Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

Realtime 3D Reconstruction

Gottfried Graber

January 2012

Thesis supervisor

Dipl.-Ing. Dr. Thomas Pock

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

EIDESSTATTLICHE ERKLÄRUNG

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

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

Englische Fassung:

STATUTORY DECLARATION

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

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

Abstract

Reconstruction of 3D geometry from 2D images is one of the most fundamental challenges in computer vision. In the past decade, numerous algorithms have been developed to solve this problem in an offline fashion. Only recently, the availability of cheap processing power in the form of GPUs and appropriate parallel algorithms made it possible to tackle this problem in a novel way and present results to the user in realtime. The aim of this thesis is to create a system that is capable of interactively reconstructing dense geometry from a single moving camera. Building on a high quality realtime tracking system, depthmaps of the scene are computed by means of a dense multiview stereo algorithm. A volumetric representation of geometry is used, where the surface is given implicitly by the zero level-set of an underlying truncated signed distance function. Reconstruction is based on robust depthmap fusion using a total variation formulation. The resulting convex energy functional is solved globally optimal using a fast primal-dual algorithm.

The application developed in this thesis is able to reconstruct arbitrary geometry on-the-fly with minimal user interaction thanks to a fully automated pipeline. The dense geometry can serve as starting point for sophisticated AR applications, where pixel-accurate interaction between computer generated content and the real world is possible.

Keywords. 3D reconstruction, dense reconstruction, realtime, interactive, total variation, convex optimization, gpu, stereo, tracking, volumetric representation, augmented reality

Kurzfassung

Die Rekonstruktion von 3D Geometrie aus 2D Bildern ist eines der fundamentalen Probleme der digitalen Bildverarbeitung. Im Laufe des letzten Jahrzehnts wurde eine Vielzahl von Algorithmen entwickelt, welche diese Aufgabenstellung offline lösen. Erst die Verfügbarkeit von billiger Rechenleistung in Form von leistungsstarken Grafikkarten sowie entsprechende parallele Algorithmen ermöglichen eine neuartige Herangehensweise, bei der die Resultate dem Benutzer in Echtzeit zur Verfügung gestellt werden. Das Ziel dieser Arbeit ist die Entwicklung eines Systems, welches die interaktive Rekonstruktion von dichter Geometrie mithilfe einer einzigen Kamera ermöglicht. Ausgehend von einem hochwertigen Framework für Echzeit Kamera-Tracking werden Tiefenkarten der Szene mithilfe von Stereo Algorithmen berechnet. Zur Darstellung der 3D Geometrie wird eine voxelbasierte Methode verwendet, wobei die Oberfläche der Objekte implizit durch das zero level-set einer vorzeichenbehafteten Distanzfunktion gegeben ist. Der Algorithmus für die Rekonstruktion basiert auf einer robusten Fusionierung der einzelen Tiefenkarten mithilfe einer Methode aus der Variationsrechnung. Die dabei verwendete konvexe Kostenfunktion wird global optimal mit einem neuartigen Primal-Dualen numerischen Verfahren gelöst.

Das Ergebnis der Arbeit ist eine Anwendung, welche in Echtzeit beliebige Geometrie rekonstruieren kann. Durch den hohen Grad an Automatisierung ist die Bedienung des Programms für den Benutzer einfach und intuitiv. Die Geometrie die man erhält kann als Ausgangspunkt für hochwertige Augmented-Reality Anwendung verwendet werden, welche die pixelgenaue Interaktion von computergenerierten Inhalten mit der echten Welt erfordern.

Schlagwörter 3D Rekonstruktion, dichte Geometrie, Echtzeit, interaktiv, Total Variation, Konvexe Optimierung, GPU, Stereo, Tracking, Volumetrische Darstellung, Augmented Reality

Acknowledgments

I would like to use the opportunity to mention the people who played a part in the realization of this thesis. First of all, I would like to thank my family and especially my father, who gave me the opportunity and provided me with the basic conditions to study to my liking. I am deeply grateful to my supervisor Thomas Pock, who guided me from my bachelor's thesis onwards. His ideas were a driving force behind this thesis, and I would like to thank him for his mentorship, support, inspiriational discussions and for always having an open ear for my questions. Without him this work would not have been possible. Many fellow students I got to know during my studies contributed to this thesis. I would like to mention Rene Ranftl, who never got tired of explaining a particular algorithm to me, and Markus Murschitz, with whom I had very fruitful and inspiring discussions. Manuel Werlberger and Markus Unger willingly shared their experience in the implementation of modern optimization algorithms on parallel hardware with me. The codebase for the raytracing algorithm used in this thesis was given to me by Markus Steinberger, who also helped me a lot with various computer graphics related questions.

Finally I would like to thank Anna for inspiration, motivation and for showing me how to see with my heart.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

The aim of 3D reconstruction is to infer the 3D structure of a scene given a set of 2D projections (images). It is one of the basic problems of computer vision, since the knowledge of 3D structure is a very strong clue about numerous other problems, including segmentation, tracking or object detection. Like many problems of early vision, 3D reconstruction is a so-called inverse problem, which can be seen by looking at the process of image formation: The 3D world is projected onto the 2D image plane. The information loss during the projection from 3D to 2D makes the problem ill-posed, meaning existence & uniqueness of a solution cannot be guaranteed.

The term *3D reconstruction* actually refers to a pipeline of processes consisting of the following components:

1. Intrinsic calibration

2. Image acquisition

3. Pose estimation

4. Reconstruction

Intrinsic calibration can be done as a pre-computation step. Image acquisition is easily done at realtime by taking a standard 30fps camera, and realtime camera pose estimation (i.e. tracking) has seen great progress in recent years, there is very few work on realtime reconstruction. Mostly this is due to the enormous computational complexity, typical high quality reconstruction pipelines take an order of hours or days of computational time.

Combining *realtime* and *3D reconstruction* opens up a variety of fascinating applications, probably the most important of which is augmented reality (AR). The knowledge of 3D structure enables computer generated content to accurately interact with the real world. The most obvious example for such an interaction is the occlusion of computer generated content behind objects of the real world. To get high quality pixel-accurate occlusions, a key requirement is that the reconstruction is *dense*, as opposed to a reconstruction consisting of a sparse point cloud. Since the latter is already achieved in realtime by state-of-the-art self-localizing systems (e.g. SLAM in mobile robotics), the step to a full dense scene representation follows naturally. Other application areas include the 3D modeling community, where a vision driven reconstruction approach can replace expensive and often complicated laser-based reconstruction systems.

1.2 Contribution

The goal of this thesis is to create a system that is able to reconstruct dense geometry from a single camera in realtime. Realtime in this context means that the time between the camera seeing an object and the corresponding reconstruction should be in the range of seconds, so that the delay in visual feedback does not distract the user. This is accomplished by taking state of the art results from different areas of computer vision (camera calibration, realtime SLAM, dense stereo matching, range image fusion) and adapting and combining them in a novel way to yield the desired system. An important enabler is the availability of highly parallel hardware in the form of powerful graphics cards, offering TeraFLOPS of computational power to the end user at an previously unseen FLOPS per money ratio. Together with appropriate parallel algorithms, this enables systems where 3D models are computed on the fly merely by filming the object of interest.

1.3 Thesis Overview

This thesis is structured as follows: Chapter 1 gives an overview of related work and methods involved with 3D reconstruction. Chapter 2 presents the methodology of the realtime 3D reconstruction system by describing the individual parts of the reconstruction pipeline, while chapter 3 gives implementation details. Chapter 4 is devoted to an evaluation of the system, where various key parameters and their impact on the reconstruction result are investigated. Finally, a conclusion is given in chapter 5.

1.4 Related Work

Outline: *This section gives an overview of related work on 3D reconstruction. The basic building blocks of the reconstruction pipeline are discussed, with special emphasis on realtime performance. Methods for both internal and external camera calibration are reviewed, where the latter leads to the problem of pose estimation and, more generally, SLAM. Next, algorithms for computing dense depth maps from multiple images as well as methods for fusing multiple depth maps into a single 3D model are presented. Finally, a detailed description of* Parallel Tracking and Mapping *(PTAM), a state of the art realtime SLAM system which forms the basis of the application developed in this thesis, is given.*

1.4.1 The Reconstruction Pipeline

As stated in the previous chapter, the task of 3D reconstruction can be split into the three distinct sub-problems of image acquisition, camera calibration and reconstruction (see figure 1.1).

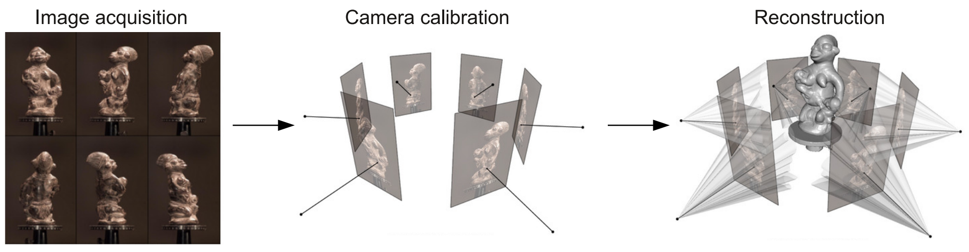


Figure 1.1: Basic building blocks of the 3D reconstruction pipeline. Here, camera calibration includes internal (projection parameters) as well as external (camera pose) parameters. Images taken from `http://www.eng.cam.ac.uk/news/stories/2005/digital_pygmalion/`

Camera calibration involves finding the internal as well as the external camera parameters. Reconstruction involves inferring 3D structure from calibrated images. While numerous algorithms exist for solving these problems individually, very specific requirements have to be met for realtime operation.

In realtime operation, image acquisition is done through a live camera. Most consumer cameras nowadays are able to operate at rates of at least $25Hz$, which satisifies the realtime requirement. To make subsequent tasks easier, a camera with a wide angle lens is beneficial. Furthermore, camera properties like shutter speed / exposure time, gain, focus, frame rate etc. should be accessible for the user to adapt for different conditions.

An overview of the mathematical concepts involved with camera calibration is given in section 1.4.2. Because the camera and therefore the internal calibration parameters typically do not change at runtime, it is reasonable to decouple internal and external calibration. Section 1.4.2.1 is devoted to internal calibration methods, external calibration is addressed in section 1.4.2.2. The task of realtime pose estimation in an unknown environment leads to the problem of simultaneous localization and mapping (SLAM), which has been a very active research area in the robot vision community. Corresponding results are presented in section 1.4.2.3.

Section 1.4.3 deals with the reconstruction process. Aiming at a dense reconstruction, methods for extracting dense geometry from images are reviewed in 1.4.3.1. To generate a single 3D model, a fusion step is necessary. Due to the realtime requirement, an online-type algorithm is needed, meaning it should be possible to add new data without the need to restart the algorithm. Appropriate methods are discussed in section 1.4.3.2.

1.4.2 Camera Calibration

The intrinsic (internal) camera parameters model the projection of world points onto the image plane [Hartley and Zisserman, 2003]

$$x = KX \tag{1.1}$$

Here, $x = (x, y, w)^T \in \mathbb{P}^2$ are homogeneous image pixel coordinates and $X = (X, Y, Z)^T \in \mathbb{R}^3$ is a point in 3D space. The *camera calibration matrix* $K \in \mathbb{R}^{3\times 3}$ contains the focal length $f = (f_x, f_y)^T$ and the principal point $p = (p_x, p_y)^T$

$$K = \begin{bmatrix} f_x & \gamma & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

The skew factor γ generally is zero. It models the skewing of pixel elements if the x- any y-axis of the sensor element array are not perpendicular. The linear relationship (1.1) is called *pinhole camera model*. Typically, a camera additionally shows distortion from the lens system, which can me modeled as a non-linear function of the projected image coordinates [Zhang, 2000, Weng et al., 1992]. Coefficients of the distortion function are counted among the internal parameters.

The extrinsic (external, exterior) camera parameters describe the pose of the camera in 3D world. They are given as 3×4 matrix $C = [R \mid t]$, where R is a 3×3 rotation matrix and t is a translation vector [Hartley and Zisserman, 2003]. Multiplying by C transforms a homogeneous 3D point from the world coordinate frame into the camera coordinate frame.

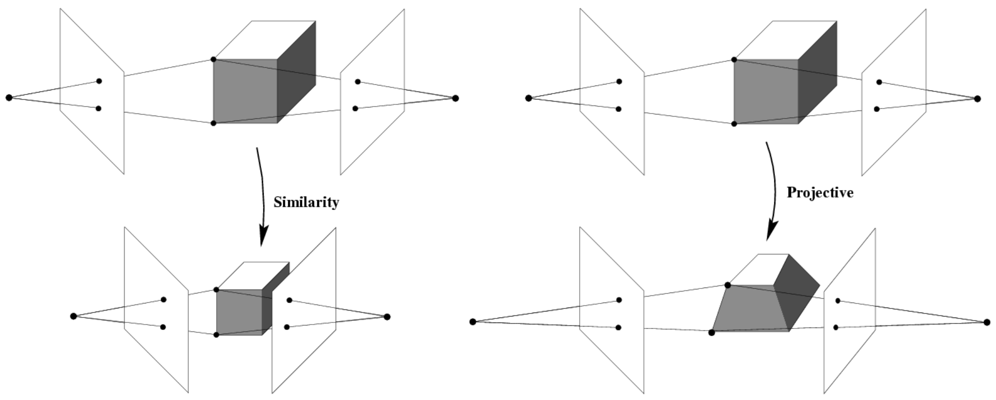
$$X_{\text{cam}} = C \cdot X_{\text{world}}$$

By combining the internal and external parameters into the *projection matrix* $P = KC = K[R \mid t]$, the complete relation between the 3D world point X and the 2D image point x (both in homogeneous coordinates) is given by

$$x = P \cdot X$$

1.4.2.1 Internal Calibration

The process of finding internal and/or external parameters is called camera calibration. Before proceeding further, a definition of terms seems appropriate, as there is some uncertainty to the meaning of “camera calibration” and “calibrated camera”. When speaking about camera calibration, it is often understood as restricted to internal calibration, although it is perfectly valid for the term to refer to external calibration. This is due to the fact that a large number of calibration methods work by estimating the projection matrix P and subsequently extracting internal parameters from the decomposition $P = K[R \mid t]$. These methods therefore calibrate internal and external parameters at the same time. On the other hand, [Hartley and Zisserman, 2003] define a calibrated camera as “a camera for which K is known”, leaving open the task of external calibration. This makes sense,



(a) Known intrinsics (b) Unknown intrinsics

Figure 1.2: Known intrinsic camera parameters allow to measure angles between rays, which enables reconstruction up to scale (a). If the intrinsic parameters are unknown, reconstruction is possible up to a projective transform (b). Figure taken from [Hartley and Zisserman, 2003].

because once the internal parameters are known, subsequent algorithms are free to assume the pinhole camera model, which, due to its linearity, is convenient. Moreover, the somewhat cumbersome term "external calibration" has long been replaced by more intuitive terms like pose estimation or camera tracking, especially in realtime operation. We will therefore adhere to the definition of Hartley and Zisserman and restrict the meaning of camera calibration to the estimation of internal parameters.

A calibrated camera allows to infer 3D information from 2D images and is thus a basic requirement for reconstruction. An important point is that the knowledge of the internal camera parameters enables the measurement of angles between rays of image points [Hartley and Zisserman, 2003, p. 209]. As a direct consequence, reconstruction from cameras with known intrinsic parameters is possible up to a similarity transform, since the reconstruction must respect the angle between rays. On the other hand, if the intrinsic parameters are not known, reconstruction is possible up to a projective transform, where the angle between rays is allowed to change (see figure 1.2).

In [Tsai, 1987] a calibration method based on a stepwise solution is presented. First, R and t are computed, then the focal length and afterwards the distortion coefficients. This method assumes the principal point to lie in the center of the image, i.e. the coordinates of the principal point are not among the parameters estimated. The method of [Zhang, 2000] is still widely used today due to its robustness and simplicity. It is based on modeling the homography between a planar calibration target and the image plane. The problem is first solved analytically in closed form, followed by a non-linear iterative optimization. The self-calibration method of [Pollefeys et al., 1999] does not require any calibration target. They showed that if the skew is zero (which is true for almost any real camera), image correspondences alone are sufficient for reconstruction up to scale.

In all of the above methods, lens distortion is included in the calibration process. The distortion model is a critical point, since a pure linear camera model is often insufficient for accurate 3D reconstruction. Especially wide-angle lenses exhibit significant distortion, the correction of which is mandatory for successful reconstruction. Distortion can roughly be divided into radial and tangential distortion. An overview of different distortion models and appropriate calibration methods is given in [Weng et al., 1992], where also exotic defects like thin prism distortion are considered. It should be noted that with increasing complexity of the distortion model, the number of distortion coefficients grows. This makes parameter estimation both harder and numerically less stable, and often it is desirable to restrict the distortion model to be simple. Indeed it has been reported in [Tsai, 1987, Zhang, 2000] that distortion is dominated mainly by the radial component. Based on this, the method of [Devernay and Faugeras, 2001] calibrates only the distortion of the camera. They motivate their algorithm by a fundamental property of the pinhole camera model: straight lines in the world are projected as straight lines in the image. By removing lens distortion such that this property is fulfilled, the camera can be treated as pinhole. Their distortion model is particularly suited for fish-eye-type wide angle lenses. It is called FOV-model since it depends only on a single parameter, the field of view.

1.4.2.2 Pose Estimation with Known Intrinsic Parameters

Estimating the external parameters of the camera is known as *pose estimation*. It consists of determining the rotation R and translation t of a calibrated (in the sense of [Hartley and Zisserman, 2003]) camera, given a set of correspondences $x_i \leftrightarrow X_i$ between image points x_i and world points X_i. Although there exist a great number of algorithms to estimate pose from uncalibrated cameras [Hartley, 1995, Pollefeys et al., 1999, Koch et al., 2000, Lhuillier and Quan, 2005], we will focus on the case where the internal parameters are known. This simplifies the task, because instead of estimating the full projection matrix P, which has 11 degrees of freedom (DOF), only the camera pose matrix $C = [R \mid t]$ with 6 DOF needs to be computed. For this, a minimum number of 3 point correspondences are needed. Therefore, the term *perspective 3-point problem* (P3P), or, more general, *perspective n-point problem* (PnP) was coined in [Fischler and Bolles, 1981].

They showed geometrically that P3P gives up to four solutions, whereas P4P can be solved uniquely in case the four points are coplanar. If the four points are nonplanar, they propose to reduce the problem to distinct P3P and find the consensus between the solutions. They further generalized this method of consensus-finding for the overdetermined case $n \geq 5$, and the resulting *random sample and consensus* (RANSAC) algorithm has since become one of the most important methods for robust parameter fitting.

Although the pose estimation problem itself is very old, research is still active. In recent times, the analytic methods for computing solutions of P4P and P5P [Horaud et al., 1989, Quan and Lan, 1999] have been replaced by non-iterative algorithms to robustly estimate pose using a large number of point correspondences [Moreno-Noguer et al., 2007, Schweighofer and Pinz, 2008].

All of the above methods for estimating pose assume that correspondences between world points X_i and image points x_i are given a priori. The question that now arises naturally is how to establish these correspondences, and ultimately how to compute 3D world points solely from 2D images.

Given the images of two cameras looking at a 3D point X, there exist constraints on the corresponding 2D projections x_1 in the first image and x_2 in the second image. These constraints are captured by the *epipolar geometry*, which models the geometric relation between world points and corresponding image points for two views [Hartley and Zisserman, 2003]. The *fundamental matrix* $F \in \mathbb{R}^{3\times3}$ is the mathematical representation of the epipolar geometry. It is independent from scene structure, hence

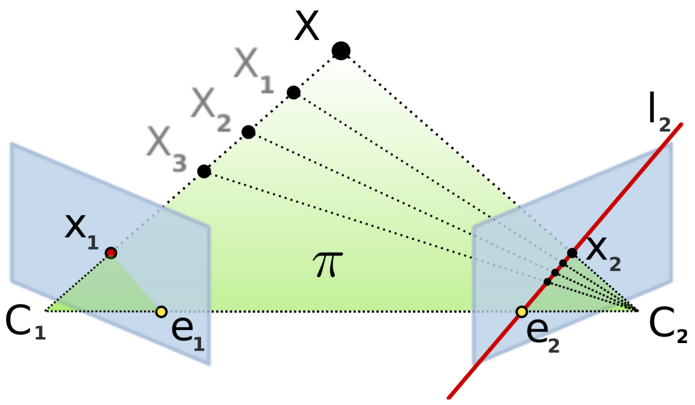


Figure 1.3: Epipolar geometry: The camera centers C_1, C_2 and the world point X define the epipolar plane π. The epipolar line l_2 defined by an image point x_1 is given by $l_2 = Fx_1$. The epipole e_1 is the projection of the camera center C_2 into the first image. The image points x_1, x_2 fulfill the epipolar constraint $x_2^T F x_1 = 0$. Image adapted from `http://en.wikipedia.org/wiki/Epipolar_geometry`

it can be computed from image points alone.

The 3D world point X is seen by two cameras, giving rise to the image points x_1 in the first image and x_2 in the second image respectively (see figure 1.3). The camera centers C_1 of the first camera and C_2 of the second camera together with the point X define the epipolar plane π. Given only the 2D coordinates of the image point x_1, it is impossible to deduce the 3D position of X, since its depth is not known. However, it must lie on the viewing ray defined by C_1 and x_1. Corresponding points in the second image lie on the so-called *epipolar line*, which is defined by the intersection of the epipolar plane and the image plane. A point x_1 in the first image therefore defines a line in the second image. Note that this relation is indeed independent from scene structure, i.e. it does not matter where along the viewing ray the 3D point actually is. Mathematically, the relation between point x_1 and epipolar line l_2 is given through the fundamental matrix:

$$l_2 = F x_1$$

In projective geometry, a point x lies on a line l if $x^T l = 0$. This directly leads to the epipolar constraint between corresponding image points:

$$x_2^T l_2 = x_2^T F x_1 = 0 \tag{1.2}$$

The projection of every 3D point along the viewing ray defined by C_1 and x_1 must lie on the epipolar line l_2 in the second image. Vice versa, if the locations of x_1 and x_2 are known, the 3D point X can be found by computing the intersection of the viewing rays. This process is known as *triangulation*. However, in real systems the epipolar constraint (1.2) generally is not fulfilled due to measurement noise, imperfect camera lenses etc. Hence, viewing rays in general do not intersect, and the position of the 3D world point has to be computed using some robust estimation method. In [P. A. Beardsley and Murray, 1994], the authors suggest to compute the midpoint of the common perpendicular between the two rays. A more sophisticated method is given in [Hartley and Sturm, 1997], where the true reprojection error is minimized by solving a sixth-order polynomial. An overview of triangulation algorithms is given in [Hartley and Zisserman, 2003].

1.4.2.3 Simultaneous Localization and Mapping (SLAM)

We have reviewed means to compute camera pose if 3D scene points are known, and to compute 3D scene points if the camera pose is known. In an unknown environment, neither 3D points nor camera pose are given a priori, which creates the following dilemma: In order to compute the camera pose (localization), one needs information about the 3D scene. Vice versa, in order to construct the 3D scene (mapping), one needs to know the camera pose. The problem of simultaneously estimating the location and building a map of the environment is known as SLAM in the robot vision community, where it is a key requirement for autonomous robot movement in an unknown environment. It is solved by a joint estimation of both the robot pose and observed landmark (map) positions in a single probabilistic formulation [Csorba, 1997, Durrant-Whyte and Bailey, 2006]. The problem is usually represented as state-space model with additive gaussian noise, where the state consists of both the robot and the landmark position. The goal is to estimate the joint posterior probability

$$P(x_k, m \mid Z_{0\ldots k}, U_{0\ldots k}, x_0)$$

where x_k is the robot location and orientation (pose) at time k, m is the set of all landmarks (the map), $Z_{0\ldots k}$ are the landmark observations, $U_{0\ldots k}$ are the robot control inputs and x_0 is the initial state. The joint posterior probability is computed from the *action model* $P(x_k \mid x_{k-1}, u_k)$ and *sensor model* $P(z_k \mid x_k, m)$. The action model describes the robot pose in terms of a state transition probability distribution on the robot state x_k. By assuming a markov process, the state at time k depends only on the immediate preceding state at $k-1$ and the control input u_k. The sensor model describes the probability of making the observation z_k, given the robot pose x_k and the current map m. In [Smith et al., 1990] these quantities are computed using an extended Kalman Filter. Action and sensor model are represented as $x_k = f(x_{k-1}, u_k) + \epsilon_k$ and $z_k = h(x_k, m) + \delta_k$, where ϵ_k and δ_k are zero-mean gaussian noise processes with covariance Q_k (robot motion noise) and R_k (landmark noise) respectively. By linearizing $f(\cdot)$ and $h(\cdot)$ using a first order Taylor expansion, the standard Kalman Filter can be applied. The probabilistic approach results in the characteristic uncertainty ellipses for the robot pose and landmarks, which reduce as the robot explores the scene. Due to the way the Kalman Filter works, all landmarks and covariance matrices have to be recalculated each time an observation is made. Although some shortcuts have been proposed to make the calculations efficient for maps of up to thousands of landmarks [Guivant and Nebot, 2001, Leonard et al., 1999], the approach remains computationally expensive.

FastSlam [Montemerlo et al., 2002] models the probability functions as a set of discrete points (particle filtering). It can represent arbitrary distributions and non-linear state transition functions directly, however it still is computationally costly.

The principles of SLAM, although originally developed for mobile robots, can be applied in pure vision systems as well. The sensor in this case is a camera, and landmarks are either provided by hand-made markers or directly extracted from the images (i.e. natural features). Due to the fact that only visual inputs are used, such systems are often called Visual SLAM (VSLAM). The restriction of not using other sensors than the camera makes the problem challenging. Whereas on a robot, motion cues are available from odometry data and often active range sensors provide accurate depth measurements, a VSLAM system has to deduce all data from images alone. Challenges include the high input rate from a live camera, motion blur from rapid camera movement, the lack of direct depth measurement and the ambiguities of reliable marker detection.

Fast VSLAM was introduced in [Davison, 2003], where the trajectory of a single camera was tracked in realtime using SLAM algorithms. Reliable camera pose tracking is a key requirement for AR applications, which call for the immediate availability of accurate camera positions. The method of [Davison, 2003] was subsequently improved for large environments and correct handling of loop closures in [Davison et al., 2007]. However, the latter work still uses the probabilistic approach found in the original SLAM algorithms. For accurate camera tracking, it would be desirable to incorporate a geometric constraint (i.e. the reprojection error) into the algorithm. Bundle adjustment is a well known algorithm to accomplish this. It works by minimizing the reprojection error of 3D points into a number of camera views, i.e. refining the coordinates of all 3D points and all camera pose parameters simultaneously. Due to the large computational effort, bundle adjustment is commonly found in offline structure-from-motion (SfM) pipelines. In a seminal work, [Klein and Murray, 2007] presented a system for realtime parallel camera tracking and mapping (PTAM), which uses bundle adjustment for globally optimizing both camera positions and 3D world points w.r.t. the reprojection error. Due to its unmatched accuracy, speed and robustness, PTAM has since been used in a variety of projects calling for high quality realtime camera tracking. It also forms the basis of the system developed in this thesis, therefore a more detailed description of PTAM is given in section 1.4.4.

1.4.3 Reconstruction

Reconstruction aims at computing a complete 3D model from a number of images of the scene taken from different viewpoints. Being one of the core problems of computer vision, it has always been a very active research area producing a great variety of algorithms.

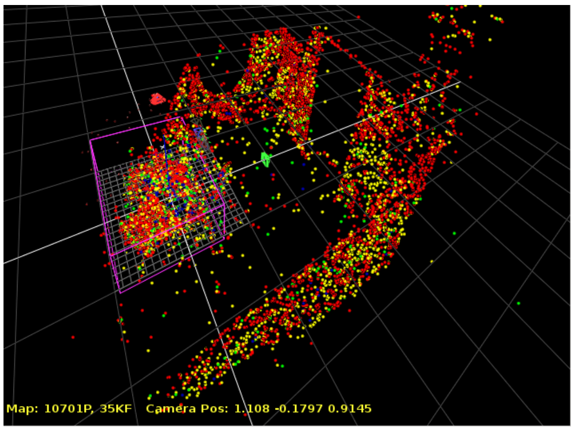
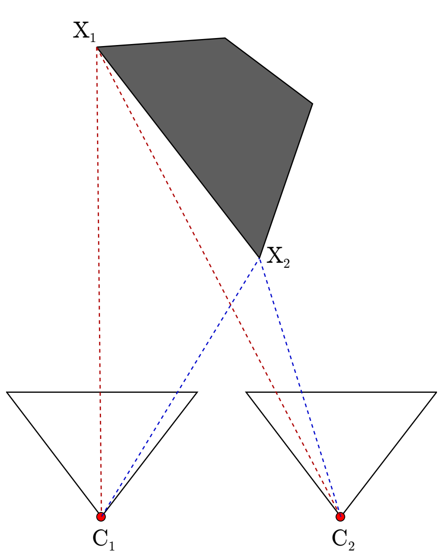


Figure 1.4: A screenshot of the map built by PTAM running in a small room. Recognizable are two perpendicular walls, one of them containing a door to another room. Even though the map contains more than 10.000 points, the reconstruction is still sparse.

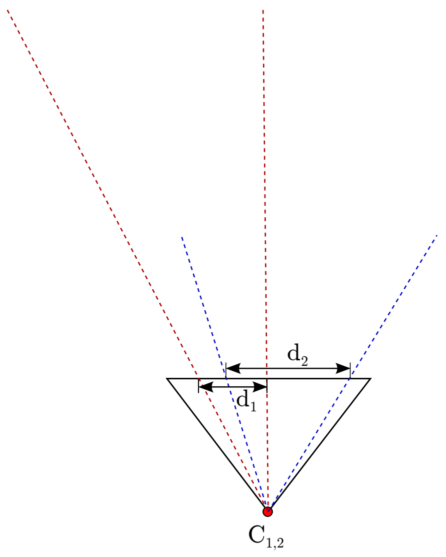
Some basic principles of reconstruction have already been presented in section 1.4.2.2: The process of triangulation computes positions of 3D points based on the epipolar geometry between two views. Indeed, this technique is used in VSLAM systems for building the map. Due to limited processing power, the map maintained by VSLAM systems typically consists of sparse point features (see figure 1.4). This does not fit our needs since we aim at a full dense reconstruction. Simply making the map dense seems not to be feasible, although [Klein and Murray, 2008] experimented with edge features (edgelets) in addition to point features, resulting in improved 3D structure of the map. Therefore, different means for extracting 3D geometry from images are needed.

1.4.3.1 Range Image Generation

Range images (depthmaps) contain for every pixel a depth value, which corresponds to the position of a world point in 3D space w.r.t. the camera frame. Depthmaps can be efficiently computed from stereo images, and a multitude of algorithms exist, see [Scharstein and Szeliski, 2002] for an extensive overview. Most stereo algorithms do not compute depth directly, but rather a dense *disparity field*. The term disparity represents the (horizontal) displacement of image pixels due to the 3D scene structure. Depth is recovered from disparity by the relation $z = \frac{bf}{d}$, where z denotes depth, d is the disparity, b the baseline between the camera centers and f the focal length. Computing disparity amounts to solving the correspondence problem: For a given pixel in the first image, what is the corresponding pixel in the second image? Most stereo algorithms as-



(a) Two cameras viewing an object.

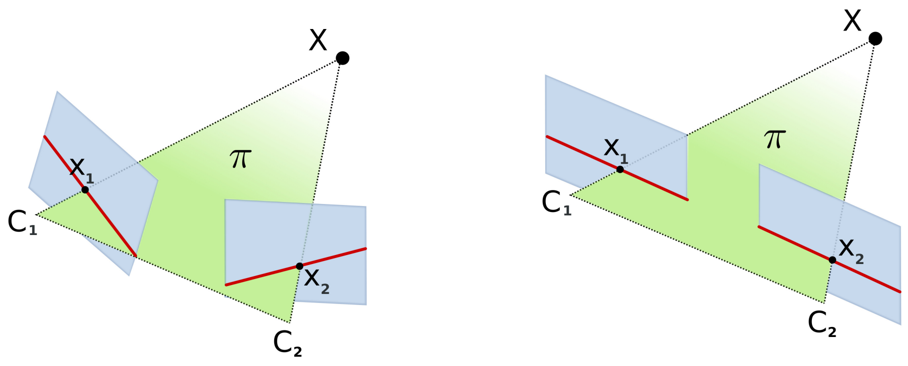


(b) The two cameras and viewing rays overlaid. Projections from the 3D point X_1 (red viewing rays) have less displacement than the projections from X_2 (blue rays).

Figure 1.5: Stereo matching: Disparity is inverse proportional to the depth of points in the scene.

sume rectified images, which simplifies the search for correspondences. The rectification procedure [Fusiello et al., 2000] guarantees that all epipolar lines are horizontal (see figure 1.6). Hence, instead of searching along arbitrary directions, one can search row-wise for correspondences. Common to most stereo algorithms is the construction of a *disparity space image* (DSI). It consists of the aggregated cost according to some similarity measure over the disparity space, which is typically sampled at pixel or sub-pixel accuracy. The goal of stereo matching is to find for every pixel the minimum cost in the DSI.

According to [Scharstein and Szeliski, 2002], stereo matching algorithms can be classified into local and global methods. Local methods use sophisticated similarity measures and aggregation schemes to compute the DSI, the best match is usually found by just taking the minimum cost along the viewing ray. Global methods on the other hand try to minimize a global energy consisting of a dataterm, i.e. a (simple) similarity measure between pixels, and a regularization term which enforces some kind of smoothness constraint on the solution. In general global methods give better results, but are often hard to compute, relying on expensive discrete optimization techniques like Graph-Cut [Boykov et al., 2001, Kolmogorov and Zabih, 2001] or Belief Propagation



(a) General stereo (b) Rectified stereo

Figure 1.6: In general stereo (a) the epipolar lines induced by the intersection of the plane π and the image plane are not horizontal. The rectification procedure computes a homography to transform both images onto a common plane (b). As a result, epipolar lines are horizontal and parallel.

[Banno and Ikeuchi, 2009]. A continuous alternative are variational methods. These are formulated in the continuous domain and can be efficiently computed on parallel hardware [Pock et al., 2008], although this approach is still not realtime capable. In recent times, local methods have made significant improvements and are now able to deliver results on par with global methods. [Zhang et al., 2008, Rhemann et al., 2011] use adaptive weights for the filter kernel to account for the fact that the center pixel and the surrounding pixels of the support window are generally on different surfaces. The algorithm presented in [Bleyer et al., 2011] additionally considers the orientation of the center pixel by estimating a 3D plane and projecting the support window onto that plane.

Often it is desirable to use more than two images for range image generation. Additional images increase the robustness and quality of the resulting depthmap, because they give additional information about the 3D structure of the scene. Disparity-based methods can not easily be generalized to multiple views due to the way disparity is defined. For multiple views, the objective has to be independent of the baseline, i.e. the camera position. This is not the case for disparity, as disparity is inherently defined by the distance between the cameras. [Okutomi and Kanade, 1993] present a multiple baseline stereo system that splits the task of multiview stereo into distinct two image stereo calculations and combines the individual result. This approach suffers from large computational complexity of $\mathcal{O}(n^2)$ w.r.t. the number of input images, since every pairwise combination of input images is run through the two-view stereo algorithm. The well known planesweep method of [Collins, 1996] arguably was the first multiview stereo algorithm with complexity $\mathcal{O}(n)$.

It works by computing homographies between image planes, where the planes move along the viewing direction of a dedicated reference view. Planesweep does not assume rectified images, which reduces the amount of preprocessing. A drawback of this method is the discrete sampling of the depth range, resulting in staircasing effects if the sampling is too coarse. The work of [Strecha et al., 2003] tackles the problem of multiview stereo by a PDE-based formulation that diffuses depth data based on the local matching confidence between all views. Their method is well suited for wide-baseline stereo and can deal with occlusions and lighting changes. However, a sparse set of intial depth values is needed to start the algorithm, this is usually taken from a camera calibration step (bundle adjustment and triangulation of 3D points). [Stuehmer et al., 2010] present a continuous variational formulation that optimizes directly for depth. Their approach is based on optical flow and can be easily generalized for multiple views. They achieve realtime performance by implementing the optimization on the GPU.

1.4.3.2 Fusion

Depthmaps allow to reconstruct dense geometry by projecting every pixel at its associated depth in space. If the camera pose matrix C and the camera calibration matrix K are known, a homogeneous pixel coordinate $x = (x, y, 1)^T$ is projected into 3D world by

$$X = C^{-1} z K^{-1} x,$$

where z is the depth of pixel coordinate x according to the depthmap. The inverse of the 3×4 camera pose matrix C is calculated by $C^{-1} = [R^{-1} \mid (-R^{-1}t)]$, where the 3×3 rotation matrix R and the translation vector t are from the decomposition $C = [R \mid t]$. Note that since R is an orthogonal matrix, its inverse can be calculated by transposition $R^{-1} = R^T$.

To obtain a full 3D model, multiple depthmaps from different viewpoints around the object of interest are required. Typically depthmaps suffer from noise and outliers, therefore a robust fusion step is required to integrate the information into a single 3D model. According to [Curless and Levoy, 1996], requirements for such an algorithm include the following properties:

- Incremental updating: As the camera explores the scene, new data becomes available. Clearly, restarting the algorithm from scratch every time new data arrives is not efficient, so incremental updates are a key requirement. Partial reconstruction results also provide an important cue on where to move the camera next to improve the 3D model.

- Ability to fill gaps: Depending on the robustness of the stereo algorithm there might be missing data in the depthmaps. Especially textureless regions are a hard challenge for similarity-based stereo algorithms, resulting in missing depth information. The fusion algorithm should be able to handle missing data, i.e. fill gaps where no depth information is available.

- Robustness: In realtime operation, errors like camera noise, camera pose uncertainty and limitations of the stereo matching algorithm will result in erroneous range data, which have to be dealt with in a robust manner to produce meaningful results.

- No restriction on topological type: The fusion algorithm should allow arbitrary genus of the 3D surface.

An important point is the representation of geometry. Basically there exist two different approaches, mesh-based and volumetric [Seitz et al., 2006]. Polygon meshes describe geometry as a set of connected planar facets. They can be stored and rendered efficiently. However, incremental updates are difficult due to the organizational overhead of removing duplicate points, updating connectivity information etc. Volumetric approaches describe geometry by means of a regularly sampled 3D grid consisting of individual elements (voxels). This technique is able to represent surfaces of arbitrary topology, which makes it a popular choice for reconstruction algorithms.

A well-studied algorithm for volumetric reconstruction is the voxel occupancy method of [Martin and Aggarwal, 1983], where the goal is to decide for each voxel if it is filled or empty. This is usually done using silhouette information from multiple images. [Laurentini, 1994] showed that algorithms based on silhouette intersection cannot fully reconstruct nonconvex objects, i.e. objects with the same silhouette but different interior. For this reason, the result of voxel occupancy methods is not the true 3D shape, but rather an approximation called the *visual hull*.

To overcome this limitation, techniques based on photo consistency were developed. *Space carving* [Kutulakos and Seitz, 2000, Slabaugh et al., 2003] and *voxel coloring* [Seitz and Dyer, 1997] exploit the photo consistency constraint between image pixels of different camera positions to decide whether a voxel is visible or not. This decision however is “hard” in the sense that it cannot be undone later on, which makes the handling of ambiguous or incorrect data difficult. Also, each voxel is processed independently, which means there is no way to enforce any kind of spatial coherence.

[Kolmogorov and Zabih, 2002] formulate the problem in an energy minimization framework. They construct an energy functional consisting of a dataterm using a photo consistency measure, a visibility term that decides if a voxel is masked by already-known geometry and a robust smoothness term which allows for discontinuities while enforcing spatial coherence on the solution. The energy functional is minimized using a graph cut algorithm.

[Curless and Levoy, 1996] use a volumetric representation and employ a signed distance function which measures for every voxel the distance to the surface. Using a continuous signed distance function instead of discrete voxel states simplifies the integration of multiple overlapping and noisy range images. The final surface is extracted by computing the zero level set of the signed distance function. In the original paper, depth values are obtained from a laser range finder. This approach was extended in [Zach et al., 2007, Zach, 2008], where the problem is cast as minimization of an energy functional and range data is calculated directly from the images using a multiview stereo algorithm. Additionally, a robust regularization is used to further improve the robustness of the method.

For further information on dense multiview reconstruction algorithms we refer to the extensive overview of [Seitz et al., 2006].

1.4.4 Parallel Tracking and Mapping

In this work, we use Parallel Tracking and Mapping (PTAM) [Klein and Murray, 2007] for tracking camera positions in realtime. PTAM marked a significant change in the design of VSLAM systems, because it did not rely on the usual probabilistic formulation of the SLAM problem. Instead, it uses high quality batch techniques for minimizing a geometrically motivated reprojection error (Bundle Adjustment). The main idea of PTAM is to split tracking and mapping into two threads. This is in contrast to probabilistic SLAM, where tracking and mapping are tightly coupled, i.e. they are performed together at each frame. Especially mapping is computationally costly and suffers from the limited amount of computation time available. If those duties are split apart, each of them can be tailored to its specific needs: The tracking thread can be made robust and fast, mapping on the other hand can be made as accurate and rich as possible, taking potentially orders of magnitude longer than the tracking thread.

Camera model and world geometry PTAM uses a right hand coordinate system for world geometry. The camera model is the FOV-model of [Devernay and Faugeras, 2001]. Camera pose is encoded as 3×4 matrix, which describes the transformation from the global world coordinate frame to the camera coordinate frame. The full relation between world points and image pixels is given by $x = \mathrm{CamProj}(\mathrm{C}_{\mathcal{CW}}\mathrm{X})$, where $x = (x, y, 1)^T$ are homogeneous image pixel coordinates, $\mathrm{CamProj}(\cdot)$ denotes the non-linear camera model including lens distortion, $C_{\mathcal{CW}}$ is the camera pose matrix ($\mathcal{CW}$ denoting the transformation direction "to $\mathcal{C}$amera from $\mathcal{W}$orld") and $X = (X, Y, Z, W)^T$ is a homogeneous 3D world point.

Initialization Before running PTAM, a map needs to be initialized. This step is user aided, after selecting a starting frame through a key press, a number of features is tracked while the camera is translated sideways. Selecting a second frame yields a stereo pair. The tracked features provide point correspondences, which enable the five-point algorithm to estimate the epipolar geometry, and 3D map points can be triangulated (see section 1.4.2.2). PTAM uses a pre-calibrated camera, which enables the reconstruction of the initial map points up to scale (see section 1.4.2.1). Targeting AR applications, metric reconstruction is not necessary and the scale of the map is set to an arbitrary fixed value by assuming a baseline of $10cm$ between the stereo images and scaling the whole map accordingly. Afterwards, the world is transformed such that the dominant plane of the map lies at $z = 0$.

Tracking The tracking threads runs at $30Hz$ and performs a two-stage tracking procedure. First, a prior camera pose is estimated from a simple motion model. Map points are projected into the image and a small number of points is searched for and used to update the camera pose. Using the improved pose estimate, a large number of features is searched for and the camera pose is further refined. Searching is done using an 8×8 image patch which is pre-warped according to the camera pose estimate coming from the motion model. FAST corners are computed in a circular region around the predicted position and zero-mean SSD is used to select the best matching position. In addition, a four-level image pyramid is constructed for every frame and feature search is done at the appropriate scale.

Given a number of reprojected world points, the camera pose is updated by minimizing a robust objective function of the reprojection error.

Mapping Starting with the map built by the stereo initialization procedure, world points are added continuously as the camera explores the scene. A key concept of PTAM are so called *keyframes*. These are special frames from the live video stream used for constructing the map. By using only a subset of all available camera frames it is possible to use high quality bundle adjustment for optimizing the map. Keyframes are selected by the tracking thread based on tracking quality, distance to the nearest keyframe and time since the last keyframe was added. This ensures that keyframes are distributed evenly in space and avoids a stationary camera corrupting the map. Keyframes generated by the tracking thread have to be processed as fast as possible by the mapping thread to ensure fast map growing. Therefore, a queue of keyframes is kept between the tracking and mapping threads and the tracker stops generating keyframes if the queue length exceeds a certain value.

Triangulating new map points requires point correspondences between images (see section 1.4.2.2). These are found by selecting the closest keyframe from the map as second view and performing feature search along the epipolar lines. If correspondences have been found, new map points are triangulated and added to the map.

The map is optimized using bundle adjustment. Bundle adjustment iteratively refines camera pose matrices μ_i and world points p_i by minimizing

$$\{\mu_2 \dots \mu_N, p_1 \dots p_M\} = \underset{\mu,p}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j \in S_i} \operatorname{Obj}\left(\frac{|e_{ji}|}{\sigma_{ji}}, \sigma_T\right)$$

e_{ji} is the reprojection error of map point p_j (which is taken from the set of image measurements S_i) into image i with camera pose μ_i (the camera pose μ_1 of the first keyframe is a fixed datum and excluded from optimization). σ_{ji} are associated standard deviations and $\operatorname{Obj}(\cdot, \sigma_T)$ is the Tukey biweight objective function with an estimate of the distribution standard deviation σ_T. For minimization, the well known Levenberg-Marquard algorithm is used. Unfortunately, this approach does not scale well with increasing number of keyframes. In the original paper, the authors reported runtimes of tens of seconds for a map with 150 keyframes on an Intel Core 2 Duo 2.66 GHz processor. Therefore, a local bundle adjustment is used to quickly integrate new keyframes and map points into the map: Upon addition of a keyframe, bundle adjustment using just the 5 nearest keyframes is performed, which converges much faster than a full global bundle adjustment using all keyframes and map points. The mapping thread uses a priority model for the following tasks:

- Add keyframes: This involves computation of FAST-corners and triangulation of new map points. Adding keyframes has top priority, and any other task is interrupted if a new keyframe arrives.
- Local bundle adjustment: If the queue of keyframes to be added is empty, local bundle adjustment is performed. This updates newly added keyframes and map points.
- Global bundle adjustment: If the mapping thread is idle (i.e. the camera does not explore), global bundle adjustment using all keyframes and all map points is performed.

1.4.5 Realtime Dense Reconstruction

We now revisist the reconstruction pipeline introduced in chapter 1.4.1 again, adding more specific requirements:

- Internal calibration
- Image acquisition
- External calibration (camera pose tracking)
- Reconstruction
  - Extract dense geometry from images
  - Integrate individual results into a single model

In this chapter we have reviewed methods and algorithms to individually solve the above tasks. However, very few work has been done to bring them together in a realtime environment.

[Stuehmer et al., 2010] use keyframes from the PTAM framework to calculate dense depthmaps in realtime. They employ a variational approach based on optical flow to robustly estimate depth using multiple views. The resulting geometry is limited to 2.5D, because a fusion of individual depthmaps is not addressed in the work.

The live dense reconstruction system of [Newcombe and Davison, 2010] also uses PTAM for tracking the camera pose in realtime. They generate a base mesh from the sparse point features of the map maintained by PTAM. This base mesh is subsequently refined using dense depth information obtained via variational optical flow. The problem of this method is that the base mesh is created once and cannot change topology later on. Thus, if a concavity in scene geometry is not captured by the initial base mesh, it cannot be recovered later in the dense refinement step. This thesis addresses these issues and aims at a full 3D reconstruction of arbitrary geometry.

Recently, Newcombe et al. presented their follow-up work of the live dense reconstruction system. They created a system called Dense Tracking and Mapping (DTAM) [Newcombe et al., 2011a], which can be seen as an improved version of PTAM. The difference is that unlike PTAM, where sparse 3D point features are used for camera tracking, DTAM tracks the camera position off a *dense* model of the world. This model is created using a simple multiview stereo algorithm to extract depth from up to hundreds of small baseline images from the live camera stream. Dense tracking results in significantly improved tracking performance, especially under difficult conditions like motion blur or camera defocus. In such cases PTAM is unable to track the camera, whereas DTAM, thanks to the overwhelming amount of data stemming from the dense approach, does not loose tracking.

Another approach which is closely related to our work is KinectFusion of [Newcombe et al., 2011b]. They use Microsoft Kinect as a cheap depth sensor to reconstruct dense geometry. As in our own work, they use a truncated signed distance function to implicitly represent geometry in a volumetric way. Because depth data from Kinect is relatively accurate, a sophisticated fusion method is not needed. Instead, a simple weighted average of the signed distance function over the voxel grid gives astonishing results. Camera tracking is done off the dense model of the world, similar to DTAM.

Chapter 2

Method

Contents

Outline: *The methodology for realtime dense reconstruction is presented. We give an overview of the system before describing the individual parts. These include camera pose tracking, selection of keyframes for dense stereo matching, multiview stereo algorithms and the integration of multiple depthmaps into the final 3D model. Finally the method for visualizing results is presented.*

2.1 System Overview

Our realtime system operates on a live camera stream. We use PTAM to track camera pose in realtime. PTAM splits tracking and mapping into two threads running on two cores of the CPU. We follow this idea and introduce a third thread which is responsible for reconstructing the scene. This multithreaded approach enables asynchronous execution of PTAM and the reconstruction task. Furthermore, PTAM is not interrupted by the computation-intensive reconstruction when the system is run on a CPU with at least 3 cores. This requirement is easily met by todays high-end desktop PC systems, where Quadcore CPUs are standard. Figure 2.1 depicts an overview of the system. The recon-

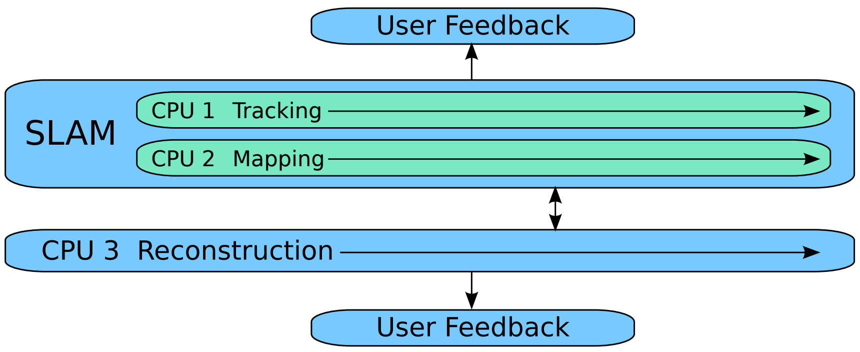


Figure 2.1: The method consists of a realtime tracking system which uses two threads and a third thread responsible for reconstructing scene geometry. On a multicore CPU, this approach minimizes interruptions and enables smooth GUI interactions even on heavy load.

struction process makes heavy use of the massive parallel computing resources provided by modern graphics cards by offloading calculations onto the graphics processing unit (GPU). For reconstruction, we consciously choose algorithms with inherent data parallelism to make optimal use of the computational power provided by the GPU. Our system requires 2 GPUs to run optimally. The reason for this is that a GPU is not designed for multitasking. Normally this is not a problem, because typical GPU calculations are finished quickly in (at most) tens of milliseconds. The calculations required for reconstruction on the other hand take an order of magnitude longer, lasting for seconds. This results in freezes of the GUI. Therefore, we require 2 GPUs, one for rendering the GUI and one for carrying out the reconstruction related calculations. Note that rendering the GUI does not require extensive computational power, so one GPU can be low-end. Running our application on a single-GPU system is possible, but results in suboptimal user experience due to short freezes and stuttering.

The system requires minimal user interaction. After initialization of the tracking system (see section 1.4.4), the volume of interest is specified. This is done through a GUI-based tool (see section 3.5), where we concentrated on intuitive handling. After setting up the volume, the reconstruction process is fully automated.

2.2 Tracking

PTAM is used for tracking the camera pose in realtime. No changes were made to the key components of the tracking system (see section 3.1). In its map, PTAM maintains a list of keyframes. Keyframe image data is stored directly from the live camera as-is. The images exhibit significant lens distortion, which must be corrected because the algorithms used for reconstruction assume a linear pinhole camera. The camera model used by PTAM is

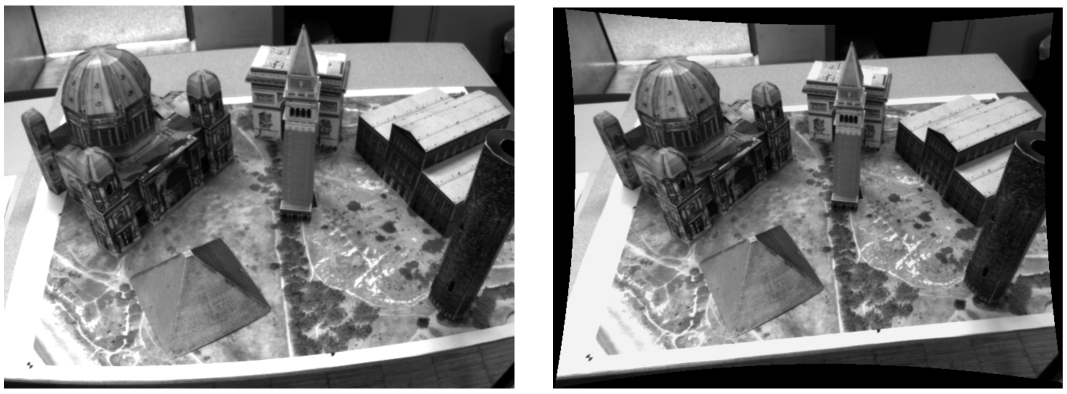

(a) Distorted image (b) Undistorted image

Figure 2.2: Camera image showing lens distortion (a) and the corrected image (b). In the corrected image, the edge of the desk is straight.

the FOV model of [Devernay and Faugeras, 2001]. It is given by

$$x_d = x_u \frac{1}{r\omega} \arctan\left(2r \tan\frac{\omega}{2}\right) \tag{2.1}$$

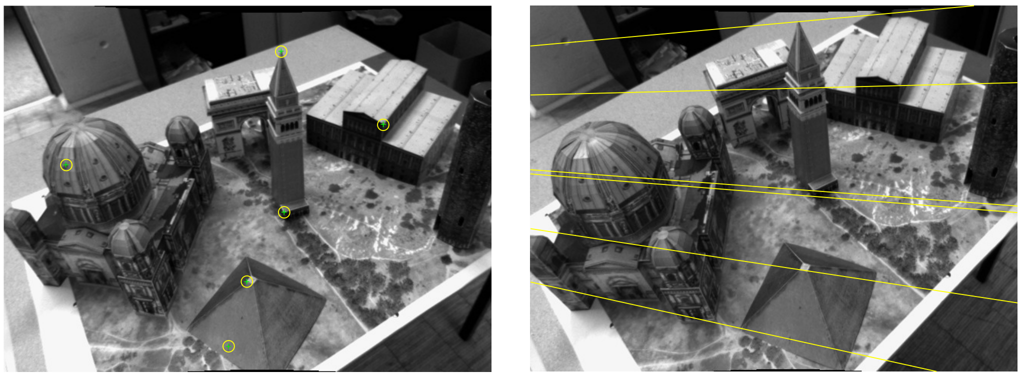
where $x_u = (x_u, y_u, 1)^T$ are normalized (i.e. undistorted, at camera $z = 1$ plane) camera coordinates, x_d are the distorted coordinates, $r = \sqrt{x_u^2 + y_u^2}$ is the radius from the image center and ω is the distortion parameter obtained from a camera calibration step. Given a distorted image, we calculate normalized camera coordinates by projecting discrete pixel raster coordinates $\hat{x}_u = (x, y, 1)^T$ onto the camera $z = 1$ plane by

$$x_u = K^{-1}\hat{x}_u$$

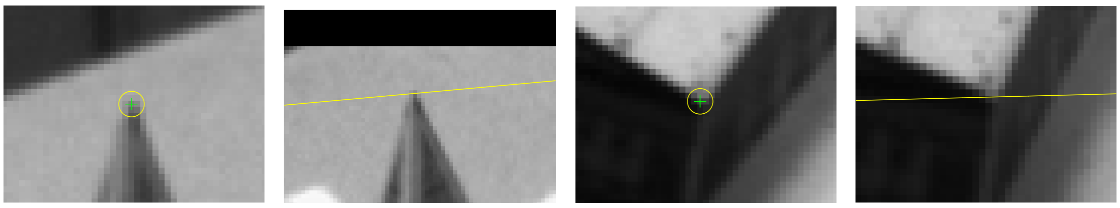
with the camera calibration matrix K. Then, distorted coordinates are computed according to (2.1) and pixel values are interpolated from the distorted image at the back-projected positions $\hat{x}_d = Kx_d$.

2.3 Depthmap Generation

Keyframes from PTAM are used to generate depthmaps of the scene. The accuracy of the camera pose is critical for calculating depthmaps, therefore we do not use pose estimates from the live camera stream but rely on keyframes. Keyframes are optimized using bundle adjustment, resulting in high quality of the associated camera pose (see figure 2.3).



(a) Image 1 (b) Image 2



(c) Detail image 1 (d) Detail image 2 (e) Detail image 1 (f) Detail image 2

Figure 2.3: Epipolar geometry from PTAM keyframe camera matrices. Shown are selected points (a) and their corresponding epipolar lines (b). As depicted in the details (c)-(f), PTAM delivers sub-pixel accuracy.

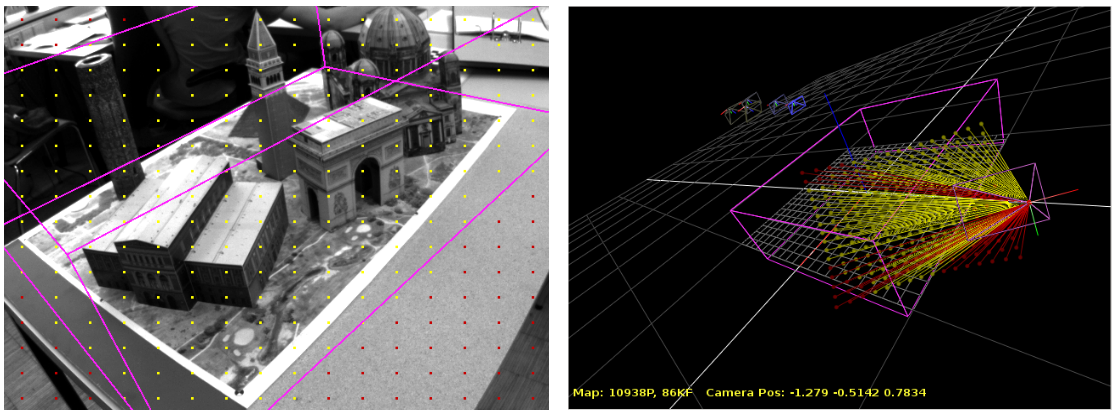
2.3.1 Keyframe Selection

Because it is not guaranteed that the camera is always looking at the volume of interest, a selection mechanism to decide whether a keyframe should be used for reconstruction is needed. Keyframe selection is based upon the percentage of image rays intersecting the volume. This guarantees that only images whose 3D points potentially lie inside the volume are used for reconstruction.

Let $\mathcal{S}$ be the set of all pixel coordinates in an image, we compute the subset $\mathcal{S}_{\text{intersect}} \subseteq \mathcal{S}$ of pixels whose viewing rays intersect the volume. The image is used for reconstruction if

$$\frac{|\mathcal{S}_{\text{intersect}}|}{|\mathcal{S}|} \geq \gamma \tag{2.2}$$

is fulfilled for a threshold $\gamma \in [0, 1]$, which is proportional to the desired percentage of viewing rays intersecting the volume. For calculating $\mathcal{S}_{\text{intersect}}$, each pixel coordinate x is projected into 3D space by $X = C^{-1}K^{-1}x$, where C is the camera pose matrix associated



(a) 2D image of intersection (b) 3D image of intersection

Figure 2.4: Result of the intersection test using a 16×16 grid (a). Viewing rays of yellow pixels intersect the volume (purple box), red pixels do not. (b) shows the same scene in 3D.

with the image. Next, the camera position C_{cam} is calculated from the pose matrix by $C_{\text{cam}} = -R^{-1}t$, where R and t are from the decomposition $C = [R \mid t]$. The two points C_{cam} and X define the viewing ray for pixel x. To determine if a viewing ray intersects the volume, we calculate the intersection of the viewing ray and the planes that define the volume. To this end, it is advantageous to use the Plücker matrix representation of a line. It is given by

$$L = AB^T - BA^T,$$

where L is a 4×4 matrix representing the line and A, B are homogeneous points on the line. Using this representation, the intersection $X_{\text{intersect}}$ of a line L and a plane $\pi = (a, b, c, d)^T$ is given by $X_{\text{intersect}} = L\pi$. By checking if X is inside[1] the volume boundaries, we decide if the viewing ray intersects the volume or not. To get a good estimate of the percentage of viewing rays intersecting the volume, it is not necessary to calculate intersections for every viewing ray. Instead, we sample image pixel coordinates based on a 16×16 grid, which results in a much smaller set $\mathcal{S}$ and consequently a drastically reduced number of computations. See figure 2.4 for an exemplary result.

If (2.2) is fulfilled, the keyframe is marked for reconstruction. It is subsequently used as *reference view* for a multi-view stereo matching algorithm, which produces a depthmap.

Once a reference view has been selected, the task remains to find good keyframes (*sensor views*) for the multi-view stereo algorithm. This selection is based on proximity

[1]For coordinate comparison, we apply an $\epsilon \approx 10^{-6}$ to account for limited numerical accuracy

to the reference view and on the angle between the optical axes (i.e. viewing direction) of the reference and the sensor view. First, the 10 nearest keyframes to the reference frame are searched for in the list of keyframes. Next, the angle between the optical axes is computed. The optical axis v is defined by the vector between the camera position C_{cam} and the projection of the principal point p into 3D space by $P = C^{-1}K^{-1}p$. The angle is then computed as $\alpha_i = \arccos\left(\langle v_{\text{ref}}, v_i\rangle\right), \quad i = 1 \dots 10$ (see figure 2.5). A keyframe is

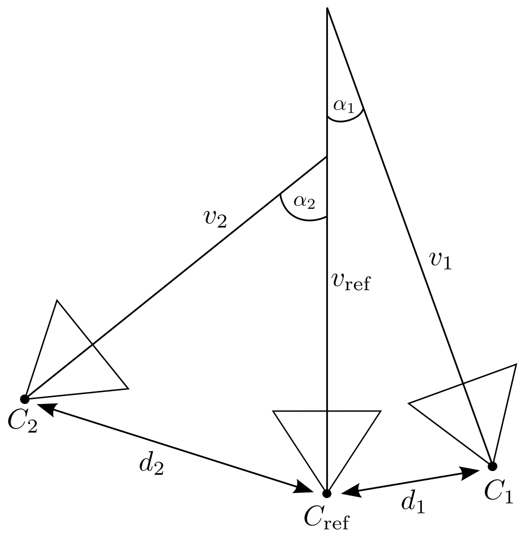


Figure 2.5: Sensor view selection is based on the euclidean distance between camera positions (d_i) and the angle between the optical axes (α_i)

selected as sensor view if α is between two user-defined thresholds, i.e. $\beta_{\text{min}} \leq \alpha \leq \beta_{\text{max}}$. By setting e.g. $\beta_{\text{min}} = 0°$ and $\beta_{\text{max}} = 15°$, one can ensure that sensor views have sufficient overlap with the reference view. The sensor keyframes are sorted according to their α and sent to the depthmap generation stage.

2.3.2 Planesweep

The images obtained from the keyframe selection step have a completely arbitrary configuration in 3D space. The stereo algorithm should therefore not assume any preconditions (e.g. rectified images etc.) and should have the following properties:

- Multiview: While the algorithm should deliver decent results when run with two images only, it should be possible to easily use multiple images.
- Robustness against large baseline: Because the configuration of keyframes depends on the user, sometimes it is not possible to find small-baseline keyframes for a given

reference view. Robustness in case of large baselines (i.e. large occlusions) is therefore required.

- Robustness against illumination changes: Arbitrary camera movement from the user results in illumination changes, e.g. if the camera looks at a lamp or outside a window.
- Fast computation: For the use in a realtime environment, the algorithm needs to be fast. Unfortunately this rules out many high quality graph-cut based algorithms.

The planesweep algorithm [Collins, 1996] suits these needs. Furthermore, it is easy to parallelize, as shown in the work of [Cornells and Van Gool, 2005]. As its name implies, planesweep works by sweeping a family of planes through 3D space. The planes are parallel to the image plane of the reference view at different depths (i.e. down the z-axis of the reference coordinate frame). At every depth, there exists a homography from each of the sensor views onto the plane (see figure 2.6). The idea of planesweep is that if the plane

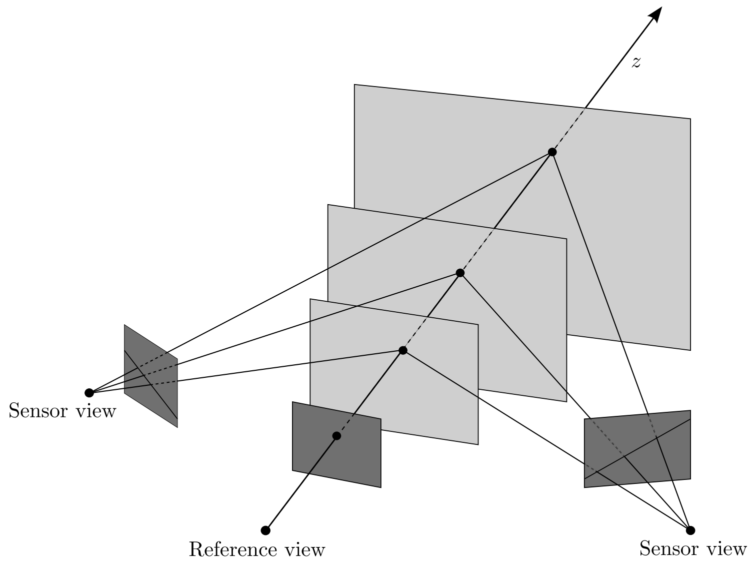


Figure 2.6: By sweeping a family of planes through 3D-space and computing pixelwise similarities of the reference view and the sensor views mapped onto the planes, a depth value for each pixel is computed. Figure adapted from [Cornells and Van Gool, 2005]

passes through the surface of the object, the pixel values of the reference view and the sensor views mapped onto the plane match (under the assumption of lambertian surfaces). The algorithm consists of computing the similarity between the reference view and the sensor view mapped onto the plane at different depths, and assigning each pixel the depth where the similarity is maximal.

The homography induced by the planes is computed the following way: Assuming a camera at the origin, i.e. $C_1 = [I \mid 0]$, a plane $\pi = (n^T, 1)$ and a second camera $C_2 = [R \mid t]$, the homography induced by the plane is computed by calculating the intersection of the viewing ray of a point $x = (x, y, z)^T$ and the plane π, and back-projecting the intersection point into the second camera. Because C_1 is at the origin, 3D points on the viewing ray of x are given by $X = (x^T, \mu)$, where μ parametrizes points on the ray. We wish to find the intersection of the ray with the plane, that is, the point X for which $\pi^T X = 0$ holds. This determines μ to $-n^T x$, and the intersection point $X = (x^T, -n^T x)^T$ is transformed by C_2 to

$$x' = [R \mid t]X = Rx - tn^T x = (R - tn^T)x \tag{2.3}$$

The homography is thus given by $H = R - tn^T$ (see [Hartley and Zisserman, 2003, p. 327]). If the two images are from the same camera, we can apply the internal calibration matrix K to get the homography from image pixel coordinates of the first view to image pixel coordinates of the second view $H = K(R - tn^T)K^{-1}$.

In the generic case we have $C_{\text{ref}} = [R_{\text{ref}} \mid t_{\text{ref}}]$ and $C_{\text{sens}} = [R_{\text{sens}} \mid t_{\text{sens}}]$ as the camera matrices of the reference camera view and a sensor view respectively. The planes $\pi(d)$ are defined in the coordinate frame of the reference view, i.e. $\pi(d) = (n^T, -d)^T$ where $n = (0, 0, 1)^T$ is the unit vector along the z-axis and d is the distance (the depth) from the reference view. This setup can be reduced to the case derived in previous paragraph by calculating the relative rotation and translation $R_{\text{rel}}, t_{\text{rel}}$ between the two cameras, which can then be plugged into (2.3). A canonical representation of the plane $\pi(d)$ is obtained by dividing by $-d$, i.e. $\pi(d) = \left(\frac{n^T}{-d}, 1\right)$. The relative rotation and translation are given by

$$R_{\text{rel}} = R_{\text{sens}} R_{\text{ref}}^{-1} \tag{2.4}$$

$$t_{\text{rel}} = t_{\text{sens}} - R_{\text{rel}} t_{\text{ref}} \tag{2.5}$$

The homography for the generic case thus computes to

$$H = K \left(R_{\text{rel}} - \frac{t_{\text{rel}} n^T}{-d} \right) K^{-1}, \tag{2.6}$$

where $n = (0, 0, 1)^T$, and d parametrizes the plane depth. The planesweep algorithm requires some additional information about scene geometry. Planes $\pi(d)$ are generated for $z_{\text{near}} \leq d \leq z_{\text{far}}$, which means that depth values are restricted to lie between z_{near} and z_{far}.

The values of z_{near} and z_{far} are calculated by projecting the corner points of the volume into the reference coordinate frame and determining the minimum/maximum depth value.

An important parameter is the depth resolution d_{step}, i.e. the sampling of the interval $[z_{\text{near}}, z_{\text{far}}]$. A coarse sampling results in poor depth accuracy, while a fine sampling increases the computational load. Because the geometric accuracy of the reconstruction is already inherently limited by the voxel size, we use the voxel resolution as basis for determining the depth resolution (see section 4.2.1).

Using normalized cross-correlation (NCC) as similarity measure makes the algorithm robust against lighting changes. The NCC is given by the following formula

$$c = \frac{1}{n} \sum_{x,y \in \mathcal{W}} \frac{(I_1(x,y) - \bar{I}_1)(I_2(x,y) - \bar{I}_2)}{\sigma_1 \sigma_2},$$

where I_1, I_2 are the two images, $\bar{I}_1, \bar{I}_2, \sigma_1, \sigma_2$ are the mean and standard deviation respectively and $\mathcal{W}$ is the window used for the calculation (e.g. 5×5). To speed up the calculation, we use precomputed sum images, which makes the run-time less dependent on the window size. Using larger windows on the one hand increases robustness, on the other hand results tend to get blurred at depth discontinuities.

The final depth value is extracted by a simple Winner-Takes-All (WTA) scheme. As the planes move along the z-axis, the best correlation value is recorded for every pixel and updated if a better match is found. The depth for each pixel is the depth of the plane at which the best match was found. This approach requires additional memory of size $M \times N$, where M, N are the width and height of the camera image. This is cheap in comparison to more sophisticated depth extraction scheme, where typically the whole cost volume needs to be stored in memory.

Figure 2.7 shows results of the planesweep using different window sizes, where one can clearly notice the noise stemming from the WTA approach. To improve depthmap quality, an optional TV-ℓ_1 image denoising was implemented. The energy functional for TV-ℓ_1 image denoising is given by

$$\min_u \left\{ \int_\Omega g \, |\nabla u| + \lambda \int_\Omega |u - f| \mathrm{d}x \right\} \tag{2.7}$$

where u is the unknown denoised image, f is the noisy input, $x = (x,y)^T \in \Omega$ are image pixel coordinates and $\Omega \subseteq \mathbb{R}^2$ is the image domain. The nabla operator is understood in the distributional sense, which enables gradient calculations for non-smooth functions. TV-

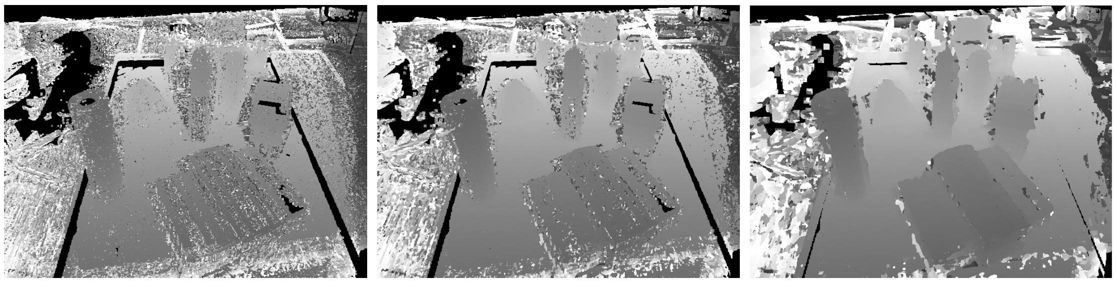

(a) 3×3 window (b) 7×7 window (c) 11×11 window

Figure 2.7: Results for 4-view planesweep using different NCC-windows.

regularization has the property of preserving sharp edges while simultaneously smoothing homgenous regions, whereas the ℓ_1 dataterm provides robustness against outliers and is particularly suited for impulse-like noise. In addition, we employ a g-weighting of the regularization term, where g is computed from the input image gradient magnitude (i.e. edges) as $g = \exp(-\alpha|\nabla I|)$. This means that the denoising process respects edges in the input image, which further improves results (see figure 2.8). To solve (2.7), the first order primal-dual algorithm of [Chambolle and Pock, 2011] is used.

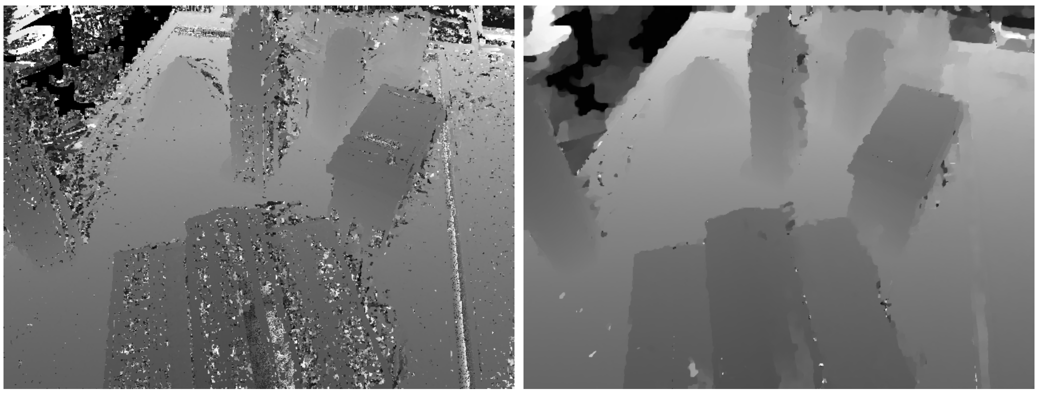

(a) Raw depthmap, 5×5 NCC window (b) TV-ℓ_1 denoised depthmap

Figure 2.8: Depthmap before and after TV-ℓ_1 denoising. Parameters were set to $\lambda = 0.3$, $\alpha = 20$, 100 iterations.

2.3.3 Total Generalized Variation Multiview Stereo

As the accuracy of the depthmaps is critical for reconstruction quality, we implemented an alternative method to generate depthmaps from multiple views. It is based on variational optical flow, where the pixel displacement between two views is parametrized by the depth of the 3D point on the corresponding viewing ray (see figure 2.9). The advantage of this method is that multiple views can be incorporated easily. Whereas disparity is defined between two images and it is cumbersome to transform a given disparity into a pixel correspondence for a third view, a 3D point parametrized by its depth is easily projected into as many views as one likes, assuming the pose of the camera views is known (i.e. the relative rotation and translation can be computed).

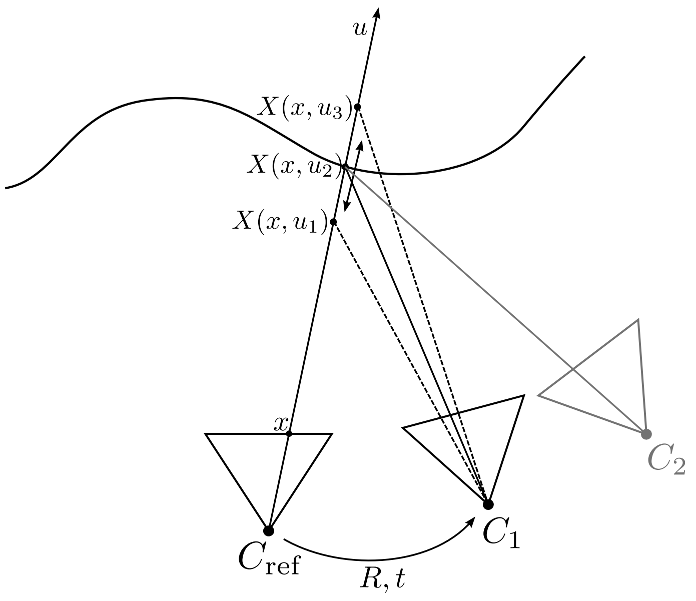


Figure 2.9: The pixel correspondence problem is formulated by parametrizing 3D points on the viewing ray by their depth and back-projecting them into the second image.

This was introduced first by [Stuehmer et al., 2010] and we have extended their method to use a more sophisticated regularizer, namely Total Generalized Variation (TGV) [Bredies et al., 2010]. Total Generalized Variation does not suffer from the staircaising effect of Total Variation. The staircasing effect stems from the fact that Total Variation favors piecewise constant (i.e. zero-order) functions, whereas TGV of order k is able to reconstruct functions of order $k - 1$. A sensible choice for the computation of depthmaps is second-order TGV, which means that piecewise linear functions can be reconstructed. This circumvents the discrete depth steps of planesweep caused by discretization of the depth range, because linear functions are able to smoothly describe slanted surfaces.

The model for TGV multiview stereo is given by

$$\min_u \left\{ TGV_\alpha^2(u) + \int_\Omega \sum_{i=1}^{N} \lambda_i |I_i(f(x,u)) - I_{\text{ref}}(x)| dx \right\}, \tag{2.8}$$

where the first term (regularization term) denotes second order TGV with weighting factors α, the second term is the data term, $x = (x,y)^T \in \Omega$ are pixel coordinates and $\Omega \subseteq \mathbb{R}^2$ is the image domain. The dataterm measures pixel similarities obtained by projecting a pixel coordinate x from the reference view into 3D space at depth u, yielding the point $X(x,u)$. This point is subsequently transformed into the coordinate frame of the second camera and back-projected onto the image plane, yielding the corresponding point x'. Under the assumption of lambertian surfaces, the pixel gray values of x and x' match if the the 3D point $X(x,u)$ lies exactly on the surface (see figure 2.9). The dataterm hence consists of simple pixelwise absolute differences of gray values. To use multiple views, the absolute differences of N projections are simply summed up.

While TGV is a convex functional, the dataterm of (2.8) is highly non-convex, as it contains the objective u as an argument of the image I_i and the transformation $f(x,u)$. We will now examine the transformation $f(x,u)$ in more detail. It is given by

$$f(x,u) = KTuK^{-1}x \tag{2.9}$$

The term $uK^{-1}x$ is the projection of a pixel coordinate $x = (x,y,1)^T$ into 3D space at depth u using the camera intrinsic calibration matrix K. This 3D point in the coordinate frame of the reference view is then subjected to the homogeneous transformation $T = [R \mid t]$, which is the relative rotation and translation between the reference view and the sensor view (see (2.4)). Once the 3D point is transformed into the coordinate frame of the sensor view, it is back-projected onto the image plane by multiplication with the intrinsic calibration matrix, yielding the corresponding point x'. The dataterm of (2.8) is convexified by using a first order Taylor linearization

$$I(f(x,u)) \approx I(f(x,u_0) + (u-u_0)\frac{\mathrm{d}I(f(x,u)}{\mathrm{d}u}\bigg|_{u_0}$$

Using the chain rule, one finds that

$$\frac{\mathrm{d}I(f(x,u)}{\mathrm{d}u} = \nabla I(f(x,u))\frac{\mathrm{d}f(x,u)}{\mathrm{d}u},$$

which means that the derivative of I w.r.t. u can be found by calculating the normal image gradient (e.g. by forward differences) and scaling the gradient by a differential vector obtained from the derivative of the transformation (2.9) w.r.t. u. Intuitively, this derivative is the direction in the image that results from a variation of u of the 3D point $X(x, u)$, i.e. it points along the epipolar line. Since all components of (2.9) are known, the derivative can be computed analytically to

$$\frac{\mathrm{d}f(x,u)}{\mathrm{d}u} = \begin{bmatrix} f_x \frac{\frac{\mathrm{d}\hat{X}(u)}{\mathrm{d}u} \cdot \hat{Z}(u) - \hat{X}(u) \cdot \frac{\mathrm{d}\hat{Z}(u)}{\mathrm{d}u}}{\hat{Z}(u)^2} \\ f_y \frac{\frac{\mathrm{d}\hat{Y}(u)}{\mathrm{d}u} \cdot \hat{Z}(u) - \hat{Y}(u) \cdot \frac{\mathrm{d}\hat{Z}(u)}{\mathrm{d}u}}{\hat{Z}(u)^2} \end{bmatrix},$$

where $\hat{X}(u), \hat{Y}(u)$ and $\hat{Z}(u)$ are the components of the 3D point $\hat{X}(x, u)$ transformed into the coordinate frame of the second view. Their derivatives are given by

$$\begin{aligned} \frac{\mathrm{d}\hat{X}(u)}{\mathrm{d}u} &= r_{11}\frac{1}{f_x}(x - p_x) + r_{12}\frac{1}{f_y}(y - p_y) + r_{13} \\ \frac{\mathrm{d}\hat{Y}(u)}{\mathrm{d}u} &= r_{21}\frac{1}{f_x}(x - p_x) + r_{22}\frac{1}{f_y}(y - p_y) + r_{23} \\ \frac{\mathrm{d}\hat{Z}(u)}{\mathrm{d}u} &= r_{31}\frac{1}{f_x}(x - p_x) + r_{32}\frac{1}{f_y}(y - p_y) + r_{33}, \end{aligned}$$

where $r_{11} \dots r_{33}$ are the entries of the relative rotation matrix between the two views, and p_x, p_y and f_x, f_y are the principal point and the focal length respectively. By combining the individual terms, we rewrite (2.8) as

$$\min_u \left\{ TGV_\alpha^2(u) + \int_\Omega \sum_{i=1}^{N} \lambda_i |\rho_i(x, u, u_0)| dx \right\}, \tag{2.10}$$

where the dataterm $\rho_i(x, u, u_0)$ is defined as

$$\rho_i(x, u, u_0) = I_i(f(x, u_0)) - I_{ref}(x) + (u - u_0)\nabla I_i(f(x, u)) \frac{\mathrm{d}f_i(x, u)}{\mathrm{d}u}\bigg|_{u_0}$$

Total Generalized Variation of order k and weighting factors α is defined as

$$TGV_\alpha^k(u) = \sup \left\{ \int_\Omega u \, \mathrm{div}^k v \, \mathrm{d}x \quad \text{s.t.} \quad \|\mathrm{div}^l v\|_\infty \le \alpha_l, \quad l = 0, \dots, k-1 \right\}$$

Formulated this way, TGV is hard to optimize because of the constraints on the divergence of v. While there exist simple algorithms for the first-order constraint, i.e. $\|v\|_\infty \le 1$, the higher order constraint on the l-divergence of v is much more difficult to deal with and has usually to be fomrulated as an additional optimization problem. Fortunately, in [Bredies et al., 2010] the authors give a recursive primal formulation of the TGV semi-norm. For second order TGV, it can be written conveniently as

$$TGV_{\alpha}^{2}(u) = \min_{u_1} \alpha_1 \|\mathrm{D}u - u_1\|_{\mathcal{M}} + \alpha_0 \|\mathrm{D}u_1\|_{\mathcal{M}} \tag{2.11}$$

where $\mathcal{M}(\Omega, \mathbb{R}^n)$ denotes the space of Radon measures on Ω.

Combining (2.10) and (2.11) yields the complete model for TGV multiview stereo

$$\min_{u,u_1} \left\{ \alpha_1 \|\mathrm{D}u - u_1\|_{\mathcal{M}} + \alpha_0 \|\mathrm{D}u_1\|_{\mathcal{M}} + \int_{\Omega} \sum_{i=1}^{N} \lambda_i |\rho_i(x, u, u_0)| dx \right\} \tag{2.12}$$

Note that the optimization problem now consists of two variables u and u_1, whereby u_1 comes from the recursive formulation of the TGV regularizer. (2.12) is optimized using the first order primal-dual algorithm of [Chambolle and Pock, 2011]. This algorithm requires the problem to be cast as a min-max saddle point problem. We therefore start by transforming (2.12) to its primal-dual formulation using Fenchel duality

$$\begin{aligned} &\min_{u,u_1} \max_{p,q,r} \alpha_1 \langle \mathrm{D}u - u_1, p\rangle + \alpha_0 \langle \mathrm{D}u_1, q\rangle + \lambda \langle \rho(x, u_0, u), r\rangle \\ &\qquad\qquad\qquad \text{s.t.} \quad \|p\|_\infty, \|q\|_\infty, \|r\|_\infty \leq 1, \end{aligned} \tag{2.13}$$

where p, q, r are the dual variables. For simplicity, we will derive the optimization procedure for the stereo case (i.e. $N = 1$) and afterwards show how to extend it for multiple views. A nice side effect of the dualization is that none of the terms of (2.13) depends exclusively on the primal variables u, u_1, they are all functions of a primal and a dual variable. As a result, one of the *proximal operators* needed for the application of the first-order primal-dual algorithm reduces to the identity.

The update iterations are given by

$$
\begin{cases}
p^{n+1} = \Pi_{\|p\|_\infty \le 1}\left\{p^n + \sigma\alpha_1(\nabla\bar{u}^n - \bar{u}_1^n)\right\} \\
q^{n+1} = \Pi_{\|q\|_\infty \le 1}\left\{q^n + \sigma\alpha_0\nabla\bar{u}_1^n\right\} \\
r^{n+1} = \Pi_{\|r\|_\infty \le 1}\left\{r^n + \sigma\lambda(I_t + (u - u_0)I_u)\right\} \\
u^{n+1} = u^n - \tau(-\alpha_1 \operatorname{div} p^{n+1} + \lambda I_u r^{n+1}) \\
u_1^{n+1} = u_1^n - \tau(-\alpha_1 p^{n+1} - \alpha_0 \operatorname{div} q^{n+1}) \\
\bar{u}^{n+1} = 2u^{n+1} - u^n \\
\bar{u}_1^{n+1} = 2u_1^{n+1} - u_1^n
\end{cases}
$$

where we used the simplifications $I_t = I(f(x, u_0)) - I_{ref}(x)$ and $I_u = \nabla I(f(x, u))\frac{\mathrm{d}f(x,u)}{\mathrm{d}u}\big|_{u_0}$. The operation $\Pi_{\|p\|_\infty \le 1}(\cdot)$ stems from the proximal operator for the dual variables and is a simple projection onto the unit ball given by the following explicit formula

$$
p = \Pi_{\|\tilde{p}\|_\infty \le 1}(\tilde{p}) \quad \Leftrightarrow \quad p_{ij} = \frac{\tilde{p}_{ij}}{\max\{1, |\tilde{p}_{ij}|\}}
$$

We choose the timesteps τ, σ according to [Chambolle and Pock, 2011].

The extension to multiple views involves multiple dual variables r_i (one for each view), which are summed up in the primal update step

$$
\begin{cases}
r_i^{n+1} = \Pi_{\|r_i\|_\infty \le 1}\left\{r_i^n + \sigma\lambda(I_i^t + (u - u_0)I_i^u)\right\} \quad i = 1 \ldots N \\
u^{n+1} = u^n - \tau(-\alpha_1 \operatorname{div} p^{n+1} + \lambda \sum_{i=1}^N I_i^u r_i^{n+1})
\end{cases}
$$

To account for large changes in depth, the procedure is implemented in a coarse-to-fine framework. Starting at a low resolution, a solution is computed which is then used as initialization for the next level. The whole method is very sensitive to the baseline between camera views. A large baseline results in occlusions, which hurts the algorithm. The implementation of a simple occlusion detection scheme based on map uniqueness [Brown et al., 2003] helps, but results still depend on a "suitable" configuration of camera views in 3D space. Figure 2.10 shows results of the algorithm for different number of sensor views. While the depthmaps are more dense and contain less outliers than the planesweep results, one can notice that depth edges are in some cases more blurred. The runtime of this method strongly depends on the algorithm parameters (number of iterations, number of warps, scale factor . . .), for high quality settings as depicted in figure 2.10, it easily

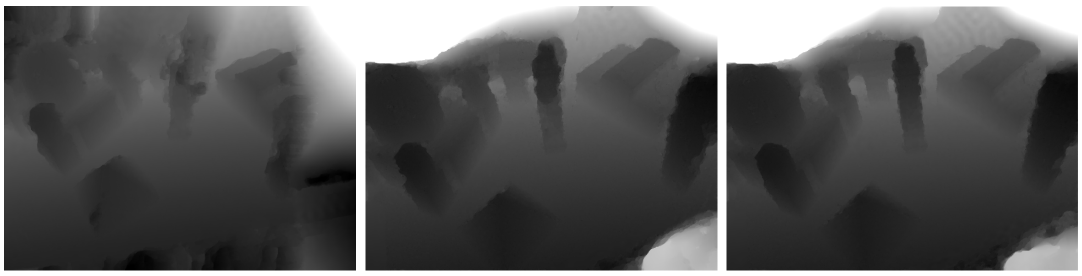


Figure 2.10: Results for TGV multiview stereo. In contrast to the planesweep results, the depthmaps are dense and contain less outliers.

takes twice as long as planesweep. Note that this still is very fast in comparison to most other state of the art stereo algorithms, additionally many of the latter are not capable of using multiple views.

2.4 Depthmap Fusion

Once the individual depthmaps have been computed, they are integrated into a common 3D model. To do this, we follow the method of [Zach, 2008, Zach et al., 2007]. Their work in turn is based on the idea of [Curless and Levoy, 1996], who represent the surface implicitly as the zero level set of an underlying truncated signed distance function (TSDF). This function is defined as $u : \Omega \to [-1, 1]$ where $\Omega \subseteq \mathbb{R}^3$. Every voxel is assigned a value, which measures its signed distance to the true surface. The value zero means that the surface goes exactly through the voxel, -1 means the voxel is part of solid geometry and $+1$ means "nothing", i.e. the voxel is part of free space. This representation is ideally suited to incrementally build up the scene geometry, as well as to implement smoothness constraints.

First, depthmaps are converted to signed distance fields f_i as depicted in figure 2.11. Each pixel is projected at a 3D point X according to the depth and voxels along the line of sight are assigned values in the interval $[-1, 1]$. Note that all voxels along the line of sight between the camera and the surface are assigned a positive value. The reasoning is that if there were another object in the line of sight, the camera would not be able to see the surface point X, thus we safely can assume that all voxels between the camera and the surface are visible. On the other hand, voxels along the line of sight *behind* the surface are assigned a value until a distance threshold η is met. This accounts for the fact that we cannot make any a priori assumptions on the thickness of the surface, it could be a thin

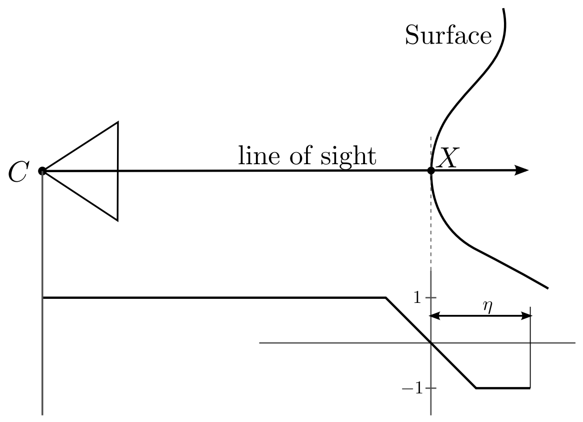


Figure 2.11: Each pixel of the deptmap projects to a 3D point X. Voxels of the distance fields f_i store the truncated signed distance value from the surface.

sheet of paper or a solid wall. The parameter η therefore in a way controls the amount of detail representable by the method.

The individual distance fields f_i are fused together by the following variational model

$$\min_u \left\{ \int_\Omega |\nabla u| + \lambda \int_\Omega \sum_{i=1}^{K} |u(x) - f_i(x)| dx \right\} \tag{2.14}$$

The first term (regularization term) enforces smoothness of the solution by minimizing the length of the level sets of u, i.e. it seeks to minimize the surface area of the reconstructed geometry. The dataterm consists of the ℓ_1-distance of u to all individual distance fields, it robustly penalizes outliers and noise. An obvious limitation of this formulation is the fact that storing the distance fields f_i has a memory complexity of $\mathcal{O}(K)$, where K is the number of distance fields. This quickly becomes infeasible even on machines with lots of memory. We therefore follow the idea of [Zach, 2008], who proposed to use a histogram compression which is able to reduce the memory complexity to $\mathcal{O}(1)$, i.e. constant memory footprint. The compression is motivated by the insight that it is not necessary to store the exact value of the TSDF to recover the zero level. Rather, each voxel maintains a histogram consisting of N bins which approximates the probability density function of u at that voxel. The TSDF is sampled at discrete positions d_i, and the exact value is replaced by the nearest d_i . Since the number of bins is constant, this allows to store an arbitrary number of signed distance fields with constant memory footprint.

The model then reads

$$\min_u \left\{ \int_\Omega |\nabla u| + \lambda \int_\Omega \sum_{i=1}^{N} h(x,i)|u(x) - d_i| dx \right\} \tag{2.15}$$

where N is the number of histogram bins and d_i are the corresponding sampled values of the signed distance function. $h(x,i)$ denotes the histogram count of bin i, i.e. how often the value d_i occurred in the distance fields at voxel x. Note that this model allows for incremental updates, if new depthmaps are available the histogram bins are updated ($h(x,i)$ changes), and the minimization algorithm adapts to the new data. Furthermore, it is convex, which means it is possible to find the global solution independent from any initialization.

For minimization we use the first-order primal-dual algorithm of [Chambolle and Pock, 2011]. The primal-dual formulation of (2.15) is given by

$$\min_u \max_{\|p\|_\infty \leq 1} \left\{ - \int_\Omega u \operatorname{div} p + \lambda \sum_{i=1}^{N} \int_\Omega h(x,i)|u(x) - d_i| dx \right\} , \tag{2.16}$$

where $p : \Omega \to \mathbb{R}^3$ is the dual variable. The algorithm consists of alternatingly performing gradient descend steps in u and gradient ascend steps in p. The update iterations are given by

$$\begin{cases} u^{n+1} = \operatorname{prox}_{\text{hist}} \left(u^n - \tau(-\operatorname{div} p^n) \right) \\ p^{n+1} = \operatorname{prox}_{\|p\|_\infty \leq 1} \left(p^n + \sigma \nabla (2u^{n+1} - u^n) \right) \end{cases} \tag{2.17}$$

The timesteps τ, σ are chosen to fulfill the convergence criterion $\tau\sigma\|\operatorname{div}\|^2 < 1$. An elementary requirement of the primal-dual algorithm is that the proximal operators are "easy" to compute. For the dual variable p, the constraint $\|p\|_\infty \leq 1$ is modeled by indicator functions of a convex set. Hence, the prox operator reduces to a projection of the following form

$$p = \operatorname{prox}_{\|\tilde{p}\|_\infty \leq 1} \quad \Leftrightarrow \quad p_{ij} = \frac{\tilde{p}_{ij}}{\max\{1, |\tilde{p}_{ij}|\}}$$

The prox operator for the primal variable u is given by the solution of the following optimization problem

$$\operatorname{prox}_{\text{hist}}(\tilde{u}(x)) = \arg\min_u \left\{ \frac{\|u - \tilde{u}(x)\|^2}{2\tau} + \lambda \sum_{i=1}^{N} h(x,i)|u - d_i| \right\} , \tag{2.18}$$

which consists of a quadratic distance term plus the histogram term. In

[Li and Osher, 2009] the authors give an explicit formula for computing the global solution of problems of the form $\arg\min_x \sum_{i=1}^{N} w_i |x - u_i| + F(x)$, where $F(x)$ is strictly convex. It is easy to see that this is exactly the case for (2.18), whereby $F(u) = \frac{\|u - \tilde{u}(x)\|^2}{2\tau}$. The solution is defined as

$$\mathrm{prox}_{\mathrm{hist}}(\tilde{u}) = \mathrm{median}\,\{d_1, ..., d_N, p_0, ..., p_N\}$$

where d_i are the distances related to the according histogram bin i and the p_i, $i = 1...N$ are computed as

$$p_i = \tilde{u} + \tau\lambda W_i\ , \quad W_i = -\sum_{j=1}^{i} h(x, i) + \sum_{j=i+1}^{N} h(x, i)$$

The idea behind the computation of the p_i is to split (2.18) into individual linear subproblems and extract the global solution as the median of the solutions of the subproblems. If $u_{\mathrm{lin}} \in (d_i, d_{i+1})$, the linear subproblem of (2.18) is given by

$$\arg\min_{u_{\mathrm{lin}}} \left\{ \frac{\|u_{\mathrm{lin}} - \tilde{u}\|^2}{2\tau} - \lambda W_i u_{\mathrm{lin}} \right\}$$

The optimality condition for u_{lin} is given by

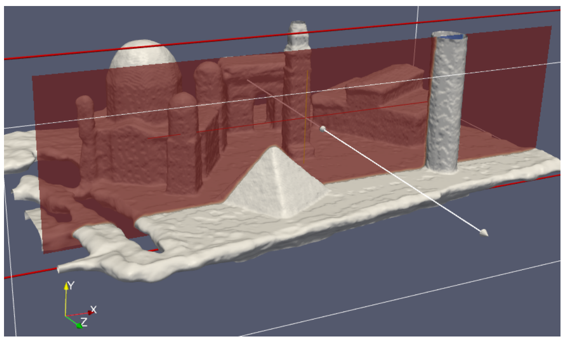
$$\frac{u_{\mathrm{lin}} - \tilde{u}}{\tau} - \lambda W_i = 0$$

and solving for u_{lin} yields

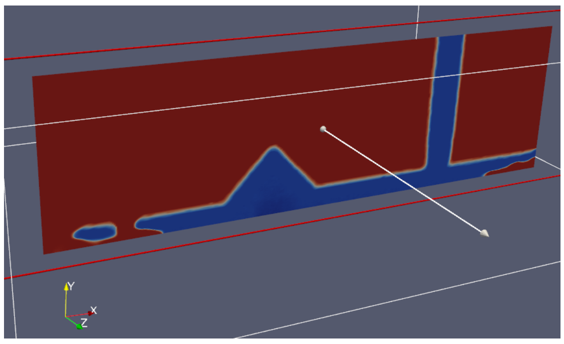
$$u_{\mathrm{lin}} = p_i = \tilde{u} + \tau\lambda W_i$$

2.5 Visualization

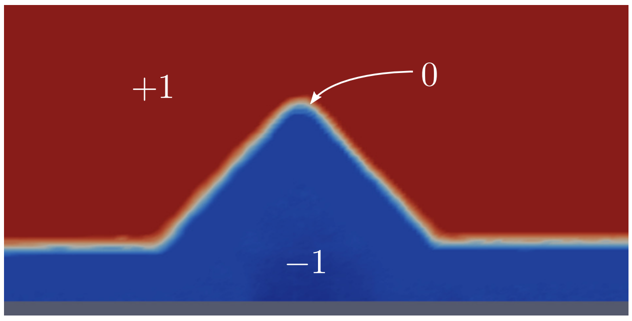
Due to the iterative nature of both the fusion algorithm and the whole system, an efficient way to visualize results is needed. A full recalculation of a triangle based mesh of the current reconstruction seems unfeasible, since the reconstruction changes almost continuously. Rather, the implicit representation of geometry is used directly. We employ a CUDA based raycaster which is capable of visualizing iso-levels of the truncated signed distance function. The codebasis of the raycaster was given to us kindly by Markus Steinberger, it is taken from his master's thesis [Steinberger, 2010]. The method is based on [Hadwiger et al., 2005]. For a virtual camera pose C_v, the viewing ray r_v of each pixel is



(a) Slice through geometry



(b) Slice of the TSDF



(c) Closeup, with values of the TSDF

Figure 2.12: Visualization of the TSDF. The raycaster finds the voxels containing the μ-level set by marching through the volume along the viewing ray of each pixel. The surface is recovered by setting $\mu = 0$.

marched until the appropriate value is encountered. Surface normals are computed from the derivative of the TSDF, which is approximated by central differences. Because the ratio of the distance between adjacent viewing rays at a certain depth and the voxel size depends on the virtual camera position, we optionally offer trilinear interpolation to get a smooth value of the TSDF for a given position along the viewing ray. The raycasting process is ideally suited for parallelization, because all rays can be computed independently from each other.

A diffuse lighting model is obtained by calculating the diffuse reflection coefficient as $c_{\text{diffuse}} = \langle n, l \rangle$, where n is the surface normal vector computed from the derivative of the TSDF and l is the light direction. By setting $l = r_v$, one gets a simple "headlight" illumination model.

To further increase the visual experience, a texture mapping methodology was implemented. It works by determining a grayvalue for each voxel v_s of the surface based on keyframe image data. To this end, the closest keyframes (in terms of euclidean distance)

C_i^{Tex}, $i = 1 \ldots 5$ for a given virtual camera pose are searched for (see figure 2.13), and each voxel of the 3D surface is projected into these keyframes by $x_i = KC_i^{\text{Tex}}\xi(v_s)$. Here, $\xi(\cdot)$ denotes the mapping from voxel indices to 3D world coordinates, C_i^{Tex} are the camera pose matrices of the 5 closest keyframes and K is the camera intrinsic calibration matrix. Next, grayvalues are sampled from the keyframe images and the final texture value is computed as the median of the 5 grayvalues and assigned to voxel v_s. This approach works

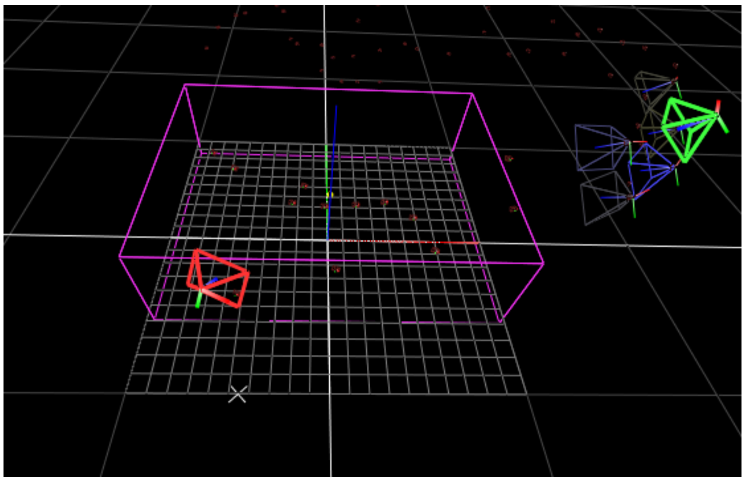
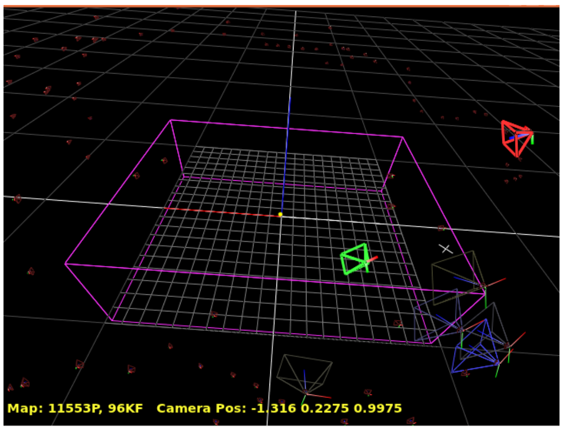
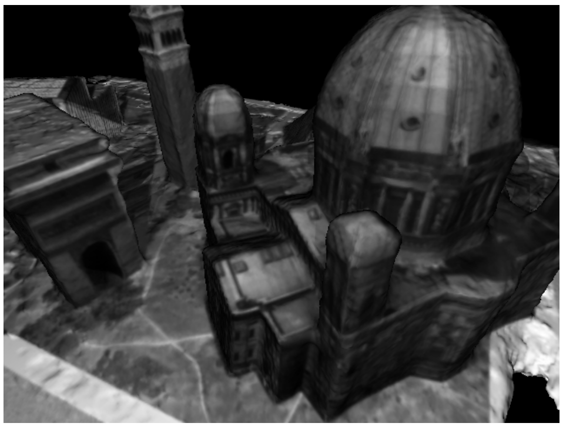


Figure 2.13: Keyframe selection for texture mapping. The current live camera position is depicted red, the virtual camera is green. Shades of blue denote the 5 closest keyframes to the virtual camera.

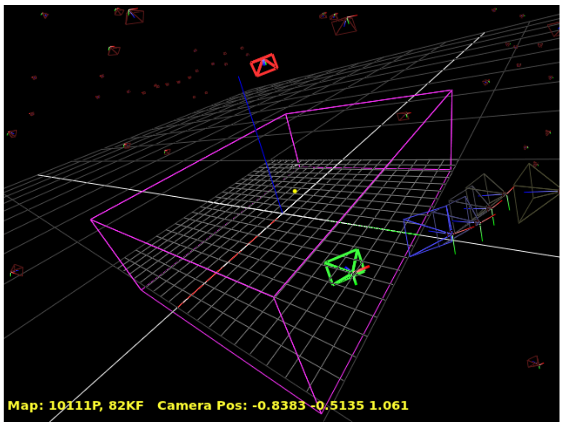
fine as long as there are enough keyframes in close proximity to the virtual camera. However, as the virtual camera can move unconstrained through 3D space, there might occur situations where even the closest keyframes are a considerable distance away (see figure 2.14). In this case, surface voxels rendered by the raycaster are occluded in the keyframes used for texturing. The resulting grayvalues therefore contain wrong information, as can be seen in figure 2.14f. This problem could be circumvented by the user by moving the camera close to the virtual camera position, so that more suitable keyframes are available. Alternatively a more sophisticated texture mapping algorithm could be used, where for each voxel an appropriate set of texture-keyframes is chosen an additional visibility checks along the viewing rays are performed. However, this conflicts with the requirement that the visualization should be fast. As our simple texture mapping method produces nice results in the majority of cases, implementation of a more sophisticated method is postponed for future work.



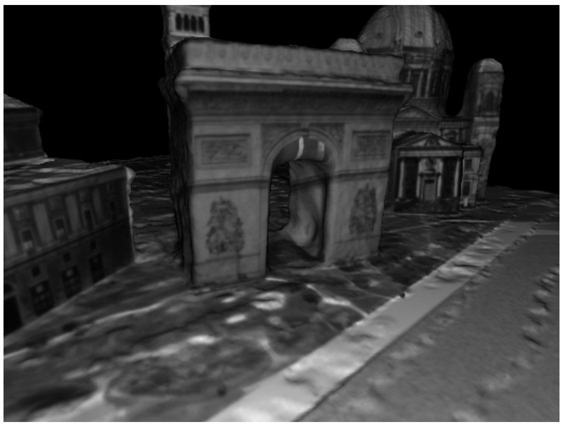
(a)



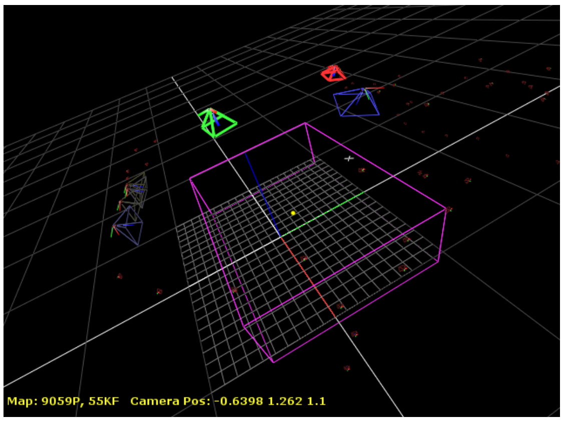
(b)



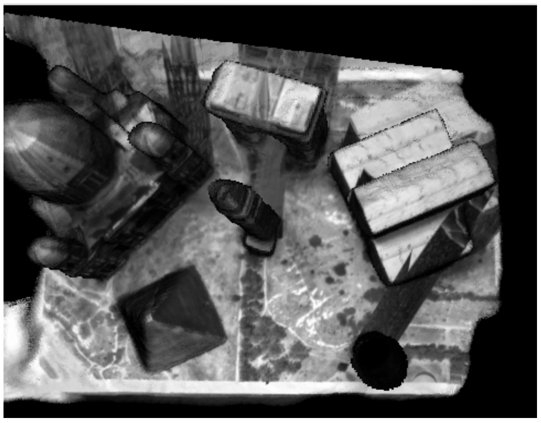
(c)



(d)



(e) Keyframes are far away from the virtual camera



(f) Resulting texture contains artefacts

Figure 2.14: Results of the texture mapping algorithm. If no keyframes are found close to the virtual camera (green), the resulting texture contains artifacts .

Chapter 3

Implementation

Contents

Outline: *In this chapter implementation details of the system are discussed. The changes made to PTAM are described, as well as the methods to speed up the planesweep algorithm. Different approaches for the volume update procedure are presented, and technical details of the visualization method are given. Finally the whole system and the interaction of the individual parts is discussed.*

3.1 Tracking

PTAM was changed in various places. Mostly this involved adaptions for the GUI and numerous visualizations for user feedback, e.g. displaying the volume in the live camera image. An important change was made to the keyframe structure, which was enhanced to store an undistorted version of the image in addition to the distorted image. This avoids having to do the undistortion every time the image is used as reference/sensor view for the multiview stereo algorithm. PTAM uses the distorted image directly and manually undistorts the feature positions during its tracking stage. For a few hundred positions, this is faster than undistorting the whole image, which involves a costly interpolation for every pixel. In our system, we heavily rely on CUDA to use the massive parallel computation capabilities of todays high end graphics cards. We therefore accelerate the undistortion procedure by making use of the bilinear interpolation offered by the GPU.

Because the reconstruction system depends on the keyframes generated by PTAM, we also experimented with the heuristic responsible for keyframe generation. A critical point for our method is the number of keyframes, thus a straightforward approach might be to change the heuristic such that more keyframes are generated. However, experiments showed that this has negative impact on tracking performance, i.e. camera tracking was lost more often. This is probably due to the mapping component having less time for global bundle adjustment when keyframes have to be processed more frequently. Experiments showed that the original heuristic was fairly well balanced for typical indoor scenes, therefore we left the heuristic unchanged.

3.2 Depthmap Generation

Calculating NCC for two images I_1, I_2 requires evaluation of the following formula

$$c = \frac{1}{n} \sum_{x,y \in \mathcal{W}} \frac{(I_1(x,y) - \bar{I}_1)(I_2(x,y) - \bar{I}_2)}{\sigma_1 \sigma_2}, \tag{3.1}$$

which is computationally expensive because computing the mean $\bar{I}_1, \bar{I}_2$ and the standard deviation σ_1, σ_2 for a windowsize of $N \times M$ requires $N \cdot M$ memory reads for each pixel position (x, y). Using the linearity of the expectation operator, calculation of the standard deviation can be simplified to

$$\sigma^2 = \mathbf{E}\{(I - \bar{I})^2\} = \mathbf{E}\{I^2\} - (\mathbf{E}\{I\})^2, \tag{3.2}$$

which is further accelerated by computing sum images of I and I^2 for the chosen windowsize. The sum images are defined as

$$I_{\text{sum}}(x, y) = \sum_{u,v \in \mathcal{W}} I(u, v)$$

for a window $\mathcal{W}$ centered at position x, y. Likewise, for the nominator of (3.1) a sum image $I_1 \cdot I_2$ is computed. Separating the sum filter (i.e. independently filtering row-wise and column-wise) further reduces the number of memory accesses from $N \cdot M$ to $N + M$. Note that computation of σ according to (3.2) can be problematic in case the two values $\mathbf{E}\{I^2\}$ and $(\mathbf{E}\{I\})^2$ cancel each other (e.g. in homogeneous image regions). The denominator of (2.18) is then close to zero and one wrongly gets a very high value for the NCC. Therefore, we check the value of the denominator and set $c = 0$ if it is below a threshold.

Whereas the homography which maps the sensor view onto the plane at depth d (see (2.6)) is computed on the CPU, interpolation, computation of the NCC and depth extraction by WTA are performed on the GPU. According to experiments, the number of memory reads has most influence performance. The usage of sum images pays off, even though this means additional memory writes to store the sum values.

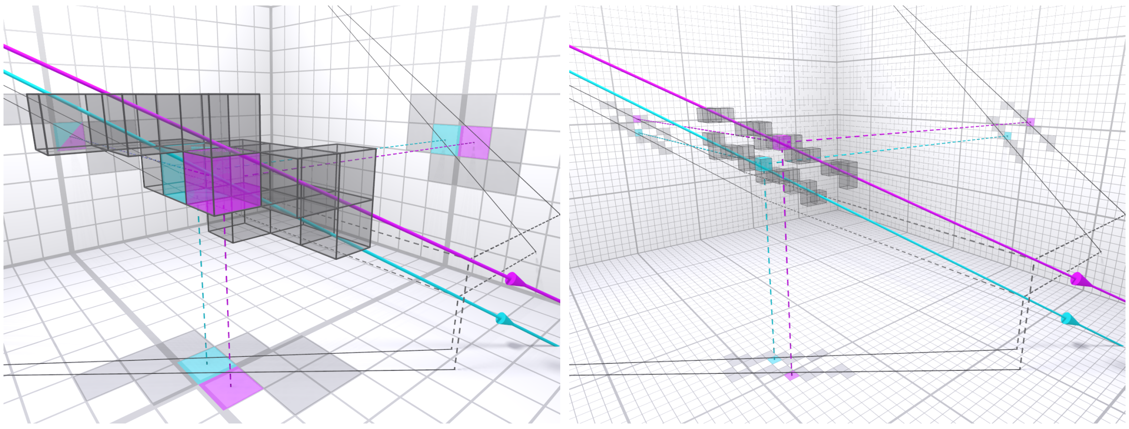
We also experimented with simpler correlation values that are faster to compute (SAD, SSD), but it turned out that these give inferior results, especially when using less than three views.

3.3 Fusion

To save memory, we use an 8-bit datatype for storing the volume histogram bins. This means that each bin can get at most 255 votes, i.e. we assume that a particular voxel is not seen from more than 255 camera positions. Because PTAM is limited in the number of keyframes (see section 1.4.4), and additionally not every single keyframe is used for reconstruction, this assumption seems reasonable.

Computing the signed distance function and updating the volume histogram (see section 2.4) can be done in two different ways. One can project each pixel of the reference view into 3D space at the depth value according to the depthmap to find the voxel containing the surface. Then one walks along the viewing ray and assigns signed distance values to voxels. On the other hand, one can calculate the distance of every voxel to the reference camera position and compare this distance to the corresponding depth value of the depthmap, which is found by projecting the voxel position into the reference view and interpolating from the depthmap. Both of the above methods assume there is a fixed bijective mapping from the global world coordinate frame to voxel indices, i.e. memory locations. We will denote the first method as ray-based since it operates on viewing rays defined by pixels of the reference view, and the second method as voxel-based. See section 4.2.2 for an evaluation of the differences of the two methods.

Figure 3.1 depicts ray-based histogram update for two adjacent viewing rays, which are computed by projecting two neighboring pixels into 3D world. Using depth information from the depthmap, the voxel containing the 3D point is found. This voxel (shown in



(a) Low voxel resolution (b) High voxel resolution

Figure 3.1: Ray-based histogram update. Every pixel of the reference camera view is projected into the volume. At low voxel resolution (a) adjacent viewing rays frequently hit the same voxels, at high resolution many voxels are not hit at all (b).

magenta and cyan) is assigned the distance value 0. Then, voxels along the viewing rays are found and their signed distance value is computed. This approach makes optimal use of the depthmap data: Exactly one viewing ray is computed for each pixel of the depthmap, and only those voxels along the ray are updated, i.e. the signed distance function only changes if there actually is new data. However, the method is sensitive to the volume resolution, i.e. the size of the voxels. At low resolution (see figure 3.1a), adjacent viewing rays often hit the same voxel, resulting in ambiguous information. At high resolution (see figure 3.1b), there are voxels between the viewing rays that are not updated at all. If this gap gets too big, the fusion algorithm cannot close the surface and one gets holes in the reconstruction.

Voxel-based histogram update (see figure 3.2) works by projecting the position of every voxel into the reference camera view. From the resulting image coordinate, the depth value is sampled from the depthmap. The distance of the voxel to the reference view and the depth value are used to determine the signed distance value for the voxel. This approach makes optimal use of the volume: Every voxel is updated exactly once. However, depending on the volume resolution there might be under- or oversampling of depth data. If the depthmap is undersampled (see figure 3.2a), precious data is lost, if it is oversampled (see figure 3.2b) one eventually gets artifacts. An experimental evaluation of different volume resolutions and histogram update methods is given in section 4.2.2.

While ray-based histogram update is more efficient in terms of the number of voxels

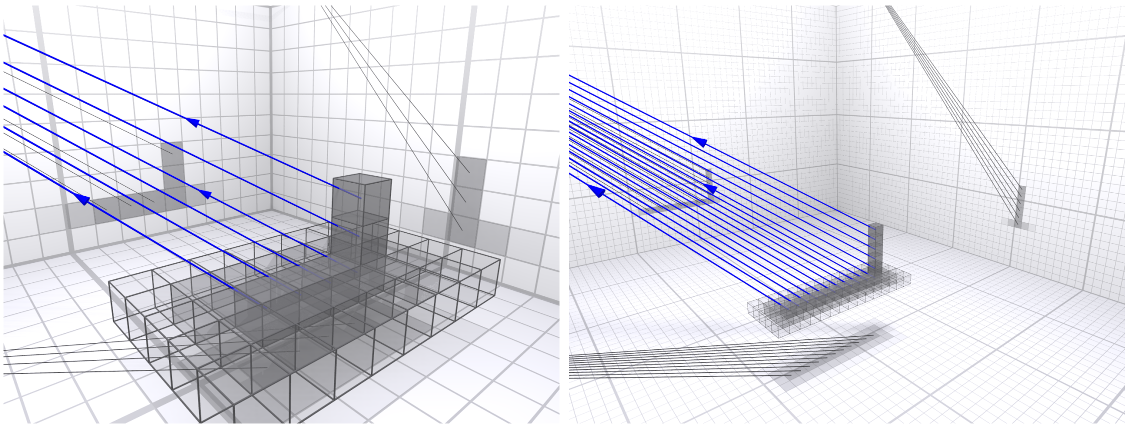

(a) Low voxel resolution (b) High voxel resolution

Figure 3.2: Voxel-based histogram update. Every voxel is projected into the reference camera view to determine the signed distance to the depth value from the depthmap. At low voxel resolution (a) information from the depthmap is not used optimally, at high voxel resolution depthmap data is oversampled (a).

that need to be changed, it is tricky to implement in parallel on the GPU. Because adjacent viewing rays can hit the same voxel, the voxel update has to be serialized, i.e. the histogram bin increment operation has to be atomic. Unfortunately, CUDA does not support atomic increment for 8-bit datatypes. Therefore, the functionality had to be implemented using the generic compare-and-swap (CAS) operation as shown in listing 3.1. We will go through this function in detail, since it is critical for correct histogram update.

Listing 3.1: Atomic increment for 8-bit datatypes

```
__device__ void atomicAdd(unsigned char* data, unsigned int index)
{
  unsigned int* dword_ptr = (unsigned int*)&(data[(index / 4) * 4]);
  unsigned char byte_pos = index % 4;

  unsigned int readback, old_value, new_value;
  unsigned char byte_value;
  do
  {
    old_value = *dword_ptr;

    byte_value = (old_value & (0xFF << byte_pos*8)) >> byte_pos*8;
    byte_value = byte_value < 254 ? ++byte_value : 255;

15      new_value = (byte_value << (byte_pos*8)) |
16                  (old_value & ~(0xFF << (byte_pos*8)));
17
18      readback = atomicCAS(dword_ptr, old_value, new_value);
19    } while (readback != old_value);
20  }
```

The function arguments `unsigned char* data` and `unsigned int index` are the pointer to the memory block containing the volume data and the (linear) voxel index into the memory block respectively. Because CAS is available for 32-bit datatypes only, we start by extracting the 32-bit aligned word which contains the byte that is to be updated, and the position of the byte in that word (lines 3 and 4). The 8-bit value is extracted from the 32-bit word and incremented, we saturate at 255 to avoid overflow (lines 12 and 13). Next, the new value of the 32-bit word is calculated, i.e. the value that results from replacing the incremented 8-bit value in the 32-bit word (line 15, see figure 3.3). Now we

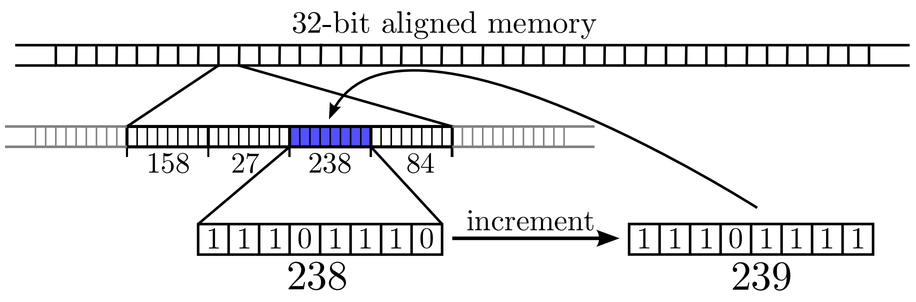


Figure 3.3: Construction of the 32-bit word for `atomicCAS()`. The old value consists of the concatenation of the four byte values $158, 27, 238, 84$ in their binary representation: $[10011110; 00011011; 11101110; 01010100]_{\text{bin}} = 2652630612_{\text{dec}}$. After replacing the third byte with the incremented value, the new 32-bit word becomes $[10011110; 00011011; 11101111; 01010100]_{\text{bin}} = 2652630868_{\text{dec}}$.

attempt to change the value of the 32-bit word in the volume memory block via compare-and-swap (line 18). CAS takes three arguments, the first one is a memory location and the second and third are two numeric values. The value pointed to by the first argument is compared to the second argument, and if they are the same, the memory is changed to the new value (the new value is “swapped in”). The comparison and change operation is atomic, meaning it cannot be interrupted by other threads. The function returns the value of the memory location before the change. This return value is used to check if the swap was successful: We expect the return value to be equal to the old 32-bit value (line 19). If it is not, it means that another thread has managed to change the memory location in the meantime, i.e. in the time since this thread read the old 32-bit value from memory (line 10). In this case, we update the old 32-bit value and loop until success.

Voxel-based histogram update does not suffer from this effect, since every voxel is updated exactly once. It also has a much more regular memory access pattern (coalesced memory access), whereas ray-based histogram update exhibits a highly irregular memory access pattern. For these reasons, voxel-based histogram update is faster than ray-based histogram update, although the number of voxels that need to be accessed is significantly higher. Note that voxel-based update scales approximately linearly with the number of voxels (i.e. the volume resolution), while ray-based update only depends on the resolution of the depthmap.

The minimization algorithm (see section 2.4) is straightforward to implement in parallel on the GPU. To numerically approximate the gradient and divergence operators, forward and backward differences are used respectively. Computing the histogram prox term (2.18) requires computation of the median of $2N+1$ elements, where N is the number of histogram bins. N elements are the distances related to the sampling of the signed distance function. These are already sorted by definition, therefore we perform an insertion sort of the remaining $N + 1$ elements.

3.4 Visualization

The raycasting procedure described in section 2.5 is accelerated by exploiting the slope of the truncated signed distance function, i.e. voxels along a viewing ray do not jump from signed distance value $+1$ to -1, but we expect at least some voxels with values in between. Therefore, when marching along the ray, a big stepsize (e.g. 4 times the voxel size) is used as long as the signed distance value is not within some threshold of the iso-level. Only when this threshold is met, the stepsize is reduced to get voxel-accurate results (see figure 3.4). This allows efficient traversal of empty space and concentrates computation resources where they are needed, i.e. near the surface.

The raycaster uses the standard OpenGL right-hand coordinate system with the negative z-axis into the screen, whereas PTAM uses a left-hand system with positive z into the screen. To convert camera pose matrices between the two, a transformation matrix of

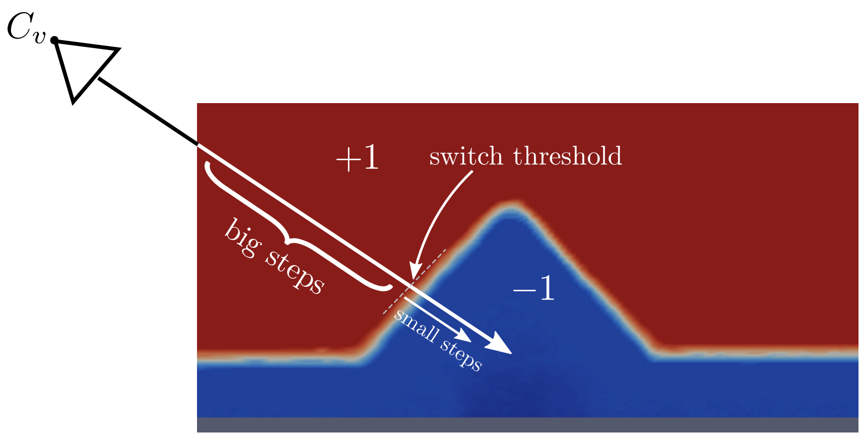


Figure 3.4: By using big steps when marching along the viewing ray through empty space, the raycasting procedure is accelerated.

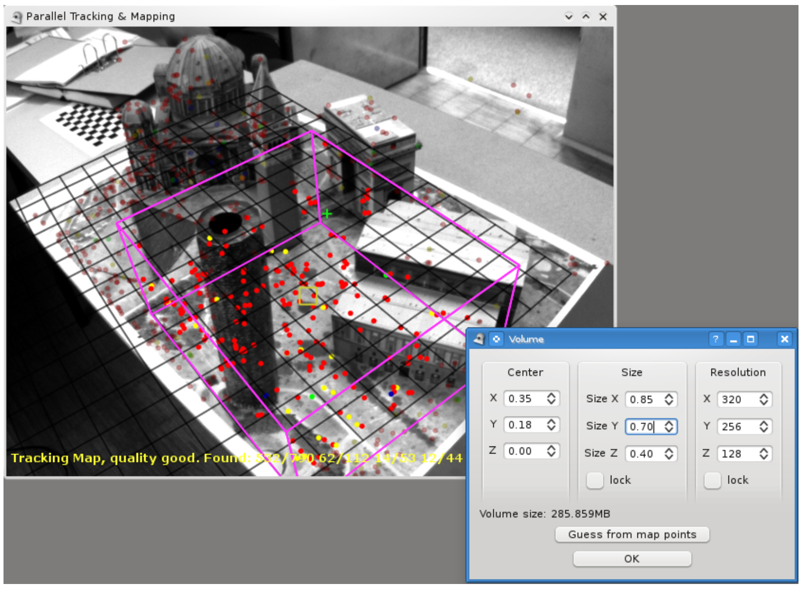
the following form is applied to the OpenGL modelview matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This transformation swaps the y- and z-axis, i.e. it transforms a right-hand system into a left-hand and vice-versa. This is essential e.g. for exporting the virtual camera pose from the raycaster to PTAM to find the closest keyframes for texture mapping.

3.5 Reconstruction System

Here we illustrate the whole reconstruction procedure and how the individual parts described in the previous chapters work together. The system is started by the user initializing the tracking system. An automation or upgrading to metric reconstruction by placing an artificial marker with known dimensions in the scene is postponed for future work. When PTAM is tracking successfully, the volume of interest needs to be specified. To this end, we provide a simple GUI where the user can change position, size and resolution of the volume. Graphical feedback is given directly through the live camera image. Because moving a 3D volume in a 2D image can be cumbersome, we provide additional cues: PTAM feature points inside the volume are rendered fully solid, while features outside are rendered opaque. This way, the user can better judge where the volume boundaries are



Tracking Map, quality good. Found: 532/790 62/112 14/53 12/44

Center	Size	Resolution
X 0.35	Size X 0.85	X 320
Y 0.18	Size Y 0.70	Y 256
Z 0.00	Size Z 0.40	Z 128
	lock	lock

Volume size: 285.859MB

Figure 3.5: GUI-dialog for setting up the volume. To aid the process, feature points inside the volume are rendered fully solid, while features outside are rendered opaque.

and what part of the scene is actually captured by the volume (see figure 3.5).

After setting up the volume, the reconstruction process is fully automated: PTAM tracks the scene and generates keyframes. For every keyframe, we decide whether it is going to be used for reconstruction or not (see section 2.3.1), and eventually notify the reconstruction thread. Communication between the tracking and the reconstruction thread is done through a queue (see figure 3.6). This way, interaction between the two threads is

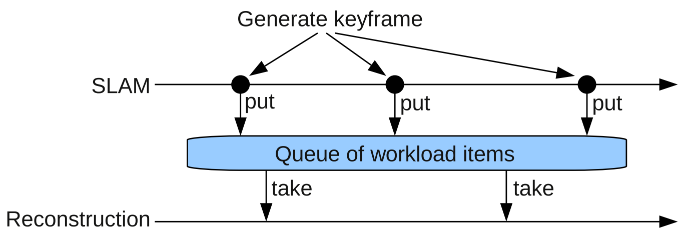


Figure 3.6: Communication of the tracking and reconstruction threads through a queue of workload items.

minimized and tracking is not disturbed by the reconstruction. We provide graphical feedback on the queue length through the GUI. In case PTAM generates keyframes at a much higher rate than they are processed by the reconstruction thread, the user can react e.g. by not moving the camera, so no further keyframes are generated and the reconstruction thread can catch up. However, in our experiments this rarely happened, as the workload items are processed quite quickly and the heuristic responsible for dropping keyframes requires a minimum number of 25 frames (i.e. approximately one second) between keyframe drops.

The minimization algorithm runs continuously in the background triggered by a timer. Ideally one would update the visualization every time the solution changes (i.e. every time an iteration of the minimization algorithm has completed), however this would drastically increase the computational load. For providing an interactive "online" experience, it suffices to update the visualization roughly every second. Hence we use a second timer to trigger visualization updates. Both timing intervals are controllable and can be set to the users preference.

Chapter 4

Evaluation

Contents

Outline: *Here we investigate the influence of various system parameters on the reconstruction result. After explaining the evaluation setup, we look into the different depthmap generation methods. Next, the parameters of the fusion algorithm are investigated, before we finally examine the system performance by providing timings of a real world example.*

4.1 Evaluation Setup

To get consistent and comparable results, it is necessary to conduct the same camera movement every time the system is run. Because this is complicated to achieve by hand, the system is fed from a prerecorded video. PTAM initialization is done at frame-accuracy by specifying start- and end-frame for the initialization algorithm. This ensures that the world coordinate frame (which is determined by the initialization, see section 1.4.4) is the same for every run. Hence the resulting geometry can easily be compared. Two different test videos were created and used throughout the experiments. Both videos are set in an indoor office scene, since this is the environment PTAM was created for and delivers best results.

One video is recorded from a Point Grey Dragonfly2 camera equipped with a 2.8mm wide angle lens. For a more controlled environment, the second video is created synthetically using POVRay[1], a high quality raytracer which is able to produce photo-realistic results.

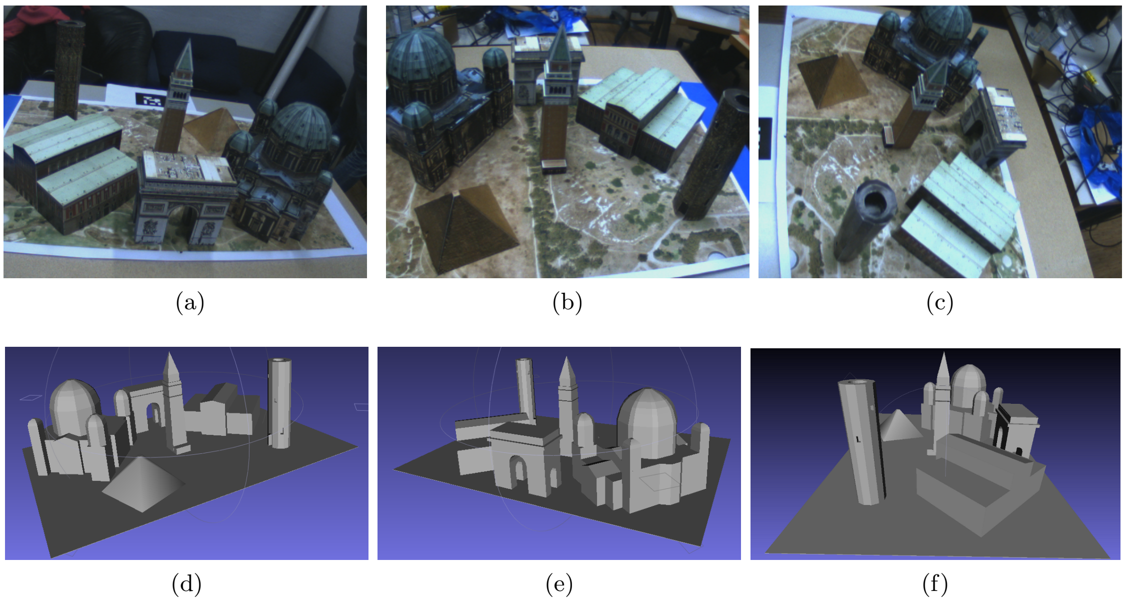
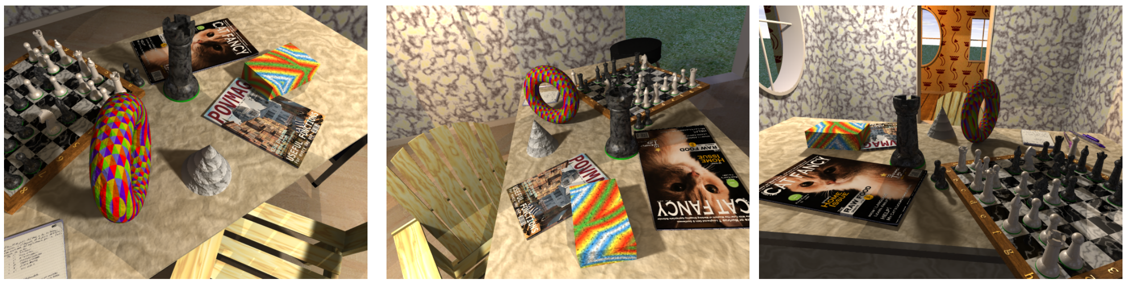


Figure 4.1: (a)-(c): Images from the City of Sights input video.
(d)-(f): renderings of the groundtruth 3D model used for comparison.

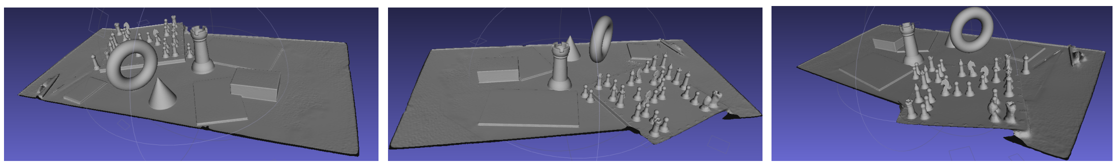
For the live video, the City of Sights model of [Gruber et al., 2010] was used. The authors provide groundtruth geometry data for comparison. The camera was moved by hand around the model, thus the video exhibits different kinds of challenges, e.g. fast and shaky motion, varying lighting conditions etc. (see figure 4.1). The camera is pre-calibrated, the camera parameters are loaded alongside the video.

The POVRay scene was created manually using freely available models for the POVRay scene description language. A camera trajectory was defined by specifying samples (i.e. camera matrices) of a real user camera motion and interpolating between those nodes using the spline subsystem of POVRay (see figure 4.3). POVRay supports the simulation of focal blur to get realistic results close to the images of a real camera lens. However, it turned out that this feature required enormous computational resources. Because the task of rendering the images of the POVRay scene was already time-limited, we opted to abandon the focal blur calculation and used the standard built-in anti-aliasing method instead. Note that the POVRay video lacks effects caused by camera motion (i.e. motion blur) due to the synthetic nature of image formation.

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



(a) (b) (c)



(d) (e) (f)

Figure 4.2: (a)-(c): Images from the POVRay input video.
(d)-(f): renderings of the groundtruth 3D model used for comparison.

Internal camera parameters are calculated from the field of view specified in POVRay. For a horizontal FOV of 75° and image dimensions of 640 × 480, the normalized[2] horizontal focal length can be calculated as

$$\frac{1}{640} \cdot \frac{320}{\tan(\frac{75}{2})} = 0.6516$$

The normalized vertical focal length then computes to 0.8688. The normalized principal point is safely assumed to be at the center of the image, i.e. at $p = (0.5, 0.5)^T$.

To create groundtruth for the POVRay video, internal data from the raytracing process is used, namely the intersection point of the ray with scene geometry. From this, the distance between camera center and geometry (i.e. depth) can be calculated for every ray/pixel, which produces a groundtruth depthmap. Pixel coordinates of the depthmap are back-projected into 3D space using the camera intrinsics and transformed into the global world coordinate frame using the known camera pose matrix. This gives a dense point cloud of groundtruth geometry data. Vertex normals are calculated and a mesh is created from the point cloud using the Poisson surface reconstruction algorithm [Kazhdan et al., 2006]. Because the point cloud is calculated from dense groundtruth-depthmaps, the resulting mesh is highly detailed (see figure 4.2).

The POVRay video consists of 3300 frames ≈ 2:11s, the City of Sights video has

[2]normalized by image dimensions

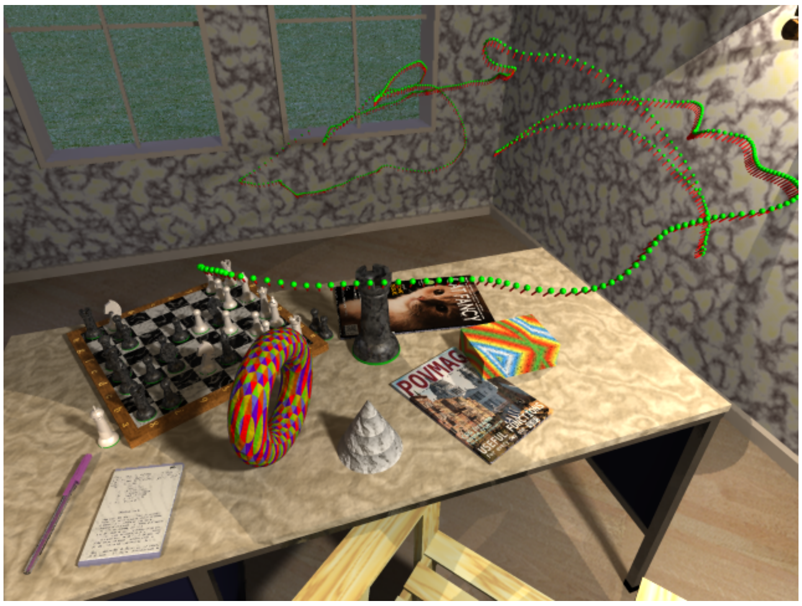


Figure 4.3: Part of the camera trajectory of the POVRay Scene. The support points for spline interpolation are taken from an actual camera movement.

5200 frames ≈ 3:28s. Experiments were carried out on a desktop PC equipped with an intel Core2Quad Q9450 Quadcore CPU running at 2.66GHz, 4GB of main memory, a nVidia Geforce GTX 480 for running CUDA calculations and a nVidia Geforce 8600GT for rendering the GUI. The computer runs a 64-bit Linux operating system.

4.2 Experimental Results

In this section we will investigate the different methods for calculating depthmaps and their parameters respectively, as well as the fusion algorithm. We will provide qualitative and quantitative results on how the individual parts of the reconstruction pipeline affect the final reconstruction result. The realtime quality of the system is apparent from the fact that the reconstruction result is available as soon as the final frame of the input video is reached. Only in a few cases some additional iterations of the fusion algorithm were necessary, e.g. because at the end of the video the solution has not been converged yet. Note that this test setup is quite challenging, because in a real use case the user might inspect the reconstruction result from time to time, resulting in periods with no camera movement. During such "spare-time" periods, both the fusion algorithm as well as the mapping component of PTAM can improve the solution. This is not the case for our video-driven test environment, where new data is arriving at a constant rate and periods without camera movement do not occur.

For all experiments we use an intersection threshold (see section 2.3.1) of $\gamma = 0.1$, and a baseline angle thresholds (see section 2.3.1) of $\beta_{\text{min}} = 0°$ and $\beta_{\text{max}} = 15°$.

4.2.1 Depthmap Generation

Planesweep As mentioned in section 2.3.2, an important parameter of the planesweep algorithm is the depth resolution. It is given by the number of planesweep steps, i.e. the number of discrete depth steps between z_{near} and z_{far}. The number of steps directly determines the depth stepsize to $d_{\text{step}} = \frac{z_{\text{far}} - z_{\text{near}}}{\#\text{steps}}$. A low depth resolution will give inferior results, while a high resolution increases the computational load. As depicted in figure

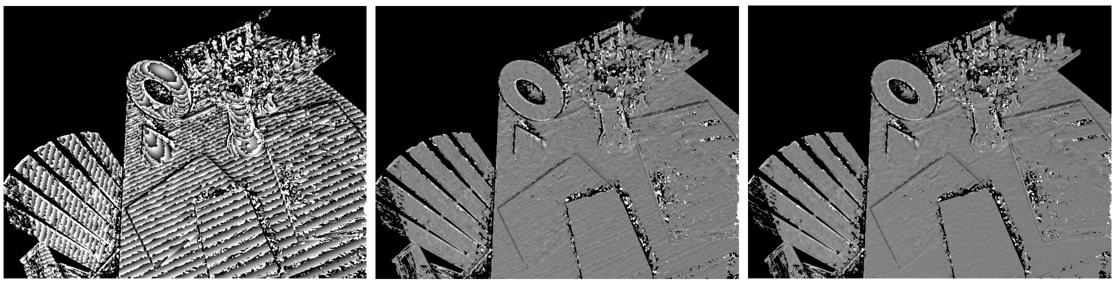
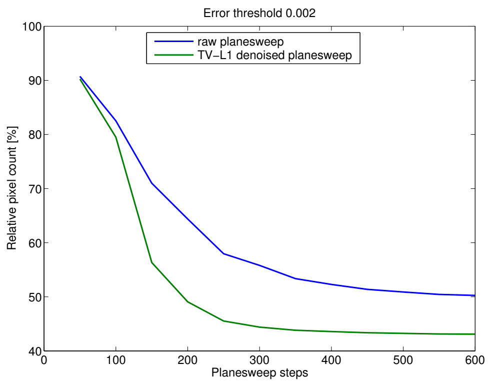

(a) 100 depth steps (b) 200 depth steps (c) 400 depth steps

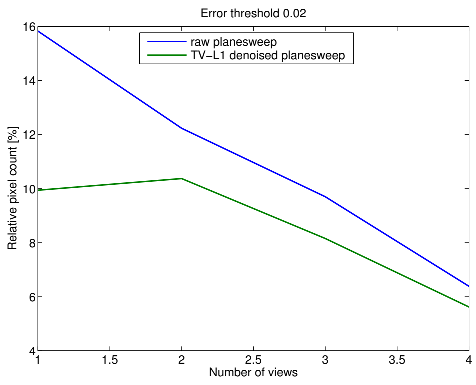
Figure 4.4: Planesweep results for different depth resolutions (4 views, 5×5 NCC). Shown are differences to groundtruth, scaled to ± 0.05 (total depth range $0.13 - 2.13$). At 100 depth steps, one can clearly see artifacts from the discrete depth sampling. These get less distinct as the depth resolution is increased and finally fade away at 400 depth steps.

4.4, a resolution of 100 depth steps yields artifacts. On the other hand, if the volume resolution is low, this might not hurt. More specifically, if the size of a voxel is bigger (i.e. very low volume resolution) than the depth stepsize d_{step}, the error from the coarse depth sampling does not hurt because it cannot be captured by the volume anyway.

Another important parameter is the number of views. More views give more information about 3D scene geometry, therefore we can expect better results when using more views. We summarize our experiments in figure 4.5. The fidelity measure used is the percentage of pixels with a depth error grater than an error threshold w.r.t. groundtruth. While the error threshold for the number of views experiment (figure 4.5b) was already set to a challenging value of 1% of the total depth range, it turned out that this threshold is



(a) Error vs. depth resolution



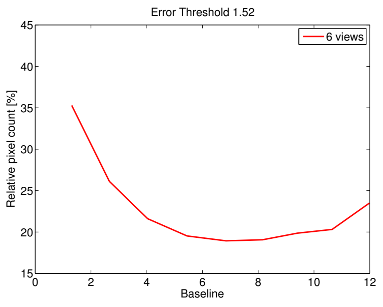
(b) Error vs. number of views

Figure 4.5: Error rates for different depth resolution and different number of views respectively. For the depth resolution experiment, the number of views was fixed to 4, for the number of views experiment, the depth resolution was fixed at 600 steps. Shown are results before (blue graph) and after (green graph) TV-ℓ_1 denoising.

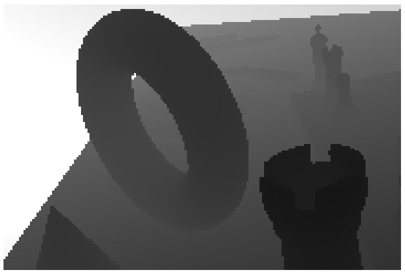
still too high for the depth resolution experiment (figure 4.5a). Here a threshold of 0.1% of the total depth range was used. One can see that increasing the depth resolution beyond 300 steps does not improve results while significantly raising the computational load. As expected, the higher the number of views, the better the results. TV-ℓ_1 denoising further improves error rates by smoothing over small patches with wrong depth information. This effect can be seen best when using a low number of views.

TGV2 Multiview Range At the core, this algorithm basically solves the same problem as optical flow algorithms. The difference is that instead of parametrizing the pixel displacement as a vector (optical flow vector), it is parametrized by the depth of the corresponding 3D point. Therefore, the method suffers from the same problems as optical flow algorithms, namely the difficulty to capture large motions i.e. baselines, and the loss of fine scale details. As described in section 2.3.3, the algorithm was implemented using a coarse-to-fine approach to account for this problem. Additionally, a simple occlusion detection algorithm was implemented, to further robustify the method. However, large baselines still is the biggest problem. This is because of the way PTAM generates keyframes. While the TGV2 algorithm would like many keyframes close to each other (small baseline), PTAM does not drop a new keyframe unless there is a minimum baseline to the closest keyframe. Unfortunately this minimum baseline is in most cases already very large for the TGV2 algorithm.

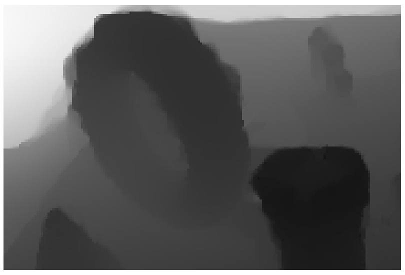
We therefore ran some experiments to gain insight on this matter by fixing the number of views and varying the baseline between the views. Figure 4.6 depicts the results of this experiment. The baseline must not be too small in order to get good depth accuracy and on the other hand must not be too large to minimize the effects of occlusions. If the baseline gets too large (see figure 4.6a), results start do get blurry and the overall error increases.



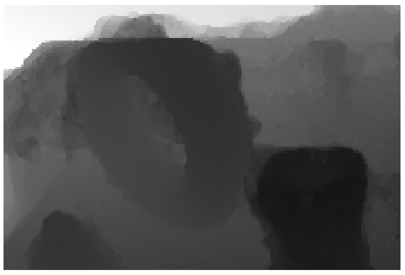
(a) Baseline vs. Accuracy



(b) Ground truth detail



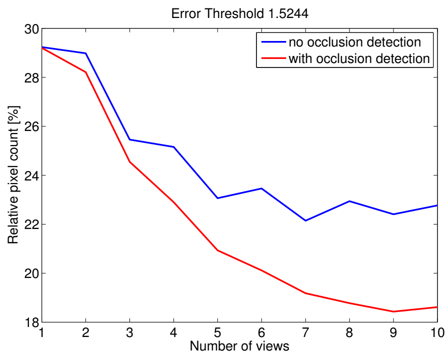
(c) 6 views, max. baseline=4



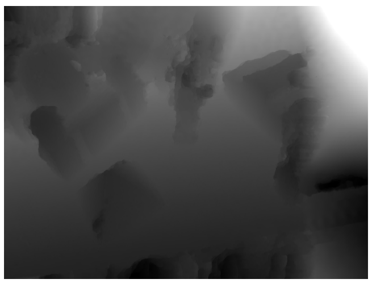
(d) 6 views, max. baseline=10.6

Figure 4.6: Influence of different baselines on depthmap accuracy. The baseline in (a) is the maximal baseline between the reference view and any other view. Note that while the overall error stays low, depthmap edges become jagged and less exact with increasing baseline.

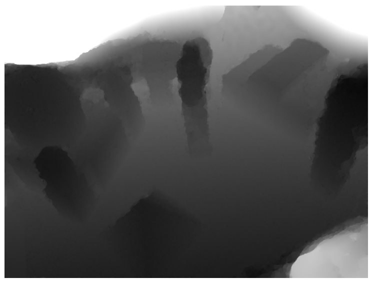
Another interesting point is how well the algorithm can deal with noisy camera matrices. For the baseline experiment we used "perfect" camera matrices from the POVRay scene, in the next experiment we will use PTAM keyframe camera matrices. These are estimates (very good ones although), and it is an interesting question how the TGV2 algorithm will perform here. Figure 4.7 depicts the results for a varying number of views. It can be seen that the algorithm performs better the more views are available, as expected. Additionally, the map uniqueness occlusion detection algorithm (see [Brown et al., 2003]), although very simple, improves results significantly (see figure 4.7a). It should be noted that while the results of figure 4.7 look quite well, the performance of the TGV2 algorithm depends critically on a "suitable" configuration of keyframes in 3D space. If this is not the case, the method produces large patches of wrong depth information due to the smoothness constraint. Such large patches are difficult to deal with for the fusion algorithm, eventually it will wrongly construct a surface from such a large patch. This is not the case for the impulse-like noise produced by the planesweep algorithm. Thus, even though the planesweep results seem to be of lower quality, the subsequent fusion can deal better with planesweep depth errors than with TGV2 depth errors.



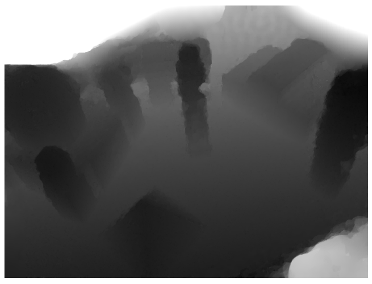
(a) Error vs. number of views



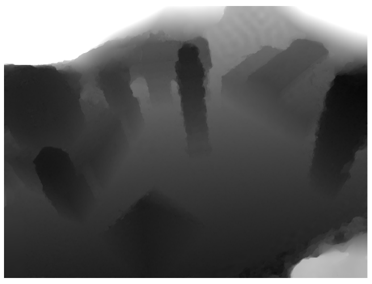
(b) 1 view



(c) 3 views



(d) 4 views



(e) 5 views

Figure 4.7: Results for the PTAM dataset. Baselines are relatively small, so results improve with increasing number of views. Note that the camera matrices of this dataset are noisy.

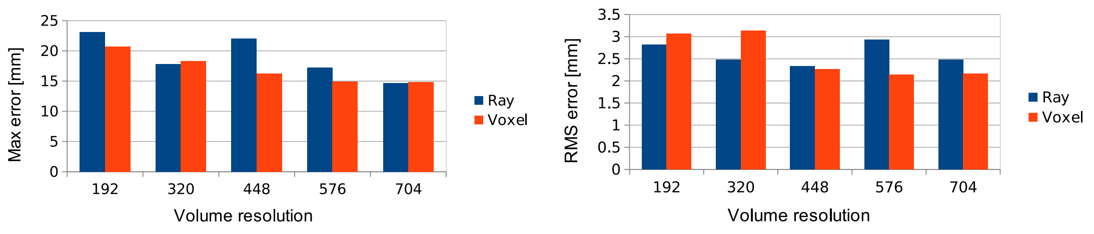
4.2.2 Depthmap Fusion

Here we will investigate the effects of ray- and voxel based histogram update (see section 3.3). The system is setup as follows: Apart from the histogram update method and the voxel resolution, the program parameters are set fixed. We then compare the reconstruction to groundtruth using the Hausdorff distance. Voxel resolutions are computed according to the physical dimensions of the volume, which are given by $(2, 1.5, 0.8)^T$ and $[2, 1.5, 0.6]^T$ for the City of Sights and POVRay scene respectively. The ratios $\frac{x}{y}$ and $\frac{x}{z}$ are used to calculate voxel resolutions such that voxel sizes are isotropic.

Table 4.1: Error rates for ray- and voxel-based histogram update

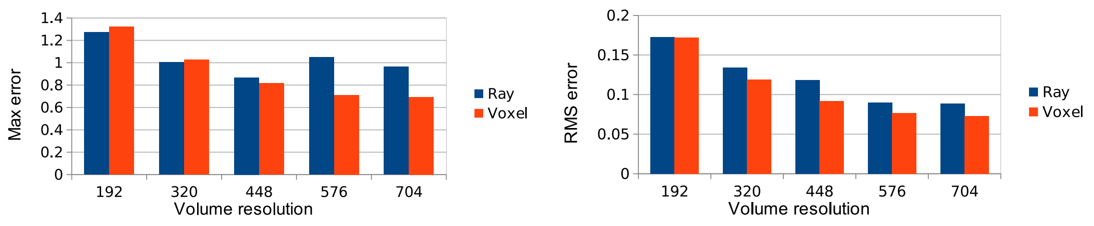
		Ray-based update		Voxel-based update	
	Volume resolution	max	RMS	max	RMS
City of Sights	$192 \times 144 \times 80$	23.16	2.82	20.75	3.01
	$320 \times 240 \times 128$	17.86	2.47	18.29	3.13
	$448 \times 336 \times 176$	22.03	2.33	16.27	2.26
	$576 \times 432 \times 224$	17.24	2.93	14.91	2.14
	$704 \times 528 \times 288$	14.65	2.48	14.88	2.16
POVRay	$192 \times 144 \times 64$	1.27	0.17	1.32	0.17
	$320 \times 240 \times 96$	1.01	0.13	1.03	0.12
	$448 \times 336 \times 128$	0.87	0.12	0.82	0.09
	$576 \times 432 \times 176$	1.05	0.09	0.71	0.07
	$704 \times 528 \times 208$	0.96	0.09	0.69	0.07

Results are shown in table 4.1 and depicted in figure 4.8.



(a) max error (b) RMS error

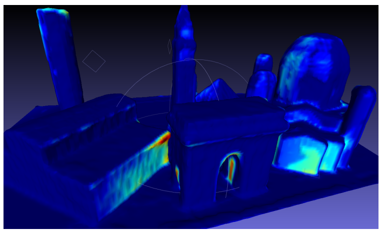
Figure 4.8: City of Sights: RMS and maximum error for ray- and voxel-based histogram update at different volume resolutions.

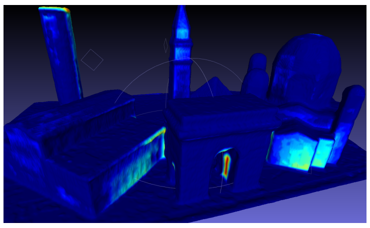


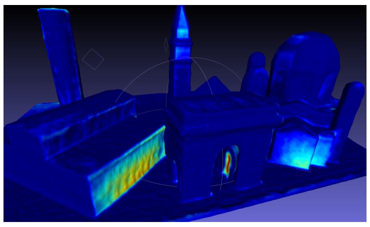
(a) max error (b) RMS error

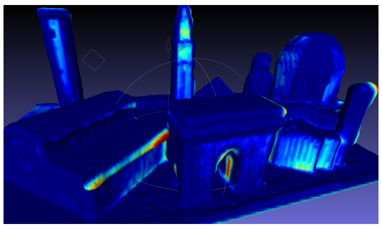
Figure 4.9: POVRay: RMS and maximum error for ray- and voxel-based histogram update at different volume resolutions.

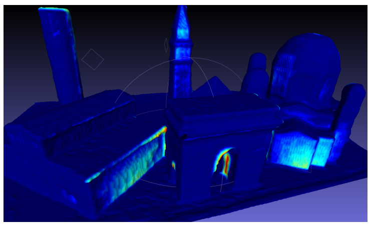
Note that the dataterm for voxel-based update is stronger, because much more voxels are changed for each depthmap than with ray-based update. For this reason, different

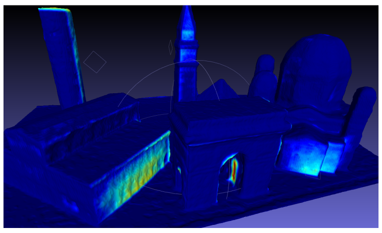

(a) Ray, $192 \times 144 \times 80$

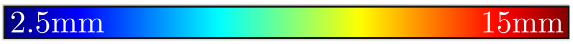

(b) Ray, $448 \times 336 \times 176$


(c) Ray, $704 \times 528 \times 288$


(d) Voxel, $192 \times 144 \times 80$


(e) Voxel, $448 \times 336 \times 176$


(f) Voxel, $704 \times 528 \times 288$



(g) Color coding

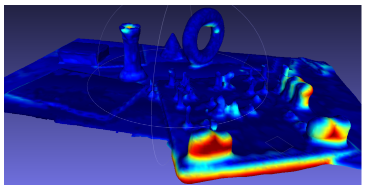
Figure 4.10: Color-coded error images of the reconstruction for ray- and voxel-based histogram update at different volume resolutions. The color coding is shown in (g). Note that according to [Gruber et al., 2010], the typical construction accuracy of the model is 3mm.

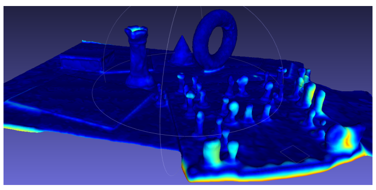
settings for the data fidelity parameter λ were used. The values for λ were tuned by hand to give the best results for each method respectively.

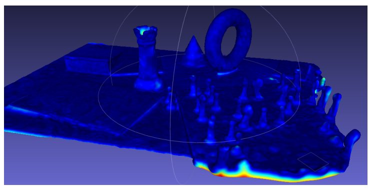
While ray-based histogram update performs well at low volume resolutions, voxel-based update wins at high resolutions. An explanation for this is that voxel-based update better scales with the volume resolution. Every voxel is projected into the reference frame and updated, whereas ray-based update might miss voxels between rays (see section 3.3). The weaker dataterm of ray-based update also means that the maximum error is higher (see figure 4.8a), i.e. the reconstruction contains more outliers. This can be seen in figure 4.10, where color-coded images of the reconstruction are depicted. The color encodes the vertex error w.r.t. groundtruth (see figure 4.10g). According to [Gruber et al., 2010], the construction/placement accuracy of the paper model is within $\pm 3mm$. Therefore the color coding does not start at an error of zero but rather at $2.5mm$.

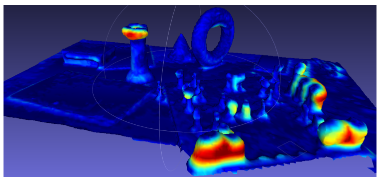
Similar results are found for the POVRay scene, see figure 4.9. Due to the synthetic nature of image formation, the unit of error can not be connected to a physical meaningful entity, it represents the internal world coordinate system used by POVRay.

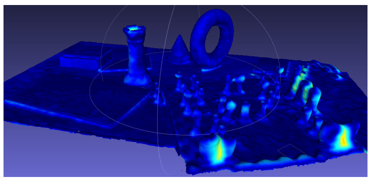
For the following experiments the volume resolution was fixed at $448 \times 336 \times 208$ for

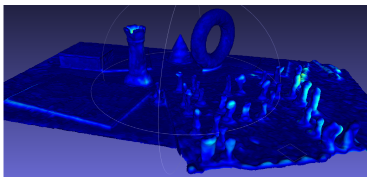

(a) Ray, 192 × 144 × 64

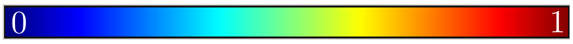

(b) Ray, 448 × 336 × 128


(c) Ray, 704 × 528 × 208


(d) Voxel, 192 × 144 × 64


(e) Voxel, 448 × 336 × 128


(f) Voxel, 704 × 528 × 208



(g) Color coding

Figure 4.11: Color-coded error images of the reconstruction for ray- and voxel-based histogram update at different volume resolutions.

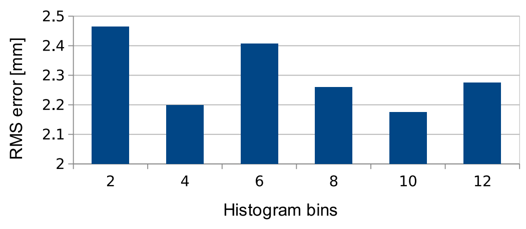
the City of Sights scene and 448 × 336 × 176 for the POVRay scene respectively. The voxel-based histogram update method is used, since it gives generally better results.

4.2.3 Realtime 3D Reconstruction

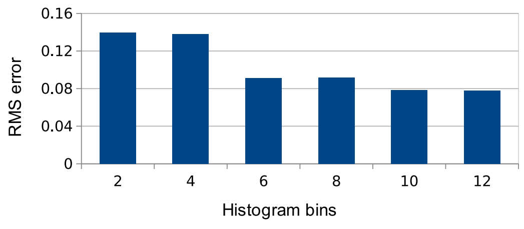
In this section we investigate the impact of the following system parameters on the reconstruction result.

- Number of histogram bins
- Depthmap generation method
- Keyframe calculation delay

Number of Histogram Bins Figure 4.12 depicts the RMS error for different number of histogram bins. Interestingly, increasing the number of bins does not improve results substantially. It seems that the differences in RMS error depicted in figure 4.12 are already due to measurement noise. Further in-depth investigations on this matter are required, open questions include e.g. how to change the slope of the TSDF with varying number of histogram bins. The results further support the motivation of the histogram compression (see section 2.4) scheme: In order to reconstruct the zero level set of the TSDF, it is not

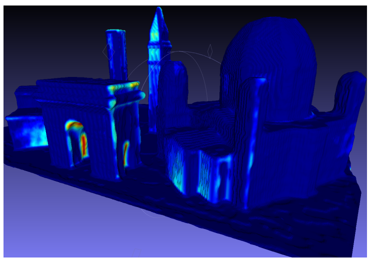


(a) RMS error City of Sights

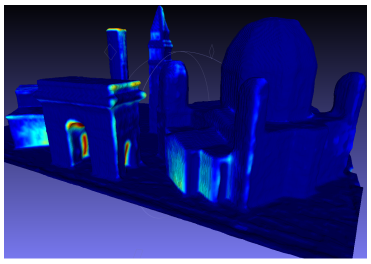


(b) RMS error POVRay

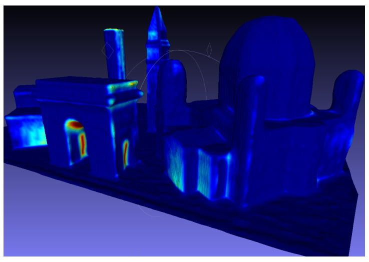
Figure 4.12: RMS Error for different number of histogram bins. Besides having an impact on memory footprint and calculation performance, it seems that more than 6-8 bins are not necessary.



(a) 2 bins



(b) 8 bins



(c) 12 bins

Figure 4.13: Color-coded error images of the reconstruction for different number of histogram bins. Note the blocky artifacts in (a) due to the extreme low number of bins. The color coding is the same as in figure 4.10.

necessary to store the exact value, it suffices to sample the TSDF at discrete positions. Note that the RMS error is not necessarily a good measure of the perceived reconstruction quality, as can be seen in figure 4.13. When using 2 bins only, the reconstruction exhibits blocky artifacts. These vanish when using more bins. Taking into account both the RMS error and the color coded difference image, it seems that a number of 6-8 histogram bins is a good tradeoff between speed and reconstruction accuracy.

Depthmap Generation Method Next, we test the different methods for depthmap generation. The volume resolution is fixed as in the previous paragraph and we use a number of 8 histogram bins throughout the experiments. Parameters are set as shown in table 4.2. When using planesweep, we additionally run the system with and without TV-ℓ_1 depthmap denoising (see section 2.3.2).

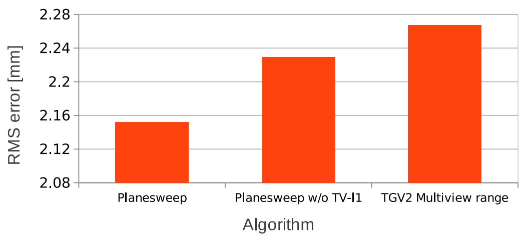
Results are shown in table 4.2 and depicted in figure 4.14. Planesweep TV-ℓ_1 denoising helps lowering the RMS error. As in the previous paragraph, we also look at the color

Table 4.2: Parameters for planesweep and TGV2 multiview range

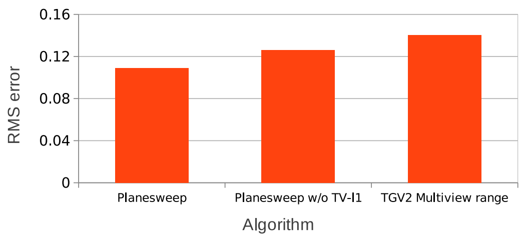
Planesweep	TGV2 Multiview Range
#depthsteps = 250	$\lambda = 10$
NCC windowsize = 5×5	#warps = 17
TV-ℓ_1 $\lambda = 0.3$	#iterations = 30
TV-ℓ_1 iterations = 100	Scalefactor = 0.9
TV-ℓ_1 edge weight $\alpha = 20$	Occlusion detection = On

Table 4.3: Error rates for Planesweep and TGV2 depthmap generation

		Error	
Scene	Method	max	RMS
City of Sights	Planesweep	18.71	2.15
	Planesweep without TV-ℓ_1	15.17	2.23
	TGV2 Multiview Range	28.25	2.27
POVRay	Planesweep	0.88	0.10
	Planesweep without TV-ℓ_1	0.97	0.126
	TGV2 Multiview Range	1.12	0.14



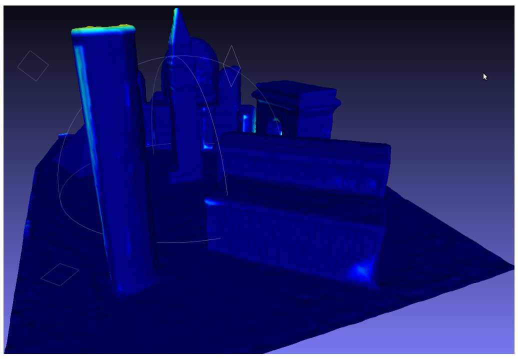
(a) City of Sights

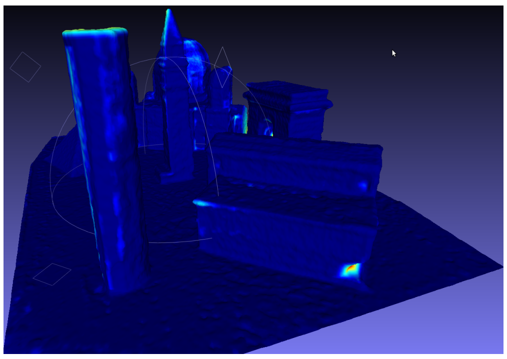


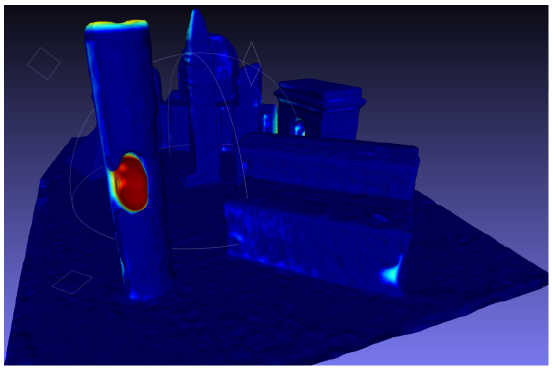
(b) POVRay

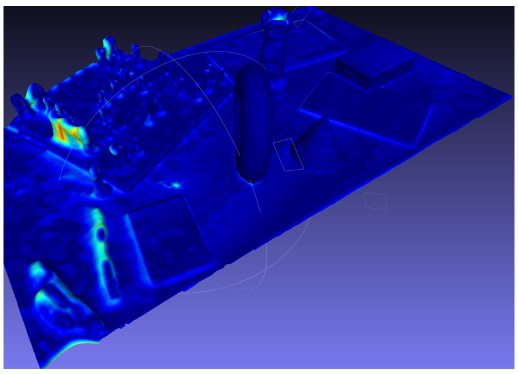
Figure 4.14: RMS and maximum error for different depthmap generation algorithms.

coded difference image of the reconstruction. It turns out that TV-ℓ_1 denoising helps in terms of surface completeness, as can be seen in figure 4.15. The surface of figure 4.15a is more complete than the surface of figure 4.15b, which means that depthmap smoothing indeed is an important step. Similar results are found for the POVRay scene. TGV2 multiview range performs well for large surfaces, e.g. the front of the Berlin Dome or the ground plane, but sometimes cannot fully recover geometry such as the tower in figure 4.15c.


(a) Planesweep


(b) Planesweep without TV-ℓ_1 denoising


(c) TGV2 Multiview Range


(d) Planesweep

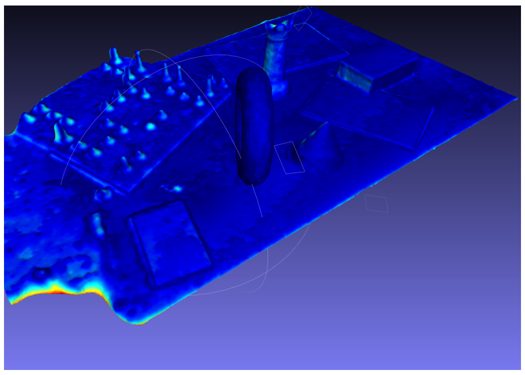
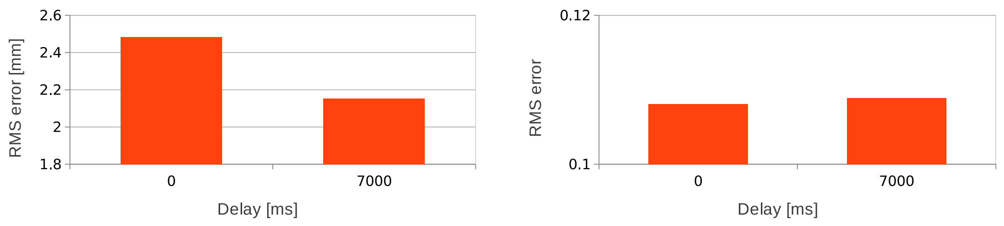

(e) TGV2 Multiview Range

Figure 4.15: Color coded difference images for different depthmap generation algorithms.

Keyframe Calculation Delay After keyframes are generated by PTAM and before they are tested for reconstruction usage (see section 2.3.1), there is a user defined delay. The reasoning behind this is to give the mapping component of PTAM some time to perform local bundle adjustment on the newly created keyframe (see section 1.4.4), thus improving the accuracy of the keyframe camera matrix. The accuracy of the camera matrix is of critical importance for the depthmap generation stage (see section 2.3). Additionally, applying a delay makes it possible that "future" keyframes are used as sensor view for the current keyframe. As an example, assuming increasing keyframe numbers it is possible for the reference keyframe with id 20 to use keyframes 21 and 23 as sensor views if the calculation is delayed. Thus, potentially more views are available for a given reference view, improving depthmap quality. For this experiment, volume resolution and number of histogram bins are set as in the previous paragraph, planesweep with TV-ℓ_1 denoising is used as depthmap generation method. Results are shown in table 4.4 and depicted in

Table 4.4: Error rates for different keyframe calculation delays

		Error	
	Delay [ms]	max	RMS
City of Sights	0	17.74	2.48
	7000	18.7	2.15
POVRay	0	0.88	0.11
	7000	0.88	0.11

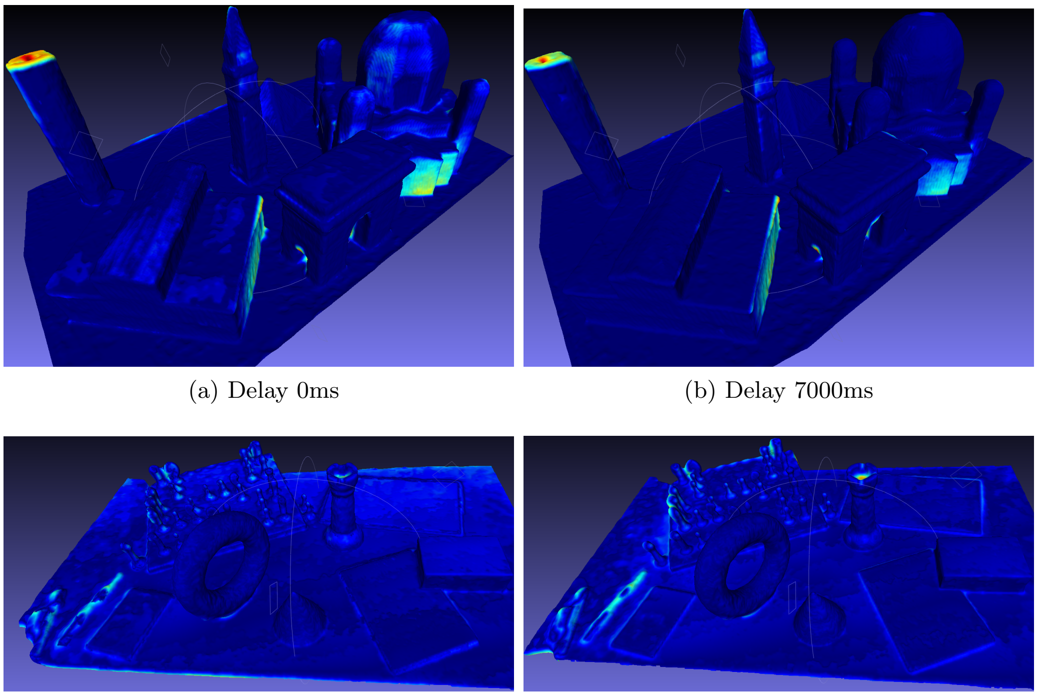


(a) City of Sights (b) POVRay

Figure 4.16: RMS and maximum error for different keyframe calculation delays. Delaying the calculation gives improvements in reconstruction accuracy.

figure 4.16. Delaying the calculation results in a small but consistent improvement of the reconstruction quality. Similar results are found for the POVRay scene, see figure 4.17c and 4.17d.

Full System and Performance In this paragraph we show an exemplary run of the system and provide intermediate results. This highlights the realtime quality of the application, the reconstruction literally is on-the-fly, with minimal delay between the camera seeing the object and the corresponding reconstruction. We also provide performance



(a) Delay 0ms (b) Delay 7000ms

(c) Delay 0ms (d) Delay 7000ms

Figure 4.17: Color coded difference images for different keyframe calculation delays. Because potentially more sensor views are available when delaying depthmap generation, the reconstruction result is better.

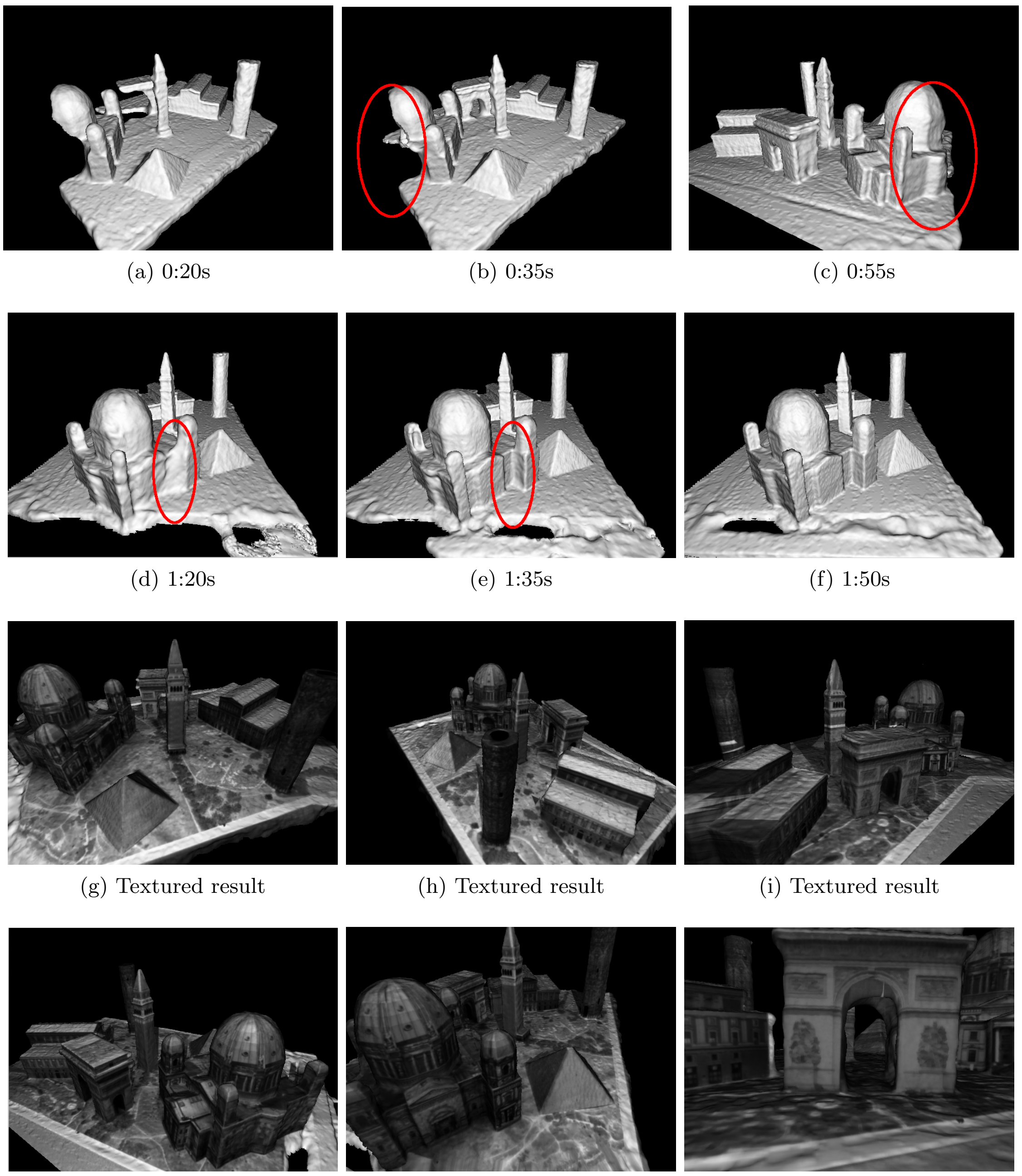
timings of the individual parts of the reconstruction pipeline. System parameters are set as shown in table 4.5. Note that although the list of parameters consists of more than 16 entries, the only important parameters that have to be specified by the user are the volume resolution and the fusion data fidelity parameter λ. All other parameters can be left at their default value, which works for almost any scene. Table 4.6 shows performance timings. One can see that calculating depthmaps ist the most expensive operation, taking slightly less than a second in the worst case of 4 views. This means that new information is fully integrated into the reconstruction result in about 1 seconds time, resulting in interactive user experience. Figure 4.18 shows some results of the system, including timestamps. As long as the camera has not seen a specific part of the scene, obviously there is no reconstruction. With the camera exploring, missing parts are added within seconds of time, see figure 4.18b, 4.18c and figure 4.18d, 4.18e. The texture mapping algorithm (see section 2.5) works nicely, although some artifacts can be seen, e.g. in figure 4.18i.

Table 4.5: Parameters for full system evaluation

Parameter	Value
Scene	City of Sights, 3467 frames, $\approx$ 2:18s
Volume resolution	$448 \times 336 \times 160$
Number of histogram bins	8
Histogram update method	Voxel-based
Keyframe calculation delay	7000ms
Depthmap algorithm	Planesweep
Planesweep depth steps	250
Planesweep NCC windowsize	5×5
Planesweep TV-ℓ_1 λ	0.3
Planesweep TV-ℓ_1 iterations	100
Planesweep TV-ℓ_1 edge weight α	20
Depthmap fusion λ	0.03

Table 4.6: System performance

Operation	Time [ms]
Planesweep 1 view	245
Planesweep 2 views	400
Planesweep 3 views	600
Planesweep 4 views	815
Histogram update (voxel-based)	15
Depthmap fusion (1 iteration)	80
Raycasting (point sampling)	1
Raycasting (trilinear interpolation)	5
Raycasting (trilinear interpolation and texture mapping)	6
Total (worst case)	916



(a) 0:20s (b) 0:35s (c) 0:55s

(d) 1:20s (e) 1:35s (f) 1:50s

(g) Textured result (h) Textured result (i) Textured result

(j) Textured result (k) Textured result (l) Textured result

Figure 4.18: Results of the system. Note that at 0:35s large parts of the scene are already reconstructed. Missing parts are due to the camera not (yet) having scene that part of the scene. The images of the textured result are taken at 2:00s.

Chapter 5

Conclusion

Contents

5.1 Summary

In this master's thesis an application for interactively reconstructing dense geometry from a single moving camera was developed. It has bee shown that it is possible to reconstruct arbitrary geometry on-the-fly, simply by pointing a camera at the object of interest. The method builds on current state of the art results from various research areas of computer vision, and adapts and combines them in a novel way to yield the desired system.

We used PTAM, a high quality realtime camera tracking method, as starting point for our work. An overview of PTAM was given alongside other related work in section 1.4.

The methodology of the realtime reconstruction system is presented in chapter 2. Section 2.3 is devoted to the generation of depthmaps, where two different approaches were discussed. Both methods make heavy use of the parallel computation capabilities of todays high performance graphics cards to meet the realtime requirement.

The robust fusion of depthmaps is addressed in section 2.4. We used a volumetric approach where the surface is represented implicitly by the zero level-set of a truncated signed distance function. Fusion is based on a variational formulation, and we showed how the resulting convex energy functional can be solved in a global optimal way on the GPU.

In section 2.5, the approach for visualizing results is described. Again, a GPU-based raycaster is used to achieve realtime performance.

Chapter 3 gives various implementation details, with special emphasis on the different methods how to update the volume histogram in section 3.3. Performance considerations are explained and we showed how the GPU is used to accelerate the different parts of the reconstruction pipeline.

Finally, a comprehensive system evaluation was done in chapter 4. The different parts of the reconstruction pipeline were analyzed individually, before the whole system was evaluated.

We want to point out that the evaluation results should be understood as performance hints only. The important point of an interactive system is after all to have a "human in the loop". Reconstruction results are highly dependent on the user moving the camera, user decisions about camera movement in turn depend on the constant stream of feedback the user gets from the system. A static evaluation as has been done in this work can not capture such a feedback loop.

5.2 Outlook

The realtime reconstruction system is limited by various factors. One of the most important is the tracking system, if tracking fails all subsequent parts of the reconstruction pipeline can not be done. Therefore, improving tracking performance is the most obvious starting point for future enhancements. The loose coupling between tracking and reconstruction makes this task easy. Our system does not make any assumptions about the tracking system, thus the tracker can easily be replaced by more powerful alternatives in the future.

Another limitation is the volumetric representation of geometry, which requires large amounts of memory. Even high end GPUs are restricted to the reconstruction of relatively small indoor scenes and objects. To enable the dense reconstruction of large (outdoor) environments, new methods and algorithms need to be developed.

The reconstruction quality of our system critically depends on the quality of the input depthmaps. While there exist many state of the art high quality stereo algorithms, most of these are not suited for realtime use. The planesweep algorithm used in this thesis is fast, but fails in untextured regions. Reconstruction of untextured scenes is thus not possible with our system. The TGV2 mutiview range algorithm to some extent overcomes this problem, but on the other hand looses fine details and is not well suited for large baselines. Here, several improvements can be made. It seems that instead of trying to get as much information as possible from very few views with sophisticated methods, it is better to simply use more input data and relatively simple and fast algorithms. Using a live camera, one has a tremendous amount of data (30 frames per second), which potentially can be used. It will be interesting to develop fast, realtime-capable algorithms that make use of this data.

Appendix A

Acronyms and Symbols

List of Acronyms

AR	Augmented Reality
CAS	Compare-and-Swap
CPU	Central Processing Unit
DOF	Degrees of Freedom
DSI	Disparity Space Image
DTAM	Dense Tracking and Mapping
FAST	Features from accelerated segment test
FLOPS	Floating-point Operations per Second
FOV	Field of View
GPU	Graphics Processing Unit
GUI	Graphical User Interface
NCC	Normalized Cross Correlation
PDE	Partial Differential Equation
PnP	Perspective n-Point Problem
PTAM	Parallel Tracking and Mapping
RANSAC	Random Sample and Consensus
SAD	Sum of absolute differences
(T)SDF	(Truncated) Signed Distance Function
SfM	Structure from Motion
(V)SLAM	(Visual) Simultaneous Localization and Mapping
SSD	Sum of squared differences

T(G)V	Total (Generalized) Variation
WTA	Winner Takes All

Bibliography

[Banno and Ikeuchi, 2009] Banno, A. and Ikeuchi, K. (2009). Disparity map refinement and 3d surface smoothing via directed anisotropic diffusion. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1870 –1877.

[Bleyer et al., 2011] Bleyer, M., Rehmann, C., and Rother, C. (2011). Patchmatch stereo - stereo matching with slanted support windows. In *British Machine Vision Conference (BMVC)*, Dundee, Scotland.

[Boykov et al., 2001] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23:1222–1239.

[Bredies et al., 2010] Bredies, K., Kunisch, K., and Pock, T. (2010). Total generalized variation. *SIAM J. Imaging Sciences*, 3(3):492–526.

[Brown et al., 2003] Brown, M. Z., Burschka, D., and Hager, G. D. (2003). Advances in computational stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25:993–1008.

[Chambolle and Pock, 2011] Chambolle, A. and Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging Vis.*, 40:120–145.

[Collins, 1996] Collins, R. (1996). A space-sweep approach to true multi-image matching. In *IEEE Computer Vision and Pattern Recognition*, pages 358–363.

[Cornells and Van Gool, 2005] Cornells, N. and Van Gool, L. (2005). Real-time connectivity constrained depth map computation using programmable graphics hardware. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 1099 – 1104 vol. 1.

[Csorba, 1997] Csorba, M. (1997). *Simultaneous localisation and map building*. Phd, University of Oxford.

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA. ACM.

[Davison, 2003] Davison, A. J. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1403–, Washington, DC, USA. IEEE Computer Society.

[Davison et al., 2007] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29:1052–1067.

[Devernay and Faugeras, 2001] Devernay, F. and Faugeras, O. (2001). Straight lines have to be straight: automatic calibration and removal of distortion from scenes of structured enviroments. *Machine Vision and Applications*, 13:14–24.

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE Robotics and Automation Magazine*, 2:2006.

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

[Fusiello et al., 2000] Fusiello, A., Trucco, E., and Verri, A. (2000). A compact algorithm for rectification of stereo pairs. *Mach. Vision Appl.*, 12:16–22.

[Gruber et al., 2010] Gruber, L., Gauglitz, S., Ventura, J., Zollmann, S., Huber, M., Schlegel, M., Klinker, G., Schmalstieg, D., and Höllerer, T. (2010). The city of sights: Design, construction, and measurement of an augmented reality stage set. In *Proc. Nineth IEEE International Symposium on Mixed and Augmented Reality (ISMAR'10)*, pages 157–163, Seoul, Korea.

[Guivant and Nebot, 2001] Guivant, J. E. and Nebot, E. M. (2001). Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE Transactions on Robotics and Automation*, 17(3):242 –257.

[Hadwiger et al., 2005] Hadwiger, M., Sigg, C., Scharsach, H., Bühler, K., and Gross, M. H. (2005). Real-time ray-casting and advanced shading of discrete isosurfaces. *Comput. Graph. Forum*, 24(3):303–312.

[Hartley, 1995] Hartley, R. (1995). In defence of the 8-point algorithm. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 1064 –1070.

[Hartley and Sturm, 1997] Hartley, R. and Sturm, P. (1997). Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157.

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

[Horaud et al., 1989] Horaud, R. P., Conio, B., Leboulleux, O., and Lacolle, B. (1989). An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics and Image Processing*, 47:33–44.

[Kazhdan et al., 2006] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Klein and Murray, 2007] Klein, G. and Murray, D. (2007). Parallel Tracking and Mapping for Small AR Workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10.

[Klein and Murray, 2008] Klein, G. and Murray, D. (2008). Improving the agility of keyframe-based SLAM. In *Proc. 10th European Conference on Computer Vision (ECCV'08)*, pages 802–815, Marseille.

[Koch et al., 2000] Koch, R., Pollefeys, M., and Gool, L. J. V. (2000). Realistic surface reconstruction of 3d scenes from uncalibrated image sequences. *Journal of Visualization and Computer Animation*, 11(3):115–127.

[Kolmogorov and Zabih, 2001] Kolmogorov, V. and Zabih, R. (2001). Computing visual correspondence with occlusions via graph cuts. In *In International Conference on Computer Vision*, pages 508–515.

[Kolmogorov and Zabih, 2002] Kolmogorov, V. and Zabih, R. (2002). Multi-camera scene reconstruction via graph cuts. In *Proceedings of the 7th European Conference on Computer Vision-Part III*, ECCV '02, pages 82–96, London, UK, UK. Springer-Verlag.

[Kutulakos and Seitz, 2000] Kutulakos, K. N. and Seitz, S. M. (2000). A theory of shape by space carving. *Int. J. Comput. Vision*, 38:199–218.

[Laurentini, 1994] Laurentini, A. (1994). The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16:150–162.

[Leonard et al., 1999] Leonard, J. J., Jacob, H., and Feder, S. (1999). A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Ninth International Symposium on Robotics Research*, pages 169–176. Springer-Verlag.

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

[Li and Osher, 2009] Li, Y. and Osher, S. (2009). A new median formula with applications to pde based denoising. *Communications in Mathematical Sciences*, 7(3):741–753.

[Martin and Aggarwal, 1983] Martin, W. N. and Aggarwal, J. K. (1983). Volumetric descriptions of objects from multiple views. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5(2):150–158.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI.

[Moreno-Noguer et al., 2007] Moreno-Noguer, F., Lepetit, V., and Fua, P. (2007). Accurate non-iterative o(n) solution to the pnp problem. In *ICCV*, pages 1–8.

[Newcombe et al., 2011a] Newcombe, R., Lovegrove, S., and Davison, A. (2011a). Dtam: Dense tracking and mapping in real-time. In *Proc. of the Intl. Conf. on Computer Vision (ICCV), Barcelona, Spain*, volume 1.

[Newcombe and Davison, 2010] Newcombe, R. A. and Davison, A. J. (2010). Live dense reconstruction with a single moving camera. In *IEEE Conference on Computer Vision and pattern Recognition.*

[Newcombe et al., 2011b] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. W. (2011b). Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, pages 127–136.

[Okutomi and Kanade, 1993] Okutomi, M. and Kanade, T. (1993). A multiple-baseline stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15:353–363.

[P. A. Beardsley and Murray, 1994] P. A. Beardsley, A. Z. and Murray, D. W. (1994). Navigation using affine structure from motion. In *Proc 3rd European Conf on Computer Vision, Stockholm*, Lecture Notes in Computer Science, pages 85–96. Springer.

[Pock et al., 2008] Pock, T., Schoenemann, T., Graber, G., Bischof, H., and Cremers, D. (2008). A convex formulation of continuous multi-label problems. In *European Conference on Computer Vision (ECCV)*, Marseille, France.

[Pollefeys et al., 1999] Pollefeys, M., Koch, R., and Gool, L. V. (1999). Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. *Int. J. Comput. Vision*, 32:7–25.

[Quan and Lan, 1999] Quan, L. and Lan, Z. (1999). Linear n-point camera pose determination. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21:774–780.

[Rhemann et al., 2011] Rhemann, C., Hosni, A., Bleyer, M., Rother, C., and Gelautz, M. (2011). Fast cost-volume filtering for visual correspondence and beyond. In *IEEE Computer Vision and Pattern Recognition (CVPR)*.

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

[Schweighofer and Pinz, 2008] Schweighofer, G. and Pinz, A. (2008). Globally optimal o(n) solution to the pnp problem for general camera models. In *BMVC*.

[Seitz et al., 2006] Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, pages 519–528, Washington, DC, USA. IEEE Computer Society.

[Seitz and Dyer, 1997] Seitz, S. M. and Dyer, C. R. (1997). Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1067–, Washington, DC, USA. IEEE Computer Society.

[Slabaugh et al., 2003] Slabaugh, G. G., Culbertson, W. B., Malzbender, T., Stevens, M. R., and Schafer, R. W. (2003). Methods for volumetric reconstruction of visual scenes. *International Journal of Computer Vision*, 57:179–199.

[Smith et al., 1990] Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In Cox, I. J. and Wilfong, G. T., editors, *Autonomous Robot Vehicles*, volume 8, pages 167–193. Springer-Verlag New York, Inc.

[Steinberger, 2010] Steinberger, M. (2010). Highly accurate multiresolution isosurface rendering using compactly supported spline wavelets. Master's thesis, Graz University of Technology.

[Strecha et al., 2003] Strecha, C., Tuytelaars, T., and Gool, L. J. V. (2003). Dense matching of multiple wide-baseline views. In *ICCV*, pages 1194–1201. IEEE Computer Society.

[Stuehmer et al., 2010] Stuehmer, J., Gumhold, S., and Cremers, D. (2010). Real-time dense geometry from a handheld camera. In *Pattern Recognition (Proc. DAGM)*, pages 11–20, Darmstadt, Germany.

[Tsai, 1987] Tsai, R. Y. (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344.

[Weng et al., 1992] Weng, J., Cohen, P., and Herniou, M. (1992). Camera calibration with distortion models and accuracy evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:965–980.

[Zach, 2008] Zach, C. (2008). Fast and high quality fusion of depth maps. *Proc. 3DPVT*.

[Zach et al., 2007] Zach, C., Pock, T., and Bischof, H. (2007). A globally optimal algorithm for robust tv-l1 range image integration. In *ICCV*, pages 1–8. IEEE.

[Zhang et al., 2008] Zhang, Y., Gong, M., and Yang, Y.-H. (2008). Local stereo matching with 3d adaptive cost aggregation for slanted surface modeling and sub-pixel accuracy.

In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1 –4.

[Zhang, 2000] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334.