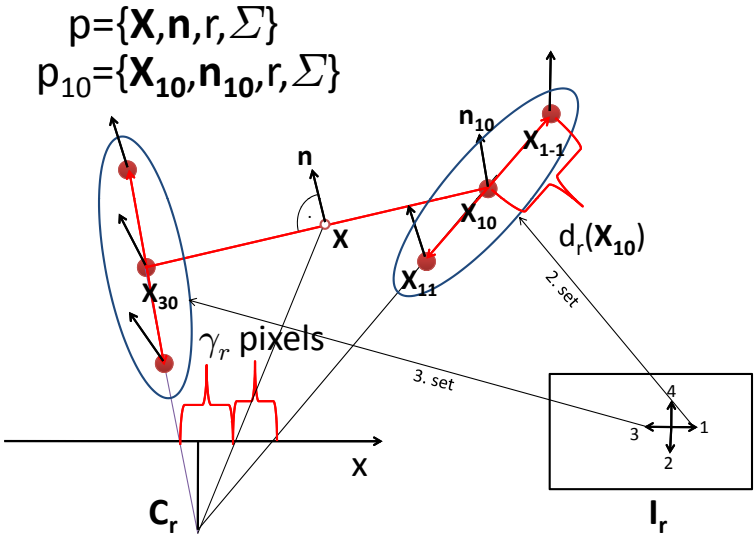


Figure 6.3.: **Expansion of patch p.** See the text.

$q(p) = \text{mean}_{i \in \Sigma} NCC(I_r(p), I_i(p))$, i.e., to the mean NCC of the texture reprojected from image r to images $i \in \Sigma$ by the patch p.

To make the growth more efficient, we use buffers V_r to suppress the growth of weak patches that become occluded. We compute buffer V_r by finding the closest visible patch to the camera r among all patches in already accepted meshes M_i (Section 6.1.7).

The growing starts from seed patches associated with image r by expanding them into new patches. The growing uses two queues. The priority queue Q_{open} contains patches prepared for the expansion ordered by their quality. Initially, Q_{open} contains all seed patches. The queue Q_{closed} contains the expanded patches. The growing process repeatedly removes the top patch $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$ from Q_{open}, expands it into $3 \times 4 = 12$ tentative patches $p_{jk} = (\mathbf{X}_{jk}, \mathbf{n}_{jk}, r, \Sigma)$, $j = 1, \ldots 4$, $k = -1, 0, 1$, out of which the four successor patches are selected and those which pass a series of tests are placed in Q_{open}, see Figure 6.3. Expanded patch p is placed into Q_{closed}. The process stops when Q_{open} becomes empty.

First, 12 patch centers are constructed in 4 groups with 3 points in each. The centers of the patches p_{j0} are constructed as the four points $\mathbf{X}_{j0}$ on the rays ρ_j obtained by backprojecting points in the γ_r 4-neighborhood of the projection of $\mathbf{X}$ into I_r. Patches p_{jk} for $k \neq 0$ are constructed with centers $\mathbf{X}_{jk}$ on ρ_{jk} in the distance $d_r(\mathbf{x_{jk}})$ from $\mathbf{X}_{j0}$. Next, $\mathbf{n}_{jk}$ are set to the normals of planes approximating (in the least squares sense) centers of patches already grown from p and not farther than $10\, d_r(\mathbf{X}_{jk})$ from $\mathbf{X}_{jk}$.

From each group, we then select a patch with the maximal quality $p_j = \arg\max_{k=-1,0,1} q(p_{jk})$. We place $p_j = (\mathbf{X}_j, \mathbf{n}_j, r, \Sigma)$ into Q_{open} if the following conditions are met: (1) no patch

has grown before on the ray (x, y) of $\mathbf{X}_j$ from the same seed as p_j, (2) the patch has high quality, i.e. $q(p_j) > 0.6$ and $q(p_j) > q_m - 0.1$, where q_m is the maximal quality on the same ray, (3) point $\mathbf{X}_j$ is not occluded by an accepted patch $V_r(x, y)$ with higher quality on the same ray, (4) none of the previously reconstructed patches is closer to $\mathbf{X}_j$ than $d_r(X_j)$, and finally (5) there are less than 10 patches on the ray of $\mathbf{X}_j$. We update $G_r(x, y)$.

Computational effort & memory requirements The maximal number of objects in memory during the growing step is $10\,N_r/\gamma_r^2$, where N_r is the number of pixels in I_r in pixels and 10 stands for the maximal number of patches on a ray in G_r. This number decreases on redundant images as the scene becomes more and more reconstructed since the growing is limited by already accepted structures (V_r).

6.1.7. Filtering and Meshing

MRF Filtering Inspired by the approach in [9], we recover a *filtered* 3D mesh from grown patches in G_r by MRF optimization. To fit MRF structures in memory for very large images, we design an approximate optimization procedure for the problem posed in [9], which does not need to load G_r in memory at once in order to get a good suboptimal solution.

We model the problem as a discrete MRF where each node may get assigned one of 11 labels. Nodes are arranged into a 2D lattice which inherits the layout from the rectangular image pixel lattice. We will coordinate it by (r, c) with pixel coordinates $(x, y) = \gamma_r(r, c)$. There are up to 10 points on each ray (r, c). Therefore, we use labels $x_s \in \{1, ..., 10\}$ to select patches along the ray and use label 0 to select no patch. The cost of a particular labeling $x = \{x_s\}$ can be written as

$$E(x|\theta) = \sum_s \theta_s(x_s) + \sum_{st} \theta_{st}(x_s, x_t) \tag{6.1}$$

where unary costs $\theta_s(x_s)$ are constants c_0 ($c_0 = 1$ in all our experiments) for $x_s = 0$ and $-\frac{q(p)+1}{2}$ for non-zero x_s.

The pairwise costs $\theta_{st}(x_s, x_t)$ are defined as follows: $\theta_{st}(0, 0) = e_{00}$ and $\theta_{st}(0, i) = e_0$ for labels $i \neq 0$, where e_{00} and e_0 are conveniently defined constants ($e_{00} = 0$ and $e_0 = 1$ in all our experiments). To define $\theta_{st}(i, j)$ for labels $i, j \neq 0$, we compute $\delta_{ij} = \frac{d_{ij}}{5\,d_r(\mathbf{X}_2)}$, where d_{ij} is the distance of the centers of patches i, j and $d_r(\mathbf{X}_2)$ is the size of the spatial detail at the center of the (closer) patch j. Then, we set $\theta_{st}(i, j) = \delta_{ij}$ if $\delta_{ij} \leq 1$ and the angle between the normals of the patches is less than 60°. We set $\theta_{st}(i, j) = \infty$ otherwise.

To obtain the filtered surface, which we store in F_r, we need to determine the optimal labeling $\hat{x}$ such that

$$E(\hat{x}|\theta) = \arg\min_x E(x|\theta) \tag{6.2}$$

by a large-scale MRF optimization.

Solving Large-scale MRF optimization We consider a pairwise Gibbs energy minimization problem for discontinuous surface reconstruction (6.2). There are methods developed, *e.g.* [16], to find global solution for submodular instances, even large-scale, which do not fit into memory. However, they do not apply here. The problem (6.2) is of general type and cannot be solved efficiently to global optimality. For our purposes it is sufficient to have just a good suboptimal solution as in other major parts of the pipeline. We apply TRW-S algorithm [86, 49]. We use the following heuristic to handle large-scale problems by parts. To get solution inside a window A we take a larger set $B \supset A$ covering all the terms, say, not farther that 50 nodes from A. We solve the "small-size problem" in B, then fix the solution inside A and process to the next window. The intuition is that the solution, *e.g.* in the upper left conner of a large image, does not really depend on the data in the lower right conner. Each next window is conditioned on the solution already fixed in all previously processed windows, so there will be no seems in the ambiguous places. Experimentally, we found that this approach gives a solution which agrees well with the solution by TRW-S applied globally. In fact, solving by parts often yields a lower energy (which is possible because TRW-S is a suboptimal method itself). Let us also note that any available method for general MRFs could be used for solving the problem by parts.

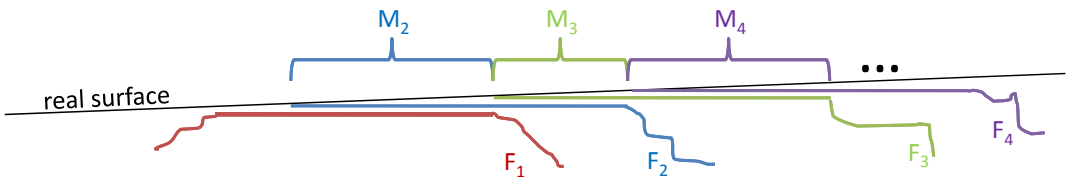
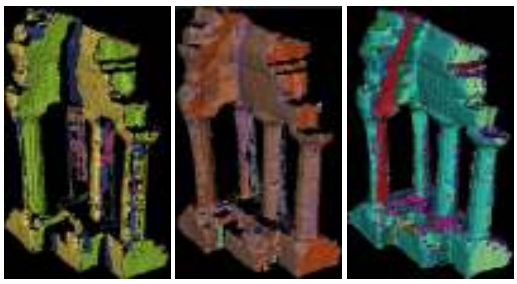


Figure 6.4.: **Meshing:** F_i – filtered meshes, M_i – accepted meshes. A consistent overlap of several (2 or 3 in our experiments) filtered meshes forms an accepted mesh.

Meshing Meshing is one of the crucial steps in our pipeline. Figure 6.4 illustrates the meshing step. Our final mesh consists of at most n (the number of cameras) parts. Each camera is associated with it's part, which is stored in a file. Therefore, we do not need to remember all reconstructed parts in one file. We detect overlapping parts of the previously filtered meshes (from the O_r) with actual (reference) filtered mesh. If a part of the actual filtered mesh is supported by at least K_a (2 or 3 in our experiments) other filtered meshes, then it is considered as an accepted mesh for the reference camera. In more details, we update the part M_r of the final mesh M, which is associated with the reference image I_r. We accept the patch $p = (\mathbf{X}, \mathbf{n}, r, \Sigma)$ of $F_r(x, y)$ on the ray (x, y) into $M_r(x, y)$ if there are at least K_a (2 or 3 in our experiments) filtered patches computed from at least K_a other reference cameras in the ball $3\, d_r(X)$.



(T-16) (T-47) (T-311)

Figure 6.5.: **Middlebury Temple data set reconstructions.** I_r-colored 3D meshes: (T-16) 16 images, 6 mintes, (T-47) 47 images, 20 minutes, (T-311) 311 images, 58 minutes.

Since meshes F_i partly overlap and growing is restricted by V_r, meshes M_j form a contiguous mesh, see Figs 6.4, 6.5, 6.8 (a), 6.10 (c) and 6.7 (b).

Computational effort & memory requirements In the filtering step, we process one G_r at a time. However, the number of variables and edge weights in the MRF problem becomes 11^2 times the number of nodes, which is too large to fit in memory for large images. Our block MRF optimization provides an acceptable suboptimal solution with negligible overhead. In the meshing step, we need to keep in memory only two or three different filtered meshes F_r and one M_r.

It is important to realize that we optimize only those parts of G_r which indeed have grown. The amount of this data is often decreasing as we proceed in reconstructing a scene from a large number of overlapping images since more and more images are covered by accepted points of M_r, therefore less image area is grown and so less nodes of G_r are optimized.

6.2. Results

To evaluate the quality of reconstructions, we present results on data sets from the standard evaluation Middlebury [68] and Strecha's [77] databases. To demonstrate the scalability of our algorithm we show reconstructions on the large Castle data set from [77] on a 294 image Marseille data set and on a 1000 image Google Street View Pittsburgh Experimental data set [2].

Figure 6.5 shows reconstructions for the three Temple data sets containing 16, 47, resp. 311 640×480 images of the Middlebury Temple. See the Middlebury evaluation page for Jancosek-3DIM09 results and their comparison. We conclude that our results are comparable to other state-of-the-art techniques. When comparing computation times on, e.g., Temple 311, we see that our method is comparatively faster as it takes 49 minutes, compared to method [21] taking 4 hours, [25] 81 hours, and [9] 6 hours.

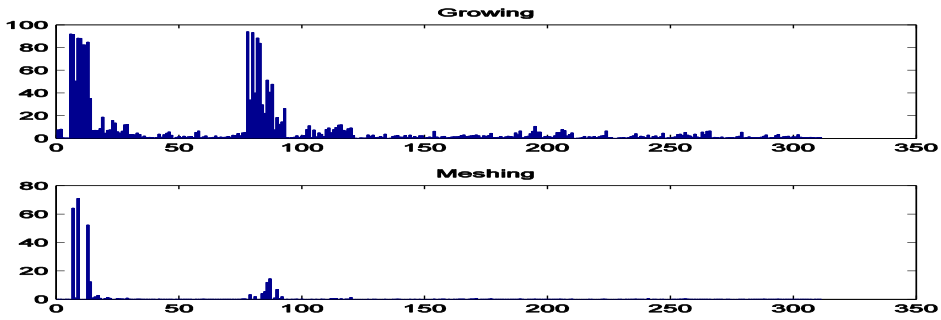
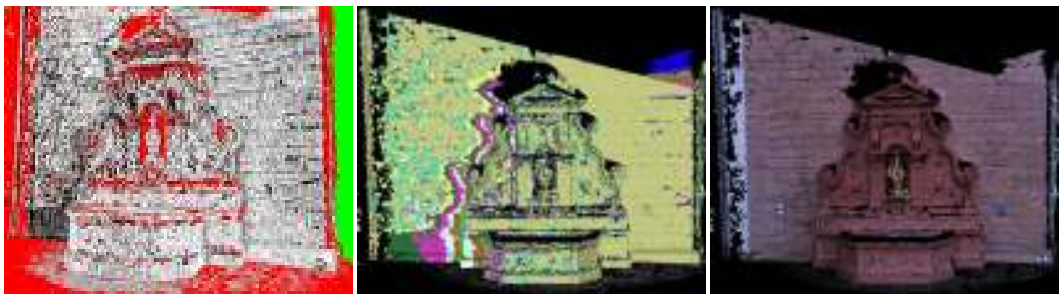


Figure 6.6.: **The fraction of pixels of the reprojection of the final mesh to I_r.** Processed in the growing (Growing) and meshing (Meshing) for Temple 311 data set. Cameras are shown on the horizontal axes in the order of processing. The fraction in % is shown on the vertical axes.



(a) (b) (c)

Figure 6.7.: **Strecha's Fountain-P11 data set reconstructions:** 11 3072 × 2048 images, 219 minutes. (a) Residuals w.r.t. ground truth (light colors represent smaller errors) (b) I_r-colored 3D mesh (c) textured 3D mesh.

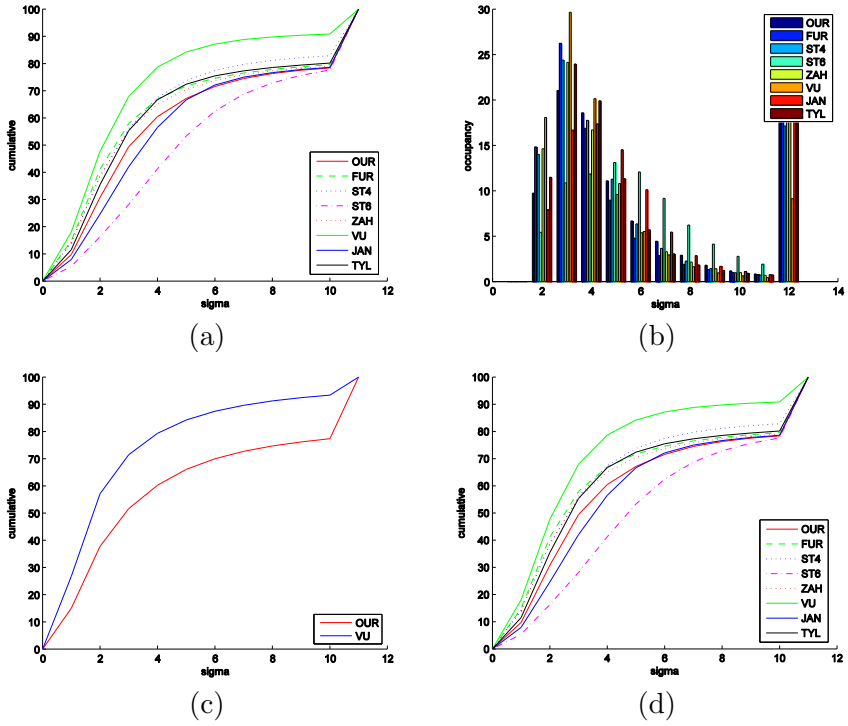
Figure 6.6 shows the fraction of pixels of the reprojection of the final mesh to I_r processed in the growing (Growing) and meshing (Meshing) for Temple 311 data set. There are two cluster of cameras (5-15) and (80-100) which support large part of the final mesh in their M_r buffers. We see that growing in other cameras was greatly reduced by using previous reconstructions in V_r.

Figure 6.7 shows our reconstructions of Strecha's Fountain-P11 data set containing 11 3072 × 2048 images. Figure 6.8 shows reconstructions of the Strecha's Castle-P30 data set containing 30 3072 × 2048 images.

Figure 6.9 shows the evaluation on the Strecha's Fountain-P11 as well as Castle-P30 data sets. See the Strecha's evaluation page for JAN09 results and their comparison. Our method is comparable to other methods on the Fountain-P11 set, although it reconstructs

(a) (b) (c)

Figure 6.8.: **Strecha's Castle-P30 dataset reconstructions:** 30 3072×2048 images, 368 minutes. (a) I_r-colored 3D mesh (b) textured 3D mesh (c) textured 3D mesh cutout.



(a) (b)

(c) (d)

Figure 6.9.: **Strecha's evaluation [77] for the Fountain-P11 (a,b) and Castle-P30 (c,d) data sets.** OUR - proposed method, FUR [21], ST4 [74], ST6 [75], ZAH [95], VU [85], JAN segmentation based method described in previous Chapter 5, TYL [83].

less points than the best method [85]. There is still room for improvement since we obtained only about 75% of points at sigma = 10 on Castle-P30 dataset.

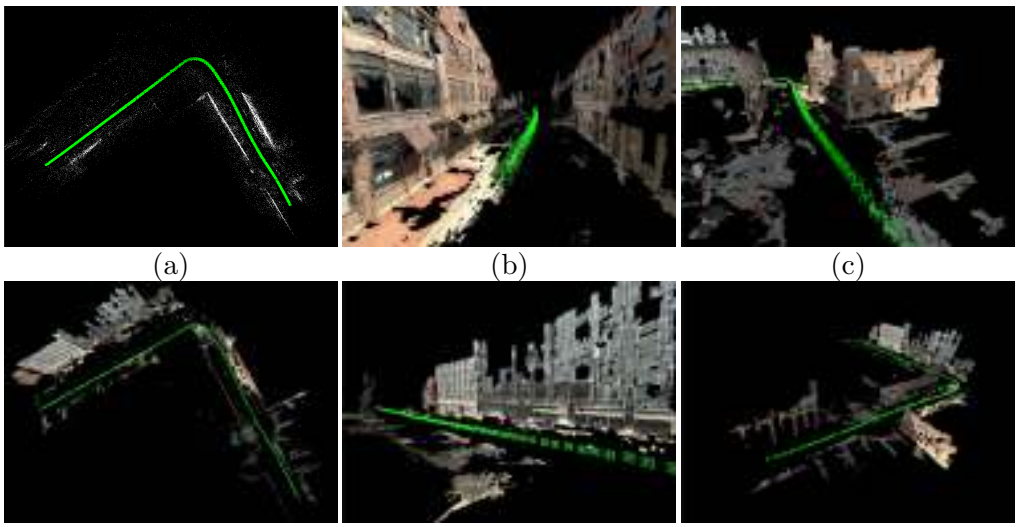


Figure 6.10.: **Google Street Views Pittsburg Experimental dataset reconstruction of 1000** 600×800 **images in 183 minutes:** (a) 3D seeds and cameras, (b–f) five different views.

Figure 6.1 shows our reconstruction of Marseille dataset containing 294 (1296×972 pixels) unorganized images computed in 92 minutes. Figure 6.1(b,c) shows textured and untextured 3D reconstruction from a general viewpoint. Cameras and sparse 3D structure, Figure 6.1(a), have been computed by [71]. Figure 6.1(d) shows reconstructed 3D structure I_r-colored to show what has been reconstructed from which central camera.

Figure 6.10 shows reconstruction of a 1000 image data set from the Google Street View Pittsburgh Experimental data set [2]. We have chosen a sequence of 200 viewpoints represented by five-tuples of perspective 600×800 images capturing complete 360° field of view. The density as well as the precision of the reconstruction varies across the scene. There are still many holes and the accuracy is not very high. This might be caused by incorrect calibrations, which were guessed from the viewing angle of the photographs, as well as distortions introduced by the process of acquisition and processing of image panoramas when making the data set.

Nevertheless, the experiments demonstrates that we were able to process a large image datasets in affordable time and thus we could aim at reconstructing city parts.

Part II.

Global volumetric

7 Weakly-supported surfaces reveal themselves through occlusion

Surfaces that do not have strong support in the input points but represent real surfaces in the scene, i.e, *weakly-supported surfaces*, are essential for achieving complete reconstructions. In this chapter we propose a novel method that can reconstruct these difficult weakly-supported surfaces.

The main idea of improving the reconstruction of weakly-supported surfaces is that even weakly-supported surfaces exhibit themselves by occluding other input points. We next review this idea for an infinite number of noiseless and outlier-less input points augmented with visibility information.

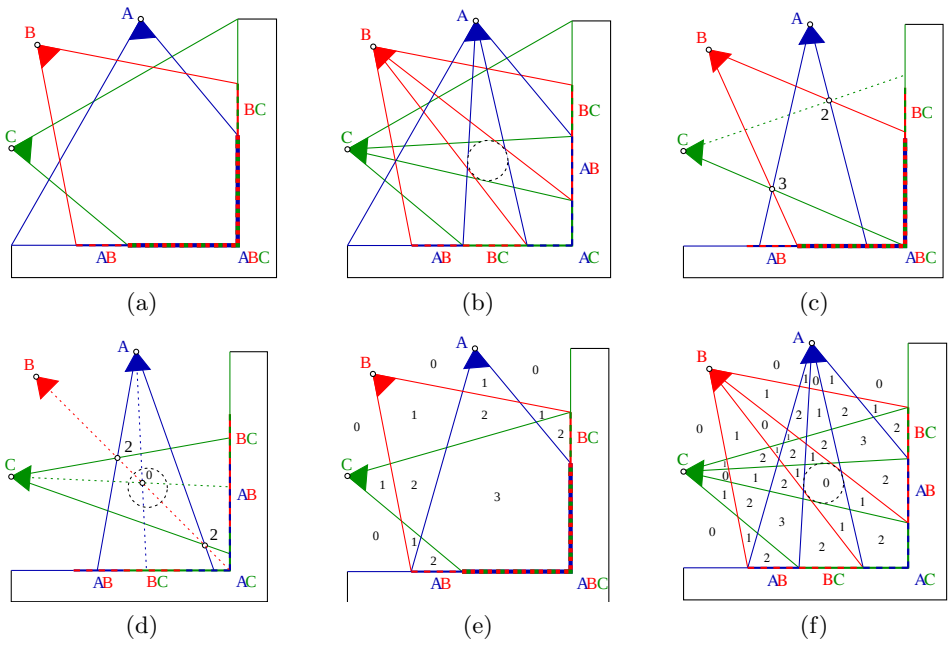


Figure 7.1.: **Occluder reveals itself through occlusion.** Three sensors with centers A, B, C, shown as blue, red, and green wedges, observe an L-shaped (black) object. Each ray r from a sensor center O to a reconstructed point X indicates that the space along the line segment OX is free. See the text for a more detailed explanation.

Figure 7.1(a) shows a two-dimensional schematic illustration of three sensors (blue, green, and red wedges) with their respective fields of view. The sensors observe the corner of an L-shaped object. Some points of the object are not observed by any sensor (shown as the solid black line), some are observed by one sensor, some by two sensors (line segments marked by AB and by BC), and some by all three sensors (corner marked by ABC). For simplicity, we assume that all points of the L-shaped object that are seen by at least two sensors are on the input. Note that we are assuming continuous case in this example. Therefore, the number of input points is infinite.

Let us next add a circular object as an occluder to the scene, Figure 7.1(b), which has no point in the input points. We can still reconstruct the same surface of L-shaped object as in Figure 7.1(a), but all points are now visible from only (and exactly) two sensors. It is important to notice here that while the set of input points has not changed, the visibility information of the input points has changed. The visibility information has changed in the way that an input point that is occluded by the occluder is not seen by the sensors in which it is occluded.

Let us now introduce a new concept that conveniently captures the effect of having an occluder in the scene. For each point in the space we can construct a measure of its emptiness. We will call this measure *free space support*. Consider the point marked by "3" in Figure 7.1(c). The emptiness of this point is supported by three observations. There are three line segments passing through this point that connect the sensor centers A, B, C with the respective input points. Hence, the free space support of the point is equal to three. The emptiness of the point marked by "2" is supported only by two observations. The ray from sensor C through the point 2 does not end in an input point. It ends in a point seen by only one sensor: sensor C. Hence the free space support of this point is equal to two. Note that there are only points that are seen by at least two sensors are on the input.

After the introduction of the occluder into the scene, Figure 7.1(d), the free space support decreases at some points because some of the rays get blocked by the occluder.

Figure 7.1(e) shows the space partitioned by sensor visibility cones into regions with constant free space support, which is denoted by black numbers. Figure 7.1(f) shows the same after introducing the occluder. We see that a region with zero free space support surrounded by non-zero free space support emerged in the scene center. Such a region provides evidence of an occluder even when no point on the occluder has been on the input.

We can consider the space with non-zero free-space-support as free and the complement as full. Therefore the evidence of the occluder interface can be detected by nonzero to zero change of the free-space-support in this ideal noiseless, outlier-less, and continuous case. Additionally, the L-shaped object has zero free-space-support and nonzero to zero change is evidence for the L-shaped object interface as well. Therefore, we can say that nonzero to zero change of the free-space-support is interface evidence for all objects.

7.1. Hallucinations

Figure 7.1(f) shows that there is another nonzero to zero change of the free-space-support that is not an evidence for any object. It is the zero to nonzero change of the free-space-support at the boundaries of visibility cones. We call these types of evidences *hallucinations*. Hallucinations cause problems in datasets where the scene is not captured from all sides. We refer the reader to the Chapter 10 for more information on solving this problem.

7.2. Space discretization by Delaunay tetrahedralization

It is impossible to compute the free-space-support for all points of the space. To be practical, the space is discretized into tetrahedra constructed by the Delaunay tetrahedralization [15] and the free-space-support is evaluated for the tetrahedra. The space discretization is constructed by tetrahedralizing the *union of the input points and sensor centers*. By having the sensor centers in the tetrahedralization, the space between the sensors centers and the input points is covered by the tetrahedralization.

The interface evidence can be easily detected by nonzero to zero change of the free-space-support in ideal noiseless, outliers-less and continuous case. In real-world scenarios however, the number of input points is finite and the amount of noise and outliers can be significant. Nevertheless, we experimentally show in the next Chapter 8 that the interface of a scene object, weakly-supported as well as strongly supported by input points, can be detected by measuring large free-space-support change of nearby tetrahedra. Let us introduce the notation first.

Tetrahedralization We denote a tetrahedron of the delaunay tetrahedralization of the input points as T.

The unit σ We introduce a constant σ that equals 2.0 times the median of the lengths of all edges of all tetrahedra. The value σ corresponds to the smallest reconstructible object.

Input points and segments (lines) of sight We assume that we have on the input a set C of sensor centers $\mathbf{c} \in C$ and a set S of pairs $(\mathbf{c}, \mathbf{p}) \in S$ where $\mathbf{c} \in C$ and $\mathbf{p}$ is an input 3D point. The set S is a set of input points augmented with the visibility information. We call the line segment defined by 3D points $(\mathbf{c}, \mathbf{p}) \in S$ the segment of sight or simply the segment. We call the line defined by 3D points $(\mathbf{c}, \mathbf{p}) \in S$ the line of sight. We call input points augmented with visibility information, i.e. the set S, the input segments of sight.

The set of input points is denoted as $P(S) = \{\mathbf{p} | (\mathbf{c}, \mathbf{p}) \in S\}$. Note that one input point $\mathbf{p}$ can be augmented (and typically is) with multiple sensor centers.

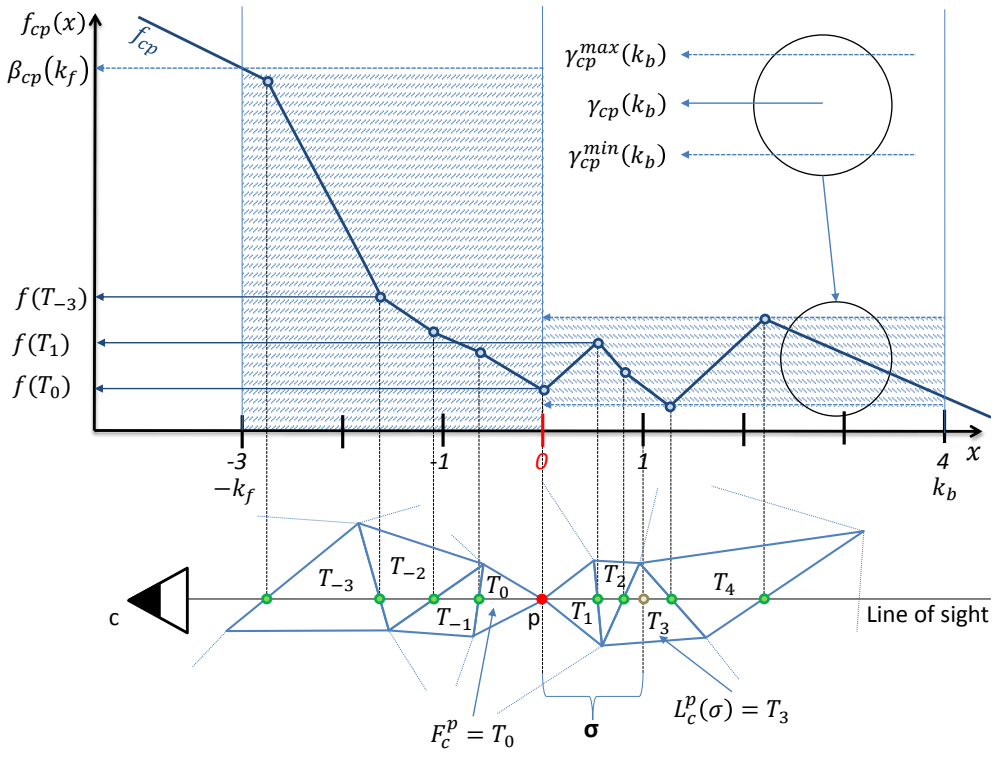


Figure 7.2.: **Free-space-support on line of sight.** Illustration of functions $f_{cp}(x)$, $\beta_{cp}(k)$ and $\gamma_{cp}(k)$. See text for more details.

Input point weights We assign to each of the input points $\mathbf{p}$ a weight denoted by $\alpha(\mathbf{p})$. The weight $\alpha(\mathbf{p}) \geq 1$ reflects the amount of input points in σ surrounding of the point $\mathbf{p}$.

Free-space-support Free space support f is a measure of emptiness. Roughly speaking, if a sensor with center $\mathbf{c}$ sees a point $\mathbf{p}$, then the segment of sight $(\mathbf{c}, \mathbf{p}) \in S$ supports the emptiness of space between $\mathbf{c}$ and $\mathbf{p}$. The free-space-support $f(T)$ of tetrahedron T is computed from segments $(\mathbf{c}, \mathbf{p}) \in S$ intersecting T as

$$\begin{aligned} f(T) &= \sum_{(\mathbf{c},\mathbf{p}) \in S_T} \alpha(\mathbf{p}) \quad \text{with} \\ S_T &= \{(\mathbf{c}, \mathbf{p}) \in S \,|\, (\mathbf{c}, \mathbf{p}) \cap T \neq \emptyset\}. \end{aligned} \tag{7.1}$$

where $(\mathbf{c}, \mathbf{p}) \cap T$ denotes the intersection of the line of sight $(\mathbf{c}, \mathbf{p})$ and the tetrahedron T.

Free-space-support on a line of sight We denote the free-space-support on a line of sight $(\mathbf{c}, \mathbf{p}) \in S$ by f_{cp}. Figure 7.2 illustrates how the free-space-support on a line of

sight is computed. The function $f_{cp}(x)$ is a piece-wise linear function that interpolates the free space supports $f(T_j)$ of tetrahedra T_j crossed by the line of sight $(\mathbf{c}, \mathbf{p}) \in S$ (bottom). Notice that the domain of the function f_{cp} is in units σ. We introduce the following notations to measure extremal values of the function f_{cp} on specific intervals:

$$\beta_{cp}(k) = max(f_{cp}(x)|x \in \langle -k, 0\rangle) \tag{7.2}$$

$$\gamma_{cp}(k) = (max(f_{cp}(x)|x \in \langle 0, k\rangle) + \tag{7.3}$$

$$min(f_{cp}(x)|x \in \langle 0, k\rangle))/2 \tag{7.4}$$

The function $\beta_{cp}(k)$ measures the maximal value of $f_{cp}(x)$ in range $\langle -k, 0\rangle$, i.e. in $k\sigma$ distance from the point $\mathbf{p}$ toward the sensor c. The function $\gamma_{cp}(k)$ measures the average of maximal and minimal values of $f_{cp}(x)$ in range $\langle 0, k\rangle$, i.e. in $k\sigma$ distance from the point $\mathbf{p}$ away from the sensor c. See Figure 7.2.

Free-space-support jump We define the following functions to be able to measure free-space-support change in a close surrounding of an input point.

$$\epsilon_{cp}^{abs}(k_f, k_b) = \beta_{cp}(k_f) - \gamma_{cp}(k_b) \tag{7.5}$$

$$\epsilon_{cp}^{rel}(k_f, k_b) = \frac{\gamma_{cp}(k_b)}{\beta_{cp}(k_f)} \tag{7.6}$$

The function ϵ_{cp}^{abs} evaluates the absolute change of the free-space-support on the line of sight $(\mathbf{c}, \mathbf{p})$ in the $\langle -k_f\sigma, k_b\sigma\rangle$ surrounding of the input point $\mathbf{p}$. The function ϵ_{cp}^{rel} evaluates a relative change of the same free-space-support.

The functions are chosen this way because we want to be able to detect big change in free space evidence near some 3D point. However, measuring just the relative change of the free-space-support is not enough because relative change of small (noise) numbers can be also significant. Therefore we study a combination of the absolute and relative changes.

Outliers level If a point $\mathbf{p}$ represents a real interface point then the value of the function γ_{cp} represents the amount of outlier lines of sight $(\mathbf{c}, \mathbf{p}) \in S$ crossing the corresponding part of full space. The function reflects average free space evidence accumulated in the space that should be occupied by an object. Only an outlier that is seen by a sensor but should be occluded by the object can contribute to this evidence, therefore the function γ_{cp} reflects the outliers level.

The first and the last crossing tetrahedron The first crossing tetrahedron of a line segment $(\mathbf{c}, \mathbf{p})$ is denoted by F_c^p. It is the tetrahedron containing point $\mathbf{p}$, the tetrahedron T_0 on Figure 7.2. We denote as $L_c^p(d)$ the tetrahedron ($L_c^p(\sigma) = T_3$ on Figure 7.2) that contains the point $\mathbf{p} + d\overrightarrow{(\mathbf{c}, \mathbf{p})} \in L_c^p(d)$ (gray point for $d = \sigma$ on Figure 7.2).

7.3. Finding surfaces by solving a graph optimization problem

Let us now introduce an efficient graph representation of the input points and formulate the surface reconstruction problem as a graph optimization problem.

7.3.1. Discretization graph

The Delaunay tetrahedralization of the input points induces its dual Voronoi graph [15] (V_d, E_d). Vertices V_d of the graph correspond to the tetrahedra. Edges E_d of the graph correspond to the faces of the tetrahedra. There is exactly one edge in E_d for every face of the tetrahedralization.

We use the Voronoi graph to construct the *Discretization graph (DG)* (V, E). First, a set E_v of *v-edges* is constructed by orienting the edges of E_d and augmenting them with normal vectors of the faces. For every edge (v_1, v_2) in E_d we add two edges, $(v_1, v_2, \mathbf{n}_{12})$ and $(v_2, v_1, \mathbf{n}_{21})$, to E_v, where $\mathbf{n}_{12}$ is the normal of the face between v_1, v_2 oriented from v_1 to v_2, $\mathbf{n}_{21} = -\mathbf{n}_{12}$. Secondly, a set $V = V_d \cup \{s, t\}$ of vertices is constructed by adding two additional vertices s, t to V_d. Finally, a set $E_s = \{(s, v) \,|\, v \in V_d\}$ of *s-edges* oriented from s to the vertices in V_d and a set $E_t = \{(v, t) \,|\, v \in V_d\}$ of *t-edges* oriented from the vertices in V_d to t are built. By putting all edges together, we get $E = E_v \cup E_s \cup E_t$.

We denote tetrahedron T that corresponds to a node $v \in V_d$ as $T(v)$. We denote oriented face F of the tetrahedralization that corresponds to an oriented v-edge $e \in E_v$ by $F(e)$.

7.3.2. Surface reconstruction using minimal s-t cut of the DG

Given the graph DG and weights $w(e)$ of all of its edges $e \in E$ an $s - t - cut$ $C(\varsigma, \tau)$ is a partition of V into two disjoint sets ς and τ where $s \in \varsigma$ and $t \in \tau$. The $C(\varsigma, \tau)$ is defined as follows:

$$C(\varsigma, \tau) = \sum_{e=(u,v)|u,v \in \varsigma \setminus \{s\}} w(e) + \sum_{e=(s,v)|v \in \varsigma \setminus \{s\}} w(e) + \sum_{e=(v,t)|v \in \tau \setminus \{t\}} w(e) \tag{7.7}$$

The minimum s-t-cut optimization problem is the minimization of the $s-t-cut$ value $C(\varsigma, \tau)$. There are many classical algorithms that efficiently solve this problem [7].

We cast the surface reconstruction problem as the minimal $s - t - cut$ problem of the graph DG. We interpret the final sets ς and τ as that tetrahedra $T(v)|v \in \varsigma$ are labeled as being free and tetrahedra $T(v)|v \in \tau$ are labeled as being full. The final surface is then reconstructed as the union of oriented faces (triangles) of the DG that are on the full-free interface. The surface is therefore guaranteed to bind a volume, i.e., to be watertight and self non-intersecting.

7.3.3. Computing edges weights of the DG

The method proposed in [52] shows that the surface separating the free space from the full space can be found as the minimal $s-t-cut$ of the DG where weights are computed as follows:

Weights of v-edges Weight $w(e)$ of an oriented v-edge $e = (u, v) \in E_v$ is set to

$$\begin{aligned} w(e) &= \sum_{(\mathbf{c},\mathbf{p}) \in S_v(e)} \alpha(\mathbf{p}) \quad \text{with} \\ S_v(e) &= \{(\mathbf{c}, \mathbf{p}) \in S \,|\, (\mathbf{c}, \mathbf{p} + \sigma \overrightarrow{(\mathbf{c}, \mathbf{p})}) \cap F(e) \neq \emptyset \\ &\qquad \wedge \overrightarrow{(\mathbf{c}, \mathbf{p})} \cdot \mathbf{n} > 0\} \end{aligned} \tag{7.8}$$

Weights of s-edges Weights $w(e)$ of s-edges $e = (s, v) \in E_s$ are set as follows: $w(e) = \infty$ if a sensor center is one of the vertices of $T(v)$, otherwise $w(e) = 0$.

Weights of t-edges Weight $w(e)$ of an t-edge $e = (v, t) \in E_t$ is set to

$$\begin{aligned} w(e) &= \sum_{(\mathbf{c},\mathbf{p}) \in S_t(v)} \alpha(\mathbf{p}) \quad \text{with} \\ S_t(v) &= \{(\mathbf{c}, \mathbf{p}) \in S \,|\, p \in L_c^p(\sigma) \wedge L_c^p(\sigma) = T(v)\} \end{aligned} \tag{7.9}$$

where $L_c^p(\sigma)$ is the last crossing tetrahedron at σ distance from the point $\mathbf{p}$ backward from the sensor c defined in section 7.2.

The weight of t-edge as non-emptiness support of tetrahedron

The weight of an t-edge $e = (v, t) \in E_t$ can be considered as non-emptiness support of tetrahedron $T(v)$. It is computed from segments of sight $(\mathbf{c}, \mathbf{p}) \in S$, which when extended by σ behind the point $\mathbf{p}$, end inside of the tetrahedron $T(v)$ i.e., it reflects the number of input points that are close to $T(v)$ and additionally that "occlude" $T(v)$ in a sensor.

The difference between the free-space-support of a tetrahedron and the v-edge weight

In [52] free-space-support $f(T)$ of tetrahedron T is not defined. However, there is a relation between v-edge weight $w(e)$ of oriented v-edge $e = (u, v) \in E_v$ and the free-space-support $f(T(u))$ of the tetrahedron $T(u)$. Let us denote the sum of the weights of all (four) incoming edges to the node u as:

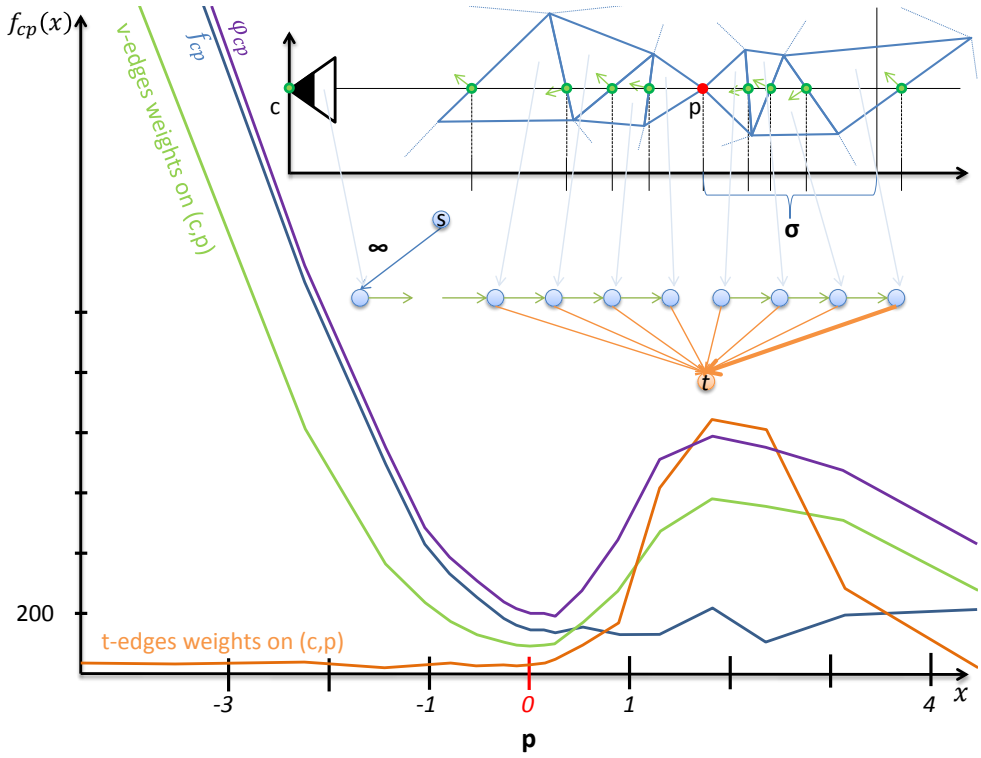


Figure 7.3.: **DG weights.** Illustration of DG related to thetrahedralization and the difference between the weight of oriented v-edges, free-space-support f and φ on line of sight $(\mathbf{c}, \mathbf{p}) \in S$. The point $\mathbf{p}$ $(x = 0)$ is a point of ground-truth strongly supported surface. See text for more details.

$$\varphi(T(u)) \quad = \quad \varphi(u) = \sum_{e=(u,v)\in E_v} w(e) \tag{7.10}$$

If σ equals zero then $f(T(u)) = \varphi(T(u))$, i.e., if $\sigma = 0$ then the free-space-support of the tetrahedron $T(u)$ equals the sum of the weights of all the incoming edges to the node u.

We define $\varphi_{cp}(x)$ on the line of sight $(\mathbf{c}, \mathbf{p}) \in S$ for the function φ in the same way as is defined the free-space-support weight $f_{cp}(x)$ on the line of sight $(\mathbf{c}, \mathbf{p}) \in S$ for the function f, in order to be able to visualize the difference between φ and f, see Figure 7.3.

Figure 7.3 illustrates the difference between the weight of the oriented v-edge $e = (u, v) \in E_v$ and the free-space-support $f(T(u))$ of the tetrahedron $T(u)$. The parameter σ is what makes the difference. The free-space-support f is in this case monotonically decreasing in front of $\mathbf{p}$ ($x = 0$ represents the $\mathbf{p}$) and almost constant (near zero) behind the $\mathbf{p}$. The weights of oriented v-edges are similar to the free-space-support of tetrahedra in front of $\mathbf{p}$ but have a small peak behind $\mathbf{p}$. This peak is caused by the parameter σ in the equation 7.8.

The minimum s-t cut optimization tends to label tetrahedra with large free-space-support as free and simultaneously it tends to label tetrahedra with large non-emptiness support as full, while the optimal cut of v-edges tends to occur on the v-edge with minimum weight in front of the point with maximal value of the non-emptiness support.

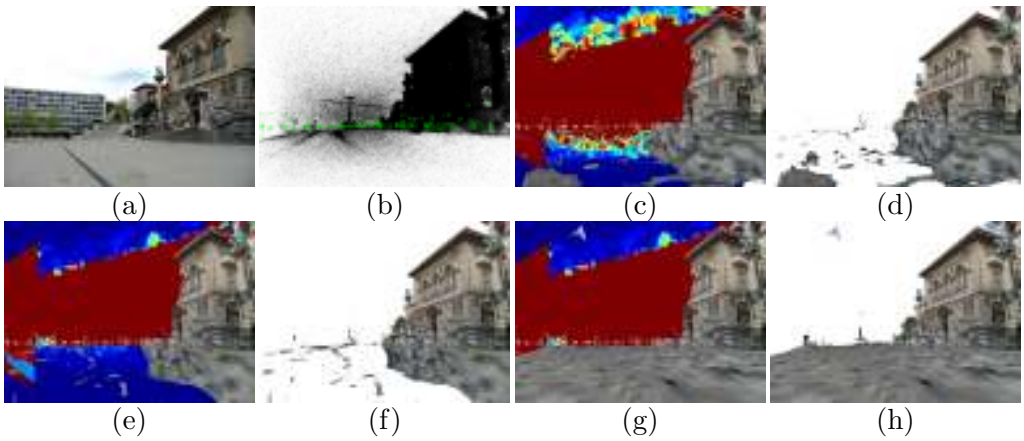


Figure 7.4.: **Results for the 'strecha' data set.** (a) Input image, (b) input 3D point cloud, (c) the free-space-support of the tetrahedra on a cut of the tetrahedralization with a plane using $\alpha_{vis} = 32$ as in the base-line method, (e), (g) the same but (e) using $\alpha_{vis}(\mathbf{p})$ instead and (g) our method. The free-space-support is different. (d),(f) are reconstructions of our implementation of the base-line method, (d) using $\alpha_{vis} = 32$, (f) using $\alpha_{vis}(\mathbf{p})$. The results are similar. (h) is the result using our method. (d), (f) demonstrate that the introduction of $\alpha_{vis}(\mathbf{p})$ into the base-line method is not sufficient to reconstruct weakly-supported ground plane. (h), (g) shows that our method, reconstructs the weakly-supported ground plane. The jet-color-map was used for the range $\langle 0, 1000 \rangle$, where the blue color corresponds to 0 and the dark red color corresponds to 1000.

7.4. Observation based approach to weakly-supported surfaces reconstruction

Figure 7.4 shows the influence of the $\alpha(\mathbf{p})$ on computation the free-space-support values of the tetrahedra. Figure 7.4 (c) shows the free-space-support values of the tetrahedra on a cut of the tetrahedralization with a plane using $\alpha(\mathbf{p}) := 32$ as in the base-line method and (e) using the adaptive $\alpha(\mathbf{p})$. When using adaptive $\alpha(\mathbf{p})$ the free-space-support describes visually the weakly-supported ground plane better. Nevertheless, in both cases, it is not enough to reconstruct the weakly-supported ground plane. We provide a synthetic example in next Section 7.4.3 in order to be able to explain the reason. We also show in the synthetic example how to modify STG t-weights so that the method described in the Section 7.3.2 gains ability to reconstruct the weakly-supported surfaces.

Le us introduce two observation based assumptions in order to be able to continue.

7.4.1. Large free-space-support jump assumption

The free-space-support of a tetrahedron, which is near or lies on the real surface, and should be labeled as outside, should be much larger than free-space-support of a nearby tetrahedron which should be labeled as inside. We assume that this holds for surfaces which are densely sampled by the input point cloud as well as for weakly-supported surfaces.

This assumption is well reviewed in the beginning of this chapter for ideal case of an infinite number of noiseless and outlier-less input points augmented with visibility information.

Furthermore, Figure 7.4(e) shows that it indeed holds. The free-space-support jump is large on both the densely sampled facade (because the facade is reconstructed) as well as on the weakly-supported ground plane.

7.4.2. Weakly-supported surfaces t-weight assumption

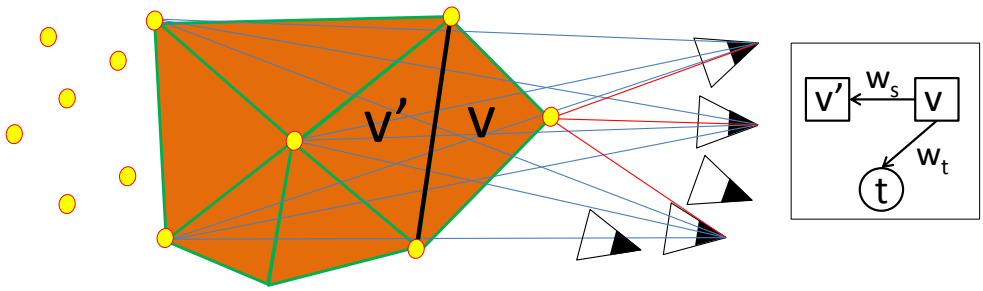


Figure 7.5.: **Weakly-supported surfaces t-weight example.** The black face occludes 3 points in 3 cameras. Therefore the corresponding edge has weight $w_s = 9$. The corresponding tetrahedron v, which is nearest to the cameras is behind just one point, which has 3 cameras associated (and ray defined by the camera center and the point intersects v). Hence the corresponding t-edge has weight $w_t = 3$. Furthermore, each point, which is occluded by the black face in c associated cameras, increases the w_s by the number c while w_t remains unchanged.

We assume that the t-edge weights for tetrahedra, which are near or lie on the real but weakly-supported surface and should be labeled as inside, are much smaller than the free-space-supports of the corresponding nodes. Let us assume that $\sigma = 0$ and denote a tetrahedron which lies on the real but weakly-supported surface, and which should be labeled as inside, by v. In this situation, the t-edge weight w_t of the tetrahedron v depends on the number of cameras associated with the four points of the tetrahedron v. The free-space-support w_s of the tetrahedron v depends on the number of cameras of all points which are occluded (in the cameras) by the tetrahedron v. Therefore, even a small number of wrongly reconstructed points makes the value w_s much greater than the value w_t (see Figure 7.5). While we assumed $\sigma = 0$ the conclusion holds for small σ, too. The parameter σ can not be set to large value (see [52]).

7.4.3. Synthetic example

The top parts of Figures 7.6(a), (b) illustrates the situation when a weakly-supported surface appears as a part of the tetrahedralization. The bottom represents the corresponding graph. The middle represents the weights for the corresponding edges. Weights for edges between nodes are light blue and weights for t-edges are dark brown. There is a large free-space-support jump on the surface; w_{56} is much higher than w_{67} (w_{78}). In this example there holds: $\infty > w_{12} > ... > w_{56} > w_{67}+t_6 > w_{78}+t_6+t_7 > w_{89}+t_6+t_7+t_8 < t_6 + t_7 + t_8 + t_9$ and $w_{56} > w_{57}$. The minimal cut in this synthetic example is the cut illustrated by the red lines. The labeling for the minimal cut does not correspond to the correct solution.

The weakly-supported surfaces t-weight assumption holds: w_{67} is much higher than t_7 and all other t_i are approximately same as t_7, Figure 7.6(a). This causes that the minimal cut will be further from the correct surface. This effect is demonstrated by the real experiment in Figure 7.4. The free-space-support jump is large near the weakly-supported ground plane, 7.4(c),(e) but using $\alpha_{vis}(\mathbf{p})$ alone in the base-line method is not sufficient to reconstruct the weakly-supported-ground plane, Figure 7.4(f). Therefore, in the place of a large jump, we multiply t_7 by a conveniently chosen x so that the cut described in Figure 7.6(b) becomes minimal and we get the correct labeling. One can see that if we set x to $w_{56} - w_{78}$, then we will achieve the correct solution. Our approach is sequential. We first compute all weights in the same way as the base-line approach. Then we search for all large jumps and multiply the corresponding t-edge weights as demonstrated in the example. This is supported experimentally in Figure 7.4(h) by the fact that our method only changes weights of t-edges as described by the synthetic example and it is sufficient to reconstruct the weakly-supported-ground plane.

It is clear that in situation where the amount of noise is significantly lower than the density of surface samples, t-edges weights of the tetrahedra which are in the σ distance to the real surface and are in the real object are greater than the free-space-supports of the corresponding nodes. Otherwise, the base-line method would not give any result. Therefore in this situation our method should give exactly the same result.

7.4.4. Setting up the weights

We denote the first tetrahedron which contains point $\mathbf{p}$ and is related to the nearest *crossing-face* of the line segment $(\mathbf{c}, \mathbf{p})$ as f_p^c. We denote the last tetrahedron which is within distance σ from the point $\mathbf{p}$ crossing the directed line $(\mathbf{p}, \mathbf{p} - \mathbf{c})$ as l_p^c. See Section 7.2.

Our approach to setting up the weights of the graph is sequential. In the first step, we compute all weights of all edges except weights of the t-edges in exactly the same way as in the base-line method but instead of α_{vis} we use $\alpha_{vis}(\mathbf{p})$. In the second step, we compute weights of the t-edges which are updated in a similar way to the base-line method, but with different weights. For each point $\mathbf{p}$ and each associated camera centre

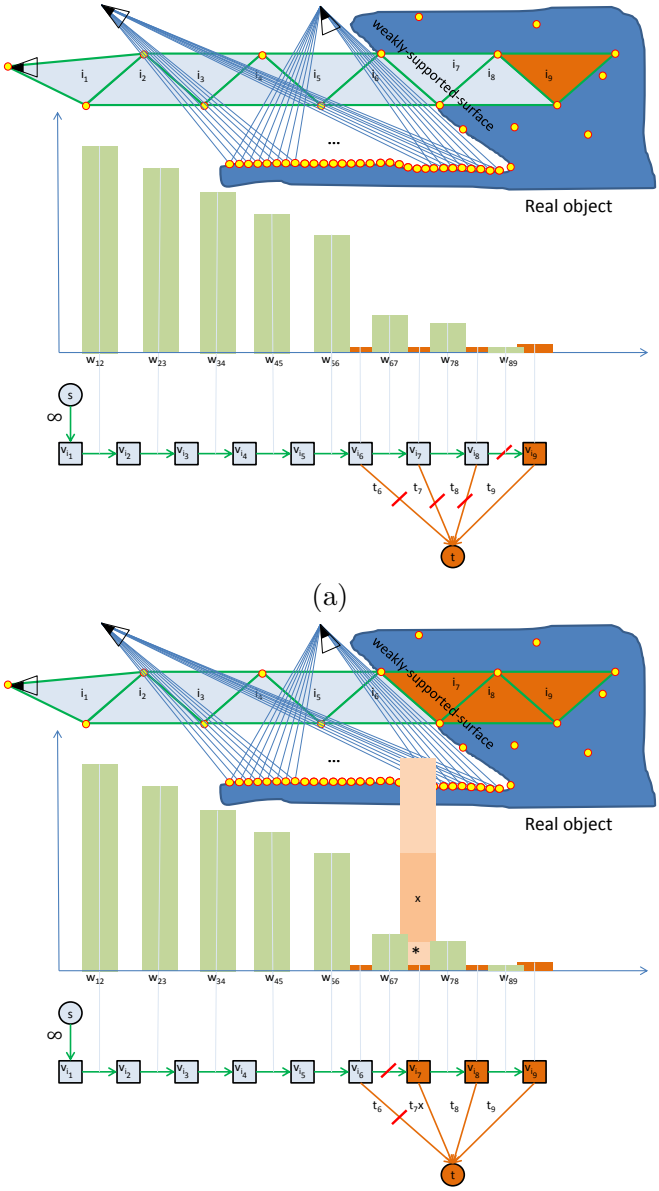


Figure 7.6.: **Representative example.** Light blue: source label (free space), brown: sink label (full space). (a,b) top: a part of the triangulation, bottom: the associated s-t graph, middle: the weights of the associated edges. (a) Minimal cut for weights computed by the base-line approach leads to a wrong solution (b) Multiplying t_7 by $(w_{56} - w_{78})$ leads to the correct solution.

$\mathbf{c}$ we take the free-space-support $e(f_p^c)$ of the f_p^c tetrahedron and the free-space-support $e(l_p^c)$ of the l_p^c tetrahedron. And we add the value $t(\mathbf{c}, \mathbf{p})$

$$t(\mathbf{c}, \mathbf{p}) = \begin{cases} \alpha_{vis}(\mathbf{p})(e(f_p^c) - e(l_p^c)) & \frac{e(l_p^c)}{e(f_p^c)} < \delta \wedge e(f_p^c) < \beta \\ \alpha_{vis}(\mathbf{p}) & \frac{e(l_p^c)}{e(f_p^c)} \geq \delta \vee e(f_p^c) \geq \beta \end{cases} \tag{7.11}$$

to the t-edge of the l_p^c tetrahedron. The parameter δ determines that there is a large jump i.e. when there should be a surface. We use very conservative approach and set $\delta = 0.5, \beta = 1000$ in all of our experiments.

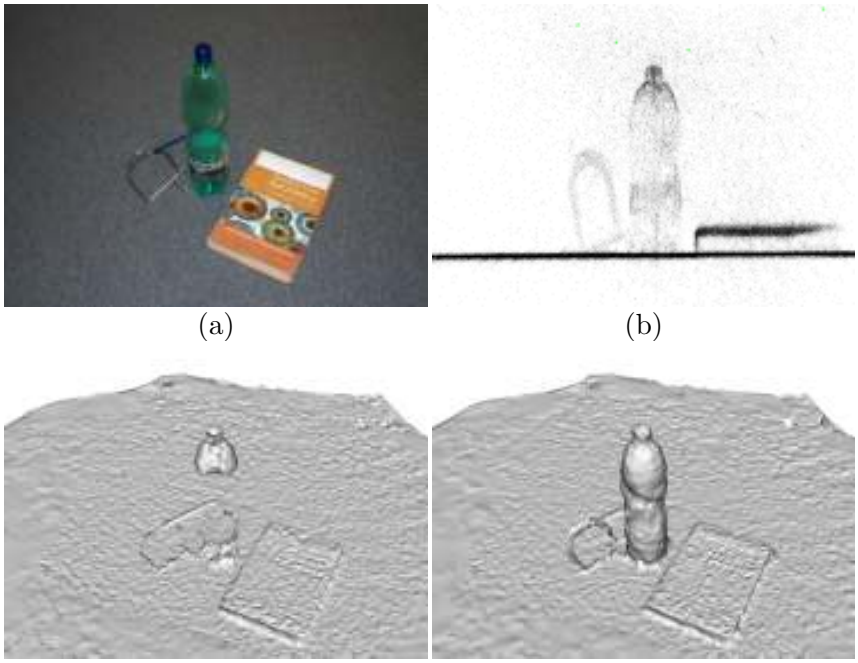
7.5. Results

All experiments were computed on Intel Core i7 CPU machine with nVidia 285GTX graphics card, 12GB RAM and 64-bit Windows 7 OS. We use four different calibrated image sets (data sets) in this chapter: bottle, strecha, castle and dragon. The bottle data set contains 24 1950×1307 images. The street-view data set 'strecha' was provided by Christopher Strecha and contains 1514 2000×1500 images. We compute the depth-map for each of the 1500 images but to reconstruct different parts of the city we choose 3D points from different volumes parts defined by boxes in order to fit into memory. The castle data set is the benchmark data set [77] and contains 30 3072×2048. The dragon data set was used in [44] and contains 114 1936×1296 images. We use the same α_{vis} and σ as in the base-line method. The parameter δ is set to 0.5 in all of our experiments. The parameter γ is set in terms of the two points re-projection pixels distance to a value in the range $\langle 2, 10 \rangle$ in order to be able to fit the data into memory.

We have to note that as our method produces more complete scenes than the base-line method but it tends to produce more hallucinations. We use the approach described in [43] to remove them.

Figures 7.7, 7.4, 7.10, 7.11, 7.9, 7.8 demonstrate that our method reconstructs weakly-supported surfaces better than the base-line method. The weakly-supported surface in the Figure 7.7(a) is the bottle. The weakly-supported surface in the date-sets 7.4, 7.10, 7.11, 7.9 is mostly the ground plane.

Figure 7.8 demonstrates that our method reconstructs densely sampled surfaces at the same level of detail as the base-line method. Additionally, our method reconstructed weakly-supported surfaces, i.e. water and ladles. Note that the ladles were not at the same place in all images. Figure 7.9 presents results on the castle data set from the standard Strecha's [77] evaluation database. The histograms (c), (d) show that our reconstruction achieves almost the same quality as the method [85] which uses an additional mesh refinement step. Figure 7.9 (e), (f) shows that, unlike the base-line method (f), our method (e) can reconstruct weakly-supported ground planes. Additionally, the histograms (c), (d) show that our reconstructions are more or less on the same level at $2\,\sigma$ and $3\,\sigma$ as the base-line method which means that our method produces results on the same level of detail as the base-line method.



(a) (b)

(c) (d)

Figure 7.7.: **Results for the 'bottle' data set.** (a) Input image, (b) input 3D point cloud, (c) results using our implementation of [52], (d) the technique presented in this work reconstructs weakly-supported surfaces (the bottle) better.

(a) (b) (c) (d)

Figure 7.8.: **Results for the 'dragon' data set.** (a) Input image, (b) input 3D point cloud, (c) result of our implementation of the base-line method, (d) result using our method. Our method reconstructs densely sampled surfaces at the same level of detail as the base-line method but also reconstructs weakly-supported surfaces i.e. water, and ladles.

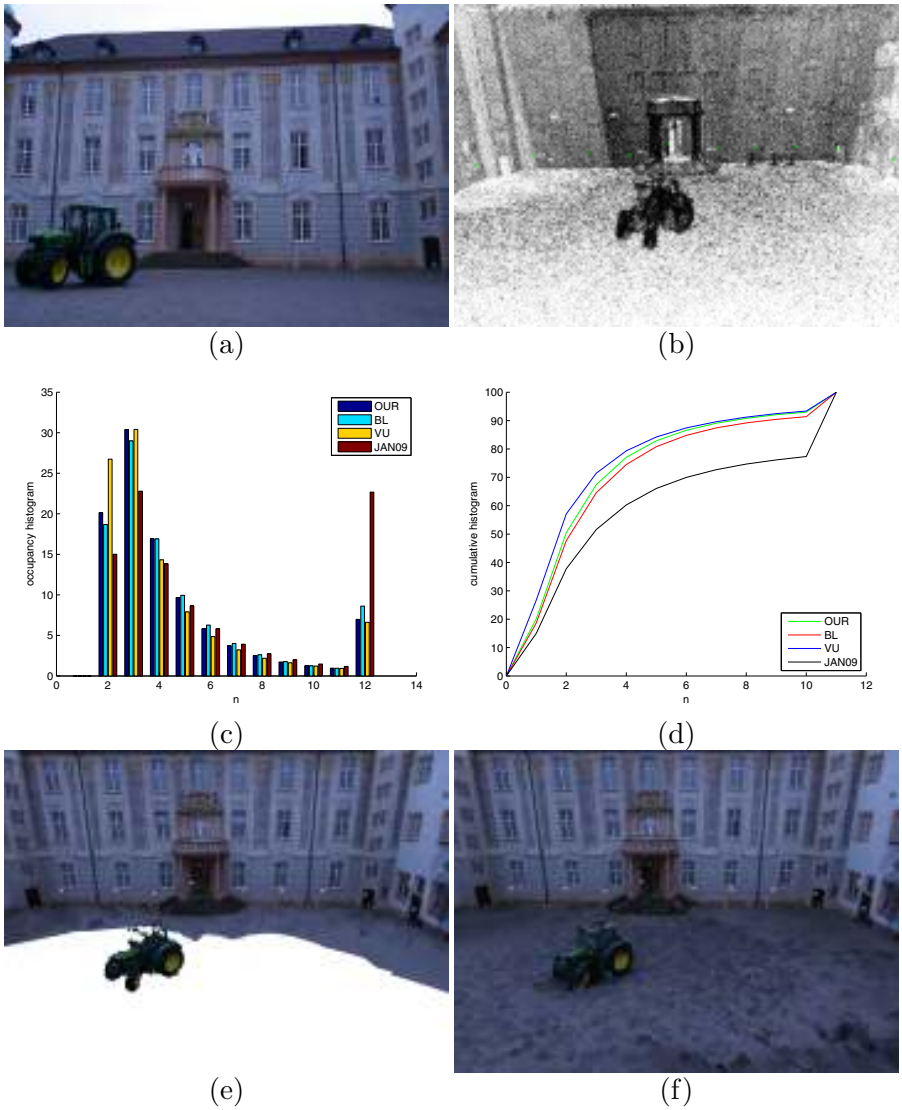
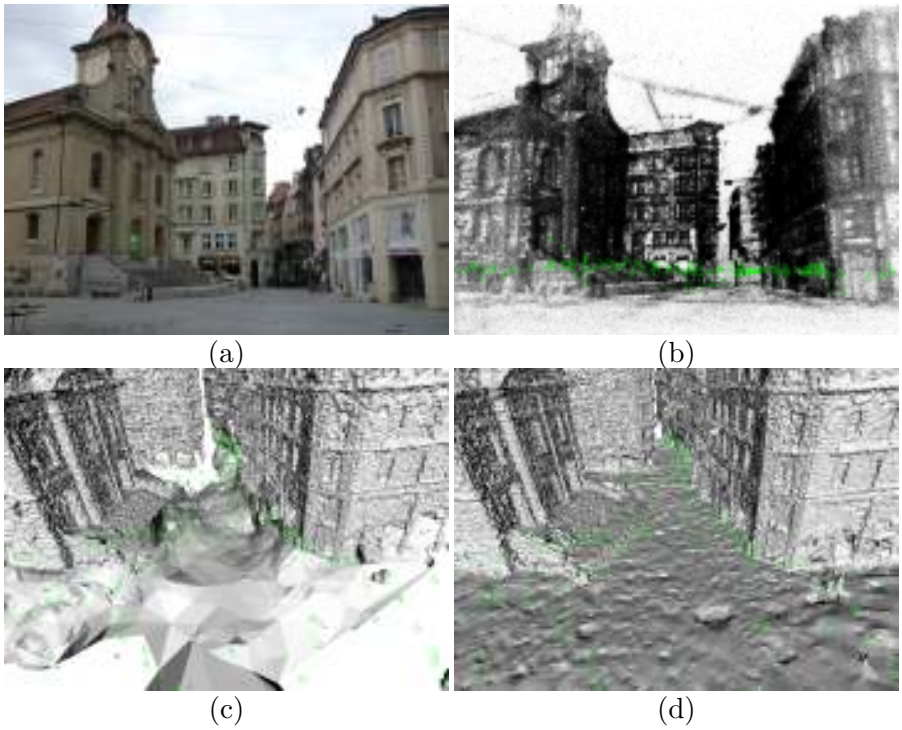


Figure 7.9.: **Results for the 'castle' data set.** (a) Input image, (b) input 3D point cloud, (c),(d) Strechas evaluation [77] for the Castle-P30 data sets. OUR - proposed method, BL - our implementation of the base-line method [52], VU [85], JAN09 [14]. (c) Histograms of the relative error with respect to $n\,\sigma$ for all views. The σ is determined from reference data by simulating the process of measurement and can vary across the surface and views, (d) relative error cumulated histograms, (e) result of our implementation of the base-line method textured, (f) result using our method textured.

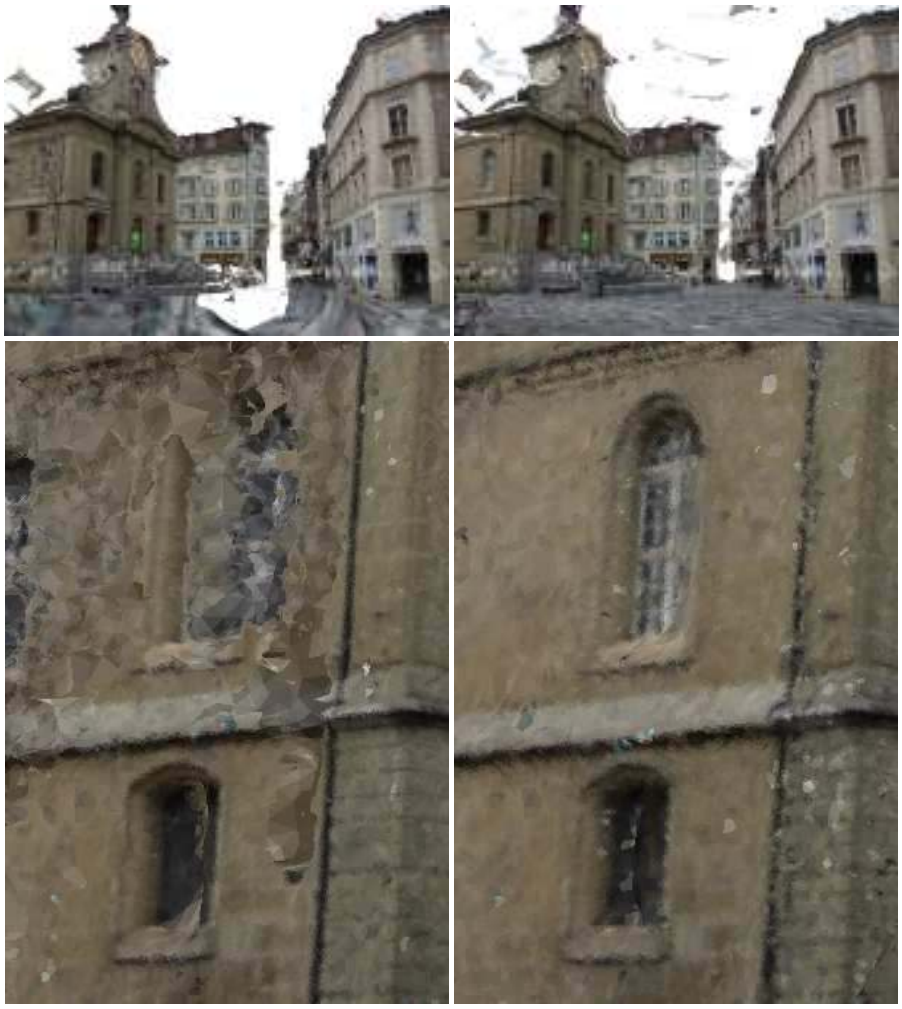
(a) (b)

Figure 7.10.: **Results for the 'strecha' data set.** (a) reconstruction using our implementation of the base-line approach, (b) reconstruction using our approach. Weakly-supported surfaces are better reconstructed using our approach than by base-line approach.



(a) (b) (c) (d)

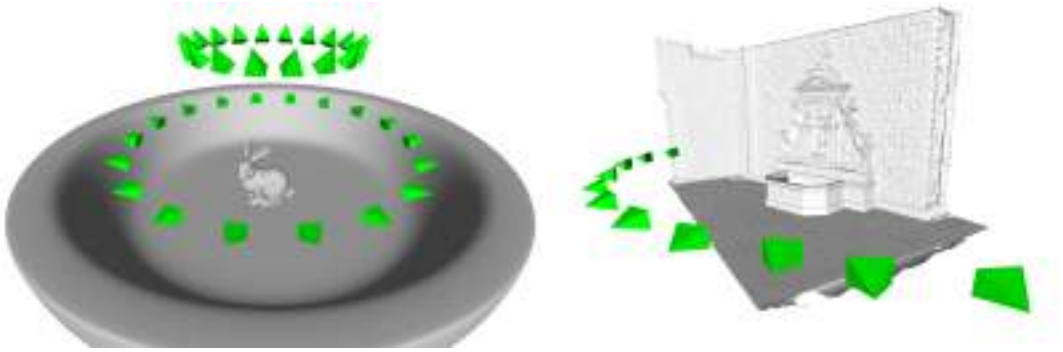
Figure 7.11.: **Results for the 'strecha' data set.** (a) Input image, (b) input 3D point cloud, (c) result of the our implementation of the base-line method untextured, (d) result using our method untextured. The weakly supported ground plane is preserved using our method.



(a) (b)

Figure 7.12.: **Results for the 'strecha' data set.** (a) result of the our implementation of the base-line method textured, (b) result using our method, textured. The weakly supported ground plane is preserved using our method and other surfaces are also constructed better.

8 Experimental evaluation of the free-space-support properties



(a) (b)

Figure 8.1.: **Ground-truth datasets** (a) Rendering of the mesh and sensors positions of the "Bunny" dataset. 36 sensors in two elevations are placed in the scene. Ground-truth for each sensor is computed directly from the mesh. Random undersampling and noisifying of the bunny object (without plate) and outlier injection is used at different levels to generate depth-maps for quantitative analysis. (b) Ground-truth laser-scan of the "Fountain" dataset and the related 11 cameras (registered to the scan).

We have introduced two observation based assumptions (i) the *Weakly-supported surfaces t-weight assumption* and (ii) the *Large free-space-support jump assumption* in the previous chapter, see sections 7.4.2, 7.4.1. We have proposed and experimentally verified a new method for weakly-supported surfaces reconstruction that is based on these the observation based assumptions in the Section 7.4. However, the method was based just on observations and assumptions.

In this chapter we experimentally evaluate how the free-space-support of thetrahedra looks in free and full space near ground-truth weakly and strongly supported surfaces in a real world scenario. Here we also experimentally confirm that the two observation based assumptions holds. Furthermore, based on the evaluations, we propose and evaluate a new method for occluder (weakly-supported surface) detection.

We evaluate the free-space-support properties under controlled levels of noise and controlled levels of undersampling. More specifically, we evaluate $f_{cp}(x)$ at 25 discrete points $x \in \{-12, 12\}$. Note that the domain of the function f_{cp} is in σ units.

We evaluate the free-space-support in two scenes (see Figure 8.1). The first scene is

a synthetically created scene of a bunny on a plate. We have downloaded the publicly available Bunny dataset from the Stanford 3D Scanning Repository[1] and we have created the plate in a 3D modeling software. We have uniformly distributed 36 sensors on two circles in two elevations around the bunny so that each sensor sees the whole bunny and, more importantly, each sensor is oriented so that the plate is visible all around the bunny. The bunny occludes a part of the plate in each view.

The plate is always strongly-supported. The bunny object is represented by $160K$ of input points and the plate object is represented by $2.4M$ of input points. We have created three datasets from the bunny on the plate scene with the 36 sensors:

- (i) original scene without undersampling and without outliers,
- (ii) bunny undersampled to 3% of the original points plus $30K$ outliers, and
- (iii) bunny undersampled to 3% of the original points plus $130K$ outliers.

The second scene is a real-world scene of a "Fountain". The "Fountain" data set is a benchmark data set [77] provided by Christopher Strecha and contains 11 calibrated images with resolution 3072×2048 together with ground-truth laser-scan. We have computed input segments of sight by MVS plane-sweeping technique described in [41]. We consider input points that are within 2σ distance to the ground-truth laser-scan as ground-truth input points.

We denote the input segments of sight i.e., the set S, of the "Bunny" (i-iii) and "Fountain" datasets as BunnySSS, BunnyWSS, BunnyWSSO and FountainSSS. Where SSS stands for Strongly Supported Surface, WSS stands for Weakly-Supported Surface and WSSO stands for Weakly-Supported Surface with large amount of Outliers.

Note that in all the datasets we know exactly which points are ground-truth interface points and in the "Bunny" datasets we additionally know which ground-truth points are from the bunny object surface and from the plate object surface. We also know which points are located inside or outside the object i.e., in full or free space.

In this section we use the following notation: $SUR(S) \subseteq S$ is the set of input segments of sight that contain ground-truth interface points of the bunny without the plate (of the fountain), $FREE(S) \subseteq S$ is the set of input segments of sight that contain input points that are located in the free space, $FULL(S) \subseteq S$ is the set of input segments of sight that contain input points that are located in the full space.

8.1. Free-space-support distribution evaluation

In this section we evaluate $f_{cp}(x)$ at points $\{-12, ..., 12\}$ just for the ground-truth interface points $p \in P(SUR(S))$. Figure 8.2 shows the Matlab boxplots[2] of values $\{f_{cp}(x) | (\mathbf{c}, \mathbf{p}) \in SUR(S)\}$ for each $x \in \{-12, ..., 12\}$. It can be seen that the free-space-support between the observer and the surface, e.g. $x \in \{-3, ..., 0\}$, is in the majority of

[1] http://graphics.stanford.edu/data/3Dscanrep/
[2] http://en.wikipedia.org/wiki/Box_plot

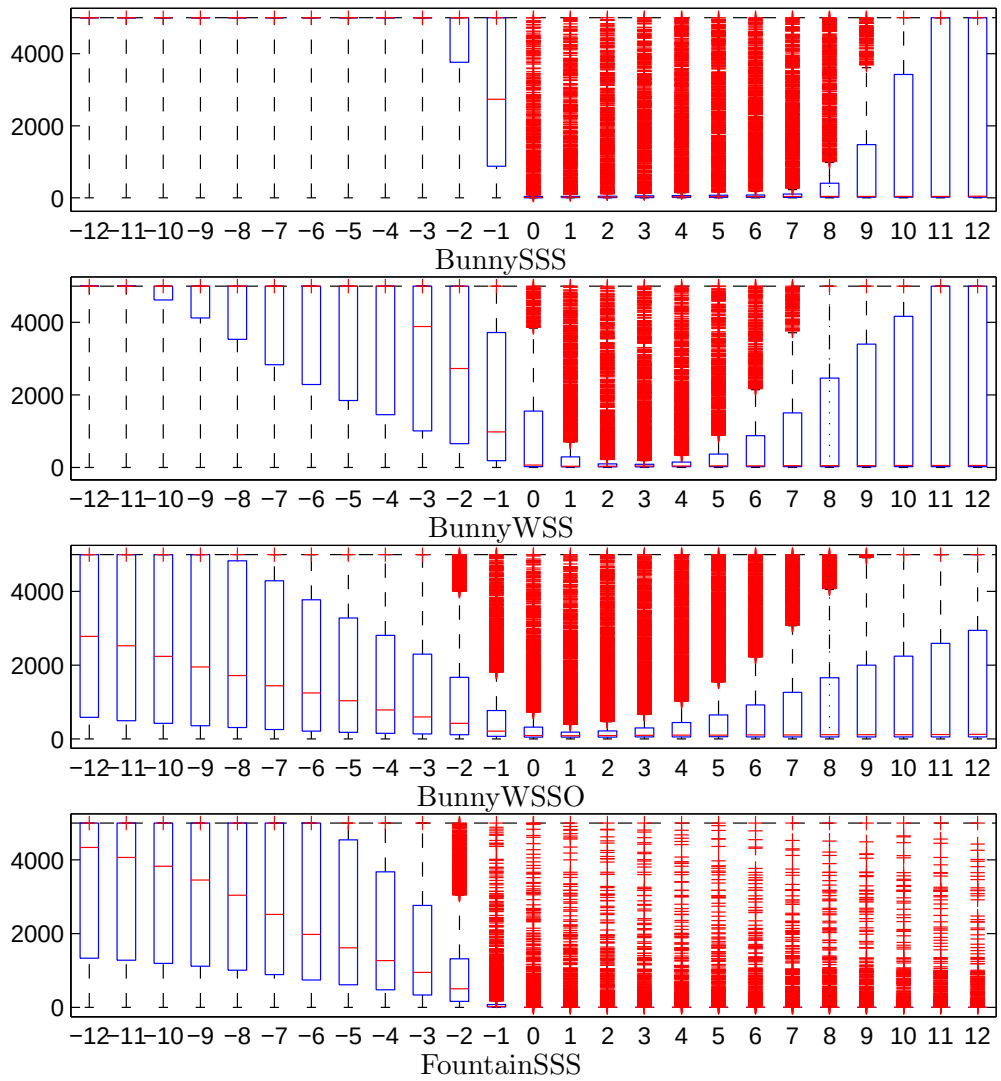


Figure 8.2.: **Free-space-support evaluation.** Free-space-support at different distances near the ground-truth surface points $p \in P(SUR(S))$ for $S \in \{\text{BunnySSS, BunnyWSS, BunnyWSSO, FountainSSS}\}$ datasets. We compute a set of free space supports $\{f_{cp}(x)|(\mathbf{c},\mathbf{p}) \in SUR(S)\}$ for each point $x \in \{-12,...,12\}$ and visualize it using the Matlab boxplot function (see text for more details).

cases relatively large compared to the free-space-support for $x \in \{1,...,4\}$, i.e., free-space-support is large in free space. Moreover, it can be seen that the free-space-support is relatively small at a small distance behind the surface, $x \in \{1,...,4\}$, i.e., the free-space-support is small in full space. This property holds for all datasets. It holds for

weakly-supported as well as for strongly-supported surfaces. Therefore, it is reasonable to measure the free-space-support jump in $\langle -3\sigma, 4\sigma \rangle$ surrounding the input points.

Results in Figure 8.2 are represented by the Matlab boxplot[3] function which shows values 25% to 75% quantile as a box with a horizontal line at the median. The red crosses show data beyond 1.5 times the interquartile range.

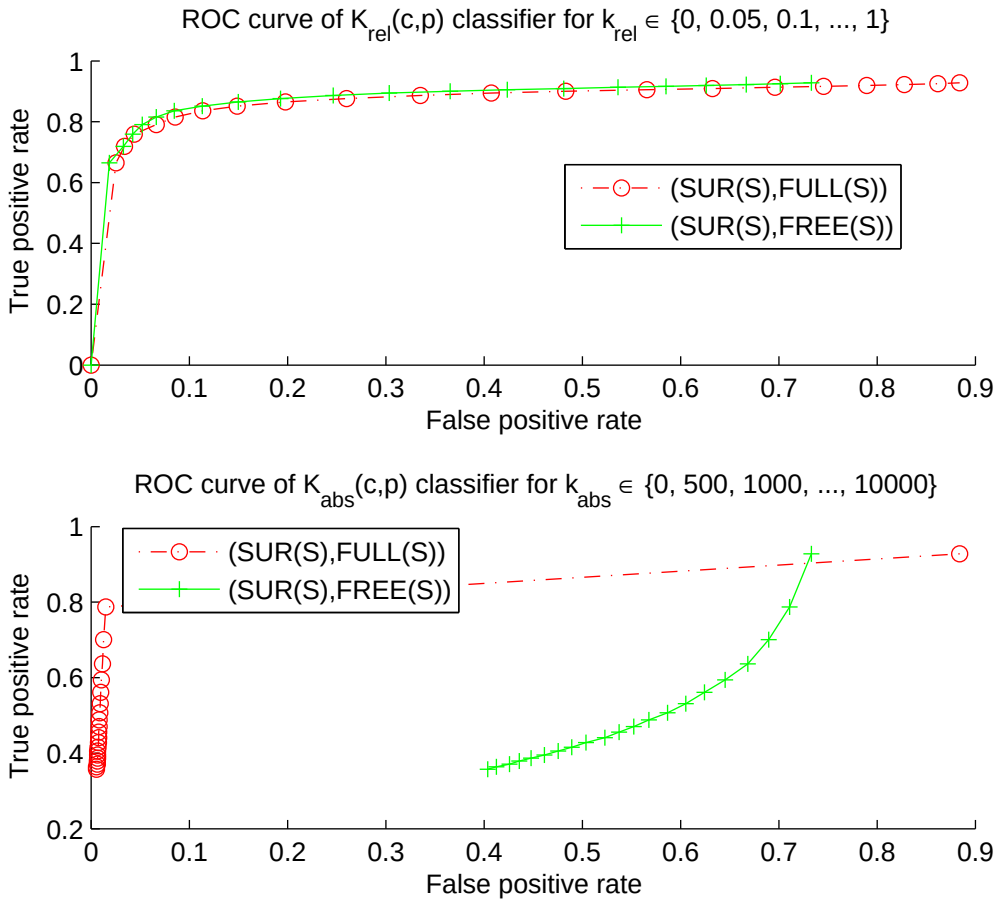


Figure 8.3.: **Free-space-support jump evaluation.** Top: relatively large number of true positives at the price of up to 10% of false positives can be detected using the relative free-space-support jump classifier $K_{rel}(\mathbf{c}, \mathbf{p})$. Bottom: absolute free-space-support jump classifier $K_{abs}(\mathbf{c}, \mathbf{p})$ can be used for detecting input points that are located in full space. The set $SUR(S)$ is considered as as positive examples and the set $FREE(S)$ ($FULL(S)$) as negative examples. See text for more details.

[3] http://en.wikipedia.org/wiki/Box_plot

8.1.1. Free-space-support jump evaluation

Based on the experiments in the previous section, we choose $k_f = 3$ a $k_b = 4$. We evaluate absolute $\epsilon_{cp}^{abs}(k_f, k_b)$ and relative $\epsilon_{cp}^{rel}(k_f, k_b)$ free-space-support jump on ground-truth interface input points $p \in P(SUR(S))$, input points located in ground-truth free space $p \in P(FREE(S))$, and input points located in ground-truth full space $p \in P(FULL(S))$. We introduce the two interface classifiers $K_{rel}(\mathbf{c}, \mathbf{p})$ and $K_{abs}(\mathbf{c}, \mathbf{p})$ in order to capture properties of the free-space-support jump functions. Given an input line of sight $(\mathbf{c}, \mathbf{p}) \in S$ the classifiers classify the point $\mathbf{p}$ as being interface point INT or non-interface point NOI according to the following rule:

$$K_{rel}(\mathbf{c}, \mathbf{p}) = \begin{cases} INT & \text{for} \quad (\epsilon_{cp}^{rel}(k_f, k_b) < k_{rel}) \\ NOI & otherwise \end{cases} \tag{8.1}$$

$$K_{abs}(\mathbf{c}, \mathbf{p}) = \begin{cases} INT & \text{for} \quad (\epsilon_{cp}^{abs}(k_f, k_b) > k_{abs}) \\ NOI & otherwise \end{cases} \tag{8.2}$$

where k_f, k_b, k_{rel} and k_{abs} are parameters of the classifiers $K_{rel}(\mathbf{c}, \mathbf{p})$ and $K_{abs}(\mathbf{c}, \mathbf{p})$.

Figure 8.3 shows receiver operating characteristic curves
(ROC[4]) of the classifiers $K_{rel}(\mathbf{c}, \mathbf{p})$ and $K_{abs}(\mathbf{c}, \mathbf{p})$. We evaluate classifier $K_{rel}(\mathbf{c}, \mathbf{p})$ ($K_{abs}(\mathbf{c}, \mathbf{p})$) on a set A of segments of sight $(\mathbf{c}, \mathbf{p}) \in A$ for classifier parameters $k_f = 3$, $k_b = 4$ and
$k_{rel} \in \{0, 0.05, 0.1, ..., 1.0\}$ ($k_{abs} \in \{0, 500, 1000, ..., 10000\}$).

For given k_{rel} (k_{abs}) we compute the rate of the results classified positively as INT out of the positive examples $A = SUR(S)$ i.e., the true positive rate. For the same k_{rel} (k_{abs}) we compute the rate of the results classified positively as INT out of the negative examples $A = FULL(S)$ ($A = FREE(S)$) i.e., the false positive rate. Finally, we plot the (false positive rate, true positive rate) as point of the ROC. We have used the set S of all segments of sights from all datasets i.e., S = BunnySSS $\cup$ BunnyWSS $\cup$ BunnyWSSO $\cup$ FountainSSS.

Figure 8.3 (top) shows that the classifier $K_{rel}(\mathbf{c}, \mathbf{p})$ can detect relatively large number, i.e. 84%, of interface input points correctly for the price of up to 10% of wrongly classified non-interface input points. We have evaluated that when allowing 10% of false positives then the parameter that gives the maximal percentage of true positives is $k_{rel} = 0.3$. The classifier $K_{rel}(\mathbf{c}, \mathbf{p})$ with this parameter $k_{rel} = 0.3$ gives 84% of true positives. The smaller the parameter k_{rel} is, the lower the percentage of the false positives the classifier $K_{rel}(\mathbf{c}, \mathbf{p})$ gives at the price of lower true positives i.e., a conservative approach to the k_{rel} parameter setting is to set a smaller value of the k_{rel} parameter.

Figure 8.3 (bottom) shows that the classifier $K_{abs}(\mathbf{c}, \mathbf{p})$ can be used to detect the input point that is located in the full part of the space. We have evaluated that when allowing just 2% of false positives for $A = FULL(S)$ then the parameter that yields the maximal percentage of true positives is $k_{abs} = 500$. The classifier $K_{abs}(\mathbf{c}, \mathbf{p})$ with this parameter $k_{abs} = 500$ gives 79% of true positives. The larger the parameter k_{abs} is,

[4] http://en.wikipedia.org/wiki/Receiver_operating_characteristic

the lower the percentage of false positives the classifier $K_{abs}(\mathbf{c}, \mathbf{p})$ yields at the price of lower true positives, i.e. a conservative approach to the k_{abs} parameter setting is to set a larger value of the k_{abs} parameter.

8.1.2. Outlier level evaluation

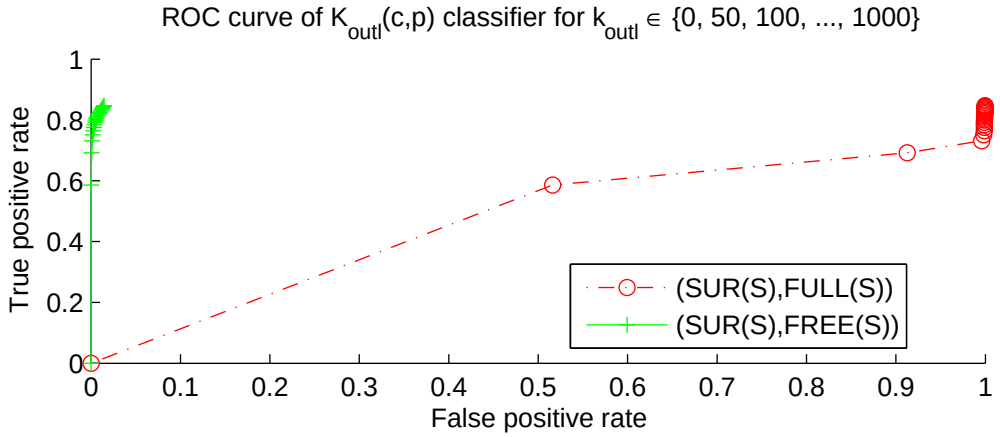


Figure 8.4.: **Outliers level evaluation.** Outliers level classifier $K_{outl}(\mathbf{c}, \mathbf{p})$ can be used for detecting input points that are located in the free space. The set $SUR(S)$ is considered as as positive examples and the set $FREE(S)$ ($FULL(S)$) as negative examples. See text for more details.

We evaluate the level of outliers $\gamma_{cp}^{max}(k_b)$ in the same way we have evaluated free-space-support jump in the previous section. We use the following classifier

$$K_{outl}(\mathbf{c}, \mathbf{p}) = \begin{cases} INT & \text{for} \quad (\gamma_{cp}(k_b) < k_{outl}) \\ NOI & otherwise \end{cases} \tag{8.3}$$

where k_b and k_{outl} are parameters of the classifier $K_{outl}(\mathbf{c}, \mathbf{p})$. We evaluate the classifier $K_{outl}(\mathbf{c}, \mathbf{p})$ for $k_{outl} \in \{0, 50, 100, ..., 1000\}$.

Figure 8.4 shows that the classifier $K_{outl}(\mathbf{c}, \mathbf{p})$ can be used to detect an input point that is located in the free space. We have evaluated that if we allow just 1% of false positives for $A = FREE(S)$ then the parameter that yields the maximal percentage of true positives is $k_{outl} = 700$. The classifier $K_{outl}(\mathbf{c}, \mathbf{p})$ with this parameter $k_{outl} = 700$ yields 83% of true positives. The smaller the parameter k_{outl} is, the lower the percentage of false positives the classifier $K_{outl}(\mathbf{c}, \mathbf{p})$ gives at the price of lower true positives, i.e. a conservative approach to the k_{outl} parameter setting is to set a smaller value of the k_{abs} parameter.

Parameter	k_{abs}	k_{rel}	k_{outl}	k_f	k_b
Estimated	0.3	500	700		
Used	0.1	1000	400	3	4

Table 8.1.: **Computed and used interface classifier K parameters.**

BunnySSS	Interface	Non-Interface
INT	$82.7\%\,TP$	$0.5\%\,FP$
NOI	$17.3\%\,FN$	$99.5\%\,TN$
BunnyWSS	Interface	Non-Interface
INT	$41.7\%\,TP$	$0.2\%\,FP$
NOI	$58.3\%\,FN$	$99.8\%\,TN$
BunnyWSSO	Interface	Non-Interface
INT	$26.6\%\,TP$	$0.2\%\,FP$
NOI	$73.4\%\,FN$	$99.8\%\,TN$
FountainSSS	Interface	Non-Interface
INT	$55.4\%\,TP$	$1.3\%\,FP$
NOI	$44.6\%\,FN$	$98.7\%\,TN$

Table 8.2.: **Decision tables.** Classification results of interface classifier $K(\mathbf{c},\mathbf{p})$ on different datasets.

8.2. Interface classifier

In this section we design a new interface classifier. The classifier takes into account the free-space-support jump and outliers level. Based on the experiments in the previous sections we can say that a large free-space-support jump coupled with a low outliers level in $\langle -3\sigma, 4\sigma\rangle$ surrounding an input point, is a good evidence for an interface. Given an input line of sight $(\mathbf{c},\mathbf{p}) \in S$ the following classifier $K(\mathbf{c},\mathbf{p})$ classifies the point $\mathbf{p}$ as being an interface point INT or a non-interface point NOI

$$K(\mathbf{c},\mathbf{p}) = \begin{cases} INT & \text{for} \quad (\epsilon_{cp}^{rel}(k_f,k_b) < k_{rel}) \wedge \\ & \quad\;\; (\epsilon_{cp}^{abs}(k_f,k_b) > k_{abs}) \wedge \\ & \quad\;\; (\gamma_{cp}(k_b) < k_{outl}) \\ NOI & otherwise \end{cases} \tag{8.4}$$

where k_f, k_b, k_{rel}, k_{abs} and k_{outl} are parameters of the classifier $K(\mathbf{c},\mathbf{p})$ and are constant in all of our experiments. All parameters k_f, k_b, k_{rel}, k_{abs} and k_{outl} were discussed and evaluated experimentally in the previous section. Table 8.1 summarizes computed and used values. The used values were chosen using a conservative approach, i.e., lower percentage of false positives at the price of lower true positives.

Table 8.2 shows the decision rates of the interface classifier on sets $S \in \{$BunnySSS, BunnyWSS, BunnyWSSO and FountainSSS$\}$ of input segments of sight. We consider the set $SUR(S)$ as positive examples (i.e. "Interface") and the set $FREE(S) \cup FULL(S)$ as negative examples (i.e. "Non-Interface").

Based on the experiments, we can say that if the classifier classifies a point as INT then it is most likely an interface point in the real world. Note that this property holds for weakly-supported surfaces as well! On the other hand if the classifier classifies a point as NOI we can't say that it is not an interface point in the real world. In the next sections we show how we use the interface classifier to modify the state-of-the-art method [52], which is not capable of reconstructing weakly-supported surfaces, so that it will obtain the ability to reconstruct weakly-supported surfaces. Although the number of true positives is around 50%, we will show that it is enough for the new method to reconstruct weakly-supported surfaces.

9 Using an interface classifier to reconstruct weakly-supported surfaces

We have introduced a way how to detect occluder just from the accumulated free-space evidences in previous chapter. Now we use this tool to reconstruct them. Our idea is to tell the optimization, described in the Section 7.3, that the positively classified occluder tetrahedra must be labeled as full. Based on the synthetic example proposed int the Section 7.4.3 we enforce weight of corresponding t-edges. This will force the optimization to label the occluder nodes as full and the weakly-supported surface will be reconstructed accordingly. It is important to note that we do not have to enforce all occluder tetrahedra. Therefore, besides that the number of true positive of our classifier is not 100% it is far enough for weakly-supported surfaces reconstruction.

Following the Section 7.4.3, all we need to do in order to preserve the weakly-supported surface is to enforce the t-weights of nodes $v \in V$ where tetrahedra $T(v)$ are located inside the occluder. We proceed sequentially. In the first iteration we compute the weights of STG according the Section 7.3.3. Then we evaluate the interface classifier and, for the segments of sight classified as INT (interface), we enforce t-weight in the place we assume is inside of the object. Weight $w(e)$ of a t-edge $e = (v, t) \in E$ is enforced to

$$
\begin{aligned}
w(e) &= w(e) \sum_{(\mathbf{c},\mathbf{p}) \in S_K(v)} \epsilon_{cp}^{abs}(k_f, k_b) \quad \text{with} \qquad (9.1)\\
S_K(v) &= \{(\mathbf{c}, \mathbf{p}) \in S \,|\, \mathbf{p} \in L_c^p(k_b \sigma) \\
&\quad \wedge L_c^p(k_b) = T(v) \\
&\quad \wedge K(\mathbf{c}, \mathbf{p}) = \mathrm{INT}\}
\end{aligned}
$$

Algorithmic overview and the implementation details of the proposed method are in section 9.2. Finally, in section 9.3, we provide an experimental evaluation of the accuracy and the ability of reconstruction of weakly-supported surfaces on synthetic as well as on real world datasets and compare them to other methods.

9.1. Comparison to the observation based approach

Both the newly proposed method and the observation based method proposed in the Section 7.4 differ from the method proposed in [52] in how the STG edge weights are

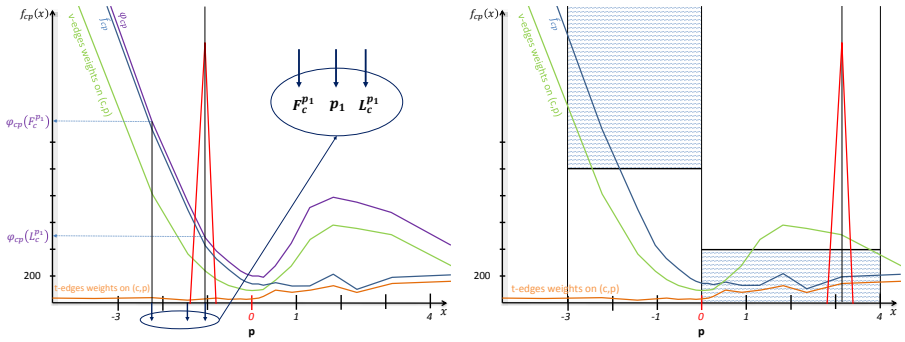


Figure 9.1.: **New interface classifier K vs observation based approach (Section 7.4)** (a,b) Point $\mathbf{p}$ ($x = 0$) represents a real interface point of a ground-truth weakly-supported surface. (a) Illustrates the typical situation when classifier K_{jan} wrongly classifies a point p_1 as an interface point and enforces t-weight (red peak) in wrong place i.e., in space that should be labeled as free. (b) Design of the new interface classifier K leads to the correct classification of interface point $\mathbf{p}$ and t-weight enforcement in correct place. Note that the t-edge weights are low because this is illustration of weights near a weakly-supported surface.

computed. First, the newly proposed method and the observation based method use adaptive $\alpha(\mathbf{p})$ weight of input point $\mathbf{p}$ while the method proposed in [52] uses a constant weight for each input point $\mathbf{p}$. Second by the newly proposed method and the observation based method use an interface classifier to enforce t-weights in order to be able to reconstruct weakly-supported surfaces. The observation based method proposed in the Section 7.4 does not define the interface classifier explicitly. However, it uses a decision rule that can be formulated as interface classifier K_{jan} as follows:

$$K_{jan}(\mathbf{c},\mathbf{p}) = \begin{cases} INT & \text{for } (\frac{\varphi(L_c^p(\sigma))}{\varphi(F_c^p)} < 0.5) \wedge \\ & (\varphi(L_c^p) < 1000) \\ NOI & otherwise \end{cases} \tag{9.2}$$

where

$$\varphi(T(u)) \quad = \quad \varphi(u) = \sum_{e=(u,v)\in E_v} w(e) \tag{9.3}$$

If σ equals zero then $f(T(u)) = \varphi(T(u))$ i.e., if $\sigma = 0$ then the free-space-support of the tetrahedron $T(u)$ equals the sum of the weights of all the incoming edges to the node u.

We define $\varphi_{cp}(x)$ on the line of sight $(\mathbf{c}, \mathbf{p}) \in S$ for the function φ in the same way as the free-space-support weight $f_{cp}(x)$ was defines on the line of sight $(\mathbf{c}, \mathbf{p}) \in S$ for the function f.

The major advantage of the newly proposed method with respect to the observation based method is that it produces much more accurate results on strongly-supported surfaces and reconstructs weakly-supported surfaces better as shown in experiments in section 9.3.

9.1.1. Why is the newly proposed method more accurate than the observation based method?

Figure 9.1(a) shows a typical situation where the classifier K_{jan} wrongly classifies the point p_1 as an interface point. While the free-space-support decreases rapidly in front of the point p_1, which is in front of a real surface point $\mathbf{p}$ ($x = 0$), the classifier K_{jan} classifies the point p_1 as interface point and enforces the t-weight behind p_1 (red peak), which leads to less accurate result. However, proposed interface classifier K searches for such point $\mathbf{p}$ and an associated sensor $\mathbf{c}$ for which the free-space-support function f_{cp} is rapidly decreasing at some (reasonably) large distance k_f in front of the point $\mathbf{p}$ (i.e., toward the sensor $\mathbf{c}$). Additionally it is reasonably small, and almost constant at (reasonably) large distance k_b behind the point $\mathbf{p}$ (i.e., backward the sensor c), see Figure 9.1(b). Afterward, based on the positive interface classification we enforce t-weight in distance k_b (deep enough) behind the point $\mathbf{p}$ in order to ensure that it is enforced inside of the real object (occluder).

Algorithm 3 Surface reconstruction preserving weakly-supported surfaces

Require: DT - Delaunay Tetrahedralization, a subset of sensors associated to each vertex of the tetrahedralization

1: **function** RECONSTRUCT(DT)
2: Compute the STG weights as in [52] (using adaptive $\alpha(\mathbf{p})$ weight of vertex $\mathbf{p} \in DT$) ▷ t_{o_1}
3: **for all** (sensor $\mathbf{c}$, vertex $\mathbf{p}$) **do** ▷ t_{o_2}
4: Compute $K(\mathbf{c}, \mathbf{p})$
5: **if** $K(\mathbf{c}, \mathbf{p}) == INT$ **then**
6: Add the value $\epsilon_{cp}^{abs}(k_f, k_b)$ to weight of t-edge (v, t) where $L_c^p(k_b\sigma) = T(v)$
7: **end if**
8: **end for**
9: Compute minimal s-t cut ▷ t_{GC}
10: Create mesh from the minimal s-t cut
11: Smooth
12: **return** Mesh
13: **end function**

9.2. Implementation Details

We use the Computational Geometry Algorithms Library (CGAL)[1] [11] to create the tetrahedralization. The tetrahedralization is created incrementally. When adding a new point o from the input point cloud we first check if there is a vertex $\mathbf{p}$ of the actual tetrahedralization within a certain distance from o. The distance is set in terms of the o and $\mathbf{p}$ re-projection distance in pixels to a value in the range $\langle 2, 10\rangle$ in order to be able to fit the data into memory. If there is such a vertex $\mathbf{p}$, then we do not add o to the tetrahedralization but associate the sensor with the vertex $\mathbf{p}$ and increase $\alpha(\mathbf{p})$ by 1. Otherwise, we will add the point o to the tetrahedralization, associate the sensor with it, and initialize $\alpha(o)$ to 1. The approach is similar to [51]. Each vertex has a set of sensors and an $\alpha(\mathbf{p})$ value associated with it. The vertices of the tetrahedralization are considered as input points, associated sensors as visibility information of input points and $\alpha(\mathbf{p})$ is the weight of the input point $\mathbf{p}$.

The implementation of the proposed method is described in Algorithm 4. We build the s-t graph. Next, we compute the weights of all edges. Our approach to setting up the weights of the graph is sequential. In the first part, we compute all weights of all edges as is described in section 7.3.3 (Line 2 in the Algorithm 4). In the second part, we evaluate on each input point $\mathbf{p}$ and each associated sensor $\mathbf{c}$ the newly proposed classifier $K(\mathbf{c}, \mathbf{p})$. If the classification result is INT, then we multiply the weight of the t-edge (v, t) where $L_c^p(k_b\sigma) = T(v)$ by $\epsilon_{cp}^{abs}(k_f, k_b)$ value (Lines 3-8 in the Algorithm 4). Table 8.1 shows classifier parameters, which we use in all of our experiments. Finally, we solve the minimal s-t cut problem using the software[2] described in [7] (Line 9 in the Algorithm 4). Finally, we do two steps of a Laplacian based smoothing (Line 22 in the Algorithm 4) as in [52]. We focus just on creating an initial surface, which can be later refined as in [85].

result name	t_{o_1}	t_{o_2}	t_{GC}	n_p	T_t
Fountain	12 : 27	2 : 44	0 : 44	$4.6M$	$30.0M$
Street-view	05 : 00	2 : 34	0 : 52	$2.9M$	$19.6M$
Lausanne	20 : 02	3 : 28	0 : 53	$4.9M$	$32.7M$

Table 9.1.: **Performance data for different results.** t_{o_1} is the time of the first part of the proposed algorithm and t_{o_2} is the time of the second one. t_{GC} is the time of solving the minimal s-t cut problem. The times are in the format: $minutes : seconds$. n_p is the number of vertices and T_t is the number of tetrahedra in the tetrahedralization. The letter M stands for million.

Table 10.1 shows the running times of different parts of the Algorithm 4 for different data sets. Table 10.1 shows that the second part, t_{o_2}, is 2 to 10 times faster than the

[1] http://www.cgal.org/

[2] http://www.adastral.ucl.ac.uk/~vladkolm/software.html

first one, t_{o_1}. Iterations in all parts are performed in parallel in our implementation. Therefore, all parts are relatively fast.

9.3. Experimental Results

All experiments were computed on Intel Core i7 CPU machine with NVIDIA 285GTX graphics card, 12GB RAM and 64-bit Windows 7 OS. We use several different calibrated data sets in this chapter: "bunny", "lausanne", "castle", "herzjesu", "fountain", and "googleStreetView". The "bunny" and "fountain" data sets are described in chapter 8. The street view data set "lausanne" was provided by Christopher Strecha and contains 1514 2000×1500 images.

The street view data set "goolgeStreetView" is Google Street View Pittsburgh Experimental image set [2]. The first 10000 camera matrices were obtained by the method proposed in [81]. We have to point out that the quality of images (compared to the quality of images from a DLSR or compact camera) was poor in this data-set. The calibration was not optimal either because the calibration was done on spherical images where the internal parameters of each perspective camera out of 6 Ladybug perspective cameras are unknown, and in the proposed pipeline we use 6 perspective cutouts from one spherical image instead of original perspective images from the Ladybug device. Therefore, we consider the "googleStreetView" data-set as the challenging one.

We compute the depth-maps for each of the 1500 (10000) images from "lausanne" ("googleStreetView") dataset but to reconstruct different parts of the city we choose 3D points from different volumes parts defined by boxes in order to fit into memory.

The "castle", "herzjesu", and "fountain" data sets are the benchmark data sets [77]. The "castle" data set contains 30 3072×2048 images, the "herzjesu" data set contains 8 3072×2048 images, and the "fountain" data set contains 11 3072×2048 images.

Let us denote the newly proposed method in this chapter as "**proposed**", the observation bsed method proposed in Section 7.4 as "**former**", method [52] as "**baseline**" and method [47] as "**poisson**".

(a) Bunny (160K) points plus 0, 25K and 130K outliers

(b) Poisson [47]

(c) Baseline [52]

(d) Former

(e) Proposed

Figure 9.2.: **Robustness to outliers** (a) The input points are visualized without plate input points in order to be able to visualize the level of outliers with respect to input points of the bunny. (b)(c),(d),(e) Results of poisson, baseline, former, and proposed methods. Baseline, former, and proposed methods are all robust to large number of outliers (130K) when the surface of the bunny is strongly supported (bunny is sampled by 160K input points).

9.3.1. Comparison with the ground truth

Let us describe how we compare our reconstruction to the ground-truth. We generate two depth-maps d_r for computed 3D mesh and d_g for the ground truth mesh for each sensor. The depth value for a pixel x of a depth-maps d is denoted as $d(x)$. We evaluate accuracy value $acc(x, c) = \frac{|d_r(x)-d_g(x)|}{\sigma}$ for each sensor $\mathbf{c}$ and for each pixel x.

We compute a histogram from all $acc(x, c)$ values as follows: Bins 1 to 10 represent the histogram of accuracy values $acc(x, c)$ of all pixels x and all sensors $\mathbf{c}$ where $d_r(x)$ and $d_g(x)$ are defined, i.e., $d_r(x) > -1 \wedge d_g(x) > -1$. Bin 11 is the number of pixels where $acc(x, c) > 10$, bin 12 is the number of pixels where the ground-truth depth-maps d_g is defined but the reconstructed depth-maps d_r is not defined, i.e.
$|\{x|d_r(x) = -1 \wedge d_g(x) > -1\}|$.

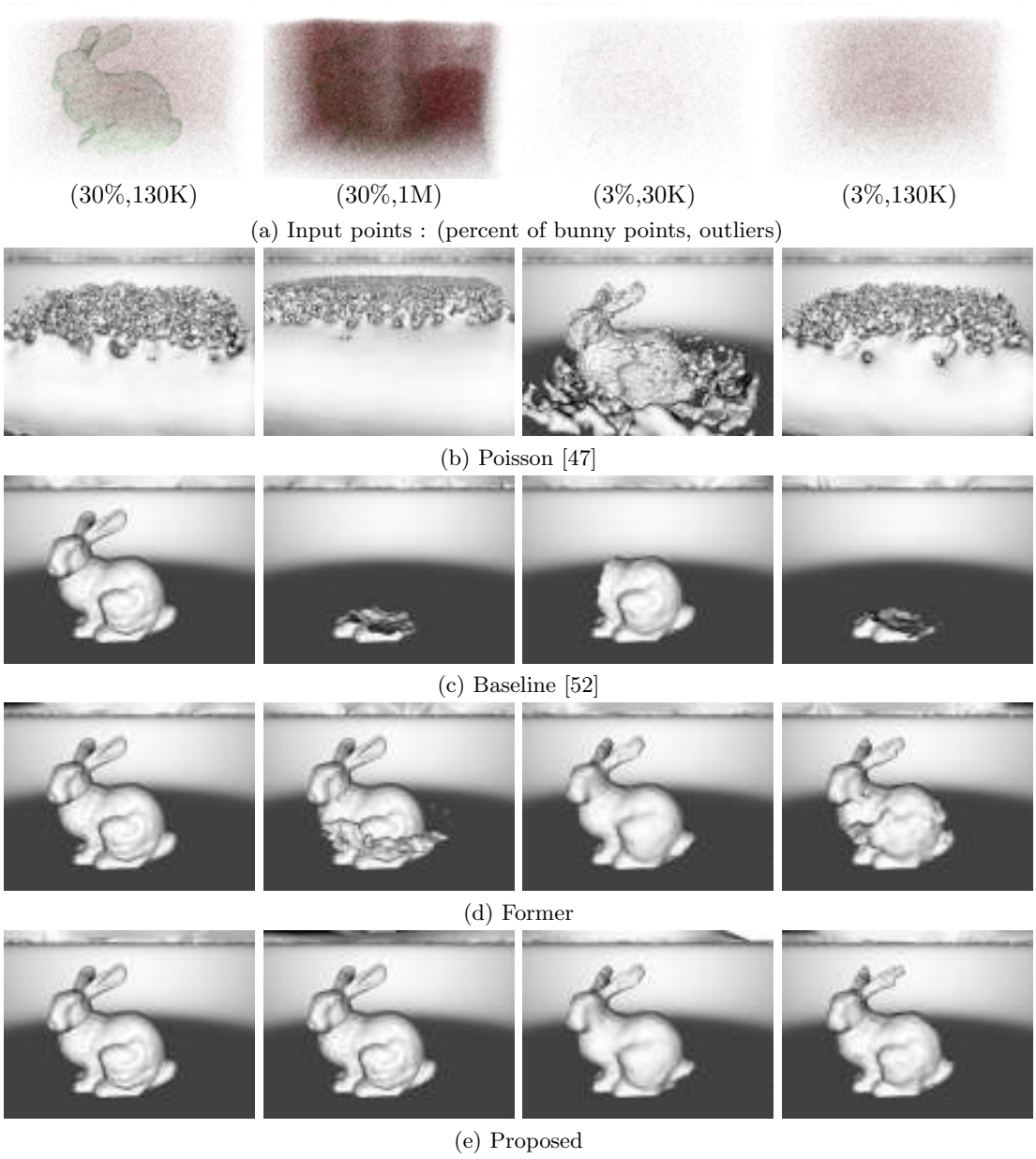
Therefore, more accurate results have a higher numbers in the first bins. The first two bins are authoritative because they contain the majority of the points. We call the aforementioned histogram occupancy histogram. We call the value of a bin X occupancy at σX or occupancy at bin X.

9.3.2. Quantitative analysis

In this section we provide a quantitative analysis of results of proposed, former, baseline, and poisson method with respect to the ground truth.

Robustness to outliers Figure 9.2 shows robustness of evaluated methods to outliers. It shows that proposed, frormer, and baseline methods are robust in the presence of 25K and 130K randomly distributed outliers in manually selected cube around the bunny. The poisson method fails completely when 130K of outliers were added to the scene.

Robustness to undersampling and outliers Figure 9.3 shows the robustness of evaluated methods to undersampling and outliers. We have randomly undersampled just the bunny object (the plate object was not undersampled) to 30% and 3% of original points. We have randomly distributed $130K$ and $1M$ outliers in manually selected cube around the bunny. Figure 9.2 shows that poisson method has completely failed in all cases. The baseline method was successful just in the first case where the surface is relatively strongly-supported by the input points and the outliers level is lower than the surface sampling by the input points. The former method can reconstruct the weakly-supported object better than the baseline method however it is not perfect in all cases. On the other hand, the proposed method is stable in all cases and it is able to reconstruct weakly-supported surfaces perfectly. This experiment also shows that while the TP classification rate showed in table 8.2 of the proposed interface classifier (section 8.2) is around 50% it is still enough to reconstruct weakly-supported surfaces.



(a) Input points : (percent of bunny points, outliers)

(b) Poisson [47]

(c) Baseline [52]

(d) Former

(e) Proposed

Figure 9.3.: **Robutness to undersampling and outliers** (a) The input points are visualized without plate input points in order to be able to visualize the level of outliers with respect to input points of the undersampled bunny. (b)(c),(d),(e) Results of poisson, baseline, former, and proposed methods.

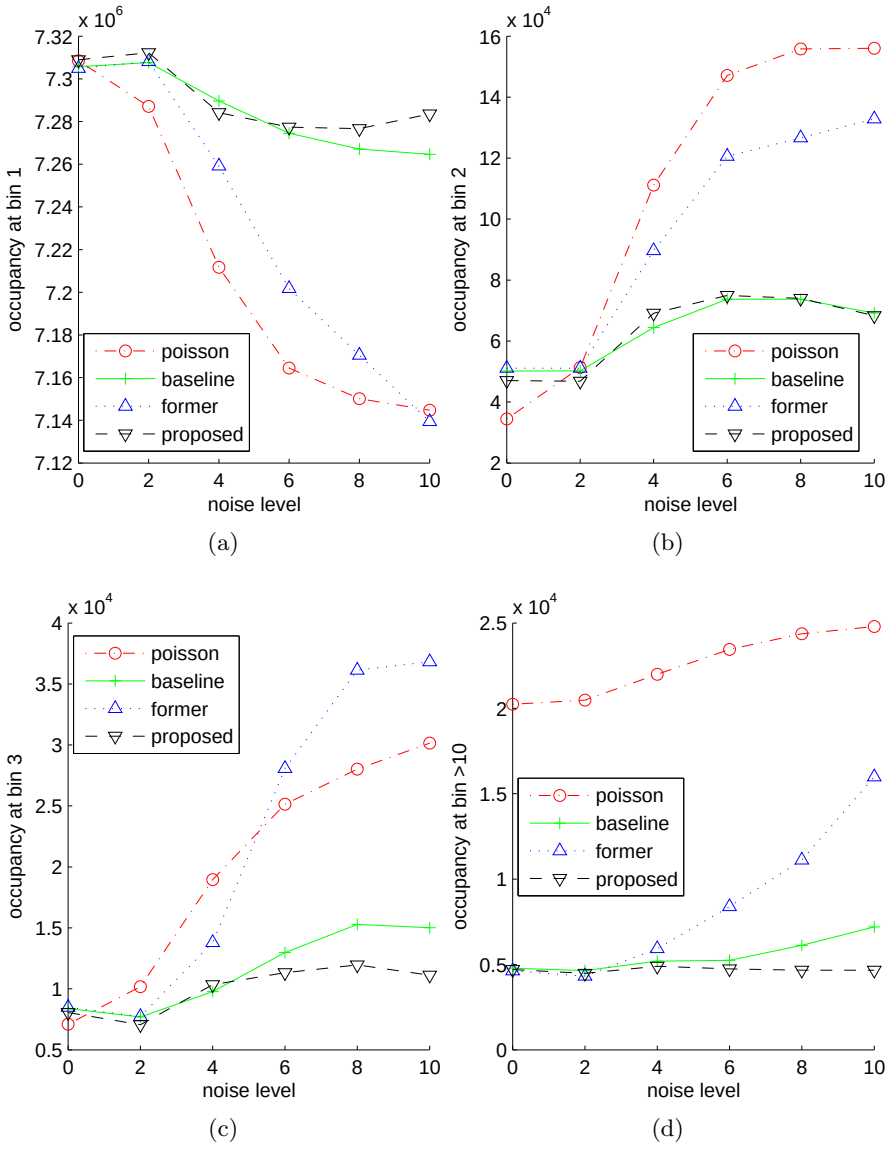


Figure 9.4.: **Robustness to noise**. Each function shows how the accuracy of a method at a cetrain bin of the occupancy histogram changes with increasing level of noise. See text for more details.

Robustness to noise Figure 9.4 shows robustness of evaluated methods to noise. We randomly noisify the bunny input points with noise with normal distribution in $\{0\,\sigma, 2\,\sigma, ..., 10\,\sigma\}$ surrounding of ground input points. Next, for each noise level we reconstruct the scene using a method and we compute the occupancy histogram of the

reconstruction (see section 9.3.1). We take a bin B from the occupancy histogram for each noise level and plot it as a piecewise linear occupancy function of the method at bin B. The occupancy function shows how the accuracy of a method at a certain bin of the occupancy histogram changes with the increasing level of noise. Figure 9.4(a)(b,c,d) shows the occupancy functions of poisson, baseline, former, and proposed methods at bin 1 (2,3,10).

Figure 9.4(a) shows that the occupancy of poisson and former methods at bin 1 decrease rapidly with increasing noise level while the occupancy of the baseline and the proposed methods at bin 1 decrease slowly and is almost at the same level.

Figure 9.4(b,c) shows that the occupancy of poisson and former methods at bins $2, 3$ increase slowly with increasing noise level while the occupancy of the baseline and the proposed methods at bins $2, 3$ increase rapidly almost identically.

Note that the more slowly the occupancy at bin 1 decreases and the more slowly the occupancy at bins ≥ 2 increases, the more accurate the reconstruction is.

Figure 9.4(d) shows that the occupancy of the poisson method at bin 11 is much higher than all other methods. Note that bin 11 of the occupancy histogram is the number of pixels where $acc(x, c) > 10$, see section 9.3.1.

We have experimentally shown that the proposed method is more accurate compared to the former method. Additionally, we showed that the proposed method produces the results at the same accuracy level as the baseline method and both the proposed and the baseline are much more accurate than the poisson method with increasing the level of noise.

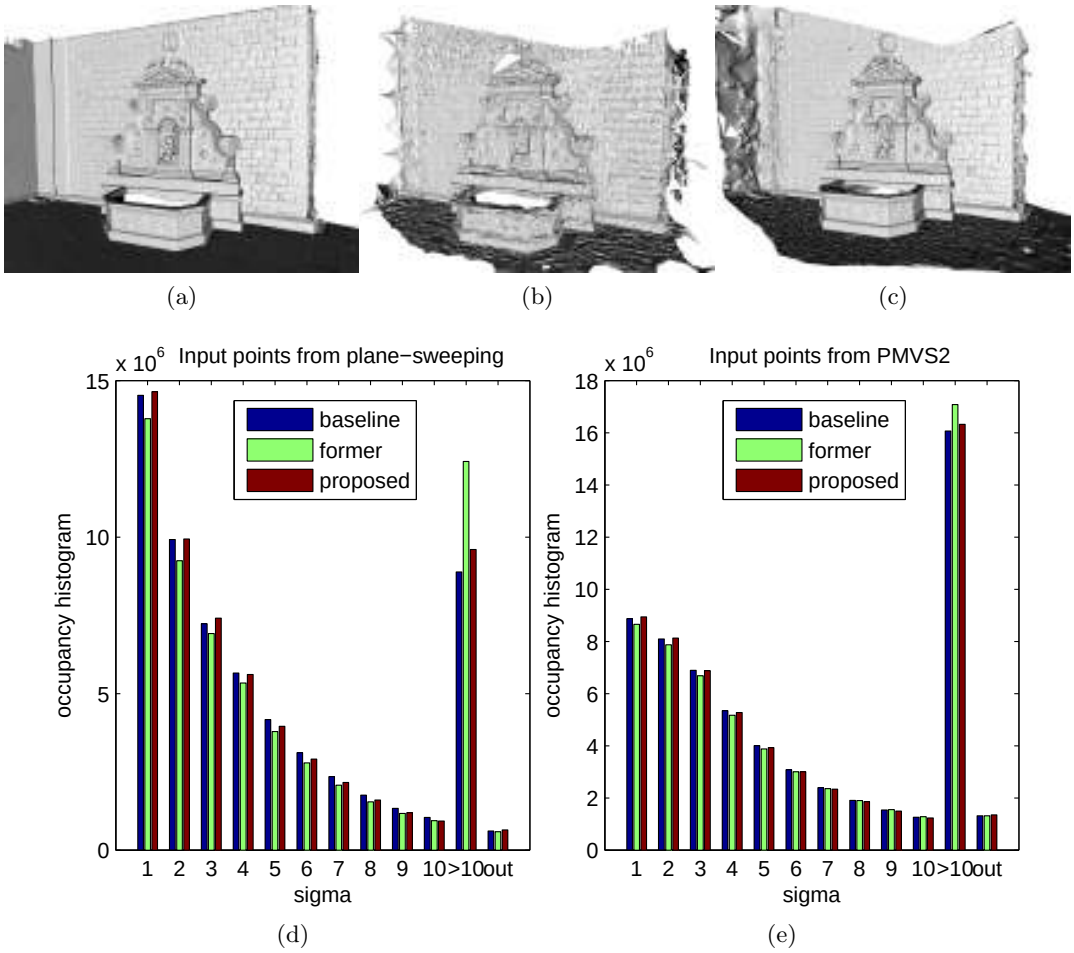


Figure 9.5.: **Accuracy evaluation on realworld dataset.** (a) ground truth generated by a laser-scan, (b) reconstruction using the proposed method on input points generated by PMVS2 software and (c) reconstruction using the proposed method on input points generated by the plane-sweeping approach described in [44]. (d),(e) Bins 1 to 10 represent the histogram of reconstruction distances from ground-truth laser-scan in units σ. Bin 11 is the number of values above 10σ and bin 12 is the number of wrong values. See text for more detailed description evaluation method. (d) Evaluations of reconstructions using baseline, former, and proposed method on input points generated by the plane-sweeping method. (e) Evaluations using input points generated by the PMVS2 software.

(a) (b) (c) (d)

Figure 9.6.: **Middlebury results.** Reconstruction results of Middlebury data sets "templeRing" and "dinoRing".

9.3.3. Accuracy evaluation on realworld dataset

Figure 9.5 shows renderings and occupancy histograms of reconstructions using baseline, former, and proposed method with respect to a ground-truth laser-scan. The laser-scan, images and calibrations were provided by Christopher Strecha and downloaded from his evaluation page [77]. We have used two sources of point clouds as input to the reconstruction methods. The first one was generated by PMVS2 software[3]. The second one was generated by the plane-sweeping method described in [44].

Figure 9.5 (a,b,c) shows the rendering of a ground truth laser-scan, the rendering of the reconstruction of proposed method on input points generated by PMVS2 and the rendering of reconstruction of proposed method on input points computed from the plane-sweeping method described in [44]. It shows that the proposed method works for input points generated by different MVS based methods.

Figure 9.5 (d) shows the occupancy histograms of reconstructions of baseline, former, and proposed methods on [44] input points. Figure 9.5 (e) shows the occupancy histograms of reconstructions of baseline, former, and proposed methods on PMVS2 input points. Figures 9.5 (d,e) show that baseline and proposed methods produce more accurate results than the former method. Additionally, it shows that the proposed method produces the results at same accuracy level as the baseline method.

9.3.4. Evaluation on Middlebury datasets

The datasets "templeRing" , and "dinoRing" are provided for benchmarking multi-view reconstruction methods [68][4].

Figure 9.6 shows results of reconstruction of the "templeRing", and "dinoRing" datasets using the method proposed in this chapter. Note that we have not used silhouette images at all.

[3]PMVS2 is software developed by Yasutaka Furukawa and Jean Ponce [22]

[4]http://vision.middlebury.edu/mview/

method	"templeRing"		"dinoRing"	
	Acc.	Compl.	Acc.	Compl.
Vu [85]	$0.45mm$	99.8%	$0.53mm$	99.7%
Sinha [69]	$0.79mm$	94.9%	$0.69mm$	97.2%
Proposed	$0.53mm$	99.3%	$0.4mm$	99.5%

Table 9.2.: **Middlebury results.**

Table 9.2 shows quantitative evaluation comparing our results with the laser-scanned ground truth. The accuracy values for the "templeRing" and "dinoRing" meshes are $0.53mm$ and $0.4mm$, respectively. More remarkably, the completeness measures are 99.3% and 99.5%. The most relevant method evaluated in the Middlebury webpage is the method proposed in [85]. The method [85] first reconstructs the scene using the "baseline" method and then it refines the reconstruction using a final mesh refinement step. It is important to note that we do not use any final mesh refinement step in the proposed method. Nevertheless, the proposed method is among the best methods in the evaluation page. Of course, the results will vary if a different depth-map estimation method is employed.

9.3.5. Results of proposed method on real-world datasets

Figures 9.8 (bottom row), 9.7 demonstrate that the proposed method reconstructs weakly-supported surfaces better than the baseline method. The weakly-supported surface in the Figures 9.8, 9.7 is mostly the ground plane. Figure 9.8 (the first two rows) shows other results using the proposed method where the weakly-supported ground plane is reconstructed.

Figure 9.7.: **Results for the 'lausanne' data set.** Reconstruction using the baseline approach is in the first two columns, Reconstruction using our approach is in the second two columns. Textured results are shown in the first and the third column of the images. Corresponding shaded results are in the second and the fourth columns. Weakly-supported surfaces are better reconstructed using the proposed approach than by the baseline approach.

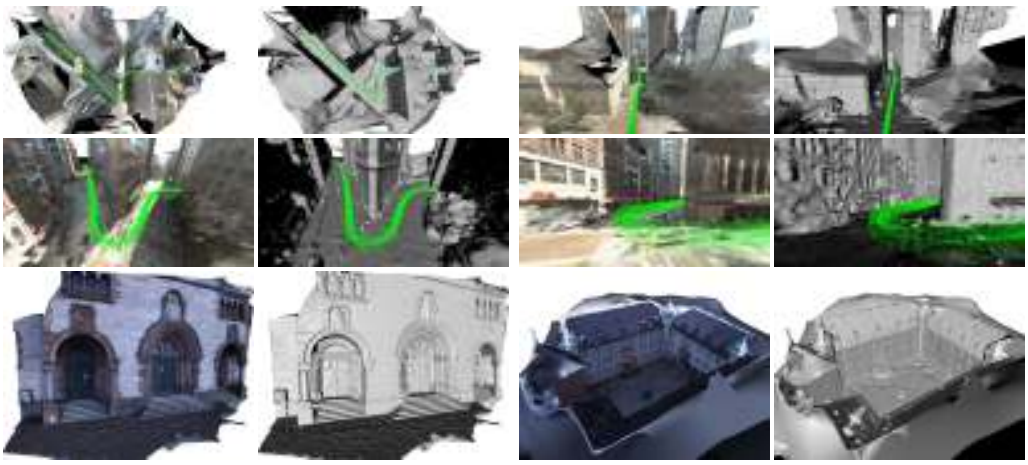


Figure 9.8.: **Results for the "googleStreetView", "herzjesu", and "castle" datasets.** Odd columns textured, even shaded.

10 Hallucinations

We have introduced the concept of occluder detection just from the visibility of input points in the beginning of the Chapter 7. We have defined the hallucinations in the Section 7.1. We have also explained in the Section 7.1 that it is not always possible to distinguish between weakly-supported real surfaces (e.g., trees or road) and hallucinated ones (e.g., the sky) from sole 3D data.

In this chapter we propose two methods for hallucination removal. First, method proposed in Section 10.1 is based on the observation that small variation of input data leads to large change of hallucinated surface. We detect these large changes and so we detect and remove hallucinated surface. This method works well but it has two disadvantages: (i) it removes also weakly-supported surfaces and (ii) it is relatively slow.

Secondly, in order to solve the two above problems we propose a new method in Section 10.2. Weakly-supported real surfaces as well as hallucinated surfaces consist of larger triangles than strongly-supported real surfaces. Therefore, removing relatively large triangles ([22, 52], Section 10.1) can lead to the removal of weakly-supported real surfaces. Therefore, we need to use prior knowledge in order to distinguish them. We focus on the special type of datasets, where sensors take measurements of a city near the ground plane, called "street view datasets". We use "sky hallucinations" to denote weakly-supported hallucinated 3D surfaces reconstructed above real 3D structures. Sky hallucinations often appear in reconstructions from street view data since there are no sensors observing the city from above. In Section 10.2, we incorporate the sky prior into our new approach to hallucinations removal in order to be able to keep the reconstructed weakly-supported surfaces while removing the hallucinated ones.

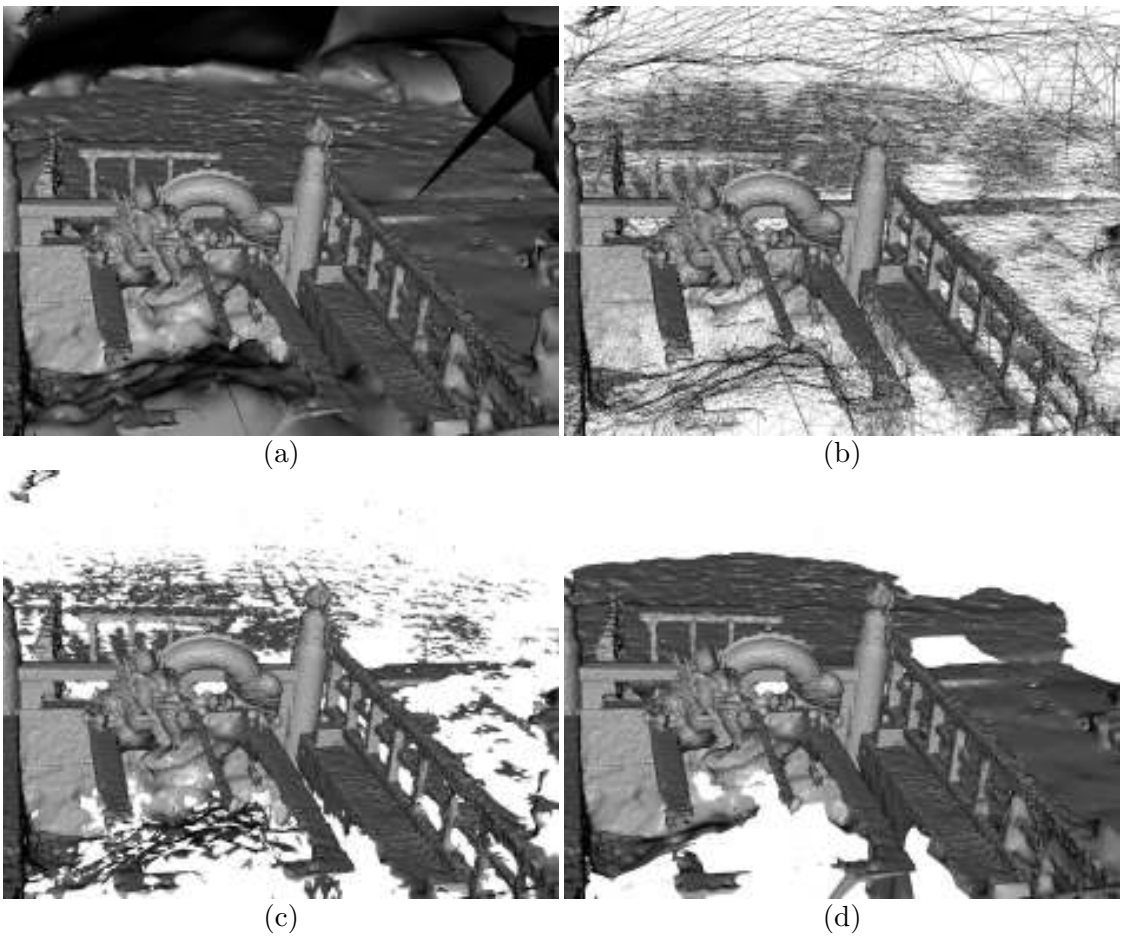
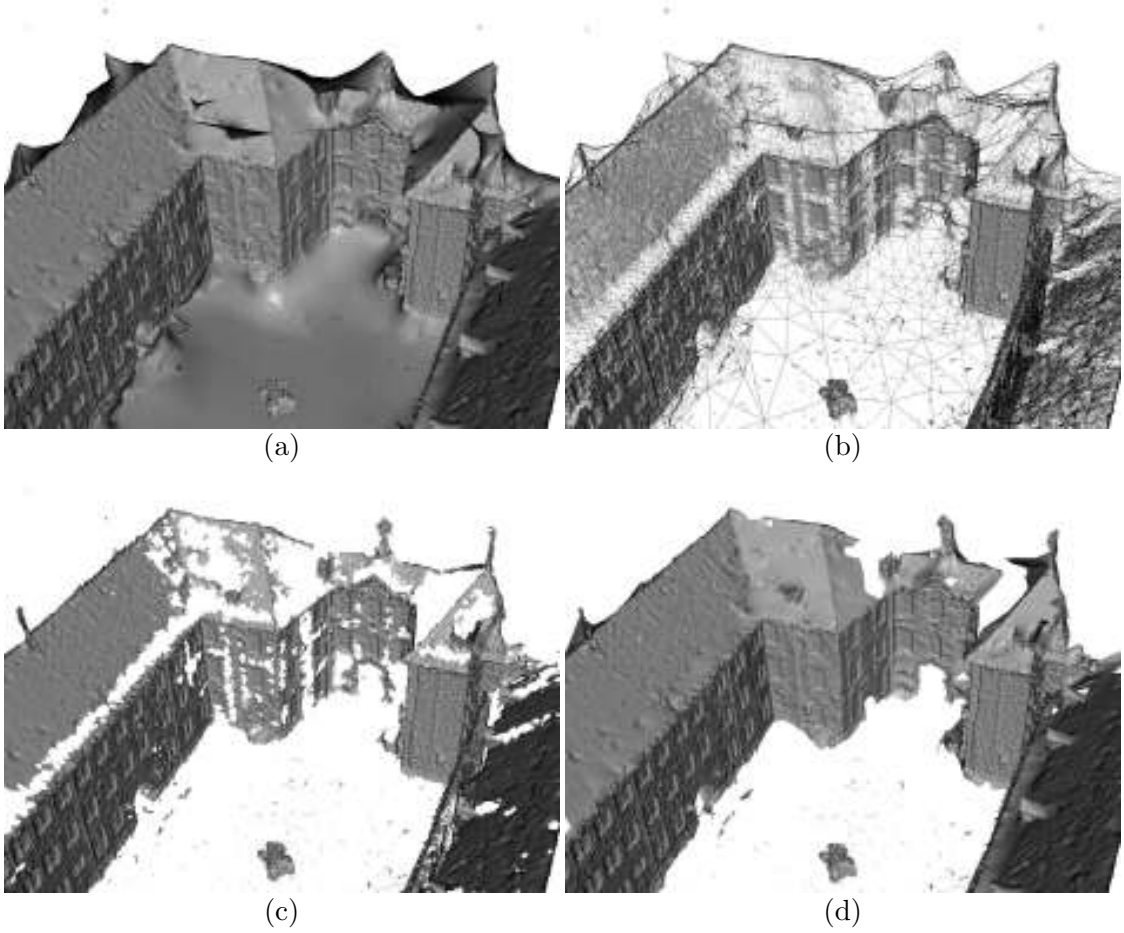


Figure 10.1.: **Dragon-P114 data set:** (a) 3D surface before removing hallucinations, (b) 3D surface wireframe before removing hallucinations, (c) 3D surface after removing hallucinations using approach proposed in [21], (d) 3D surface after removing hallucinations using our method.

10.1. Hallucination-free Multi-View Stereo

We present a multi-view stereo method that avoids producing hallucinated surfaces which do not correspond to real surfaces. Our approach to 3D reconstruction is based on the minimal s-t cut of the graph derived from the Delaunay tetrahedralization of a dense 3D point cloud, which produces water-tight meshes, see Section 7.3. This is often a desirable property but it hallucinates surfaces in complicated scenes with multiple objects and free open space. For example, a sequence of images obtained from a moving vehicle often produces meshes where the sky is hallucinated because there are no images looking from the above to the ground plane. We present a method for detecting and removing such surfaces. The method is based on removing perturbation sensitive parts of the



(a) (b)

(c) (d)

Figure 10.2.: **Castle-P30 data set:** (a) 3D surface before removing hallucinations, (b) 3D surface wireframe before removing hallucinations, (c) 3D surface after removing hallucinations using approach proposed in [21], (d) 3D surface after removing hallucinations using our method.

reconstruction using multiple reconstructions of perturbed input data. We demonstrate our method on several standard datasets often used to benchmark multi-view stereo and show that it outperforms the state-of-the-art techniques.

10.1.1. Motivation

The approach by using some triangle (or related tetrahedron) property like average edge length [21], maximal edge length, triangle area, radius of circumscribed sphere of the related tetrahedron, and so on, can not be used to remove hallucinated surfaces well in general. The reason is that it is possible (and we demonstrate it in our experiments) that a reconstruction pipeline will produce the mesh where one can often find two subsets of

triangles of the same average edge length of the mesh such that the first subset corresponds to a hallucination but the second one corresponds to a real surface. Therefore, it is in general not possible to find a threshold on these dimensional parameters which would separate hallucinated triangles, see Figure 10.5 .

The main idea behind our approach is motivated by the following observation. The surfaces which have strong support from the data are mostly not affected by small perturbations. By the strong support we mean that there are many reconstructed points near the real surface. On the other hand the surfaces, which are originally created due to false positive points, are usually strongly affected by perturbations because false positives are generated randomly and usually sparsely distributed far from true surfaces.

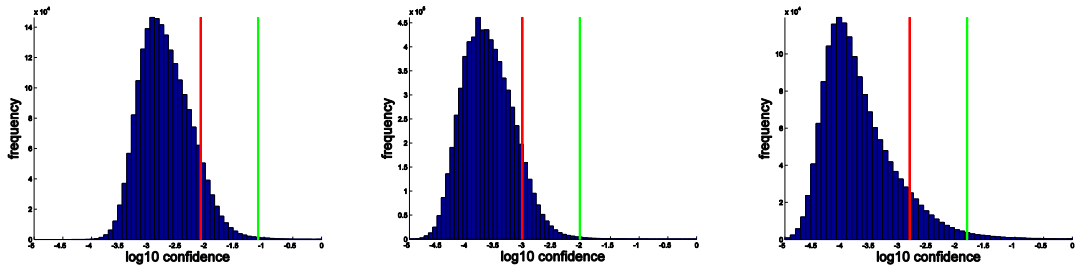
We assign a confidence value to each triangle based on the sensitivity to perturbations (see Section 10.1.5). We build the s-t graph [18] from the triangulation and the confidences and solve it by the implementation [7]. If a triangle is labelled as sink we deem the triangle as hallucinated.

One can argue that large triangles will be mostly created from false positive 3D points. That is often true but large triangles are also created in texture-less parts of the scene (see Figure 10.2) and in parts which have very oblique viewing angles (see Figure 10.1). Such triangles are important because they make the reconstruction complete. The main contribution of our method is to keep such triangles in the reconstruction while removing the triangles that are hallucinated.

We have to point out that our method may keep unseen parts of the surface when they are a part of the visual hull of the scene (see Figure 10.6 with the roof region behind dormers is filled). We do not consider such parts as hallucinations.

10.1.2. Reconstruction pipeline

Our MVS pipeline is similar to the pipeline proposed in [85]. First, we compute feasible camera pairs based on the epipolar geometry as in [45]. Next, we detect and match SIFT features [57] in the feasible camera pairs using [88]. We triangulate matches and create seeds. A seed is a 3D point with a set of cameras it was triangulated from. For each camera we compute the minimal and the maximal depth based on the related seeds. Then, we perform the plane-sweeping and filtering (see Section 10.1.3) at several scales. To remove hallucinated surfaces, we run a mesh computation k times from differently perturbed data. In each iteration we perturb (see Section 10.1.4) the point cloud generated by plane-sweeping and use it as the input to our implementation of the method proposed in [52]. We do not perform mesh refinement as in [85] but do mesh smoothing as in [52]. This gives us k meshes. We remove hallucinated surfaces from the first mesh using other meshes (see Section 10.1.5 and 10.1.6). This gives us the resulting mesh.



(a) (b) (c)

Figure 10.3.: **Histogram of log_{10} of confidences for:** (a) Castle-P30 dataset, (b) Fountain-P11 dataset and (c) Dragon-P114 dataset with s (red) and t (green) values. (see text)

10.1.3. Plane-sweeping and filtering

Our plane-sweeping is slightly different from the state-of-the art [23, 91, 13]. We do plane sweeping for each reference camera with respect to the set of α nearest feasible target cameras (we use $\alpha = 4$ in all of our experiments). For each pixel $\mathbf{p}$ and for each depth d (corresponding to a plane parallel to the reference image plane) we compute the photo-consistency $f(\mathbf{p}, d)$ of the depth as follows. For each pair of the reference r and target t cameras we compute photo-consistency value $c(\mathbf{p}, d, r, t)$ as the NCC between 5×5 window centered in the pixel on the reference image and its projection to the target image. The projection is generated by the homography inducted by the plane and consistent with the epipolar geometry between the reference and target images. The NCC value is in the range $\langle -1, 1 \rangle$, where -1 corresponds to the worst photo-consistency and 1 corresponds to the best one. The photo-consistency $f(\mathbf{p}, d)$ equals the maximum of $c(\mathbf{p}, d, r, t)$ over all target cameras t and fixed r. The reconstructed depth $\gamma(\mathbf{p})$ of the pixel $\mathbf{p}$ is chosen as the depth d for which $f(\mathbf{p}, d)$ is maximal and $f(\mathbf{p}, d) > \delta$ (we use $\delta = 0.8$ in all of our experiments). If there does not exist such depth, then we set $\gamma(\mathbf{p})$ as unknown. This plane-sweeping strategy produces a lot of true positive 3D points, but a lot of false positive ones too. Therefore, we perform a simple but fast and effective filtering after plane-sweeping. For each 3D point we search for other 3D points in its small neighborhood. If there are at least β (we use $\beta = 2$ in Fountain dataset and $\beta = 3$ in all others) 3D points from β different cameras, then we accept the point. We consider all depths which were filtered out as unknown. We choose this approach because we have experimentally verified that this approach produces more true positives than, for example [91, 13]. On the other hand it still (even after filtering) produces some false positives. But this is not critical because we are later using a strong tool [52] which can effectively deal with noise.

10.1.4. The principle of the removal of hallucinated surfaces

Method [52] tends to produce closed meshes and hence it generates false (hallucinated) surfaces in places which are not well captured by any camera. The hallucinated surfaces are often related to missing cameras which would otherwise lead to cleaning the space between the cameras and the real surface of the scene. When cameras are not present, method [52] hallucinates surfaces from sparsely distributed false positive points which are present in the point cloud. On the other hand, surfaces, which are strongly supported by the data, i.e. where the point cloud is dense near the real surface, or which are strongly supported by a visibility prior, and hence do not have to be strongly supported by the data, are not affected by the sparsely distributed false positive points. We build our approach on this observation. We introduce a small amount of false positive points into the original point cloud several times and filter out unstable hallucinated surfaces. We implement it by adding a small amount of noise into the depth-maps constructed by plane-sweeping in each iteration. Consider a depth-maps and all the pixels with unknown depth in that map. We randomly choose γ (we use $\gamma = 0.1$ in all of our experiments) percent of the pixels with unknown depth and assign them depths randomly. The new depth is chosen randomly from the four times the depth range of the camera. The values were selected experimentally and were sufficient in all of our experiments.

10.1.5. Perturbation based triangle confidence computation

We assign a confidence value to each triangle of each of the k meshes. Let's assume the $i-th$ mesh and the $j-th$ triangle t. For each $k-th$ mesh $k \neq i$ we find the nearest triangle t_k to the triangle t. We measure the distance of triangles by the distance of the triangle centers $(\mathbf{c_t}, \mathbf{c_{t_k}})$. Now, for each pair (t, t_k) of triangles we compute $d(t, t_k) = \min\{d(\mathbf{c_t}, t_k), d(\mathbf{c_{t_k}}, t)\}$ over all pairs t_k, t (with fixed t) where $d(\mathbf{c_t}, t_k)$ is the distance of the point $\mathbf{c_t}$ to the plane defined by triangle t_k. To compute the confidence of the triangle $\delta(t)$ we use Gaussian kernel voting to cluster values $d(t, t_k)$.

10.1.6. Graph-cut based hallucinations removing

To remove triangles with high confidence, we formulate a minimum s-t cut problem [18]. We create a graph from the mesh such that the nodes correspond to triangles. If two triangles are neighboring, then we create the edge between the corresponding nodes. We compute $90th$ percentile s of all triangle confidences to find the threshold on the triangle confidence. The threshold at the $90th$ percentile is very conservative because majority of confidences are from small triangles which have usually similar confidences that are close to zero, see Figure 10.3. We introduce value $t = 10\,s$. Value s should correspond to triangles which should be definitely in the final mesh, value t to triangles which should definitely be removed. We assign $(s - \delta(t))^2$ value to each s-edge and $(t - \delta(t))^2$ value to each t-edge. To each edge between nodes we assign value $s + (t - s)/4$. This value is established experimentally to remove isolated triangles. We use this value in all of our

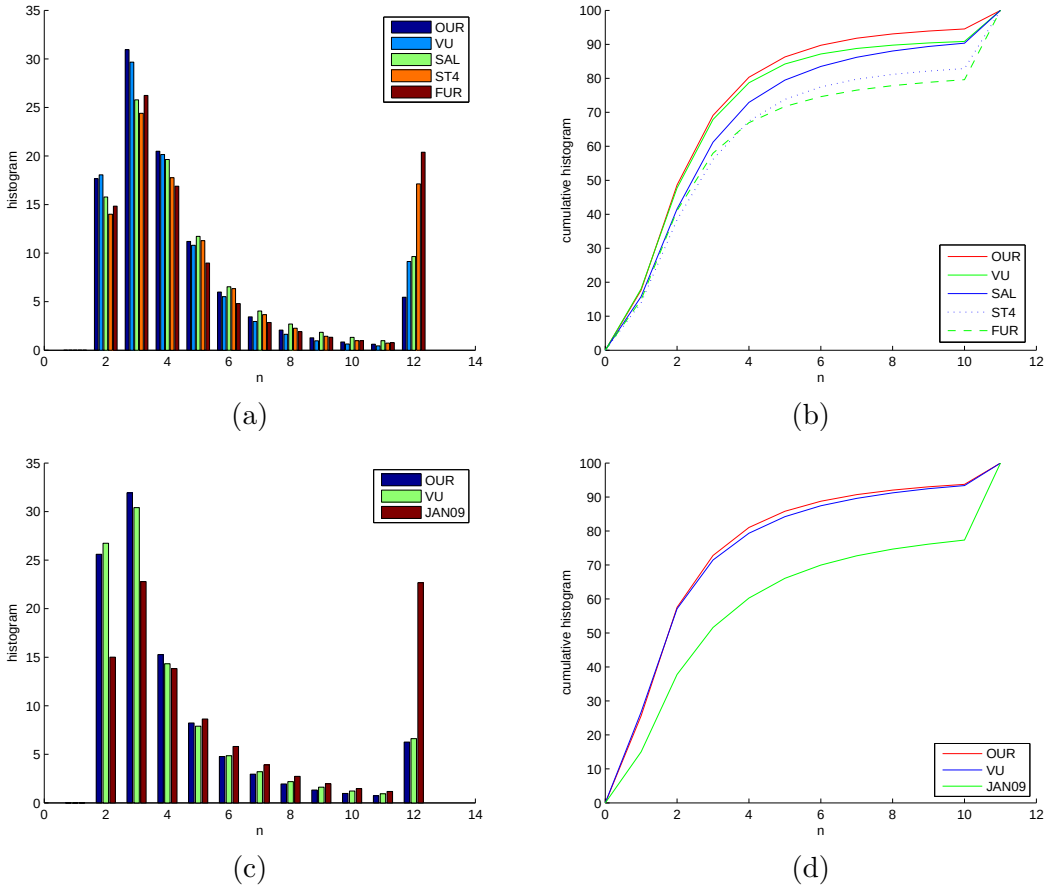


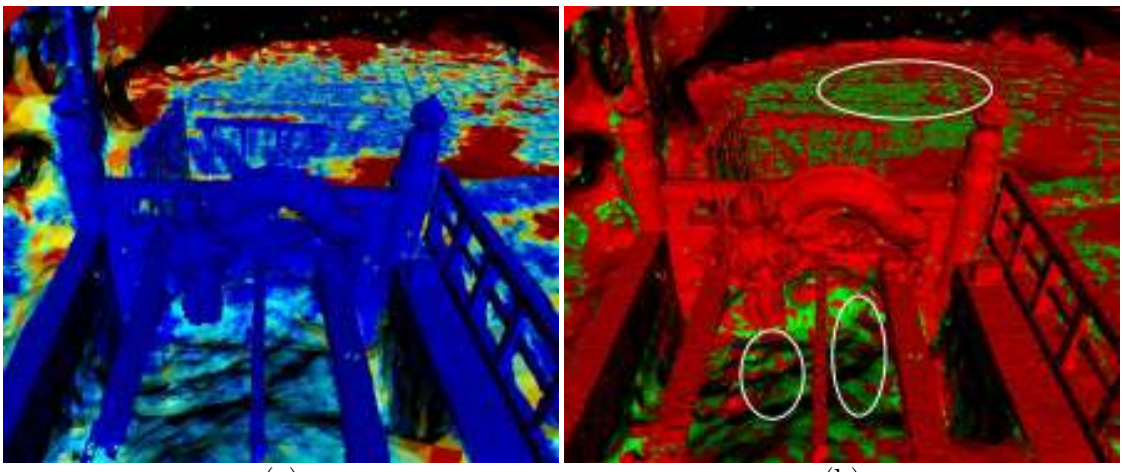
Figure 10.4.: **Strecha's evaluation [77] for the Fountain-P11.** (a,b) and Castle-P30 (c,d) data sets. OUR - proposed method, VU [85], SAL [66], ST4 [74], FUR [21], JAN09 [45]. (a,c) histograms of the relative error with respect to $n\,\sigma$ for all views. The σ is determined from reference data by simulating the process of measurement and can vary across the surface and views. (b,d) relative error cumulated histograms.

experiments. To solve the s-t cut problem we use the implementation described in [7]. The final mesh consists of the triangles represented by the s-nodes.

10.1.7. Performance discussion

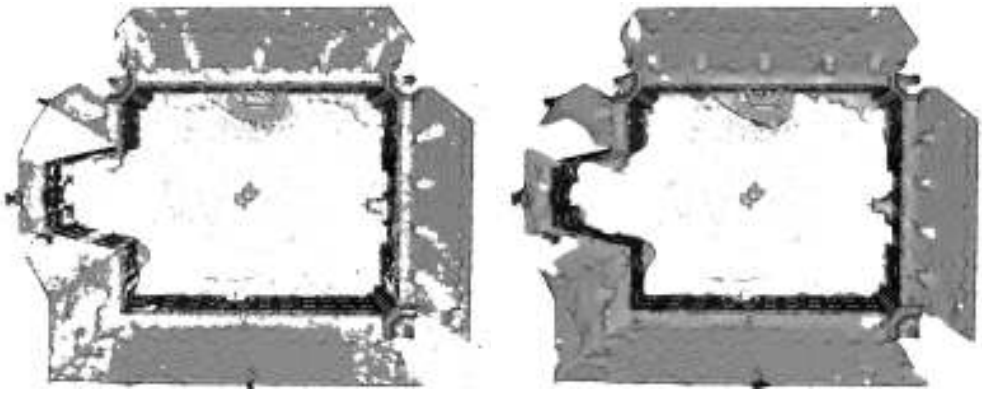
In this section we provide the time performance discussion of our pipeline. We discuss how to make significant speedup, too.

The plane-sweeping was performed on the original scale and on two, three and four times sub-sampled images. The plane-sweeping is implemented on GPU. The computation time of one depth-maps varies from a few minutes on 3072×2048 resolution to



(a) (b)

Figure 10.5.: **Dragon-P114 data set:** (a) 3D surface before removing hallucinations colored by average edge length of the triangle, blue - smallest, red - 10-times average edge length of the whole mesh and more (using JET color model), (b) red - triangles with average edge length smaller than 3-times or larger than 6-times average edge length of the whole mesh, green - triangles with average edge length between 3-times and 6-times average edge length of the whole mesh, top ellipse - real surface, bottom ellipses - hallucination. Conclusion: average edge length does not separate real surface triangles from the hallucinated ones.



(a) (b)

Figure 10.6.: **Castle-P30 data set (top view):** (a) 3D surface after removing hallucinations using approach proposed in [21], (b) 3D surface after removing hallucinations using our method.

a second on the smallest scale (768×512). The time complexity is cubic, because the number of depths scales with the image resolution. The computation time of the filtering

step is approximately tens of seconds. We have to point out that the plane-sweeping and filtering is performed only once for each camera at each scale. The computation time of one perturbation iteration using our implementation of [52] is approximately tens of minutes. The time depends on the number of input points. In our experiments the number of points varies in the range of one to six millions. The computation time of consistencies and graph-cut based hallucinations removal is approximately minutes.

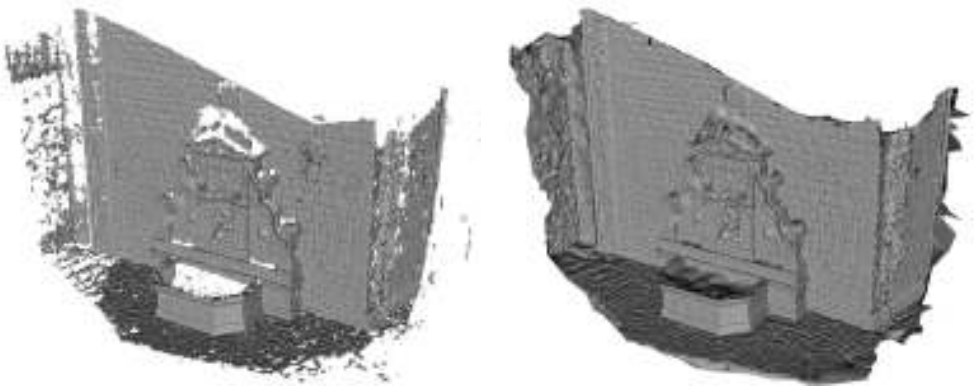
We have tested our method with $k = 10$. We think that $k = 3$ should be enough, too. The code can be later optimized with respect to that only small amount of 3D points are changing in each iteration. Therefore we would build the triangulation from original points and remember it. Later we would add perturbed points (0.1% of original points in all of our experiments) and update the weights to the triangulation in each iteration. This optimization would cause that the computation time of all iterations should be similar to the computation time of one iteration.

10.1.8. Results

Based on the experiments we observed that using the criterial function based on the triangle (or related tetrahedron) property like average edge length, maximal edge length, triangle area and so on, can not be used to remove hallucinated surfaces with sufficient quality. Figure 10.5 shows the input mesh (including hallucinations) colored by the average edge lengths using JET color model. Red color represents the triangles whose average edge lengths are greater or equal to 10-times average edge length of the whole mesh. Blue color represents the triangles which average edge lengths is close to zero. Areas marked by the ellipses show two different parts of the mesh. Both of them (see the distribution of colors) contain the triangles with the same average edge lengths. However, the top set of triangles represents real object part and the bottom ones are hallucinated. This example demonstrates that it is impossible to find a threshold which would separate the hallucinated part from the real one in general. We carried out several experiments on this dataset using maximal edge length, triangle area and maximal radius of circumscribed sphere of the related tetrahedron, and all of them produced similar results.

To evaluate the quality of our reconstructions, we present results on data sets from the standard Strecha's [77] evaluation database. We show the result for three different outdoor datasets: Fountain-P11, Castle-P30, Dragon-P114. The first two datasets are Strecha's datasets [77] and the last one is the data set of a dragon's sculpture in Kyoto. Strecha's Fountain-P11 data set contains 11 3072×2048 images. Strecha's Castle-P30 data set contains 30 3072×2048 images. Dragon data set contains 114 1936×1296 images.

Figure 10.4 shows the evaluation on the Strecha's Fountain-P11 as well as Castle-P30 data sets. See the Strecha's evaluation page for JAN10 results and their comparison. The histograms show that our reconstructions are more or less on the same level at $2\,\sigma$ and $3\,\sigma$ as the method [85] which uses an additional mesh refinement step. The cumulative histograms shows that our method outperforms all other methods in completeness. In



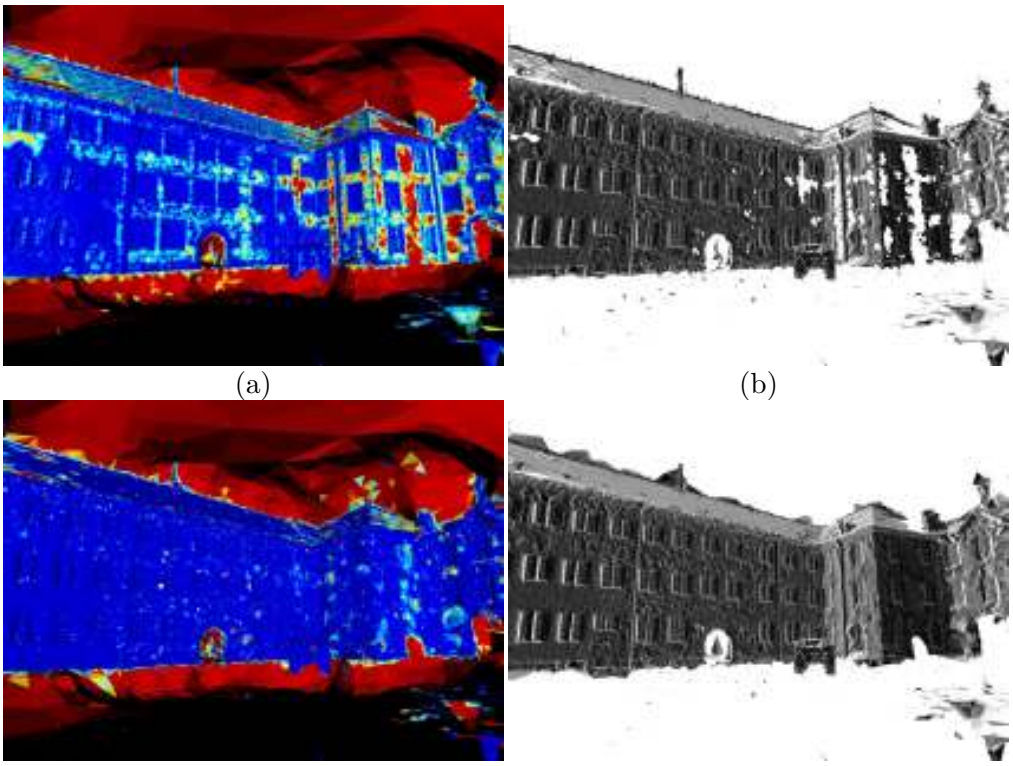
(a) (b)

Figure 10.7.: **Fountain-P11 data set:** (a) 3D surface removing hallucinations using approach proposed in [21], (b) 3D surface after removing hallucinations using our method.

Figure 10.4 (a) and (b) we are comparing our results with four best methods [85, 66, 74, 21]. For complete results, we refer the reader to the challenge website [73]. This experiment demonstrates that methods based on the opportunistic approach [52] produce complete and accurate results and outperforms the other state-of-the-art methods, and it is therefore important to deal with its negatives which was the goal of this chapter.

We made several experiments to demonstrate that our method produces better results than the state-of-the-art approach proposed in [21], which solves this problem by removing large triangles, i.e. they discard the triangles which average edge length is greater than six times the average edge length of the whole mesh.
Figures 10.1 and 10.5 show the comparison of the results computed using the approach proposed in [21] with the results computed using our method on Dragon-P114 data set. Figures 10.6 (a) and (c) show that our method can better deal with larger triangles which are on the ground and which cover surfaces captured at oblique angles. Figures 10.6, 10.8 and 10.2 show the comparison of the results computed using the approach proposed in [21] with the results computed using our method on Castle-P30 data set. Figures 10.6 (a) and (c) show that our method can better preserve large triangles which are in between the windows where is a lack of the texture but which are not hallucinated. Figure 10.7 shows the comparison of the results computed using the approach proposed in [21] with the results computed using our method on Fountain-P11 data set. It shows that our method can avoid discarding important low-textured parts of the scene.

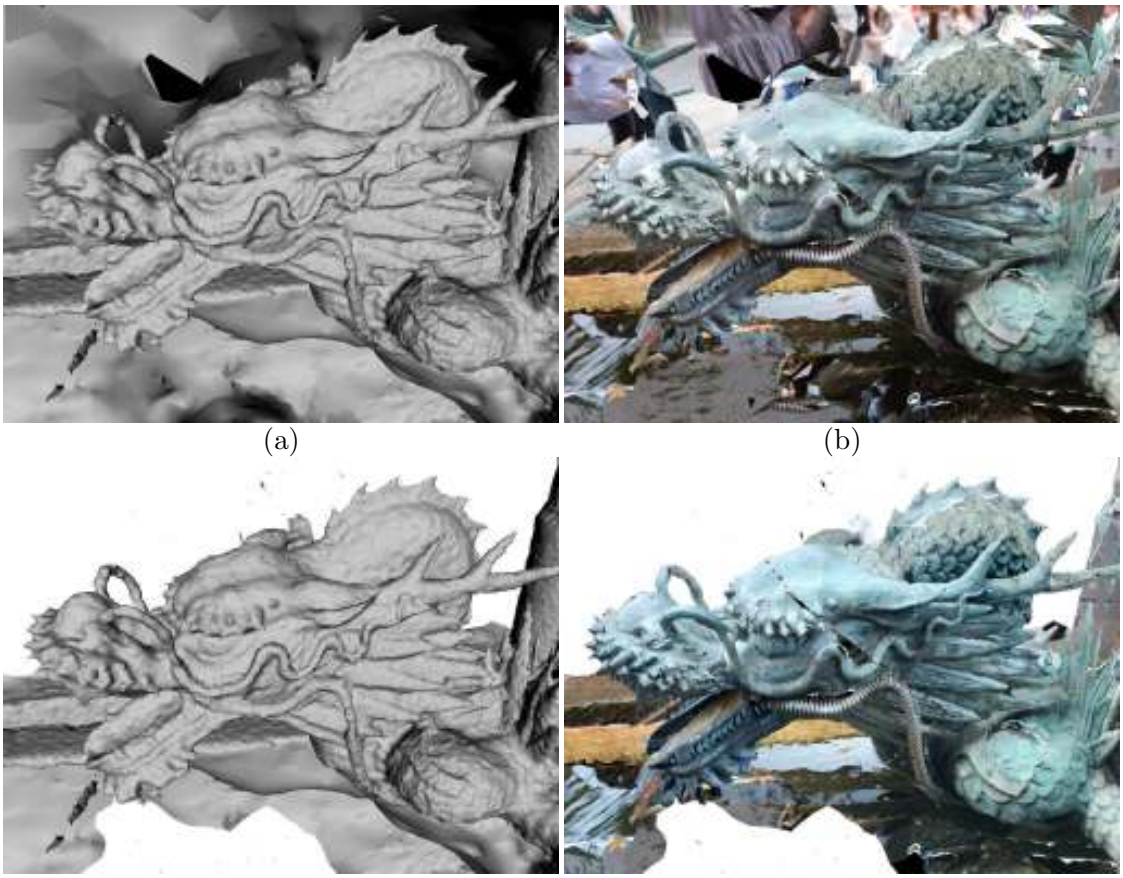
Figure 10.9 shows the detailed view of the dragon's head before and after removing hallucinations using our method (untextured and textured). We have used the nearest camera to texture each triangle without texture color unification.



(a) (b)

(c) (d)

Figure 10.8.: **Castle-P30 data set:** (a) 3D surface before removing hallucinations colored by average edge length of the triangle, blue - smallest, red - 10-times average edge length of the whole mesh and more (using JET color model), (b) 3D surface after removing hallucinations using approach proposed in [21], (c) 3D surface before removing hallucinations colored by the perturbation based confidence (blue - smallest confidence, red - largest confidence), (d) 3D surface after removing hallucinations using our method.



(a) (b)

(c) (d)

Figure 10.9.: **Detailed view of the dragon's head.** (a) 3D surface before removing hallucinations untextured, (b) 3D surface before removing hallucinations textured, (c) 3D surface after removing hallucinations using our method untextured, (d) 3D surface after removing hallucinations using our method textured

10.2. Volumetric based hallucination Removal

In Section 10.2.1, we describe an easy approach that removes dust, bubbles and large triangles. In Sections 10.2.2 and 10.2.3, we describe a more advanced approach that exploits prior knowledge in datasets from city environments, where input points were computed from sensors located near the ground plane.

10.2.1. Removing dust, bubbles and large triangles

Our goal is to reconstruct objects that are defined by the full label and typically consist of thousands of tetrahedra. We segment all full (free) tetrahedra that have a common face into connected components, which we call full (free) objects. We consider all small full (free) objects as hallucinations and remove them by relabeling them to free (full). Full (free) object is small when it consists at most of d (we use d=10 in all of our experiments) tetrahedra. We say that we remove dust (bubbles). Finally, we use an easy technique that removes very large triangles from the final mesh. It is the same approach as the one used in [22]. We compute average edge length over all edges in the mesh and remove all triangles where the maximal edge is larger than 100-times the average length.

10.2.2. Finding weakly-supported full tetrahedra

After s-t cut optimization and dust and bubbles removal (isolated small full (free) connected components), we have an initial full-free labeling of tetrahedra. Due to properties of the proposed method we can be sure that what is labeled as free is free in the real world. On the other hand what is labeled as full does not have to be full in the real world. Following this property we can compute a full space support of all tetrahedra labeled as full in a similar way as the free-space-support is computed. We do it as follows. Given the initial labeling we consider interface input points of tetrahedralization that are on the interface defined by the full-free labeling. After the s-t cut optimization, we do the following steps:

1. We introduce a new weight $\theta_{in}(i)$ for every tetrahedron i. We call it "full space support". Initially, we set it to zero.

2. For each segment of sight (c, p), where p is the interface vertex, we do the following. We iteratively traverse all crossing tetrahedra T_i one by one from first tetrahedron behind the surface vertex on the line of sight (c, p) while they are labeled as full. The value $\theta_{in}(i)$ of each such tetrahedron T_i is increased by $\alpha(p)$.

This means that the full space support of an full tetrahedron T_i is such a number of segments of sight (c, p) where (i) interface vertex p occludes (occludes = projects to the same area as tetrahedron T_i in the sensor c) the tetrahedron T_i in the sensor c and (ii) all tetrahedra between the surface vertex and the tetrahedron T_i crossing the line of sight (c, p) are labeled as full.

Now we can relabel all full tetrahedra that are weakly-supported by the full space support value as free, i.e. when the full space support is smaller than a threshold.

On the other hand, such relabeling can also destroy small surface tetrahedra which typically have small full space support. Therefore, it is not sufficient to use only this thresholding approach. We set t-weights (of the actual graph) of full tetrahedra that are weakly-supported by the full space support to zero and run optimization again instead. Therefore, the optimization process will tend to label them as free. Figure 10.10(f) shows $\theta_{in}(i) \geq 30$ in red. Table 10.1 shows the overall time of $\theta_{in}(i)$ computation in t_{SKY} column.

In street view like datasets, however, threshold that would differentiate weakly-supported sky from the weakly-supported ground plane does not exist in general. The next section describes how we deal with this problem.

10.2.3. Using sky prior to removing sky hallucinations

In this section we describe a technique that we use to remove sky hallucinations from results on street view like datasets. Since our goal is to remove sky hallucinations, we use a sky prior to dividing all tetrahedra into sky tetrahedra and the non-sky tetrahedra. Our goal is to use full space support only for tetrahedra that are high above the ground plane, i.e., for the sky tetrahedra to distinguish between weakly-supported ground plane and weakly-supported sky surface. First we compute the gravity vector from the first sensor. We assume that the first sensor oriented such that the vertical direction in the image is also vertical in the world. Based on the gravity vector, we choose the top part of images that most likely captures sky tetrahedra. The size of the top part can be controlled by the parameter u (we use $u = 0.3$ in all of our experiments).

result name	t_{o_1}	t_{o_2}	t_{GC}	t_{SKY}	n_p	n_c	n_t	n_K	T_t	T_f
Fountain	12 : 27	2 : 44	0 : 44		$4.6M$	2.7	113.1	39.7	$30.0M$	$60.1M$
Street-view	05 : 00	2 : 34	0 : 52	0 : 26	$2.9M$	2.5	097.5	34.2	$19.6M$	$39.2M$
Lausanne	20 : 02	3 : 28	0 : 53	1 : 17	$4.9M$	2.7	205.3	50.1	$32.7M$	$65.4M$

Table 10.1.: **Performance data for different results.** t_{o_1} is the time of the first part of the proposed algorithm and t_{o_2} is the time of the second one. t_{GC} is the time of solving the minimal s-t cut problem. t_{SKY} is the time of $\theta_{in}(t)$ computation. The times are in the format: $minutes$: $seconds$. n_p, n_c, n_t are defined in Section 9.2. T_t is the number of tetrahedra and T_f is the number of faces in the tetrahedralization. The letter M stands for one million, i.e. 10^6.

For example, for image height with 1000 pixels, we may set the top part to the top 300 rows. This is the sky prior. We call sky vertices all vertices of tetrahedralization that project to the sky area of an associated image.

We call sky tetrahedra all tetrahedra that have at least one sky vertex associated. Next, we set t-weights (of the actual graph) of tetrahedra originally labeled as full with $\theta_{in}(i) < q$ to zero and run optimization again (we use $q = 30$ in all of our experiments).

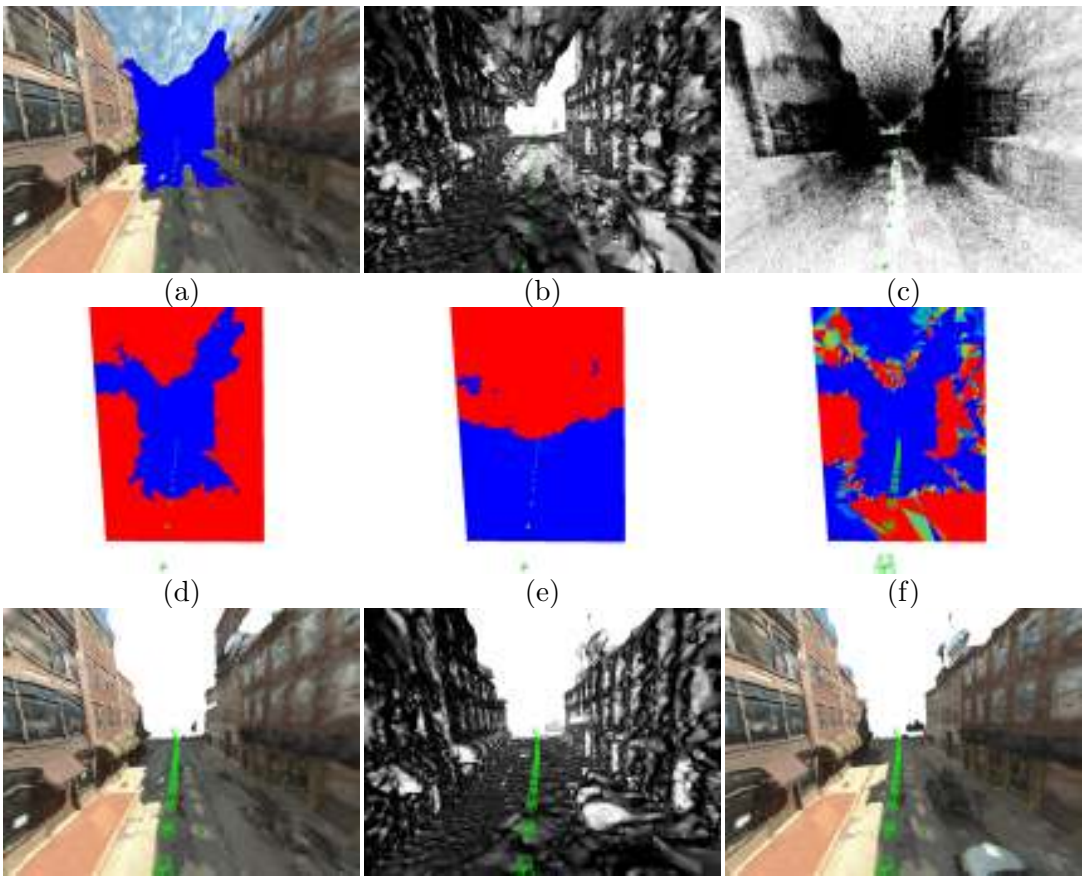


Figure 10.10.: **Removing sky hallucinations.** (e) Input points computed from input images using plane-sweeping method described in [44], (c,f,g,h) plane manually placed into the scene colored by values of crossing tetrahedra. (c,d) result of the [52] method (without using sky prior based hallucination removal), (c,f) plane colored blue (free), red (full), (h) plane colored by $\theta_{in}(t)$ blue (0) to red (≥ 30) using JET color map, (g) plane colored by sky thetrahedra blue (non sky) and red (sky), (i) reconstruction using the [52] method with using sky prior based hallucination removal (j,k) reconstruction using the proposed method. The original method (c) does not reconstruct weakly-supported ground plane, cars, and trees and the sky is hallucinated. The hallucinated sky is removed (i) using the proposed sky prior to remove hallucinations. The hallucinated sky is removed and additionally weakly-supported ground planes, cars, and trees are preserved when using the proposed method (j,k).

Figure 10.10 illustrates the hallucination removal process. Figure 10.10(e) shows sky

tetrahedra colored in red. Figure 10.10(g) shows the reconstruction using method [52] with using sky prior based hallucination removal. Figure 10.10(h,i) shows the reconstruction using the proposed method of using sky prior. The method [52] (a,b) does not reconstruct weakly-supported ground planes, cars, and trees and the sky is hallucinated. The hallucinated sky is removed using the proposed sky prior to remove hallucinations. The hallucinated sky is removed and additionally weakly-supported ground plane, cars, and trees are preserved when using the proposed method Figure 10.10(h,i).

Algorithm 4 Sky-hallucinations free surface reconstruction preserving weakly-supported surfaces

Require: DT - Delaunay Tetrahedralization, a subset of sensors associated to each vertex of the tetrahedralization

function RECONSTRUCT(DT, sky)
  Compute the s-t graph weights ▷ t_{o_1}
  for all (sensor c, vetrex p) **do** ▷ t_{o_2}
    Compute $K(c,p)$
    if $K(c,p) == INT$ **then**
      Add the value $\epsilon_{cp}^{abs}(k_f, k_b)$ to weight of t-edge (v,t) where $L_c^p(k_b\sigma) = T(v)$
    end if
  end for
  Compute minimal s-t cut ▷ t_{GC}
  Remove dust and bubbles
  if $sky == TRUE$ **then**
    Precompute $\theta_{in}(i)$ (for each tetrahedron T_i) ▷ t_{SKY}
    for all tetrahedra T_i **do**
      if (T_i is sky tetrahedron)&&($\theta_{in}(i) < q$) **then**
        Set weight of t-edge (v,t) where $T_i = T(v)$ to zero
      end if
    end for
    Compute minimal s-t cut
    Remove dust and bubbles
  end if
  Create mesh from the minimal s-t cut
  Smooth
  return Mesh
end function

Robustness of the method Let us discuss a situation when we mark all tetrahedra as sky thetrahedra i.e., $u = 1.0$. In this situation full tetrahedra that have low full-space support i.e., $\theta_{in}(i) < q$ should be relabeled as free. However, the value $q = 30$ is very conservative and therefore the majority of very strongly-supported surfaces will remain unchanged in such a situation but all other surfaces will be removed. Figure 10.10 shows

that the top parts of facades were not removed despite that they were in the sky part. On the other hand, using more advanced techniques that detect a sky in images can lead to even better results.

11 Scalability

In this chapter we describe a new method for the large-scale 3D multi-view reconstruction problem. The proposed method can be, for instance, used to reconstruction Urban Environment. The input to our method is a set of calibrated cameras and a cloud of 3D points, where each point has associated a subset of cameras which it was created from as was defined in Chapter 7.

Since we are working with huge data, we can't solve the problem globally using any of the method proposed in Chapters 7, 9, and 10 that are all based on the same global volumetric optimization approach described in the Section 7.3. Therefore, we divide 3D space into parts represented by boxes and we reconstruct each part separately. Each box should be as large as possible in order to fit input 3D points which lie inside the box into the memory. This property should guarantee that the reconstruction of a part will be detailed and complete. The problem which we address in this chapter is that the 3D reconstruction of the points, which lie inside a single box, may contain artifacts (hallucinations) when the box is being reconstructed independently from the rest of the scene. The artifacts are mostly located near the box borders. Therefore, we propose a method to reconstruct individual boxes as if the complete scene was reconstructed globally by propagating constraints between neighboring boxes. We show experimentally that our method works on real datasets.

11.1. Semi-global 3D reconstruction by parts

We follow here the basic approach described in Section 7.3. The motivation is that we want to obtain the same result on a part (box) which we would obtain if solving the problem globally using all the input 3D points. Let us denote the global DG of all 3D points as G. Since we cannot solve the problem globally due to memory limitations, we take just points which lie inside the box and create a smaller local DG g. The goal now is to compute all the weights of the local DG g such that they match corresponding weights of the global DG G but using an out-of-core approach.

First, we compute local DG g weights using all the points inside the box. Next, we update the weights of the local DG g using all the points outside the box which would contribute to the weights of the local DG g but without storing them all together in the memory. We do it as follows.

First, we divide the input 3D point cloud into files. We create one file for each camera.

Each file contains all points from the input 3D point cloud which are associated with the related camera. This is done in a preprocessing step.

Next, we take all cameras which are incident to the box and iterate through associated files. For each file we iterate trough the line segments (the camera centre, a point $\mathbf{p}$) without adding them into the memory. We update the graph weights for each line segment which intersects the box and the point $\mathbf{p}$ lies outside the box.

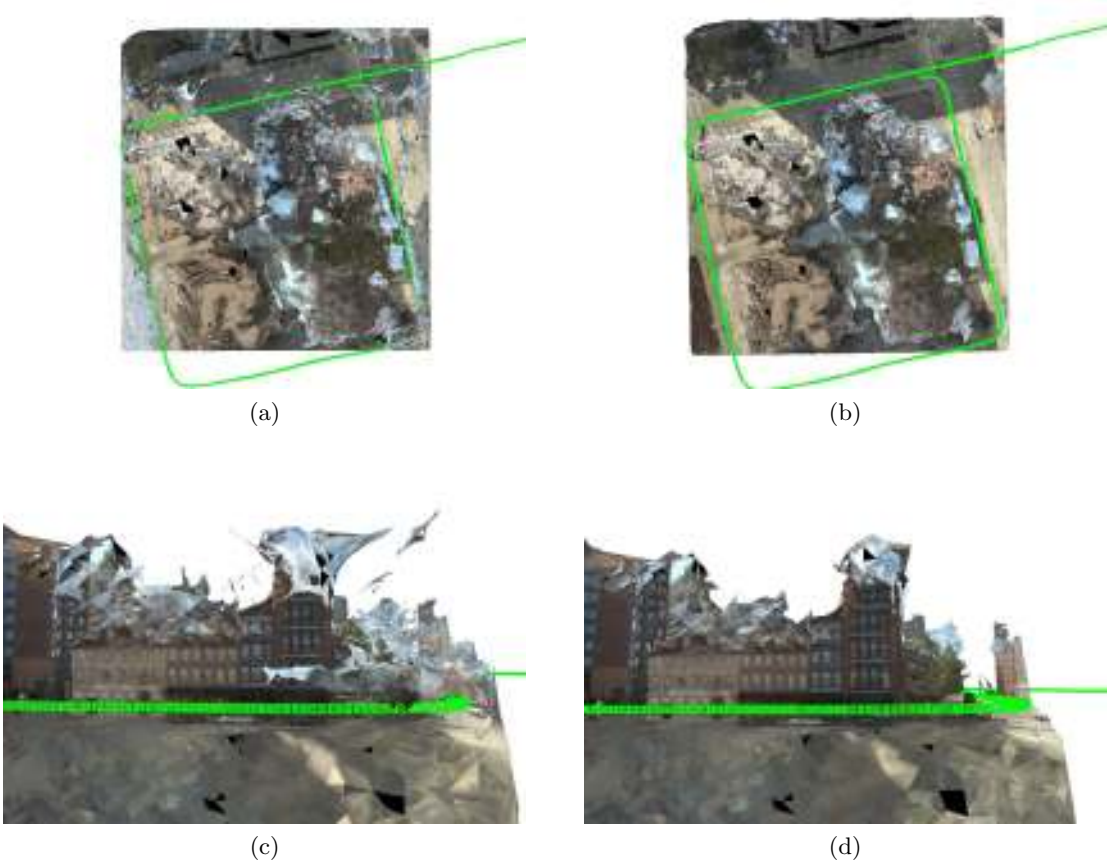
Finally, we solve the minimal s-t cut problem for the local DG g and obtain a final 3D reconstruction of the box. This process is fast enough and we never process all input points at a time but only a small subset.

11.2. Results

Figure 11.1 (a),(c) shows the reconstruction of the points which are inside a box of a part using the method described in the Section 7.3.

Figures 11.1 (b),(d) show the reconstruction of the part using the proposed method. One can see that the original approach produces a lot of artifacts and hallucinations (especially near the part boundaries). On the other hand the proposed method produces much cleaner result without artifacts.

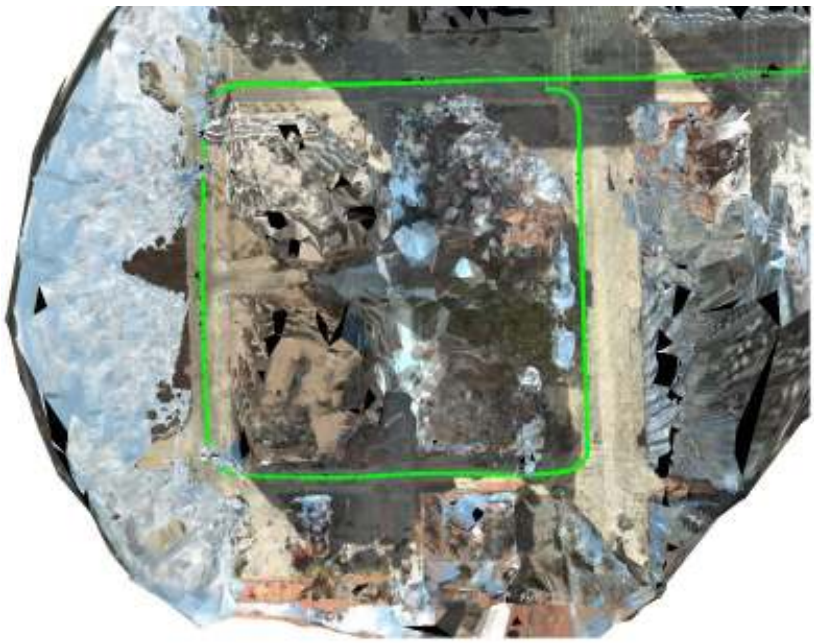
Figure 11.2 shows the final reconstruction using the proposed method consisting of joined reconstructions of all parts (boxes). The parts are connected smoothly without artifacts.



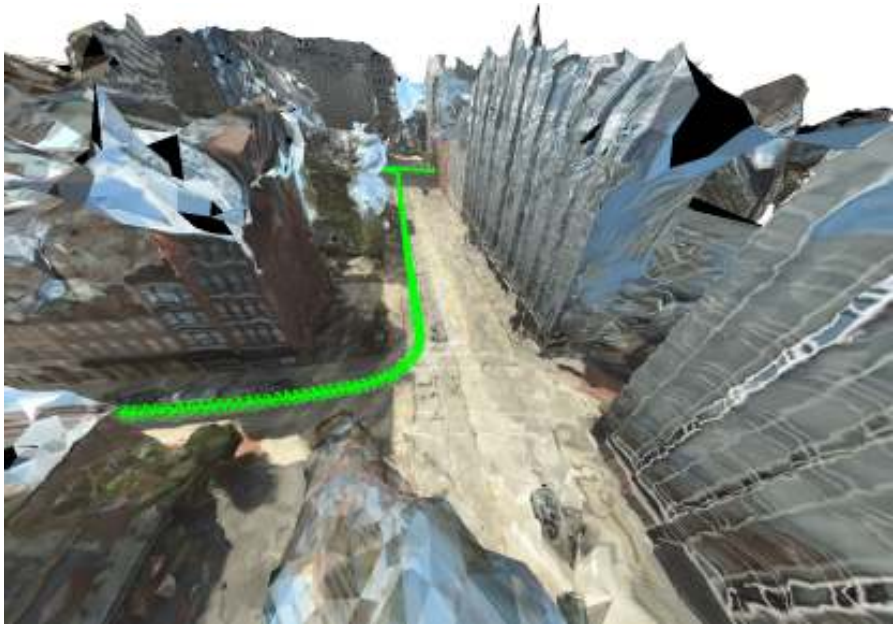
(a) (b)

(c) (d)

Figure 11.1.: **Reconstruction of a part.** (a),(c) 3D reconstruction of one part (box) using original method produces visible artifacts especially on the part borders. (b), (d) 3D reconstruction of the part using the proposed semi-global method produces no artifacts.



(a)



(b)

Figure 11.2.: **Reconstruction of all parts.** 3D reconstruction of all parts using proposed semi-global method produces results without artifacts and all parts are joining smoothly.

12 Results

We have joined all methods proposed in Chapter 9, Section 10.2, and Chapter 11 into one pipeline as result of our research. We have created the CMPMVS binary[1] that is publicly available for non-commercial purposes.

In this chapter, we provide relevant experimental results of the CMPMVS binary that were conducted during our PhD. study.

[1] http://ptak.felk.cvut.cz/sfmservice/websfm.pl?menu=cmpmvs

(a)

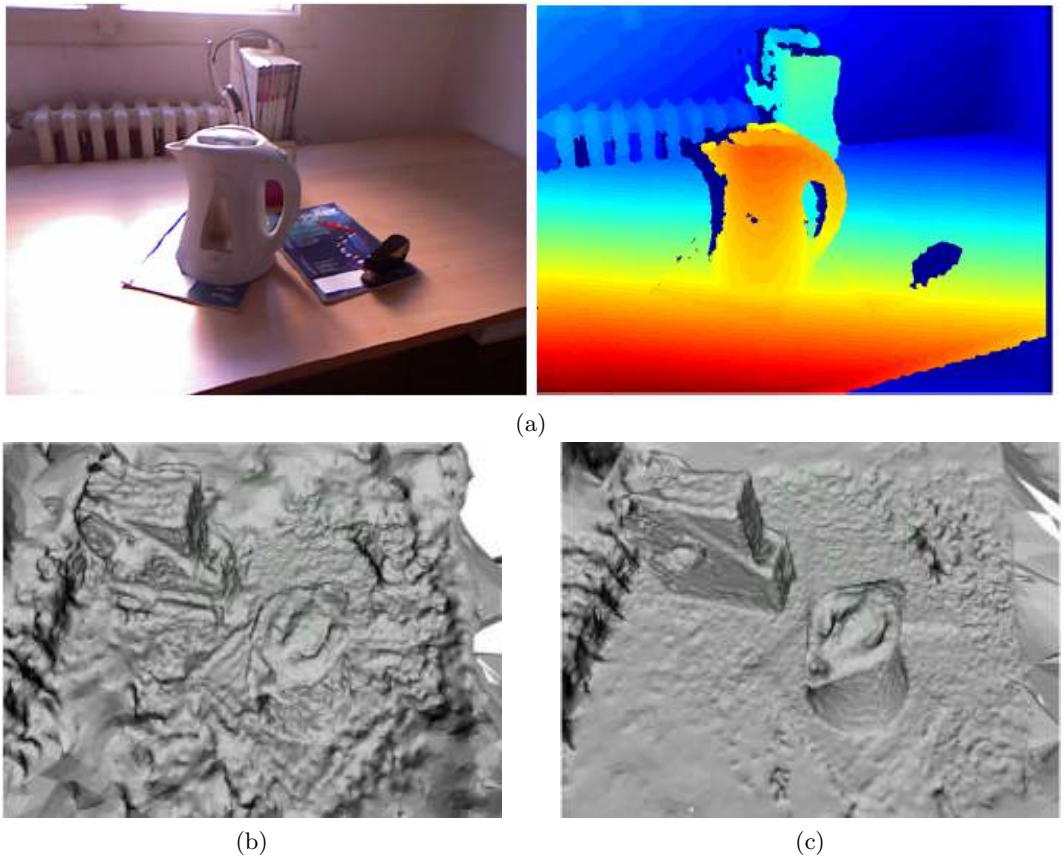
(b)

(c)

Figure 12.1.: **Large scale 3D reconstruction.** 3D reconstruction from 418 images of historical building setled in San Sebastian city.

12.1. CMPMVS on large data

Figure 12.1 shows 3D reconstruction from 418 images of historical building settled in San Sebastian city. Images provided by Pablo Vidaurre. The reconstruction consists of 19 parts and contains 22680890 vertices and 45363514 faces. Figure 12.1 (c) shows the reconstruction where each part is colored by different random color.



(a)

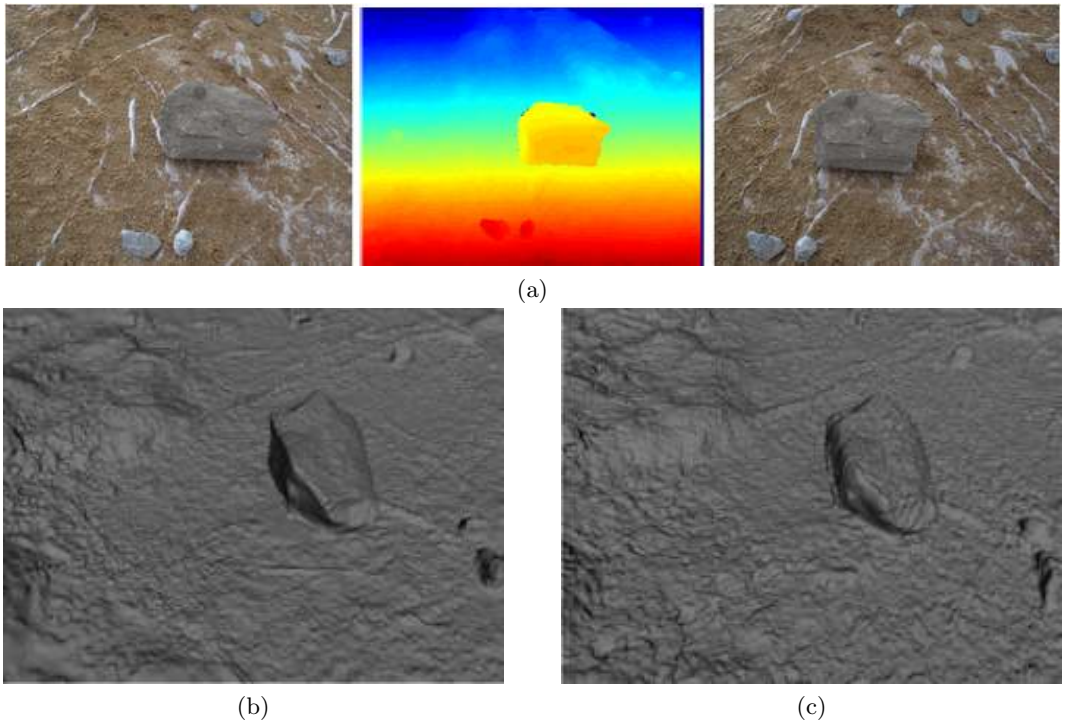
(b)

(c)

Figure 12.2.: **Kinect: images vs depth.** (a) one of 50 input RGB-Depth image pairs (b) Reconstruction from registered Kinect 2D data. (c) Reconstruction from registered Kinect 3D data.

12.2. CMPMVS and Kincet

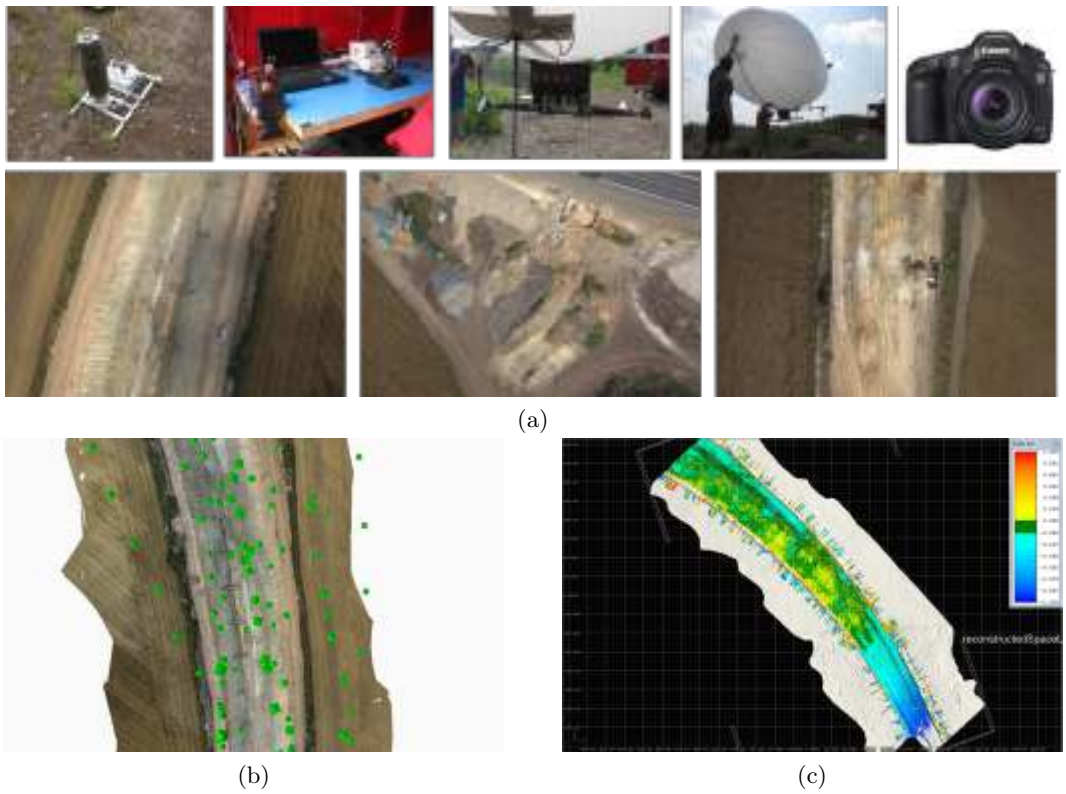
Figure 12.2 (a) shows one of 50 input RGB-Depth image pairs that has been acquired and processed. Figure 12.2 (b) shows the 3D reconstruction from CMPMVS using just images produced by Kinect. Figure 12.2 (c) shows 3D reconstruction from CMPMVS using just depth-maps produced by Kinect. The images were registered by a SFM pipeline and relative position of Kinect depth sensor vs Kinect visual sensor was precalibrated. The calibration of the Kinect depth sensor was computed from calibration of images with respect to the known fixed relative transformation. See [70] for more details.



(a) (b) (c)

Figure 12.3.: **MVS vs Kinect.** (a) one of 70 input fixed RGB-Depth-RGB triplets. The RGB comes from an SLR camera. The depth comes from Kinect. (b) reconstruction from SLR images. (c) reconstruction from registered kinect 3D data.

Figure 12.3 shows the comparison of 3D reconstruction from high quality images captured by a SLR camera versus 3D reconstruction from Kinect depth-maps. Both computed by the CMPMVS. See [70] for more details.



(a)

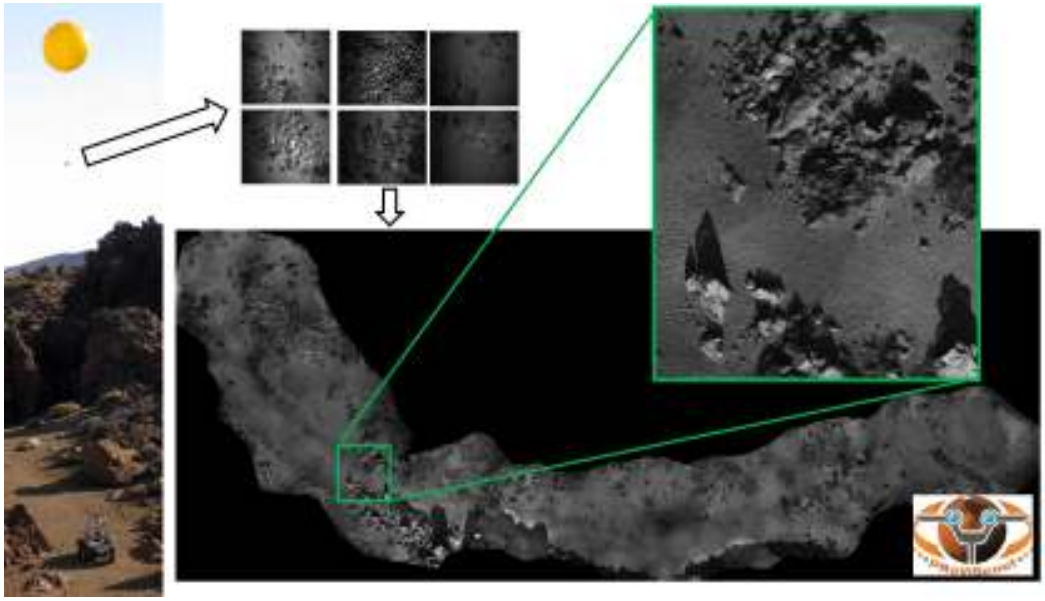
(b)

(c)

Figure 12.4.: **MVS vs Laser-Scan.** (a) Top row: antena, RC-airship and SLR camera used for capturing. Property of airshipclub.com. Bottom row: a sample of input images captured by the airship. (b) 3D reconstruction from images. Green pyramids represent camera positions. (c) comparison of the 3D reconstruction from images with a laser-scan.

12.3. CMPMVS vs laser-scan

Figure 12.4 shows the comparison of the 3D reconstruction from CMPMVS of images captured by an radio controlled airship versus laser-scan made from ground. The reconstruction was made from 1451 images of 21Mpx resolution. The experiment was made for ATOM [3] project.



(a)

Figure 12.5.: **Mars terrain reconstruction for autonomous navigation on Mars.** Left: the rover with the tethered balloon carrying a camera. Middle top: a sample of captured images. Right bottom: the final true ortho-photo map generated by CMPMVS. Right top: a closeup of the true ortho-photo.

12.4. CMPMVS for Mars surveying

Figure 12.5 shows true ortho-photo map generated by CMPMVS from images captured by a baloon thethered to ProvisG Mars rover. The balloon captured 1808 images of resolution 2448×2050. The final reconstruction contained 2621534 vertices and 5242825 faces. The final ortho-photo was of 10000×24627 resolution and approximately 2 centimeters per pixel. The experiment was made for ProVisG [65] project.

13 Conclusions

In this thesis, different aspects of the problem of 3D surface reconstruction from images or other depth sources were studied.

In **Chapter 4**, we have formulated the seed normal generation for fixed seed position as an optimization problem with a criterial function f_s based on the similarity of reprojection of images on a 3D ring. We have demonstrated by experiments that the criterial function f_s is often unimodal in certain area around its global maximum and all values in this area are greater than the values outside of this area. We have designed an effective method for finding the global optimum, which provided the correct solution in 94% of our test data.

We have formulated the seed normal and position detection as an optimization problem with a criterial function f_d based on the similarity of reprojection of images on a 3D ring. We have described a method starting with initial estimates of seed positions, improving them, and computing normals with high probability of success. We have demonstrated its high effectivity 75% on very bad initial estimates of seed position, that means approximately 2 pixels of reprojection error on the image.

We have shown that there exist nonhomogenous textures which are not discriminative. Our method is able to detect such situations when it is not possible to detect seed normals e.g. the texture is not discriminative.

In **Chapter 5**, we have presented a segmentation based multi-view stereo method. Our technique can process large scale images in affordable time. We have demonstrated that the technique achieves acceptable quality of reconstruction. We have demonstrated that our method can deal with homogeneous planar parts of scene surfaces better than patch based approaches. The advantage of our method is that it uses minimal amount of image data needed to build a dense 3D shape. Its shape is scene dependent and can cover large homogeneous image parts. Therefore, we do not need to verify as many hypotheses as growing based approaches and we can deal better with homogeneous image areas.

In **Chapter 6**, we have presented a scalable multi-view stereo. Our technique can process an unlimited number of images since it loads only a small subset of data in memory. We have demonstrated that the technique achieves acceptable quality of reconstruction in affordable time and computational effort and that it can process large sets of real images. In particular, we have demonstrated that our approach is scalable with growing image size, the number of images, and increasing the redundancy in image data.

In **Chapters 7, 8, and 9**, we have presented a new surface reconstruction method that can reconstruct surfaces strongly supported by input points with the same accu-

racy as the state-of-the-art methods and moreover it can reconstruct weakly-supported surfaces (e.g., low-textured walls, windows, cars, ground planes). We have assumed calibrated sensors and input points augmented with visibility information of sensors on the input. We have introduced an observation that it is also possible to reconstruct a surface that does not contain input points and we demonstrate it by an example. We assume an infinite number of ideal noiseless, outliers-less input points in the example. We have accumulated free-space-support from visibility information of input points and we show that nonzero free-space-support is the evidence for free space and the zero free-space-support is the evidence for full space or hallucination in the example. Therefore, the nonzero to zero change is the evidence of a surface (interface). Based on this observation, we have defined and study free-space-support of tetrahedra in tetrahedralization of (real-world) input points that can be noisy, contain huge amount of outliers, and the number of input points is finite. We have designed an interface classifier based on the experiments. We have experimentally showed that the number of false positives (wrongly classified non-interface point as interface) is negligible and that the number of true positives (especially on weakly-supported surfaces) is considerable. Finally, we have proposed a new method that extends an existing state-of-the-art (baseline) method by using an interface classifier, so that the existing state-of-the-art method gains the ability to reconstruct weakly-supported surfaces. The newly proposed method strongly follows existing (former) method that introduces the problem of weakly-supported surfaces and is able to reconstruct them. However, we have discussed and experimentally showed that the newly proposed method produces more accurate results and reconstructs weakly-supported surfaces better.

In Section 10.1, **Chapter 10** we have proposed a hallucination-free multi-view stereo method. We have demonstrated that the quality of our reconstructions is comparable to the best state-of-the art methods on several benchmark datasets. We have shown experimentally that our method produces more complete reconstructions while removing falsely generated surfaces.

In Section 10.2, **Chapter 10** we have proposed new fast post-processing method that can detect and remove particular types of hallucinations that mainly occur in outdoor scenes that are not observed from above (i.e., sky hallucinations) when using state-of-the-art methods.

In **Chapter 11**, we have proposed an effective method for solving multi-view stereo problem. We have proposed a method which can produce exactly the same reconstruction of a box as if all related points outside the box were used but without actually storing them in the memory. We have experimentally showed that our method works on real data.

In **Chapter 12**, we have showed different results relevant to this thesis that were conducted during our PhD. study.

Bibliography

[1] Image modeller. `http://imagemodeler.realviz.com`.

[2] Google street view pittsburgh experimental data set. In *Google Cityblock Research Dataset V.1.7.*, 2008.

[3] ATOM. Automatické trojdimensionální monitorování terénu. tačr ta02011275, 2012.

[4] C. Baillard and A. Zisserman. Automatic reconstruction of piecewise planar models from multiple views. In *CVPR*, pages 559–565, 1999.

[5] C. Bibby and I. Reid. Fast feature detection with a graphics processing unit implementation. In *Proc. International Workshop on Mobile Vision*, 2006.

[6] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 69–78, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[7] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[8] Y. Boykov and V. Lempitsky. From photohulls to photoflux optimization. In *BMVC*, 2006.

[9] N. Campbell, G. Vogiatzis, C. Hernandez, and R. Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *ECCV*, 2008.

[10] J. Cech and R. Sara. Efficient sampling of disparity space for fast and accurate matching. In *BenCOS*, pages 1–8, 2007.

[11] CGAL. Cgal, Computational Geometry Algorithms Library, 2013.

[12] J. Clark. Hierarchical geometric models for visible surface algorithms. In *Comm. of the ACM*, pages 19(10):547–554, 1976.

[13] R. Collins. A space-sweep approach to true multi-image matching. In *CVPR*, 1996.

[14] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[16] A. Delong and Y. Boykov. A scalable graph-cut algorithm for n-d grids. In *CVPR*, 2008.

[17] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

[18] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[19] J.-S. Franco and E. Boyer. Fusion of multi-view silhouette cues using a space occupancy grid. In *ICCV*, 2005.

[20] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski. Towards internet-scale multi-view stereo. In *CVPR*, 2010.

[21] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. In *CVPR*, 2007.

[22] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[23] D. Gallup, J. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *CVPR*, 2007.

[24] M. Goesele, B. Curless, and S. Seitz. Multi-view stereo revisited. In *CVPR*, pages II: 2402–2409, 2006.

[25] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. Seitz. Multi-view stereo for community photo collections. In *ICCV*, 2007.

[26] L. Guan, J.-S. Franco, and M. Pollefeys. 3d occlusion inference from silhouette cues. In *CVPR*, 2007.

[27] M. Habbecke and L. Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *CVPR*, 2007.

[28] C. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, 1998.

[29] R. Hartley and A. Zisserman. Multiple view geometry in computer vision. In *Cambridge University Press*, 2003.

[30] M. Havlena, A. Ess, W. Moreau, A. Torii, M. Jancosek, T. Pajdla, and L. Gool. Awear 2.0 system: Omni-directional audio-visual data acquisition and processing. In *1st Workshop on Egocentric Vision, CVPR*, 2009.

[31] M. Havlena, M. Jancosek, J. Heller, and T. Pajdla. 3d surface models from opportunity mer navcam. In *EPSC 2010: European Planetary Science Congress Abstracts*, volume 5, page 2, Goettingen, Germany, September 2010. Europlanet Research Infrastructure, Copernicus Gesellschaft mbH. CD-ROM.

[32] M. Havlena, M. Jancosek, B. Huber, F. Labrosse, L. Tyler, T. Pajdla, G. Paar, and D. Barnes. Digital elevation modeling from aerobot camera images. In *EPSC-DPS 2011: European Planetary Science Congress Abstracts*, volume 6, page 2, Goettingen, Germany, October 2011. Europlanet Research Infrastructure, Copernicus Gesellschaft mbH. CD-ROM.

[33] M. Havlena, A. Torii, M. Jancosek, and T. Pajdla. Automatic reconstruction of mars artifacts. In *EPSC 2009: The European Planetary Science Congress*, volume 4, abstract EPSC2009-280, page 2, September 2009.

[34] E. Hernandez, G. Vogiatzis, and R. Cipolla. Probabilistic visibility for multi-view stereo. In *CVPR*, 2007.

[35] L. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. In *Genome Research*, pages 9:1106–1115, 1999.

[36] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[37] V. Hlavac, R. Sara, J. Matas, T. Pajdla, J. Kostliva, V. Franc, P. Doubek, M. Havlena, M. Jancosek, A. Torii, and R. Tylecek. Stereoscopic imaging project summary report. Research Report CTU–CMP–2009–15, Center for Machine Perception, K13133 FEE zech Technical University, Prague, Czech Republic, October 2009.

[38] M. Jancosek and T. Pajdla. Exploiting visibility information in surface reconstruction to preserve weakly supported surfaces. In *International Scholarly Research Notices, vol. 2014, Article ID 798595, 20 pages*, 2014. doi:10.1155/2014/798595.

[39] M. Jancosek and T. Pajdla. Effective seed generation for 3d reconstruction. In *13th Computer Vision Winter Workshop (CVWW)*, pages 83–90, 2008.

[40] M. Jancosek and T. Pajdla. Segmentation based multi-view stereo. In *14th Computer Vision Winter Workshop (CVWW)*, 2009.

[41] M. Jancosek and T. Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR*, 2011.

[42] M. Jancosek and T. Pajdla. Multi-view stereo for urban enviroment, 2011. In *Czech Technical University in Prague, local WORKSHOP*, 2011.

[43] M. Jancosek and T. Pajdla. Removing hallucinations from 3D reconstructions. Research Report CTU–CMP–2011–12, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, September 2011.

[44] M. Jancosek and T. Pajdla. Hallucination-free multi-view stereo. In K. Kutulakos, editor, *Trends and Topics in Computer Vision*, volume 6554 of *Lecture Notes in Computer Science*, pages 184–196. Springer Berlin Heidelberg, 2012.

[45] M. Jancosek, A. Shekhovtsov, and T. Pajdla. Scalable multi-view stereo. In *IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1526–1533, Sept 2009.

[46] L. Jianguo, L. Eric, C. Yurong, X. Lin, and Z. Yimin. Bundled depth-map merging for multi-view stereo. In *CVPR*, 2010.

[47] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *SGP*, 2006.

[48] H. Kim, M. Sarim, T. Takai, J.-Y. Guillemaut, and A. Hilton. Dynamic 3d scene reconstruction in outdoor environments. In *3DPVT*, 2010.

[49] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. In R. G. Cowell and Z. Ghahramani, editors, *AI and Statistics*, pages 182–189. Society for Artificial Intelligence and Statistics, 2005.

[50] K. Kutulakos and S. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000.

[51] P. Labatut, J. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *ICCV*, 2007.

[52] P. Labatut, J. Pons, and R. Keriven. Robust and efficient surface reconstruction from range data. In *Computer Graphics Forum*, 2009.

[53] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, February 1994.

[54] M. Lhuillier and L. Quan. A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):418–433, March 2005.

[55] X. Li, C. Wu, C. Zach, S. Lazebnik, and J. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV*, 2008.

[56] S. Liu and D.-B. Cooper. Ray markov random fields for image-based 3d modeling: Model and efficient inference. In *CVPR*, 2010.

[57] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[58] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pages 674–679, 1981.

[59] D. Martinec and T. Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *CVPR*, 2007.

[60] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J. Frahm, R. Yang, D. Nister, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *ICCV*, pages 1–8, 2007.

[61] B. Micusik and J. Kosecka. Multi-view superpixel stereo in urban environments. *International Journal of Computer Vision*, 2009.

[62] H. P. Moravec. Towards automatic visual obstacle avoidance. In *Proc. 5th Int. Joint Conf. Artifficial Intell.*, page 584, 1977.

[63] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011.

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

[65] ProVisG. Planetary robotics vision ground processing. fp7-space-218814, 2012.

[66] N. Salman and M. Yvinec. Surface reconstruction from multi-view stereo. In *Modeling-3D*, 2009.

[67] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[68] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, 2006.

[69] N. S. Sinha, M. P., and M. Pollefeys. Multi-view stereo via graph cuts on the dual of an adaptive tetrahedral mesh. In *ICCV*, 2007.

[70] J. Smisek, M. Jancosek, and T. Pajdla. 3d with kinect. In A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, editors, *Consumer Depth Cameras for Computer Vision*, Advances in Computer Vision and Pattern Recognition, pages 3–25. Springer London, 2013.

[71] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH*, 2006.

[72] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, November 2008.

[73] C. Strecha. evaluation page. `http://cvlab.epfl.ch/~strecha/multiview/denseMVS.html`.

[74] C. Strecha, R. Fransens, and L. Van Gool. Wide-baseline stereo from multiple views: a probabilistic account. In *CVPR*, 2004.

[75] C. Strecha, R. Fransens, and L. Van Gool. Combined depth and outlier estimation in multi-view stereo. In *CVPR*, 2006.

[76] C. Strecha, R. Tuytelaars, and L. Van Gool. Dense matching of multiple wide-baseline views. In *ICCV*, 2003.

[77] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*, 2008.

[78] H. Tao and H. Sawhney. Global matching criterion and color segmentation based stereo. In *WACV*, 2000.

[79] E. Tola, V. Lepetit, and P. Fua. Daisy: An efficient dense descriptor applied to wide baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, 2010.

[80] A. Torii, M. Havlena, M. Jancosek, Z. Kukelova, and T. Pajdla. Dynamic 3D scene analysis from omni-directional video data. Research Report CTU–CMP–2008–25, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, December 2008.

[81] A. Torii, M. Havlena, and T. Pajdla. From google street view to 3d city models. In *OMNIVIS '09: 9th IEEE Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras*, 2009.

[82] A. Torii, M. Havlena, T. Pajdla, and B. Leibe. Measuring camera translation by the dominant apical angle. In *CVPR*, 2008.

[83] R. Tylecek and R. Sara. Depth map fusion with camera position refinement. In *CVWW*, 2009.

[84] G. Vogiatzis, C. Hernindez Esteban, P. Torr, and R. Cipolla. Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2241–2246, December 2007.

[85] H. Vu, R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. In *CVPR*, 2009.

[86] M. Wainwright, T. Jaakkola, and A. Willsky. MAP estimation via agreement on (hyper)trees: Message-passing and linear-programming approaches. In *IEEE Trans. on Information Theory*, pages 51(11):3697–3717, 2005.

[87] T. Werner and A. Zisserman. New techniques for automated architectural reconstruction from photographs. In *ECCV*, pages 541–555, 2002.

[88] C. Wu. SiftGPU. `http://cs.unc.edu/~ccwu/siftgpu`.

[89] C. Wu. Towards linear-time incremental structure from motion. In *3DV*, 2013.

[90] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore bundle adjustment. In *CVPR*, 2011.

[91] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *CVPR*, pages I: 211–217, 2003.

[92] X. Zabulis and K. Daniilidis. Multi-camera reconstruction based on surface normal estimation and best viewpoint selection. In *3DPVT*, pages 733–740, 2004.

[93] X. Zabulis and G. Kordelas. Efficient, precise, and accurate utilization of the uniqueness constraint in multi-view stereo. In *3DPVT*, pages 137–144, 2006.

[94] C. Zach, T. Pock, and H. Bischof. A globally optimal algorithm for robust tv-l1 range image integration. In *ICCV*, 2007.

[95] A. Zaharescu, E. Boyer, and R. Horaud. Transformesh: A topology-adaptive mesh-based approach to surface evolution. In *ACCV*, 2007.

[96] G. Zeng, S. Paris, L. Quan, and F. Sillion. Accurate and scalable surface representation and reconstruction from images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):141–158, January 2007.