Christoph Bauernhofer

Dense Reconstruction On Mobile Devices

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme

Computer Science

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Thomas Pock

Institute for Computer Graphics and Vision

Graz, Austria, Dec. 2015

Abstract

The aim of 3D reconstruction is to infer 3D geometry of the scene from a given set of 2D images. Being one of the most fundamental problems in computer vision, many algorithms have been developed in the last years to solve this problem on desktop computers. Modern mobile devices such as tablets and smartphones, however, deliver unprecedented computational power in everyone's pocket making it possible to tackle this problem on mobile platforms as well.

The aim of this master's thesis is to create the fundamental building blocks: *dense tracking* and *dense depthmap* computation, for a novel reconstruction system on mobile devices. The developed tracking system operates directly on images without an intermediate representation like keypoints. Depthmaps are computed by using a dense multi-view stereo algorithm and optimized by minimizing a global spatially regularized energy functional. Using the graphics processing unit (*GPU*) and highly parallelized state-of-the-art algorithms on these mobile devices, enables us to perform high quality, dense depthmap computations within several seconds.

Keywords. 3D reconstruction, mobile, interactive, stereo, dense tracking, total variation, convex optimization, GPU

Kurzfassung

Das Ziel der 3D Rekonstruktion ist die Berechnung der 3D Geometrie einer Szene, aus einer gegebenen Menge von 2D Bildern. Da dies eines der fundamentalen Probleme in der digitalen Bildverarbeitung darstellt, wurde in den letzten Jahren eine Vielzahl von Algorithmen entwickelt, die das Problem mit leistungsstarker Desktop Hardware lösen. Moderne mobile Geräte wie Tablets und Smartphones liefern jedoch eine noch nie dagewesene Rechenleistung in jedermanns Hosentasche und machen es nun möglich, diese Aufgabenstellung auch auf mobilen Plattformen zu lösen.

Das Ziel dieser Masterarbeit ist die Entwicklung der Grundbausteine *dichtes Kamera-Tracking* und *dichte Tiefenkartenberechnung*, für ein neuartiges Rekonstruktionssystem auf mobilen Geräten. Das entwickelte Tracking-System arbeitet direkt auf den Bildern ohne Verwendung einer Zwischenrepräsentation wie Schlüsselpunkten. Tiefenkarten werden über einen dichten Stereo Algorithmus berechnet und mithilfe eines räumlich regularisierten Energiefunktionals global optimiert. Erst die Verwendung des Grafikprozessors mobiler Endgeräte und der Einsatz von modernsten, hochparallelisierten Algorithmen ermöglicht die Berechnung von dichten und qualitativ hochwertigen Tiefenkarten innerhalb weniger Sekunden.

Schlüsselwörter 3D Rekonstruktion, mobil, interaktiv, Stereo, dichtes Tracking, Total Variation, Konvexe Optimierung, GPU

Affidavit

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

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date Signature

Acknowledgments

There are several people who played an important part in the realization of this thesis and I would like to use the opportunity to mention them. First and foremost, I would like to thank my family and especially my mother, who gave me the opportunity to study to my liking. Without her help I would not be where I am today.

Furthermore, I am deeply grateful to my supervisor Prof. Dr. Thomas Pock, who guided me through the last two years of my work at the Institute for Computer Graphics and Vision and also through this thesis. Without his help, continuous support and inspirational discussions, this work would have not been possible. I would also like to thank my colleagues who contributed to this thesis. At this point, I especially want to mention Gottfried Graber, who always had an open ear for all my questions, never got tired of explaining algorithms to me and helped me during implementation.

Finally, I would like to thank my girlfriend Anastasiya. Her inspiration, motivation and accompaniment through all my ups and downs during this thesis helped me to complete this work successfully.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Motivation

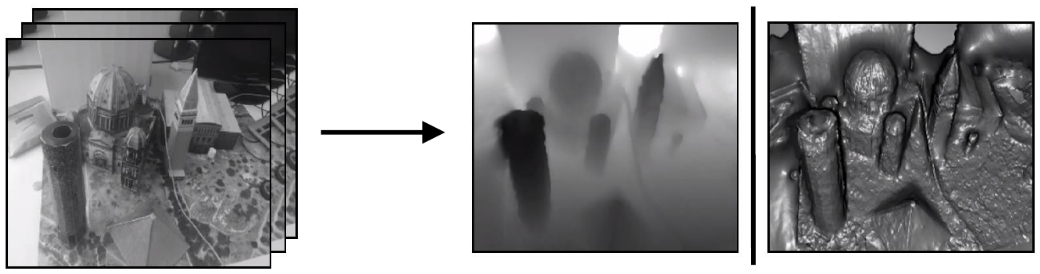


Figure 1.1: Given a set of 2D images, our system performs dense tracking in real-time and computes high quality depthmaps of the scene within several seconds on a mobile device.

State-of-the-art mobile devices like smartphones and tablets are nowadays equipped with all kind of sensors and very powerful processing units to gather information about the environment. The most powerful sensors are cameras since they deliver a huge amount of information at high frequency. However, for numerous applications like image segmentation, pose estimation or object detection, it is crucial to have 3-dimensional information about the world.

By taking a picture, the 3D world gets projected onto the 2D image plane resulting in an information loss during the projection. In contrast, the goal of a visual based reconstruction system is to deduce the 3D scene structure given a set of 2D images. Due to the information loss during the projection the reconstruction is generally known to be a hard and ill-posed problem where the uniqueness and existence of a solution is not guaranteed.

3D reconstruction actually involves several tasks including image acquisition, pose estimation and the depth computation or reconstruction itself. Using this technology on mobile devices opens up a great variety of unforeseen applications, where augmented reality (*AR*) probably has the biggest potential. By knowing the 3D geometry of the scene, all kinds of computer generated content can interact with the real world. The own living room could be used e.g. as a landscape within an AR game. In order to fully exploit these new possibilities, it is crucial to have accurate, high quality reconstructions. Therefore, the key requirements are a robust tracking system and a high quality *dense* depth computation algorithm.

1.2 Contributions and Outline

The goal of this master's thesis is to create the fundamental building blocks: *dense tracking* and *dense depthmap* computation, for a novel reconstruction system on mobile devices. Using the graphics processing unit (*GPU*) and highly parallelized state-of-the-art algorithms on these devices, enables us to perform high quality reconstruction just within several seconds. The system is run on an Android[1] platform and the user simply captures a sequence of images while moving the device. After that, the reconstruction starts and the system shows in an interactive way how the 3D model of the captured scene emerges. This usecase is easy enough for everyone since no special know-how or experience is needed. Therefore, anyone's smartphone or tablet can be used as a 3D scanner, delivering high quality depthmaps on the fly.

This thesis is structured as follows: Chapter 2 gives an overview of fundamental methods needed for 3D reconstruction and related work on mobile devices and desktops, since

[1]https://www.android.com/

the latter one formed the principles used on mobile platforms. Chapter 3 explains our approach, including *dense tracking*, *dense depthmap computation*. This chapter is followed by a chapter, where we give details about the implementation. In chapter 5 we evaluate our system and discuss the results. Finally, chapter 6 gives a conclusion and an outlook on future work.

2 Related Work

Contents

This section gives an overview and introduction of related work on dense 3D reconstruction on mobile and also desktop platforms since the latter one formed the basic principles. First, the camera and distortion models are discussed, followed by internal camera calibration. Next, epipolar geometry and triangulation methods are reviewed leading to external camera calibration and, more generally, to simultaneous location and mapping (SLAM). Here, traditional probabilistic and more recent geometric SLAM approaches are shown. Finally, dense reconstruction methods with special emphasis on real-time performance and state of the art mobile approaches are discussed.

2.1 Camera Model

The pinhole camera model is a common model of an ideal camera and follows the principle of *central projection*. A point in 3D space $\mathbf{X} = [X, Y, Z]^{\mathrm{T}} \in \mathbb{R}^3$ gets mapped onto a 2D point on the image plane at depth f with homogeneous coordinates $\mathbf{x} = [x, y, w]^{\mathrm{T}} \in \mathbb{R}^2$ where $x = fX/Z,\ y = fY/Z,\ w = f$.
This projection is characterized completely by only 5 parameters, which are called the

intrinsic parameters of the camera, also known as the *camera calibration matrix* $\mathbf{K} \in \mathbb{R}^{3\times 3}$

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

where f is the focal length, f_x, f_y are the focal length multiplied with the pixel scale factors m_x, m_y, γ is the skew factor and $\mathbf{p} = (p_x, p_y)^{\mathrm{T}}$ is called the principal point. The projection itself may now be written as

$$\mathbf{x} = \mathbf{KX} \tag{2.1}$$

The focal length f is the distance between the pinhole (camera center) and the image plane. γ, the axis skew factor, models shear distortion of the projected image. This parameter is usually 0 and can therefore be neglected. The principal point $\mathbf{p}$ describes the point of intersection between the cameras principal axis and the image plane. See Figure 2.1 for an illustration. Note that in a true ideal camera f_x and f_y take the same value but due to flaws in the digital camera sensor, non uniformly scaling in post processing or unintentional lens distortion they might differ from each other.

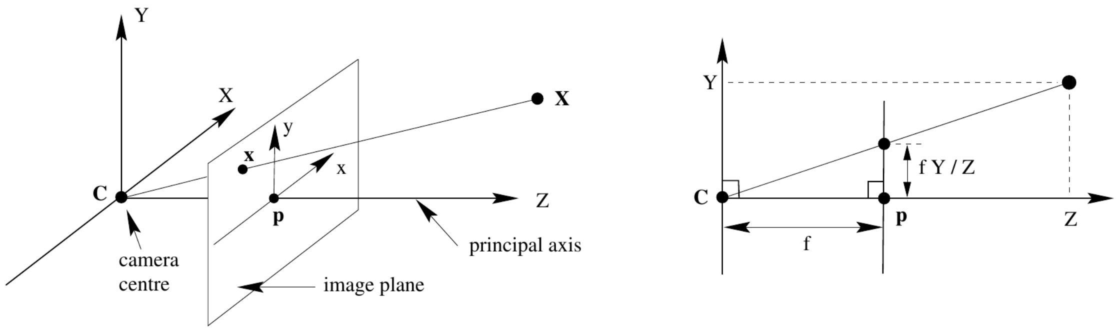


Figure 2.1: Pinhole camera model which depicts the projection of the 3D point $\mathbf{X}$ onto the point $\mathbf{x}$ on the image plane. Furthermore the pinhole $\mathbf{C}$, focal length f and principal point $\mathbf{p}$ are illustrated. Note that in a real camera system the image plane lies behind the pinhole and the resulting image is 180 degrees rotated. (Figure taken from [17]).

Furthermore it is to mention that not only f_x and f_y and the principle point $\mathbf{p}$ might differ from each other, but that a camera generally also shows lens distortion due to different constraints in the manufacturing process of the camera lens. The most significant effect is the *radial lens distortion (dr)* which means that straight lines in the real world get projected to curves on the image plane. If the lens is not mounted exactly in the center *tangential lens distortion (dt)* is the result. See Figure 2.2 for an illustration. In order to compensate for lens distortion, *dr* and *dt* are modeled as a non-linear function of the projected image coordinates. One of the most common models, that is also used in this work, is a standard polynomial model, first introduced in [1]. A pixel $\mathbf{x}$ is undistorted by

$$\begin{aligned} x' &= x\left(1 + k_1 r^2 + k_2 r^4\right) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y' &= \underbrace{y\left(1 + k_1 r^2 + k_2 r^4\right)}_{\text{dr}} + \underbrace{2p_2 xy + p_1(r^2 + 2y^2)}_{\text{dt}} \end{aligned} \tag{2.2}$$

where $\mathbf{x}' = [x', y']^{\mathrm{T}}$ is the undistorted pixel position, k_1, k_2, p_1, p_2 are the four distortion parameters and $r^2 = x^2 + y^2$.

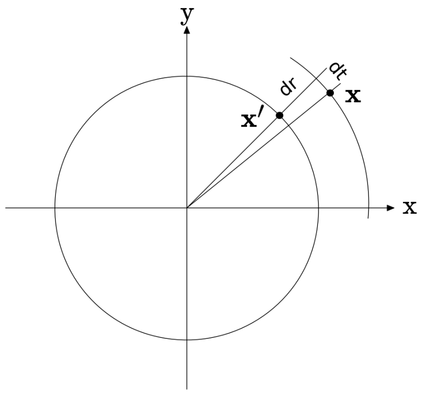


Figure 2.2: The effects of radial and tangential lens distortion is shown. The ideal projected point position is shown as point $\mathbf{x}'$ but due to radial and tangential distortion effects the projection results in the point $\mathbf{x}$.

In contrast to *intrinsic*, the *extrinsic camera parameters* describe the rotation and translation of the camera in the Euclidean coordinate system. Specifically, such a transformation is denoted by

- $\mathbf{t}$, a 3×1 translation vector representing the camera translation
- $\mathbf{R}$, a 3×3 rotation matrix defining the orientation of the camera coordinate system

which together form the 3×4 exterior or *camera pose matrix* $\mathbf{C} = [\mathbf{R} \mid \mathbf{t}]$. Thus, let $\mathbf{X}_{world} = [X_w, Y_w, Z_w, W_w]^{\mathrm{T}}$ be a point in the world coordinate frame, then $\mathbf{X}_{cam} =$

$[X_c, Y_c, Z_c, W_c]^{\mathrm{T}}$ is the representation of the same point in the camera coordinate frame

$$\mathbf{X}_{cam} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0_3} & 1 \end{bmatrix} \mathbf{X}_{world}. \tag{2.3}$$

Figure 2.3 depicts the transformation between world and camera coordinate frames.

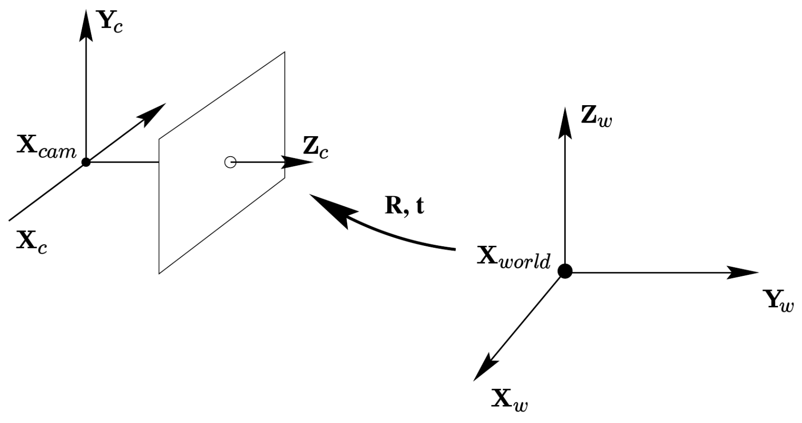


Figure 2.3: Relationship between world and camera coordinate systems.

Combining 2.1 and 2.3 leads to the full linear mapping between an homogeneous point $\mathbf{X}$ in 3D world space and its corresponding homogeneous point $\mathbf{x}$ on the image plane:

$$\mathbf{x} = \mathbf{KR}[\mathbf{I} \mid \mathbf{R}^{\mathrm{T}}\mathbf{t}]\mathbf{X}$$

where $\mathbf{R}[\mathbf{I} \mid \mathbf{R}^{\mathrm{T}}\mathbf{t}] = [\mathbf{R} \mid \mathbf{t}] = \mathbf{C}$ which leads to the *projection matrix* $\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] = \mathbf{KC}$ that gives

$$\mathbf{x} = \mathbf{PX}$$

2.2 Camera Calibration

As we have seen a pinhole camera model consists of the 2, *internal* and *external*, camera parameters. Camera calibration is now referred to as the process of determining these values. This can be achieved by finding internal and external parameters separately from each other or by jointly calibrating them together [16, 32]. In that case the projection matrix $\mathbf{P}$ is estimated and since $\mathbf{P} = \mathbf{KC}$ and $\mathbf{C} = [\mathbf{R} \mid \mathbf{t}]$ all relevant information can be decomposed afterwards.

2.2.1 Internal Camera Calibration

The procedure of finding the internal camera calibration matrix **K** is called internal camera calibration or simply camera calibration. We therefore call a camera with known **K** a *calibrated camera.*

When dealing with 3D reconstruction from a given set of 2D images a calibrated camera is a crucial requirement in order to get meaningful results. The reason for this is the projective ambiguity, which means that only intersection and tangency is preserved or as [17] say, that the angle between a pair of rays can not be measured (see Figure 2.4).

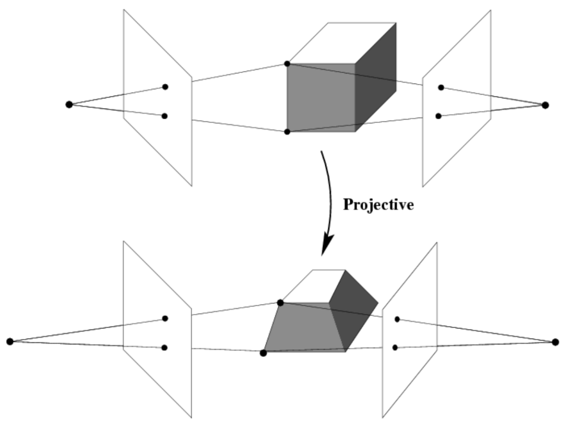


Figure 2.4: Projective ambiguity since intrinsic camera parameters **K** are unknown. (Figure taken from [17]).

In contrast, when **K** is known only rotation, translation and scaling are free parameters in the reconstruction step since the angle between a pair of rays has to be preserved, which is also known as the similarity ambiguity (see Figure 2.5).

Camera calibration is the process of estimating a model for an uncalibrated camera. In order to find these parameters, real world 3D calibration objects and their specifications are needed so that 3D point to 2D pixel correspondences can be established. The calibration procedure may be categorized according to the dimensions of the calibration target:

- **3D based calibration**, a 3D object is observed whose geometry is known very precisely [9].

- **2D based calibration**, instead of observing a 3D object a simple planar pattern is observed from several different angles [39, 40]

- **Self calibration**, instead of using any calibration target at all solely image information is used to perform the camera calibration [16, 32].

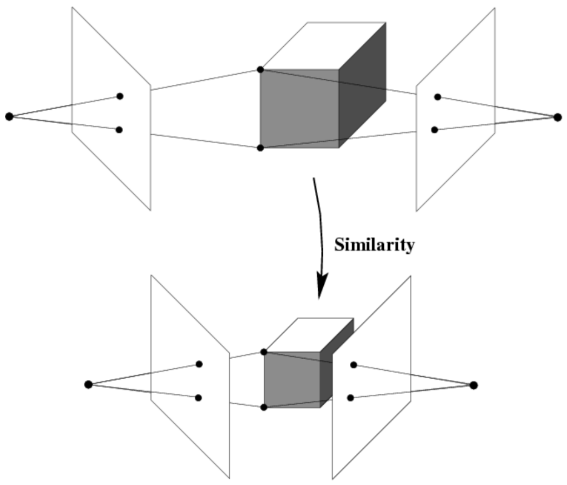


Figure 2.5: Similarity ambiguity since intrinsic camera parameters $\mathbf{K}$ are known. The angle between rays can be measured and only rotation, translation and scaling are variable. (Figure taken from [17]).

If possible, a calibration object should be used since the mathematical problem that naturally arises in self calibration is much harder to solve due to the larger number of variables that need to be estimated.

2.2.2 Epipolar Geometry

Given a two camera system (stereo vision) with corresponding 2D image projections of the 3D world, its geometry is called *epipolar geometry.* Referring to Figure 2.6, a 3D point $\mathbf{X}$ is observed by 2 cameras from distinct positions $\mathbf{C}_1$ and $\mathbf{C}_2$ in 3D space. The corresponding 2D projection $\mathbf{x}_1$ in the first image and $\mathbf{x}_2$ in the second image fulfill a number of constraints which are defined by the epipolar geometry. Intersecting the line joining the two camera centers $\mathbf{C}_1$ and $\mathbf{C}_2$ with both camera images result in the so-called *epipoles* $\mathbf{e}_1$ and $\mathbf{e}_2$. $\mathbf{C}_1$, $\mathbf{C}_2$ and the 3D point $\mathbf{X}$ form together the *epipolar plane* π. The *epipolar line* is formed when a given image point and the corresponding epipole are joined or formulated alternatively, when the epipolar plane and the image plane are intersected. Knowing solely the camera position $\mathbf{C}_1$ and coordinates of the 2D projection $\mathbf{x}_1$ it is impossible to reconstruct the original 3D point $\mathbf{X}$ due to the fact that its depth is unknown.

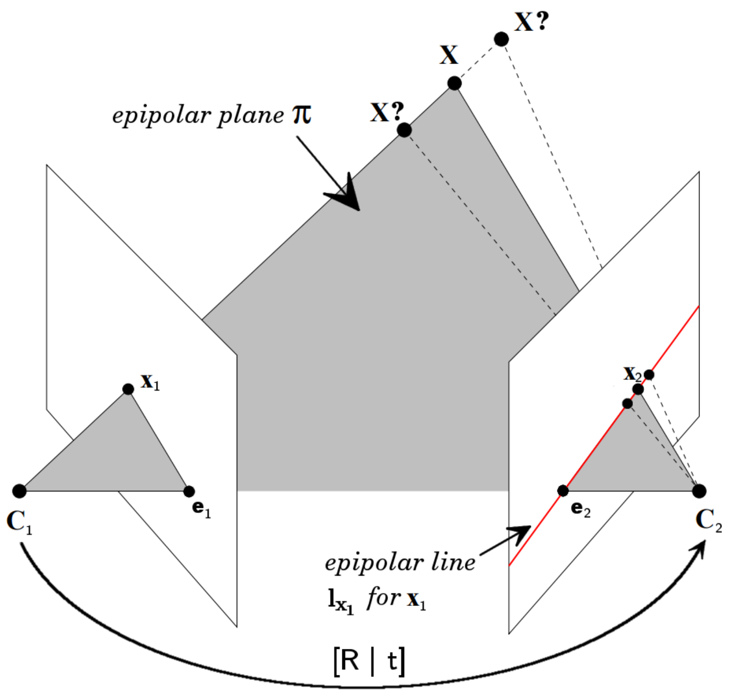


Figure 2.6: Epipolar geometry with camera centers $\mathbf{C}_1$, $\mathbf{C}_2$ and their relative pose $[\mathbf{R} \mid \mathbf{t}]$. $\mathbf{C}_1$, $\mathbf{C}_2$ and 3D point $\mathbf{X}$ (but also the projections $\mathbf{x}_1$, $\mathbf{x}_2$ and the epipoles $\mathbf{e}_1$ and $\mathbf{e}_2$) lie on a common epipolar plane π. The point $\mathbf{x}_1$ is mapped to the epipolar line $\mathbf{l}_{x_1}$ and the epipolar constraint $\mathbf{x}_2^\mathrm{T}\mathbf{F}\mathbf{x}_1 = 0$ is fulfilled.

It is only known that $\mathbf{X}$ has to lie on the viewing ray defined by $\mathbf{C}_1$ and $\mathbf{x}_1$. The point $\mathbf{x}_2$ on the other hand fulfills the so-called *epipolar constraint* which states the $\mathbf{x}_2$ has to lie on the epipolar line $\mathbf{l}_{x_1}$. As mentioned before $\mathbf{l}_{x_1}$ is formed by intersecting the epipolar plane π with the second image plane. Furthermore corresponds the 2D projection $\mathbf{x}_1$ to all 3D points on the viewing ray between $\mathbf{C}_1$ and $\mathbf{x}_1$ which means that the projection of all that 3D points onto the second image plane result exactly in $\mathbf{l}_{x_1}$. Therefore a point $\mathbf{x}_1$ on the first image plane defines a corresponding epipolar line in the second image. Mathematically, the epipolar constraint is expressed by the *fundamental matrix* $\mathbf{F} \in \mathbb{R}^{3\times3}$:

$$\mathbf{x}_2^\mathrm{T}\mathbf{F}\mathbf{x}_1 = 0 \tag{2.4}$$

since the mapping between points and lines is represented by the fundamental matrix

$$\mathbf{l}_{x_1} = \mathbf{F}\mathbf{x}_1 \tag{2.5}$$

and it's known from projective geometry that a point $\mathbf{x}$ lies on a line $\mathbf{l}$ when $\mathbf{x}^{\mathrm{T}}\mathbf{l} = 0$ is fulfilled. Note that $\mathbf{F}$ is independent from the scene structure since it is computed solely from image points [17].

2.2.3 Triangulation

Given the 2D projections $\mathbf{x}_1$ and $\mathbf{x}_2$, the camera positions $\mathbf{C}_1$ and $\mathbf{C}_2$ and a known camera calibration matrix $\mathbf{K}$ one can find the corresponding 3D point $\mathbf{X}$. This is done by intersecting the viewing rays defined by $\mathbf{C}_1, \mathbf{x}_1$ and $\mathbf{C}_2, \mathbf{x}_2$ respectively. This process is commonly referred to as *Triangulation*.

In the absence of noise the epipolar constraint is fulfilled and thus the problem is trivial to solve (see Figure 2.6). For real cameras this is commonly not the case and noise is present due to flaws in the lens manufacturing process, imperfect measurements etc. In that case the viewing rays will generally not meet and therefore the best point of intersection has to be found by using more robust and sophisticated approaches.
The midpoint method introduced by [30] is a well known and widely used method where the midpoint of the common perpendicular to the two viewing rays is computed. The polynomial method [15] is an optimal method of triangulation and seeks the points $\mathbf{x}'_1$ and $\mathbf{x}'_2$ that globally minimize the cost function

$$d(\mathbf{x}_1, \mathbf{x}'_1)^2 + d(\mathbf{x}_2, \mathbf{x}'_2)^2$$

where $d(\mathbf{x}, \mathbf{y})$ is the Euclidean distance, subject to the epipolar constraint 2.4. The minimization problem gets reformulated to

$$d(\mathbf{x}_1, \lambda_1)^2 + d(\mathbf{x}_2, \lambda_2)^2$$

where λ_1 and λ_2 are all possible choices of corresponding epipolar lines and finally a sixth-order polynomial is solved.

2.2.4 External Camera Calibration

Given a camera calibration matrix $\mathbf{K}$ and a set of correspondences between 3D points $\mathbf{X}_i$ and 2D projections $\mathbf{x}_i$ the process of finding the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ of a calibrated camera with reference to some world coordinate system is referred to as *external camera calibration* or often called by the more intuitive term *pose estimation*.

Although it is possible to estimate the pose of an uncalibrated camera [16, 32], the knowledge of $\mathbf{K}$ makes the problem mathematically easier. Instead of estimating the full camera projection matrix $\mathbf{P} = \mathbf{KC}$, merely the camera pose $\mathbf{C} = [\mathbf{R} \mid \mathbf{t}]$ has to be found. In other words, the degrees of freedom (*DOF*) drop from 11 for $\mathbf{P}$ to 6 for $\mathbf{C}$ since $\mathbf{K}$ gives an additional 5 DOF for intrinsic parameters. The *perspective n-point problem* (*PnP*) [10] states that the rotation $\mathbf{R}$ and translation $\mathbf{t}$ of a calibrated camera can be estimated given n $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$ point correspondences. Since a calibrated camera is used and $\mathbf{K}$ is known, the angle θ between two viewing rays can be measured (see Figure 2.7). PnP uses solely this information:

$$d_{12}^2 = \overline{\mathbf{CX}}_1^2 + \overline{\mathbf{CX}}_2^2 - 2\overline{\mathbf{CX}}_1\overline{\mathbf{CX}}_2 cos\theta_{12}$$

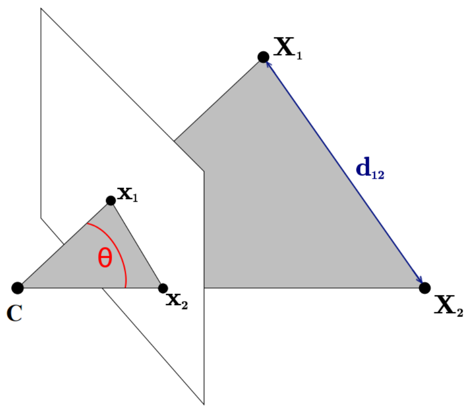


Figure 2.7: Perspective n-point problem uses the fact that the angle between rays can be measured.

A minimum of $n >= 3$ is needed to perform the pose estimation. Therefore the term *perspective 3-point problem* (*P3P*) was introduced [10]. Using only 3 point correspondences P3P will give typically two but up to four solutions. *P4P* though, is already overdetermined and a unique solution can be found in case of coplanarity of the points $\mathbf{X}_i$. In case of non-planarity *P3P* is performed four times and a consensus between the solutions is found. Since then, their generalized approach of finding consensus between solutions is known as *random sample and consesus* (*RANSAC*).

2.3 SLAM

In section 2.2 we have seen how 3D points of the scene can be computed if the camera poses are known and vice versa, how the camera poses are estimated if the 3D scene is known. The question that naturally arises now is how one can calculate 3D points and camera poses in an unknown environment. That means that neither a 3D model of the region of interest, nor the camera pose is given a priori. This circumstance is generally known as a chicken-egg-problem since one needs a 3D model in order to perform the *localization* or pose estimation. On the other hand has the camera pose to be known to compute the *mapping* or 3D reconstruction. Therefore both distinct parts are dependent on each other but none of them is existing at the beginning. In literature, this problem of simultaneously performing pose estimation and map building is called *Simultaneous Localization And Mapping* (*SLAM*). Even though the expression SLAM got coined by research in the field of mobile robotics the same topic was studied in another field of computer vision, the photogrammetry. Here the problem is known as *structure from motion* (*SfM*) where the aim is to build a map from measurements taken by a moving sensor. The main difference to traditional SLAM is though, that the processing time and the actual sensor trajectory are negligible. Furthermore, most SfM algorithms are batch based which means that an incremental map building is not required. Therefore SfM is often found in offline working approaches.

2.3.1 Probabilistic Approach

Probabilistic SLAM algorithms are jointly estimating the pose and the map using a single probabilistic formulation. In [7] an introduction to SLAM given and the probabilistic form of the SLAM problem is described. SLAM got coined by research in the field of mobile robotics, where the autonomous movement of a robot in an unknown environment is a major task. The estimate of the pose and landmark positions (map) is given by the conditional joint probability

$$P(\mathbf{x}_k, m \mid \mathbf{z}_{0:k}, \mathbf{u}_{0:k}, \mathbf{x}_0) \tag{2.6}$$

where $\mathbf{x}_k$ is a state vector describing the robot orientation and location at time k, m is the map (set of all landmarks), $\mathbf{z}_{0:k}$ are landmark observations, $\mathbf{u}_{0:k}$ is the history of input controls and $\mathbf{x}_0$ is the initial state of the robot [7]. Furthermore an *observation model* and a *motion model* are defined and 2.6 is computed by these two quantities. The observation model describes the probability of making an observation $\mathbf{z}_k$, if the robots pose $\mathbf{x}_k$ and landmark locations m are known in the form $P(\mathbf{z}_k \mid \mathbf{x}_k, m)$. The motion model $P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k)$ describes the probability of the new pose $\mathbf{x}_k$ given only the old vehicle position and orientation $\mathbf{x}_{k-1}$ and input control $\mathbf{u}_k$. From that it can be seen that the pose transition is assumed to be a Markov process since not the whole vehicle pose history is used for computation but only the immediate preceding pose at time $k-1$.

In order to solve the probabilistic SLAM problem several different approaches have been introduced. A commonly used way is to represent the observation and motion model as a state-space model with additive gaussian noise where then an extended kalman filter (*EKF*) [26] can be applied [6]. The observation model is then defined in the form

$$P(\mathbf{z}_k \mid \mathbf{x}_k, m) \Longleftrightarrow \mathbf{z}(k) = h(\mathbf{x}_k, m) + \mathbf{v}_k$$

where $h(\cdot)$ is a function that describes the geometry of the observation and $\mathbf{v}_k$ is zero mean uncorrelated gaussian noise of the landmarks with a covariance matrix $\mathbb{R}_k$. The motion model is now described as

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \Longleftrightarrow \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k$$

where $f(\cdot)$ models the robots motion and $\mathbf{w}_k$ is zero mean uncorrelated gaussian noise of the robots kinematic with a covariance matrix $\mathbb{Q}_k$. The main drawback of this approach is that during the observation update step the covariance matrices and all landmarks have to be recalculated every time a new observation is made. This makes this approach computationally extremely expensive. Due to various optimizations real-time EKF-SLAM systems with several thousands of landmarks have been implemented though [14, 22].

Another important alternative to EKF-SLAM is to represent the motion model as a set of discrete sample points that follow a non-gaussian probability distribution. This approach is known as FastSLAM or Rao-Blackwallized particle filter [27]. In their work, they divide the SLAM problem in two separate parts. First the localization of the vehicle is performed using a particle filter. Secondly, the feature estimation depending on the robot's pose is done by using a Kalman filter conditioned by the pose. This combination of the two, Kalman and particle, filters is known as the Rao-Blackwallized particle filter.

By using a tree structure the computational cost could be reduced but the approach still remains computationally costly [27].

All previously discussed SLAM methods assume that a robot is exploring an unknown environment which means that all kinds of odometry data or accurate depth measurements from active range sensors are given. The basic principles of SLAM can be nevertheless also applied on pure visual systems (VSLAM). In that case there is only a single sensor given, the camera. This circumstance makes the problem a lot more challenging since all informations have to be extracted from images alone and one has to deal with high input frame rates and motion blurred images due to fast camera movements or too slow shutter speed. Fast VSLAM was one of the first visual SLAM systems using solely a single camera [5]. They were able to estimate the ego-motion of the camera in real-time through an unknown scene using a commodity desktop PC.

2.3.2 Geometric Approach

Inspired by optimization techniques which were used by *structure from motion* (*SfM*) systems, geometric SLAM algorithms are the second important class of SLAM systems. Instead of using a probabilistic formulation, the SLAM problem is reformulated as an optimization problem which can be solved by using a least-square solver which is today better known as *bundle adjustment*. By minimizing the reprojection error of 3D scene points within several camera views a geometric constraint is incorporated. This means that all 3D points as well as all camera poses are refined simultaneously within one optimization. The main drawback of this method is that the computational effort is large and therefore bundle adjustment is often found in offline SfM systems.

In order to compensate for this, Parallel Tracking and Mapping (*PTAM*) introduced by [19] divided the SLAM problem into two distinct parts running in two separate threads. This marked a change in the way how VSLAM systems are build because now tracking and mapping were not tightly coupled anymore, i.e. both parts are not performed together at each frame. This is beneficial since the mapping part is especially computationally costly and can therefore not be performed in real-time. The tracking thread on the other hand runs at $30Hz$, delivering fast and accurate pose estimations using the current map. In the beginning the map is built by a stereo initialization procedure and afterwards the map is extended whenever the camera explores new regions. In order to reduce the computational complexity this map extension is performed only at *keyframes*. Matching of corresponding points in other keyframes is done by using patch-based correlation. Finally, the accuracy of the map is further improved at each map update by bundle adjustment optimization of a fixed number of keyframes. Even though PTAM is capable of building a map consisting of several thousand 3D points and tracking them in real time, the reconstruction is still considered as being sparse (see Figure 2.8).

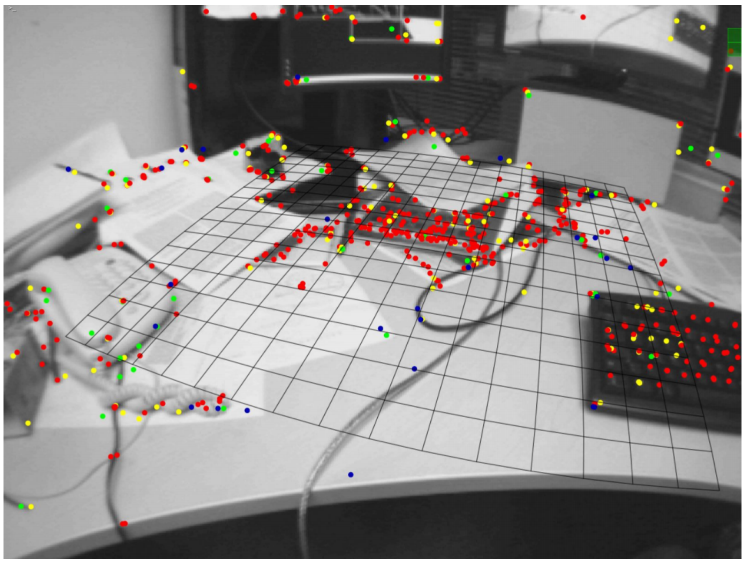


Figure 2.8: The map generated by PTAM contains nearly 3000 points, whereas about 1000 are attempted to be found in the current frame. Finally, 660 are successfully observed and shown as dots. Furthermore is the dominant plane shown as a grid. (Figure taken from [19])

2.3.2.1 Mobile Approach

In [20] the basic PTAM framework got adopted to work on mobile phones. Due to hardware limitations of the phones available at the time of publishment (*Apple iPhone 3G*[1]) the original concept had to be changed. Instead of a two-threaded system running on a multi-core CPU, tracking and mapping are performed still in two separate threads but on a single core. Therefore the bundle adjustment only runs in the time gaps where the tracking thread is waiting for a new frame. Furthermore, only several tens or hundreds of points are mapped and tracked resulting in poor pose estimation and reconstruction quality.

[1]https://www.apple.com/

With the evolution in mobile hardware, multi-core CPUs are nowadays available on mobile devices too. This fact is used in [24] - mapping and tracking are decoupled and run on different cores resulting in a better performance than in [20].

2.4 Dense Reconstruction

All previously reviewed methods work in a two-step scheme. First, discrete feature observations have to be extracted from the images and matched to each other. Second, the pose estimation is then done using solely this set of observations - the original images itself are not used anymore. This additional abstraction step reduces the problem's overall complexity enormously but also brings major drawbacks with it. Depending on the feature type, typically only image corners, line segments or blobs are extracted, the rest of the image is neglected. Furthermore these extracted features have then be matched to each other requiring computationally rich scale- and rotation-invariant feature descriptors but also outliers have to be eliminated using robust methods like RANSAC. Regarding reconstruction, the most obvious drawback is though, that the reconstruction of all keypoint-based approaches is sparse. To overcome this problem [28] generate a base mesh from the sparse point set of PTAM's map and gradually refine it using dense depth estimations from a variational optical flow. The main problem of this approach is though, that concavities which are not captured by the extracted base mesh may not be recovered in the following dense refinement procedure.

In order to counteract the drawbacks of feature point based approaches, dense monocular SLAM methods have been proposed [25]. See Figure 2.9 for a comparison between dense and sparse methods. The main difference is that the additional feature extraction step is not performed but instead these methods work directly on the images itself, for both localization and mapping. First, the scene is modeled as a dense surface rather than just a set of points. Second, localization or tracking is done using whole image alignment. In that way the need of discrete feature observations is removed and *all* information available in the image can be used resulting in a significantly improved tracking accuracy and robustness. Dense Tracking and Mapping (*DTAM*) [29] is the dense counterpart to PTAM and can be seen as an improved version of it. The surface model is created by using a multiview stereo algorithm from up to hundreds of small baseline images with a global

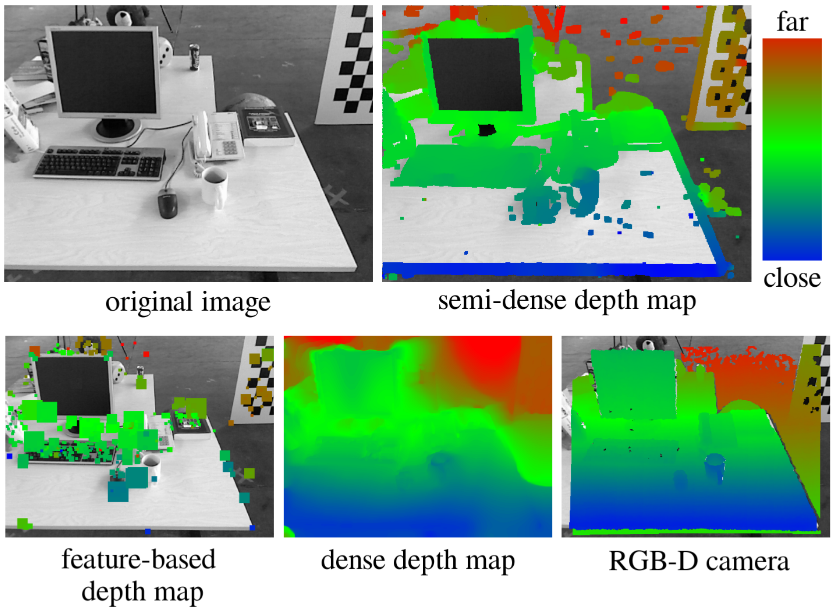


Figure 2.9: Comparison of dense and sparse reconstruction methods. On the top right, the semi-dense approach has dense depth data in information rich image regions, i.e. edges and corners [8]. The bottom line shows feature-based depthmap [19], a fully dense depthmap [25] and the ground truth RGB-D data [37]. (Figure taken from [8])

spatially regularized energy functional that gets minimized in a non-convex optimization framework. In comparison to PTAM they showed that especially under difficult conditions like camera defocus or motion blur DTAM does not loose tracking due to the huge amount of data generated by this dense approach. However, real-time performance is only achieved using a powerful desktop GPU.

In [8] a feature less, semi-dense approach was introduced where the depthmap is calculated only in image regions which carry information, i.e. edges and corners. Due to the reduced amount of data and a probabilistic depthmap representation, real-time performance was achieved on a commodity *CPU*. The main purpose of this method is though a real-time monocular visual odometry system, rather than a reconstruction system.

2.4.1 Mobile Approach

Lately, dense reconstruction methods were also proposed for mobile platforms. The semi-dense system of [8] got adopted to work on *Android*[2] systems [35]. This is achieved by

[2]`https://www.android.com/`

lowering the image resolution from 640×480 to 320×240 used for mapping. The tracking follows, like in the original system, a pyramid scheme in order to handle larger inter-frame motions. But due to the lower processing power of the mobile CPU, the largest pyramid level was reduced even further to a resolution of 160×120 to ensure real-time performance. Furthermore, all computation-heavy algorithmic steps which are suited for parallelization are optimization using *SIMD* parallelization. On ARM[3] processors, this functionality is achieved by NEON instructions. Finally, a low resolution collision mesh is generated out of the semi-dense depthmap giving a virtual vehicle the possibility to interact with the real world.

In [38] the *inertial measurement unit* (*IMU*) is used additionally to the image information to improve tracking accuracy. Also, the map has to be initialized using a two-view-initialization. ORB features [33] are extracted, matched and outliers are removed using RANSAC in combination with the 5-point algorithm. Finally, the initial map is subsequently refined with bundle adjustment making the initialization computationally costly. Depthmaps are computed using a multi-resolution scheme but the final reconstructed 3D model is still sparse. In [21] the method of [38] got further improved resulting in an increased reconstruction accuracy. The system is still point cloud-based, though.

[3]https://www.arm.com/

3 Approach

Contents

Our approach of dense reconstruction on mobile devices is presented in this section. We first give an overview of the system before the individual parts are described in more detail. These include dense camera pose tracking with all mathematical properties needed, a multiview stereo algorithm used for bootstrapping the system, an adaptive stereo algorithm to improve the initial depthmap and an optimization algorithm that minimizes a global spatially regularized energy functional.

3.1 System Overview

Our system performs dense tracking and dense depthmap creation of a given set of images on an Android tablet. The input images can originate from either precaptured frames or from the live camera stream. Figure 3.1 shows the principal process flow of our system. Once the input frame source is selected and the images are captured, a keyframe is chosen. Starting with a random or homogeneous initial depthmap the tracking for every frame in the sequence is performed. Now a coarse depthmap is calculated using the first rough pose estimates. From here on the iterative procedure starts, all frames are tracked again with the newly computed depthmap and depending on the computing stage either a coarse depthmap is generated again or a more sophisticated algorithm for depthmap generation is performed. This can be an adaptive stereo algorithm which improves the initial depthmap or an optimization algorithm that minimizes a global spatially regularized energy functional.

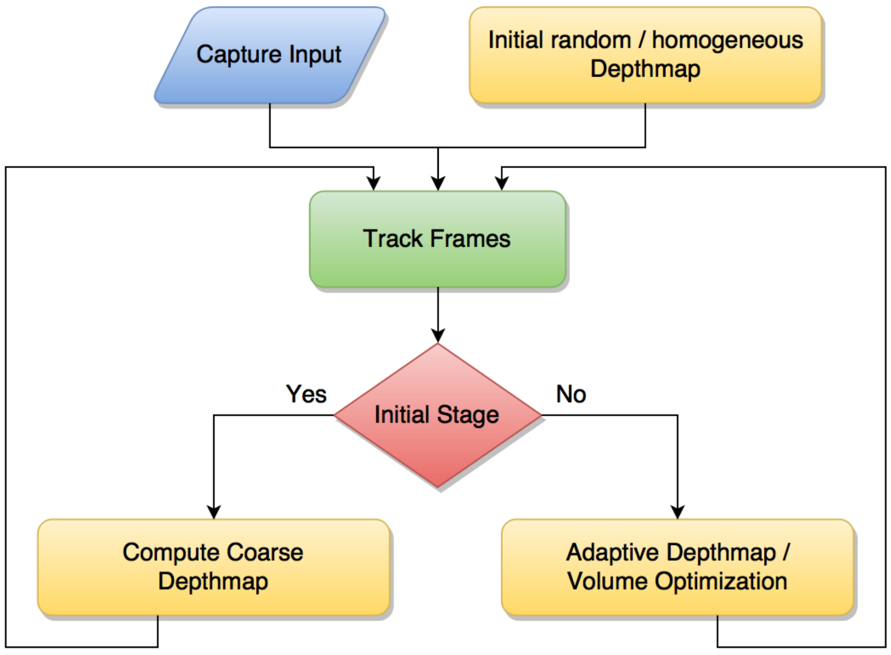

Capture Input
Initial random / homogeneous Depthmap
Track Frames
Yes
Initial Stage
No
Compute Coarse Depthmap
Adaptive Depthmap / Volume Optimization

Figure 3.1: System overview

The system itself consists of three separate threads. The first one is responsible for the live camera stream and enqueues incoming frames into an input buffer. The second thread is handling all tasks regarding GUI in- and output. The third thread performs the actual computations, namely dense frame tracking and depthmap generation. For this thesis we made the decision to perform both tasks in one thread on the GPU. The main reasons for this are ease of implementation and the non-vital necessity of running tracking and mapping really simultaneously because in this work we capture a set of frames and perform all computations iteratively afterwards. CUDA[1] is used for all general purpose computations (*GPGPU*) on the graphic chip.

[1]`http://www.nvidia.com/object/cuda_home_new.html`

3.2 Dense Tracking

Given a reference image $\mathbf{I}_{ref}$ and a current image $\mathbf{I}$, the goal of dense tracking is to estimate the camera motion by aligning the images $\mathbf{I}_{ref}$ and $\mathbf{I}$. For this, assumptions have to be made. This approach is based on the photo consistency assumption, which is illustrated in Figure 3.2. The photo consistency assumption states that a 3D world point $\mathbf{X}$ observed by two cameras results in the same brightness in both camera images. Therefore it is defined as

$$\mathbf{I}_{ref}(\mathbf{x}) = \mathbf{I}(\tau(\mathbf{x}, \boldsymbol{\xi})) \tag{3.1}$$

where $\mathbf{x} = [x, y]^{\mathrm{T}} \in \mathbb{R}^2$ are 2D pixel coordinates, $\mathbf{I}(\mathbf{x})$ is the current image at pixel location $\mathbf{x}$, $\mathbf{I}_{ref}(\mathbf{x})$ is the reference image at pixel location $\mathbf{x}$, $\boldsymbol{\xi} = [w_1, w_2, w_3, w_4, w_5, w_6] \in \mathbb{R}^6$ is a 6-dimensional vector in the $\mathfrak{se}(3)$ Lie algebra defining the minimal representation of a rigid body transformation and $\tau(\mathbf{x}, \boldsymbol{\xi})$ is the transformation function that takes a pixel $\mathbf{x}$ from one image to the other using the transformation parameters $\boldsymbol{\xi}$. In the following sections we follow [18, 23, 29].

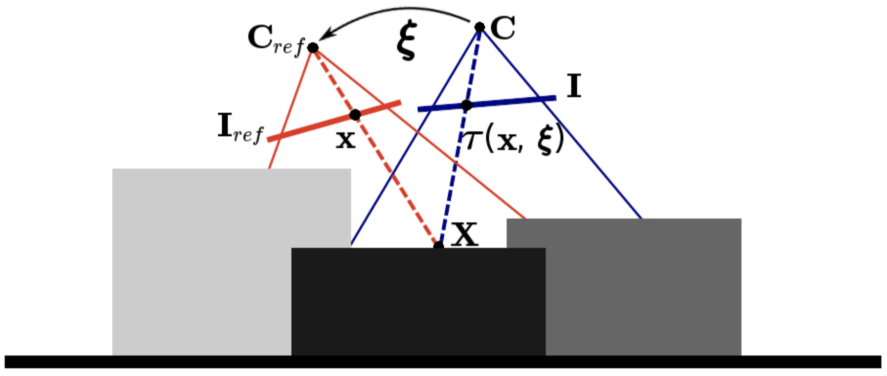


Figure 3.2: Photo consistency assumption - an observed 3D world point $\mathbf{X}$ yields the same brightness in both images $\mathbf{I}_{ref}$ and $\mathbf{I}$. The aim is to find the transformation parameters $\boldsymbol{\xi}$ such that the warped image matches the other one.

Using 3.1, an optimization model seeks to find now the *optimal transformation* between those two images by minimizing

$$\min_{\boldsymbol{\xi}} \|\mathbf{I}(\tau(\mathbf{x}, \boldsymbol{\xi})) - \mathbf{I}_{ref}(\mathbf{x})\|^2 \tag{3.2}$$

which means that we are aligning the image $\mathbf{I}_{ref}$ and $\mathbf{I}$ by minimizing the photometric error. The transformation or warping function is defined as

$$\tau(\mathbf{x},\boldsymbol{\xi}) = \pi(g(G(\boldsymbol{\xi}), d(\mathbf{x})\pi^{-1}(\mathbf{x}))). \tag{3.3}$$

The projection function $\pi(\mathbf{X})$ is projecting the 3D point $\mathbf{X}$ onto the image plane and is given by $\pi(\mathbf{X}) = \mathcal{P}(\mathbf{KX}) = \left[\frac{f_x X}{Z} + p_x, \frac{f_y Y}{Z} + p_y\right]^{\mathrm{T}}$, where $\mathbf{K}$ is the intrinsic camera calibration matrix, $f_{x,y}$ is the pixel scaled focal length, $p_{x,y}$ is the principal point and $\mathcal{P}(\cdot)$ is the projection function that takes homogeneous coordinates to euclidean coordinates by dividing by the last coordinate. $\pi^{-1}(\mathbf{x})$ is the inverse projection function defined as $\pi^{-1}(\mathbf{x}) = \mathbf{K}^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$. The depth is given by the depthmap d which corresponds to the reference image $\mathbf{I}_{ref}$. We can therefore now define $\mathbf{X} = d(\mathbf{x})\pi^{-1}(\mathbf{x})$, that is the 3D point $\mathbf{X}$ constructed by projecting the pixel coordinate $\mathbf{x}$ to the depth $d(\mathbf{x})$. $G(\boldsymbol{\xi}) = e^{[\boldsymbol{\xi}]_\times}$ is the exponential map that associates elements of the $\mathfrak{se}(3)$ Lie algebra to elements of the underlying SE(3) Lie group. The actual transformation of the 3D point $\mathbf{X}$ with transformation matrix $\mathbf{T} \in \mathrm{SE}(3)$ is denoted by $g(\mathbf{T},\mathbf{X}) = \mathbf{TX}$. With that being known the transformation function defined in (3.3) can be written shorter as

$$\tau(\mathbf{x},\boldsymbol{\xi}) = \pi(g(G(\boldsymbol{\xi}), \mathbf{X})). \tag{3.4}$$

3.2.1 Inverse Compositional Model

Since (3.2) has to be solved iteratively, the minimization problem has to be reformulated and therefore different models are used. As we will see later in 3.2.2, an iteratively reweighted least squares (*IRLS*) algorithm will be applied to solve the problem. Here, the model choice is crucial since some formulations need computationally expensive Jacobian recalculations in every iteration of the IRLS algorithm. The *forward additive* model is defined as $\min\limits_{\boldsymbol{\delta\xi}} \|\mathbf{I}(\tau(\mathbf{x},\boldsymbol{\xi}_0 + \boldsymbol{\delta\xi})) - \mathbf{I}_{ref}(\mathbf{x})\|^2$ and seeks to find incremental update steps $\boldsymbol{\delta\xi}$ of the transformation parameters $\boldsymbol{\xi}$, given some initial estimate $\boldsymbol{\xi}_0$. The *forward compositional* model is defined similarly as $\min\limits_{\boldsymbol{\delta\xi}} \|\mathbf{I}(\tau(\tau(\mathbf{x},\boldsymbol{\delta\xi}),\boldsymbol{\xi}_0) - \mathbf{I}_{ref}(\mathbf{x})\|^2$. Both models have the drawback that the Jacobian has to be recalculated in every iteration, though.

Therefore, we chose the *inverse compositional* model which is defined as

$$\min_{\boldsymbol{\delta\xi}} \|\mathbf{I}_{ref}(\tau(\mathbf{x}, \boldsymbol{\delta\xi})) - \mathbf{I}(\tau(\mathbf{x}, \boldsymbol{\xi}_0))\|^2. \tag{3.5}$$

Compared to the forward additive or forward compositional model the roles of the current image and the reference image are switched and both are transformed. This means that the inverse model seeks for an incremental update making the reference image more similar to the current image, which also gets transformed with the current estimate. From here on the transformation is updated according to $\boldsymbol{\xi}_{k+1} = \boldsymbol{\xi}_k \boxplus (\boldsymbol{\delta\xi})^{-1}$ and the whole procedure is run iteratively. Since both, $\boldsymbol{\xi}_k$ and $\boldsymbol{\delta\xi}$, are elements of the Lie algebra, a composition of two transformations, defined by the Lie algebra group product, is denoted by the $\boxplus$ operator. Note, that an *inverse* update step $(\boldsymbol{\delta\xi})^{-1}$ has to be added since the incremental update is performed on the reference image and not on the current image. Since (3.5) is non-convex and non-linear in the argument $\boldsymbol{\xi}$, we linearize around $\boldsymbol{\delta\xi} = 0$ by computing the first order Taylor approximation and get

$$\mathbf{I}_{ref}(\tau(\mathbf{x}, \boldsymbol{\delta\xi})) \approx \mathbf{I}_{ref}(\tau(\mathbf{x}, 0)) + \left.\frac{\partial \mathbf{I}_{ref}(\tau(\mathbf{x}, \boldsymbol{\delta\xi}))}{\partial \boldsymbol{\delta\xi}}\right|_{\boldsymbol{\delta\xi}=0} \boldsymbol{\delta\xi} \tag{3.6}$$

The derivative can be further expanded using the chain rule

$$\left.\frac{\partial \mathbf{I}_{ref}(\tau(\mathbf{x}, \boldsymbol{\delta\xi}))}{\partial \boldsymbol{\delta\xi}}\right|_{\boldsymbol{\delta\xi}=0} = \left.\frac{\partial \mathbf{I}_{ref}}{\partial T(\mathbf{x}, \boldsymbol{\delta\xi})}\right|_{T=T(\mathbf{x},0)=\mathbf{x}} \left.\frac{\partial \tau(\mathbf{x}, \boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \nabla \mathbf{I}_{ref} \left.\frac{\partial \tau(\mathbf{x}, \boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} \tag{3.7}$$

The first part is simply the gradient of the reference image, it will be denoted by $\nabla \mathbf{I}_{ref}$. The second term is the derivative of the transformation w.r.t. the transformation parameters $\boldsymbol{\xi}$ evaluated at $\boldsymbol{\xi} = 0$. By using the definition of the transformation given in (3.4) and applying the chain rule, the derivative expands to

$$\left.\frac{\partial \tau(\mathbf{x}, \boldsymbol{\xi})}{\partial \boldsymbol{\xi})}\right|_{\boldsymbol{\xi}=0} = \left.\frac{\partial \pi}{\partial g}\right|_{g=g(G(0),\mathbf{X})} \left.\frac{\partial g}{\partial G}\right|_{G=G(0)} \left.\frac{\partial G(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0}. \tag{3.8}$$

Now we will address each individual term in (3.8). The first one is the derivative of the projection π which is given by

$$\mathbf{J}_\pi = \left.\frac{\partial \pi}{\partial g}\right|_{g=g(G(0),\mathbf{X})} = \frac{\partial \pi(\mathbf{X})}{\partial \mathbf{X}} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \end{bmatrix} \tag{3.9}$$

Next, there is the derivative of the transformation of the 3D point $\mathbf{X}$ (see (A.4), (A.5))

$$\mathbf{J}_g = \left.\frac{\partial g}{\partial G}\right|_{G=G(0)} = \frac{\partial \mathbf{IX}}{\partial \mathbf{I}} = \mathbf{X}^{\mathrm{T}} \otimes \mathbf{I}_{3\times 3} \tag{3.10}$$

Next one is the derivative of the exponential map evaluated at $\boldsymbol{\xi} = 0$ which is derived in detail in (A.15)

$$\mathbf{J}_G = \left.\frac{\partial G(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \begin{bmatrix} \mathbf{0}_{3\times 3} & -[\mathbf{e}_1]_\times \\ \mathbf{0}_{3\times 3} & -[\mathbf{e}_2]_\times \\ \mathbf{0}_{3\times 3} & -[\mathbf{e}_3]_\times \\ \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} \end{bmatrix} \tag{3.11}$$

We saw now that the individual Jacobians do not depend on $\boldsymbol{\xi}$ and therefore the full Jacobian can be precomputed once for all iterations. It is given by

$$\begin{aligned} \mathbf{J} &= \nabla \mathbf{I}_{ref} \left.\frac{\partial \tau(\mathbf{x},\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \nabla \mathbf{I}_{ref} \mathbf{J}_\pi \mathbf{J}_g \mathbf{J}_G = \\ &= \nabla \mathbf{I}_{ref} \begin{bmatrix} \frac{f_x}{Z} & 0 & -f_x \frac{X}{Z^2} & -f_x \frac{XY}{Z^2} & f_x(1+\frac{X^2}{Z^2}) & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -f_y \frac{Y}{Z^2} & -f_y(1+\frac{Y^2}{Z^2}) & f_y \frac{XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix} \end{aligned} \tag{3.12}$$

3.2.2 Optimization

As mentioned in 3.2, we assume that the photo consistency assumption holds. This means that for all n pixels $\mathbf{x}_j$ with $j = 1, ..., n$ in the image, (3.1) equally holds. Based on this we define the *residual* of j-th pixel as the brightness difference between the reference image and the current image as

$$r_j(\boldsymbol{\xi}) = \mathbf{I}(\tau(\mathbf{x}_j,\boldsymbol{\xi})) - \mathbf{I}_{ref}(\mathbf{x}_j). \tag{3.13}$$

Ideally, all residuals would be zero. A true camera, however, produces sensor noise leading the residuals to follow a probabilistic *sensor model* $p(r_j \mid \boldsymbol{\xi})$ distribution. We follow [18] and assume t-distributed errors. Their analysis showed that a normal distribution or Tukey weights poorly fit the data (see figure 3.3). Since the t-distribution covers outliers with its heavy tails, it is well suited to model distributions with outliers included. Furthermore, the t-distribution gives the possibility to specify a *degrees of freedom* ν of the distribution, in addition to the mean μ and the variance σ^2.

The weighting function $w(r)$ which is derived from t-distribution is defined as

$$w(r_j) = \frac{\partial \log p(r_j)}{\partial r_j} \frac{1}{r_j} = \frac{\nu + 1}{\nu + \left(\frac{r_j}{\sigma}\right)^2} \tag{3.14}$$

where the variance σ^2 is calculated with

$$\sigma^2 = \frac{1}{n} \sum_j r_j^2 \frac{\nu + 1}{\nu + \left(\frac{r_j}{\sigma}\right)^2}. \tag{3.15}$$

Note that the function is defined recursively and therefore has to be solved iteratively.

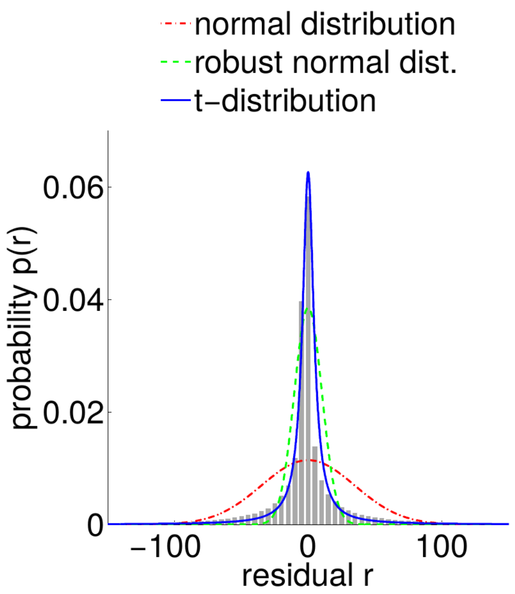


Figure 3.3: [18] analyzed the error probability distributions. Normal distribution and Tukey weights fit the data, in comparison to the t-distribution, poorly. (Figure taken from [18]).

3.2.2.1 Algorithm

For optimization, we start from (3.2), define the weighting diagonal matrix $\mathbf{W}$ with $\mathbf{W}_{jj} = w(r_j)$ and use our residual definition (3.13) to write

$$\min_{\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{W}^{1/2} r(\boldsymbol{\xi})\|^2 \tag{3.16}$$

and replacing $r(\boldsymbol{\xi})$ with the linearized version yields in

$$\min_{\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{W}^{1/2} r(\boldsymbol{\xi})\|^2 \approx \frac{1}{2}\|\mathbf{W}^{1/2}(r(\boldsymbol{\xi}_0) + \mathbf{J}(\boldsymbol{\xi} - \boldsymbol{\xi}_0))\|^2 = \frac{1}{2}\|\mathbf{W}^{1/2}(r(\boldsymbol{\xi}_0) + \mathbf{J}(\Delta\boldsymbol{\xi}))\|^2 \tag{3.17}$$

where $\Delta\boldsymbol{\xi} = \boldsymbol{\xi} - \boldsymbol{\xi}_0$. By looking at (3.17) we see that it is the weighted least-squares solution of the linear system $\mathbf{W}(\mathbf{Ax} + \mathbf{b}) = \mathbf{0}$, where $\mathbf{A} = \mathbf{J}$, $\mathbf{x} = \Delta\boldsymbol{\xi}$ and $\mathbf{b} = r(\boldsymbol{\xi})$. Therefore, we solve it by deriving w.r.t $\Delta\boldsymbol{\xi}$ and setting to zero

$$\begin{aligned} \frac{\partial \frac{1}{2}\mathbf{W}\|\mathbf{J}\Delta\boldsymbol{\xi} + r(\boldsymbol{\xi}_0)\|^2}{\partial \Delta\boldsymbol{\xi}} &= \mathbf{J}^{\mathrm{T}}\mathbf{W}(\mathbf{J}\Delta\boldsymbol{\xi} + r(\boldsymbol{\xi}_0)) = \mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}\Delta\boldsymbol{\xi} + \mathbf{J}^{\mathrm{T}}\mathbf{W}r(\boldsymbol{\xi}_0) = 0 \\ \mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}\Delta\boldsymbol{\xi} &= -\mathbf{J}^{\mathrm{T}}\mathbf{W}r(\boldsymbol{\xi}_0) \\ \Delta\boldsymbol{\xi} &= -(\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J})^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{W}r(\boldsymbol{\xi}_0). \end{aligned} \tag{3.18}$$

Note that here a huge performance gain can be achieved. The Jacobian matrix $\mathbf{J} \in \mathbb{R}^{MN\times 6}$ has dimensions $MN \times 6$ where M is the height and N the width of the image. $\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J} \in \mathbb{R}^{6\times 6}$, on the other hand, is much smaller. Here we want to point out that the product $\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}$ can be computed without storing the potentially huge matrix $\mathbf{J}$ explicitly. This is achieved by summing the outer products of the rows of $\mathbf{J}$, i.e.

$$\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J} = \sum \mathbf{W}_{jj}\mathbf{J}_j^{\mathrm{T}}\mathbf{J}_j \tag{3.19}$$

where $\mathbf{J}_j$ is the j-th row of $\mathbf{J}$. Furthermore, we know $\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}$ is symmetric, which means that we can improve the performance even further since we need to compute and store only to upper/lower triangular part having solely 21 instead of 36 values.

3.2.2.2 Multi Level Tracking

The linearization performed in (3.17) is only valid for small $\boldsymbol{\xi}$. This means that only small translational and rotational motions can be handled. In order to overcome this drawback, we apply a *coarse-to-fine* scheme by using image pyramids. The lowest layer gets initialized with the original image having $M \times N$ pixels. A scaling factor α; $0 < \alpha < 1$ specifies how much the image resolution is reduced in each layer with $M_{i+1} = \alpha M_i$ and $N_{i+1} = \alpha N_i$. After determining the image resolution on the higher level, the intensity image get scaled down using bilinear interpolation. For the depth pyramid another scaling method is used. Here a $k \times k$ window is used to count the number of entries having a valid depth value, i.e. $z > 0$. Now the depth image gets downsampled and normalized by that number. In that case we are not introducing non-existing depth values during scaling. The scaling is repeated until a defined minimum size is reached.

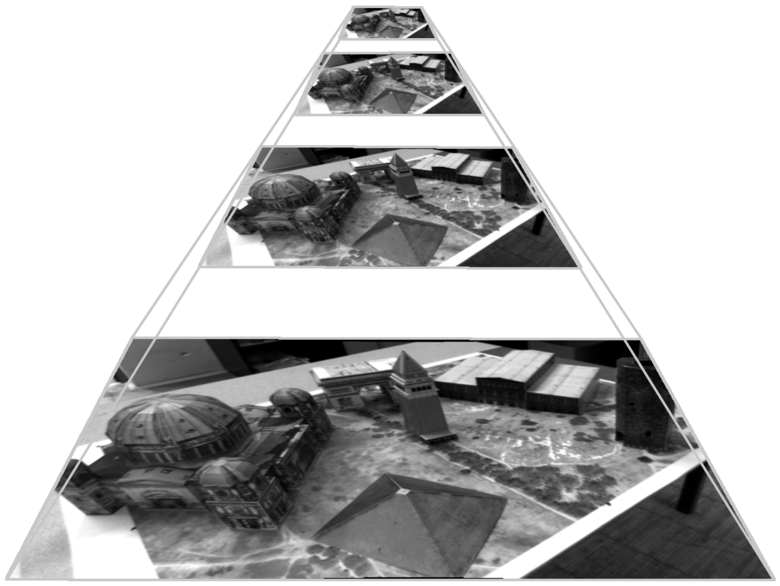


Figure 3.4: Schematic illustration of an image pyramid with four levels. Each level gets scaled down by scaling factor α until a defined minimum size is reached. Tracking starts at the highest level and uses the current estimate as an initialization for the next level.

The tracking algorithm starts now on the highest level having the lowest resolution. This way, big motions can be detected, although the estimates are still imprecise. After the algorithm converged on the higher level, it continues with the next level using the current estimate determined in the higher level as an initial estimate. Finally, the lowest level with the original image is processed.

3.3 Depthmap Generation

When computing the depth of a scene a large amount of data is always beneficial in order to be robust and get high quality results. Therefore one can roughly choose between two types of approaches when real-time requirements have to be met. Either a lot of fast but relatively inaccurate depthmaps are computed using simple two-view stereo algorithms, which are then fused together in a 3D reconstruction volume increasing the robustness and reconstruction quality. Or more information from multiple images is used already in the depthmap generation step using muli-view stereo algorithms. Since the resulting depthmaps can be again fused in a volume, multi-view stereo algorithms have a higher potential to deliver good results. Therefore, we seek a stereo algorithm with the following properties:

Multi-view

The algorithm should use the information of multiple images but it should be possible to get meaningful results from solely an image pair.

No preconditions

The stereo algorithm should not assume that the input images meet any kind of preconditions like rectification, no blur etc.

Illumination changes

Since the system is used in mobile devices the user might operate near to a window or lamp, therefore the algorithm must be robust against illumination changes.

Large baseline

Robustness against large baselines is required because it cannot be predicted how fast the user will move the device.

Computation speed

For running the system on a mobile device, the stereo algorithm needs to be fast since computing capabilities are limited.

The *planesweep algorithm* introduced in [3] fits these requirements. In the work of [4], the system got adopted and they showed that the algorithm can be run parallelized, decreasing the computation time even further.

3.3.1 Planesweep

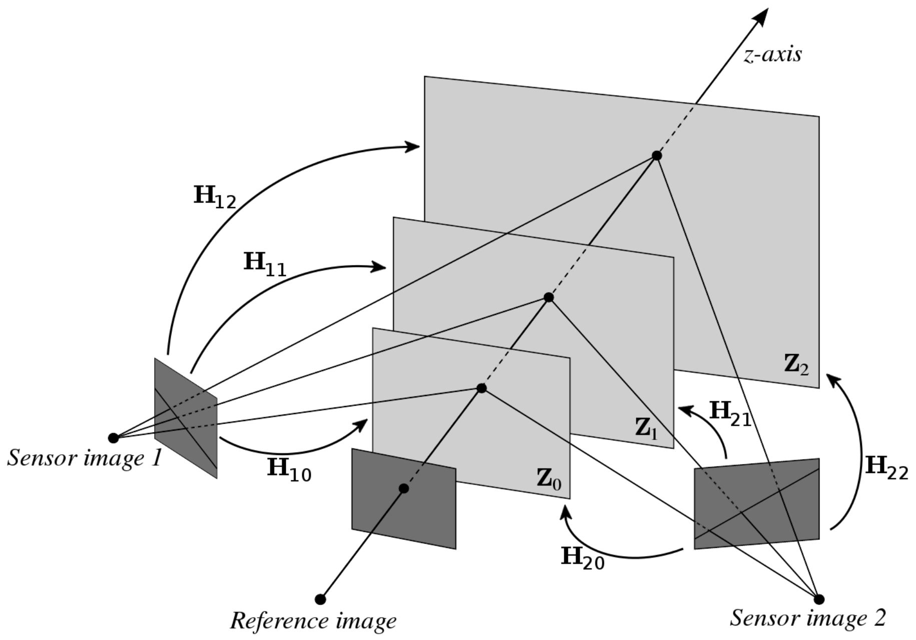


Figure 3.5: Planesweep (Figure adapted from [4]).

The planesweep is a well known stereo algorithm and it works, as its name suggests, by sweeping a number of planes through the 3D space. The basic principle of the algorithm is the following (see Figure 3.5 for an illustration). Starting from the reference image plane, other planes are located parallel to it at different depths by following the z-axis of the reference coordinate system. The inserted planes are bound within a $\mathbf{Z}_{near}$ and $\mathbf{Z}_{far}$ plane, which define the volume. For each depth, a homography is calculated between the corresponding plane and each of the sensor images (therefore the camera pose has to be known for each image). The algorithm measures now the similarity error between the reference image and each of the sensor images transformed to the planes at different depths. This makes sense, because as the surface of an object in the scene is passed through by a plane, the projected pixel values of the reference image and the sensor images transformed onto that plane match. In that way a cost volume is built up with the similarity errors at different depths. Finally, the depth of the plane, for which the cost is minimal, is assigned to each pixel individually. This is generally known as the winner takes all (*WTA*) algorithm.

3.3.1.1 Homography Calculation

Given a reference camera at origin $\mathbf{C}_{ref} = [\mathbf{I} \mid \mathbf{0}]$, a point $\mathbf{x} = [x, y, z]^{\mathrm{T}}$ in the reference view, a plane $\boldsymbol{\pi} = (\mathbf{v}^{\mathrm{T}}, 1)^{\mathrm{T}}$ and a second sensor camera with arbitrary position and orientation in 3D space $\mathbf{C}_{sens} = [\mathbf{R}_{sens} \mid \mathbf{t}_{sens}]$. The homography $\mathbf{H}$ induced by the plane $\boldsymbol{\pi}$ is now calculated by back-projecting $\mathbf{x}$ and calculating the point of intersection with $\boldsymbol{\pi}$ resulting in an intersection point $\mathbf{X}_{\boldsymbol{\pi}}$. $\mathbf{X}_{\boldsymbol{\pi}}$ is now projected into the second camera $\mathbf{C}_{sens}$ resulting in a point on the second image plane $\mathbf{x}'$ (Figure 3.6 is depicting the process).

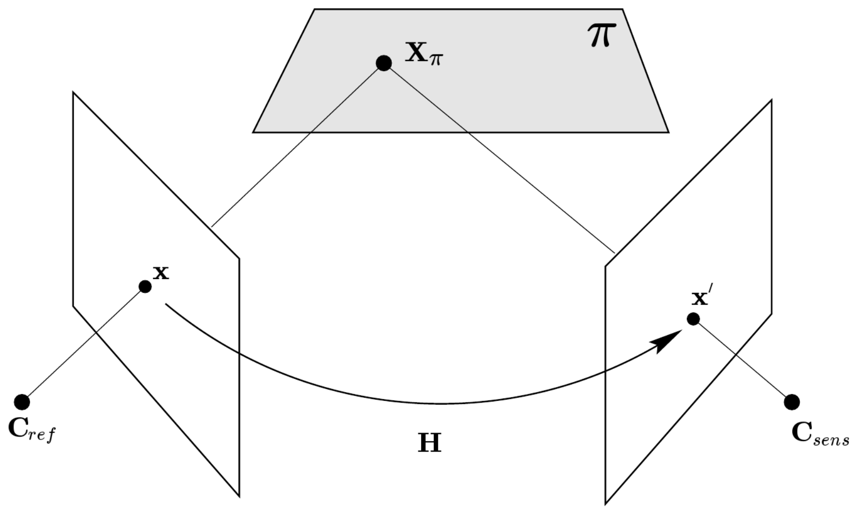


Figure 3.6: Homography induced by a plane. (Figure adapted from [17]).

The composition of the perspectivities between the two image planes and $\boldsymbol{\pi}$ is the homography $\mathbf{H}$ and is given by

$$\begin{aligned} \mathbf{x} &= \mathbf{H}_{ref\boldsymbol{\pi}}\mathbf{X}_{\boldsymbol{\pi}} \\ \mathbf{x}' &= \mathbf{H}_{\boldsymbol{\pi}}\mathbf{X}_{\boldsymbol{\pi}} = \mathbf{H}_{\boldsymbol{\pi}}\mathbf{H}_{ref\boldsymbol{\pi}}^{-1}\mathbf{x} = \mathbf{H}\mathbf{x} \end{aligned} \tag{3.20}$$

Now we know how the homography is theoretically built. We will see now how to actually compute $\mathbf{H}$. The reference camera is at the origin, therefore all points on the ray $\mathbf{X} = (\mathbf{x}^{\mathrm{T}}, \alpha)$ project to $\mathbf{x}$, where α parameterizes the 3D points on the viewing ray. Since we wish to find the point of intersection $\mathbf{X}_{\boldsymbol{\pi}}$ between the ray $\mathbf{X}$ and the plane $\boldsymbol{\pi}$, $\boldsymbol{\pi}^{\mathrm{T}}\mathbf{X} = 0$ must hold. This determines now α to $-\mathbf{v}^{\mathrm{T}}\mathbf{x}$ and the point of intersection

$\mathbf{X}_{\boldsymbol{\pi}} = (\mathbf{x}^{\mathrm{T}}, -\mathbf{v}^{\mathrm{T}}\mathbf{x})^{\mathrm{T}}$. The projection into the second view is therefore given by

$$\begin{aligned} \mathbf{x}' &= [\mathbf{R} \mid \mathbf{t}]\mathbf{X}_{\boldsymbol{\pi}} \\ \mathbf{x}' &= \mathbf{R}\mathbf{x} - \mathbf{t}\mathbf{v}^{\mathrm{T}}\mathbf{x} = (\mathbf{R} - \mathbf{t}\mathbf{v}^{\mathrm{T}})\mathbf{x} = \mathbf{H}\mathbf{x} \end{aligned} \tag{3.21}$$

where the homography is given by $\mathbf{H} = (\mathbf{R} - \mathbf{t}\mathbf{v}^{\mathrm{T}})$. If both images originate from the same calibrated camera, the intrinsic calibration matrix $\mathbf{K}$ can be used to get the homography for image pixel correspondences by

$$\mathbf{H} = \mathbf{K}(\mathbf{R} - \mathbf{t}\mathbf{v}^{\mathrm{T}})\mathbf{K}^{-1}. \tag{3.22}$$

In the general case, the reference camera does not necessarily have to be at the origin but can be located and rotated arbitrarily in the 3D space, i.e. $\mathbf{C}_{ref} = [\mathbf{R}_{ref} \mid \mathbf{t}_{ref}]$. By calculating the relative motion $\mathbf{R}_{rel}, \mathbf{t}_{rel}$ between the reference and sensor camera and by using that in 3.22, the homography for the general case can be computed. The relative motion is given by

$$\mathbf{R}_{rel} = \mathbf{R}_{sens}\mathbf{R}_{ref}^{-1} \tag{3.23}$$

$$\mathbf{t}_{rel} = \mathbf{t}_{sens} - \mathbf{R}_{rel}\mathbf{t}_{ref}. \tag{3.24}$$

Since the reference camera is not anymore in the origin, the sweeping planes have to be defined also in the coordinate system of the reference camera. With an unit vector $\mathbf{v} = [0, 0, 1]^{\mathrm{T}}$ along the z-axis and a distance z, the planes are now defined with $\boldsymbol{\pi}(z) = (\mathbf{v}^{\mathrm{T}}, -z)^{\mathrm{T}} = (\frac{\mathbf{v}^{\mathrm{T}}}{-z}, 1)^{\mathrm{T}}$ (see Figure 3.6). Using this and (3.23), (3.24) the homography for the general case is finally defined by

$$\mathbf{H} = \mathbf{K}\left(\mathbf{R}_{rel} - \frac{\mathbf{t}_{rel}\mathbf{v}^{\mathrm{T}}}{-z}\right)\mathbf{K}^{-1}. \tag{3.25}$$

3.3.1.2 Matching Methods

The planesweep algorithm measures the similarity error between the reference image and each of the sensor images mapped to the planes at different depths. The specific matching method defines how the similarity for a given pixel is actually computed between the two images. The following listing shows which matching methods are used in this work:

Single Pixel

Single pixel matching is the simplest method where the absolute difference of the pixel values in the two images give the cost: $c = |\mathbf{I}_{ref}(x, y) - \mathbf{I}_{sens}(x, y)|$

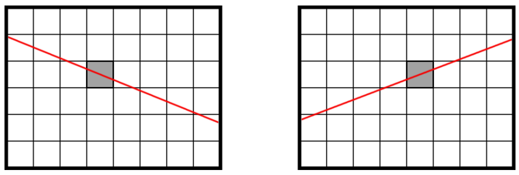


Figure 3.7: Single pixel matching

1 × 5 Epipolar Window

This is method requires the computation of the epipolar line in the reference image. This is done by calculating the plane spanned by the two camera centers and the current pixel. This plane is then intersected with the reference's image plane (see 2.6). The matching cost is then given by the absolute difference of the interpolated pixel values within the 1 × 5 window $\mathcal{W}$ along the epipolar line: $c = \sum_{x,y \in \mathcal{W}} |\mathbf{I}_{ref}(x, y) - \mathbf{I}_{sens}(x, y)|$

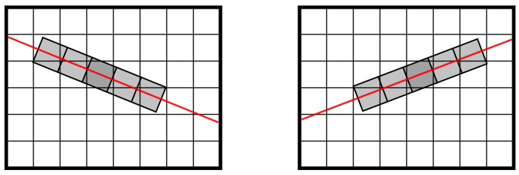


Figure 3.8: 1 × 5 epipolar window matching

3 × 5 Epipolar Window

Similar to the forgoing method with the only difference that the matching window $\mathcal{W}$ is now 3 × 5. Using a bigger window makes the matching more robust but fine details are lost. The cost is again given by: $c = \sum_{x,y \in \mathcal{W}} |\mathbf{I}_{ref}(x, y) - \mathbf{I}_{sens}(x, y)|$

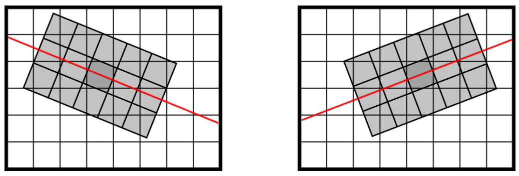


Figure 3.9: 3 × 5 epipolar window matching

3 × 3 Zero Mean

Using a 3 × 3 zero mean method as a similarity measure makes the matching more robust against lighting changes. The matching cost is given by:
$c = |(\mathbf{I}_{ref}(x, y) - \bar{\mathbf{I}}_{ref}(x, y)) - (\mathbf{I}_{sens}(x, y) - \bar{\mathbf{I}}_{sens}(x, y))|$
where $\bar{\mathbf{I}}_{ref}, \bar{\mathbf{I}}_{sens}$ are the 3 × 3 mean images. Note that we are precalculating the mean images to speed up the computation, therefore the window is aligned along the image.

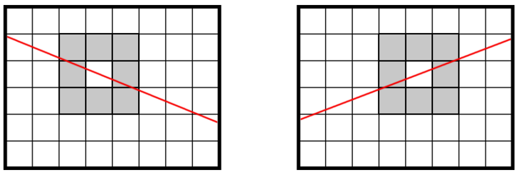


Figure 3.10: 3 × 3 zero mean matching

3.3.1.3 Speedup

The planesweep algorithm places planes at different depths parallel to the reference image plane. Afterwards, a homography is calculated for each depth and for each sensor image. Having d depth planes and n sensor images, $d \times n$ homographies have to be computed. This can be reduced to $2 \times n$ computations by using the following idea. Since the inserted planes are following the z-axis of the reference coordinate system, they lie on a line. This means that the mapping by the homography $\mathbf{H}$ of the point x in each depth plane onto one sensor image, results in points which lie again on a line. When the inserted planes are equidistant to each other, their mappings in the sensor image are not. Mappings of far planes are located closer to each other. By using an *inverse* step size for the distance between the planes in 3D space, equidistant projections can be achieved. The inverse step size z_{inv} is defined as

$$\begin{aligned} \psi_{far} &= \frac{1}{z_{near}} \\ \psi_{near} &= \frac{1}{z_{far}} \\ z_{inv} &= \frac{\psi_{far} - \psi_{near}}{d} \end{aligned} \tag{3.26}$$

where z_{near} and z_{far} are the depth values of the near and far planes and d is the number of inserted planes. Now the homography $\mathbf{H}_0$ for the first plane is calculated and also the homography $\mathbf{H}_1$ for the second plane, which is z_{inv} away from the first one. The 2D point $\mathbf{x} = (x, y)^{\mathrm{T}}$ in the reference image gets transformed by $\mathbf{H}_0$ and $\mathbf{H}_1$ resulting in 2D point $\mathbf{x}_0$ and $\mathbf{x}_1$ in the sensor image. The displacement vector $\mathbf{v}$ within the sensor image is therefore given by their difference, i.e. $\mathbf{v} = \mathbf{x}_1 - \mathbf{x}_0$. The projections of the different depth planes can therefore be calculated easily by starting at $\mathbf{x}_0$ and going n times the vector $\mathbf{v}$.

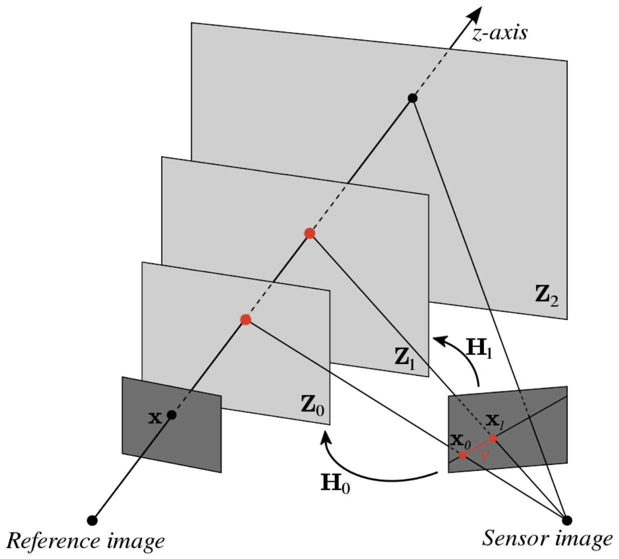


Figure 3.11: Speeding up the planesweep by using an *inverse* step size between the planes in 3D space. This gives equidistant projections in the sensor image which gives a start point $\mathbf{x}_0$ and a displacement vector $\mathbf{v}$ in the sensor image. By using $\mathbf{x}_0$ and $\mathbf{v}$ every depth mapping can be computed easily.

3.3.2 Adaptive Planesweep

The cost volume used by the planesweep algorithm is defined by the near plane $\mathbf{Z}_{near}$ and the far plane $\mathbf{Z}_{far}$ and all inserted planes lie within that range. The resolution of the depthmap is therefore depending on n, the number of planes inserted. Assuming that the range between $\mathbf{Z}_{near}$ and $\mathbf{Z}_{far}$ is not too big, n needs to be large, i.e. 1024 or 2048, in order to achieve smooth results. The obvious drawback is that the computation time is increasing drastically, though. The idea behind the adaptive planesweep is that homographies are calculated for each pixel individually. This means that the planes within the cost volume do not exist in the described way anymore. Instead of having one plane per depth, $m \times n$ planes are used, where m and n are the image dimensions. So each pixel has its own depth planes and therefore also its own near and far plane. The adaptive planesweep follows an iterative scheme and is outlined in algorithm 3.1.

Algorithm 3.1 Adaptive planesweep algorithm

```
1: Calculate the initial depthmap using the basic planesweep algorithm as explained in 3.3.1
2: Set $z0_{near} = z_{near}$ and $z0_{far} = z_{far}$
3: Use the depthmap from step 1 and set it as the current depthmap $\mathbf{DM}_{cur}$
4: Apply a $3 \times 3$ median filter to $\mathbf{DM}_{cur}$
5: Update $\mathbf{Z}_{near}$ and $\mathbf{Z}_{far}$ based on $\mathbf{DM}_{cur}$, the shrinking factor $\beta$ and the current iteration $j$:
   $\mathbf{Z}_{near}(x,y) = \min(z0_{near},\ \max(z0_{far},\ \mathbf{DM}_{cur}(x,y) - \frac{\beta}{j}))$
   $\mathbf{Z}_{far}(x,y) = \min(\mathbf{Z}_{near}(x,y),\ \max(z0_{far},\ \mathbf{DM}_{cur}(x,y) + \frac{\beta}{j}))$
6: Calculate new plane depths $\mathbf{Z}_i(x,y)$ based on the updated $[\mathbf{Z}_{near}, \mathbf{Z}_{far}]$ range
7: Compute new current depthmap $\mathbf{DM}_{cur}$ by using $\mathbf{Z}_i(x,y)$ (rest stays the same as explained in 3.3.1)
8: If depthmap is smooth enough return otherwise goto step 4
```

Since the near plane $\mathbf{Z}_{near}$ and the far plane $\mathbf{Z}_{far}$ are adjusting adaptively, the range, within which the depth planes are lying in, is shrinking. The shrinking speed is controlled by the shrinking factor β. In order to stabilize the algorithm, outliers within the depthmap are removed by applying a 3×3 median filter. In that case the near and far plane adaptation gets more robust. By reducing the range, defined by $\mathbf{Z}_{near}$ and $\mathbf{Z}_{far}$, the depthmap resolution is increasing whereas the computational time stays the same since the number of inserted planes n stays the same.

3.3.3 Volume Optimization

The adaptive planesweep algorithm improves the quality of the coarse initial depthmap by adjusting the near and far plane of the cost volume for each pixel adaptively and individually. For gradient rich image regions this approach works well since a lot of information is available and furthermore, the amount of outliers is also decreased by a median filter. The

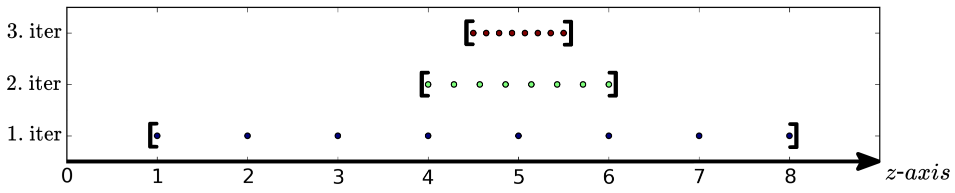


Figure 3.12: Adaptive planesweep - the depth is assumed to be at $z = 5$ and shrinking factor $\beta = 1$. In each iteration the depth resolution for the pixel gets increased because the near and far plane adaptively adjusts.

problem of this approach is though, that each pixel is treated individually and no global solution is found. This problem is especially notable in image regions with few features since homogeneous regions do not give a discriminative enough photometric similarity. Therefore, we mostly follow [29] where a total variational based approach is used to optimize the depthmap where

- the *data term* is defined by the photometric similarity error
- the *regularization term* is defined by the smoothness constraint

So the key idea is that featureless image regions are more likely to result in false minima and therefore a regularization term has to penalize deviations from a spatially smooth depthmap but in the same time edges and discontinuities have to be preserved.

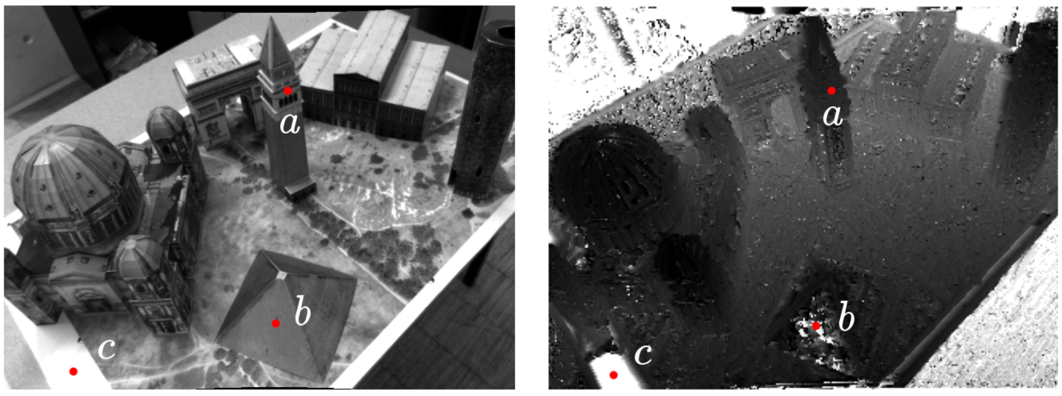


Figure 3.13: Featureless image regions are prone to false minima. The points a, b, c show different well textured parts. The corresponding depthmap is depicted on the right.

In Figure 3.13 an inverse depthmap is extracted from the costvolume $\mathbf{C}$ which is built by the planesweep algorithm. The inverse depth is calculated by taking the plane within $\mathbf{C}$ where the matching costs are minimal, i.e. by computing $\arg\min_z \mathbf{C}(\mathbf{x}, z)$ for every pixel $\mathbf{x}$ within the reference image. Since the matching cost within $\mathbf{C}$ is already the sum of individual two view stereo matchings it contains more robust information. Using this, the variational cost volume optimization model is given by

$$\min_{\boldsymbol{\psi}} \left\{ \int_{\Omega} |\nabla \boldsymbol{\psi}(\mathbf{x})|_{\epsilon} + \lambda \mathbf{C}(\mathbf{x}, \boldsymbol{\psi}(\mathbf{x})) \; d\mathbf{x} \right\} \tag{3.27}$$

where $\boldsymbol{\psi}(\mathbf{x}) : \Omega \rightarrow \mathbb{R}$ is the depthmap, $\mathbf{x} \in \Omega$ are pixel coordinates and $\Omega \subset \mathbb{R}^2$ is the image domain. The first term is the regularization term which enforces smoothness of the solution by penalizing deviations from a spatially smooth depthmap. The second term is the data term defined by the photometric similarity error within the cost volume $\mathbf{C}$. The regularization parameter λ is controlling the smoothness of the resulting depthmap.

3.3.3.1 Regularization

The choice of the regularizer is crucial since a reconstructed inverse depthmap of a typical scene consists of smooth regions together with sharp discontinuities due to occlusions. The *Huber norm* over the gradient of the inverse depthmap is used , i.e. $|\nabla \boldsymbol{\psi}(\mathbf{x})|_{\epsilon}$,since it perfectly fits our needs. The Huber norm, a composition of two convex functions, is defined as

$$|x|_{\epsilon} = \begin{cases} \frac{|x|_2^2}{2\epsilon}, & \text{if } |x| \leq \epsilon \\ |x|_1 - \frac{\epsilon}{2}, & \text{otherwise} \end{cases} \tag{3.28}$$

where $\epsilon > 0$ is a small parameter controlling the tradeoff between a quadratic regularizer and a total variation (TV) regularizer. Therefore, if $|\nabla \boldsymbol{\psi}| \leq \epsilon$ a L_2^2 norm is used, resulting in a smooth reconstruction while otherwise a L_1 norm allows sharp depth discontinuities at depth edges. Compared to a pure TV regularizer the Huber norm has another advantage. By setting ϵ to a small value the stair-casing effect is reduced (which is a typical artifact of the TV regularizer).

3.3.3.2 Optimization

Figure 3.14 shows that the Huber norm is a convex function whereas the photometric similarity error function is not, i.e. the composition of these two terms in (3.27) is a *non-convex* function. In order to cope with that, several possible solutions exist. The

$$\min_{\boldsymbol{\psi}} \left\{ \int_\Omega \underbrace{|\nabla \boldsymbol{\psi}(\mathbf{x})|_\epsilon}_{\text{convex}} + \underbrace{\lambda \mathbf{C}(\mathbf{x}, \boldsymbol{\psi}(\mathbf{x}))}_{\text{non-convex}} \, d\mathbf{x} \right\}$$

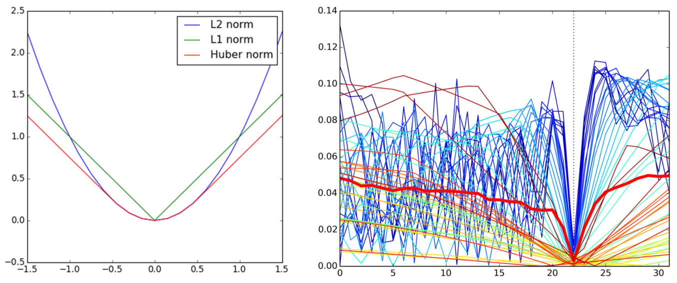


Figure 3.14: Convex Huber norm versus non-convex photometric cost.

typical way to get a convex approximation is to linearize the cost volume. Afterwards this approximation is solved iteratively within a coarse-to-fine pyramid scheme. The drawback of this approach is though, that this might lead to a loss of details in the reconstruction. Like [29], we follow another key observation. The data term can be globally optimized by performing a trivial point-wise exhaustive search over the full range of possible inverse depth values within the costvolume. Also, we can solve the convex regularization term efficiently by using convex optimization algorithms. Finally, [36] showed that the energy functional in (3.27) can be approximated by decoupling the data and regularization term. In our case the coupling is achieved by using an auxiliary variable $\boldsymbol{\beta} : \Omega \to \mathbb{R}$. The resulting variational optimization model is therefore given by

$$\min_{\boldsymbol{\psi},\boldsymbol{\beta}} \left\{ \int_\Omega |\nabla \boldsymbol{\psi}(\mathbf{x})|_\epsilon + \frac{1}{2\Theta}(\boldsymbol{\psi}(\mathbf{x}) - \boldsymbol{\beta}(\mathbf{x}))^2 + \lambda \mathbf{C}(\mathbf{x}, \boldsymbol{\beta}(\mathbf{x})) \, d\mathbf{x} \right\} \tag{3.29}$$

where the coupling term $\mathbf{A}(\mathbf{x}) = \frac{1}{2\Theta}(\boldsymbol{\psi}(\mathbf{x}) - \boldsymbol{\beta}(\mathbf{x}))^2$ is used to drive both, the original and auxiliary variable, together, i.e. that $\boldsymbol{\psi} = \boldsymbol{\beta}$ as $\theta \to 0$. If $\boldsymbol{\psi} = \boldsymbol{\beta}$ is fulfilled, we see that (3.29) results in the original optimization model (3.27). At a first glance it seems that the model got more complicated now because instead of having one optimization problem in $\boldsymbol{\psi}$ we have to solve two coupled optimization problems in $\boldsymbol{\psi}$ and $\boldsymbol{\beta}$. In fact, the optimization got easier as we will see now. Since we are optimizing for two variables, the optimization process is split into two sub problems:

- The model can be globally minimized w.r.t. $\boldsymbol{\psi}$. By looking at (3.29) we can see that the first two terms of the model, $\int_\Omega |\nabla \boldsymbol{\psi}(\mathbf{x})|_\epsilon + \mathbf{A}(\mathbf{x})\; d\mathbf{x}$, are a slightly modified version of the TV-L_2^2 ROF denoising model [34]. Since it is convex in $\boldsymbol{\psi}$ we can perform an efficient optimization using a primal-dual approach [2].

- The model can be globally minimized w.r.t. $\boldsymbol{\beta}$. Although it is non-convex in the auxiliary variable $\boldsymbol{\beta}$, the second two terms, $\int_\Omega \mathbf{A}(\mathbf{x}) + \lambda \mathbf{C}(\mathbf{x}, \boldsymbol{\beta}(\mathbf{x}))\; d\mathbf{x}$, can be globally optimized by performing a trivial point-wise exhaustive search over the full range of possible inverse depth values within the costvolume.

3.3.3.3 ROF Model

The first two terms of (3.29), $\int_\Omega |\nabla \boldsymbol{\psi}(\mathbf{x})|_\epsilon + \mathbf{A}(\mathbf{x})\; d\mathbf{x}$, are a slightly modified version of the TV-L_2^2 ROF denoising model. By setting $\lambda = \frac{1}{2\Theta}$, $\mathbf{u} = \boldsymbol{\psi}$, $\mathbf{f} = \boldsymbol{\beta}$ and using a $L1$ norm for the regularizer we finally arrive at the TV-L_2^2 ROF image denoising model which is defined as

$$\min_{\mathbf{u}} \left\{ \int_\Omega |\nabla \mathbf{u}| + \frac{\lambda}{2}(\mathbf{u} - \mathbf{f})^2 \; d\mathbf{x} \right\} \tag{3.30}$$

We will first see how the standard ROF model is solved and afterwards show how to handle the Huber norm for the regularization term. For optimization we use the first-order primal-dual algorithms proposed in [2].

The primal-dual formulation of (3.30) is given by

$$\min_{\mathbf{u}} \max_{||\mathbf{p}||_\infty \leq 1} \left\{ -\int_\Omega \mathbf{u} \;\mathrm{div}\; \mathbf{p} + \frac{\lambda}{2} \int_\Omega (\mathbf{u} - \mathbf{f})^2 \; d\mathbf{x} \right\} \tag{3.31}$$

and the corresponding discretized version reads

$$\min_{\mathbf{u}} \max_{\mathbf{p}} \left\{ -\langle \mathbf{u}, \mathrm{div}\; \mathbf{p} \rangle + \frac{\lambda}{2} ||\mathbf{u} - \mathbf{f}||^2 - I_{||\mathbf{p}||_\infty \leq 1}(\mathbf{p}) \right\} \tag{3.32}$$

where $\mathbf{f} \in \mathbb{R}^{MN}, \mathbf{u} \in \mathbb{R}^{MN}$ are images of size $M \times N$, $\mathbf{p} \in \mathbb{R}^{dMN}$ is the dual variable with $d = 2$ for 2D images and $I(\mathbf{p})$ is the indicator function of the convex set. By defining the non-linear functions $G(\mathbf{u}) = \frac{\lambda}{2}||\mathbf{u} - \mathbf{f}||^2$ and $F^*(\mathbf{p}) = I_{||\mathbf{p}||_\infty \leq 1}(\mathbf{p})$ the resolvent operators for the primal and dual update can be computed.

The solution to the resolvent operator for the primal update is given as the point-wise update

$$\mathbf{u} = (\mathbf{I} + \tau\sigma G)^{-1}(\tilde{\mathbf{u}}) \Longleftrightarrow \mathbf{u}_{i,j} = \frac{\tilde{\mathbf{u}}_{i,j} + \tau\lambda\mathbf{f}_{i,j}}{1 + \tau\lambda} \tag{3.33}$$

Since the constraint $||\mathbf{p}||_\infty \leq 1$ is modeled by the indicator function F^* for the dual variable $\mathbf{p}$, the resolvent operator reduces to a projection of the form

$$\mathbf{p} = (\mathbf{I} + \sigma\partial F^*)^{-1}(\tilde{\mathbf{p}}) \Longleftrightarrow \mathbf{p}_{i,j} = \frac{\tilde{\mathbf{p}}_{i,j}}{\max(1, |\tilde{\mathbf{p}}_{i,j}|)} \tag{3.34}$$

3.3.3.4 Huber-ROF Model

The basic primal-dual ROF model (3.31) can be easily extended to use a Huber normed regularizer [2]. For this, the non-linear function $F^*(\mathbf{p}) = I_{||\mathbf{p}||_\infty \leq 1}(\mathbf{p})$ is replaced by $F^*(\mathbf{p}) = I_{||\mathbf{p}||_\infty \leq 1}(\mathbf{p}) + \frac{\epsilon}{2}||\mathbf{p}||^2$. Therefore the formulation of the primal-dual Huber-ROF model is given by

$$\min_{\mathbf{u}} \max_{\mathbf{p}} \left\{ -\langle \mathbf{u}, \text{div } \mathbf{p} \rangle + \frac{\lambda}{2}||\mathbf{u} - \mathbf{f}||^2 - I_{||\mathbf{p}||_\infty \leq 1}(\mathbf{p}) - \frac{\epsilon}{2}||\mathbf{p}||^2 \right\} \tag{3.35}$$

Since $F^*(\mathbf{p})$ got replaced, the resolvent operator for the dual update is now given by the following point-wise update

$$\mathbf{p} = (\mathbf{I} + \sigma\partial F^*)^{-1}(\tilde{\mathbf{p}}) \Longleftrightarrow \mathbf{p}_{i,j} = \frac{\frac{\tilde{\mathbf{p}}_{i,j}}{1+\sigma\epsilon}}{\max(1, |\frac{\tilde{\mathbf{p}}_{i,j}}{1+\sigma\epsilon}|)} \tag{3.36}$$

3.3.3.5 Final Algorithm

Algorithm 3.2 Final depthmap computation algorithm

```
1: Initialize $\boldsymbol{\psi}$ and $\boldsymbol{\beta}$ by finding the minimal matching costs within the cost volume $\mathbf{C}$, i.e. by computing $\arg\min_z \mathbf{C}(\mathbf{x}, z)$ for every pixel $\mathbf{x}$.
2: Fix $\boldsymbol{\beta}$ and solve for $\boldsymbol{\psi}$ by using the primal-dual Huber-ROF model where the update steps are given by (3.33) and (3.36).
3: Fix $\boldsymbol{\psi}$ and solve for $\boldsymbol{\beta}$ by performing a point-wise exhaustive search over $\boldsymbol{\beta}$.
4: Update $\theta$ by $\theta^{i+1} = \theta^i(1 - \gamma n), i \leftarrow i + 1$ and go back to step 2 if $\theta^i > \theta_{stop}$, otherwise return.
```

The minimization itself is performed by optimizing for $\boldsymbol{\psi}$ and $\boldsymbol{\beta}$ alternately and consists of 4 steps. It is outlined in algorithm 3.2. The first step initializes the algorithm. Step (2) − (4) are performed n times until a stopping criteria is fulfilled. The parameter θ defines the coupling of $\boldsymbol{\psi}$ and $\boldsymbol{\beta}$ where a lower value of θ results in a tighter coupling. The decreasing rate of θ is defined by γ where a higher value is decreasing θ faster.

3.3.3.6 Increasing Accuracy

The proposed volume optimization algorithm minimizes a global spatially regularized energy functional. Hence, the resulting depthmap contains a low amount of outliers but it is still coarse unless we use a very high number of planes for the planesweep algorithm. In order to counteract that problem we can increase the accuracy of the depthmap without increasing the computational time. Figure 3.15 depicts the idea.

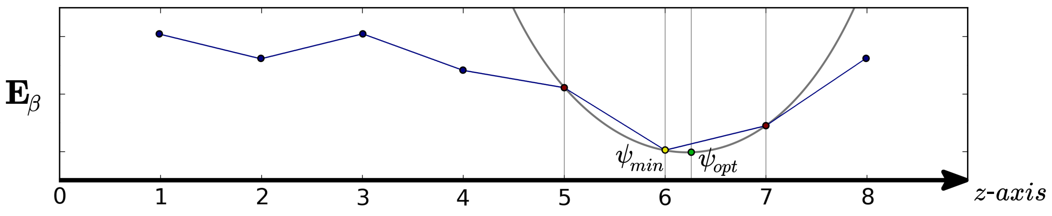


Figure 3.15: The minimum of a quadratic function constructed by the points around discrete minimum $\boldsymbol{\psi}_{min}$ of the auxiliary energy $\mathbf{E}_{\boldsymbol{\beta}}$ gives the optimal depth $\boldsymbol{\psi}_{opt}$.

The energy for the auxiliary variable is given by $\mathbf{E}_{\boldsymbol{\beta}}(\mathbf{x}) = \int_\Omega \mathbf{A}(\mathbf{x}) + \lambda \mathbf{C}(\mathbf{x}, \boldsymbol{\beta}(\mathbf{x}))\, d\mathbf{x}$ and the coupling term is given by $\mathbf{A}(\mathbf{x}) = \frac{1}{2\Theta}(\boldsymbol{\psi}(\mathbf{x}) - \boldsymbol{\beta}(\mathbf{x}))^2$. By computing the minimum of a quadratic function constructed by the points around discrete minimum $\boldsymbol{\psi}_{min}$ of the auxiliary energy $\mathbf{E}_{\boldsymbol{\beta}}$, we achieve subsample accuracy resulting in an optimal depth $\boldsymbol{\psi}_{opt}$ for the pixel $\mathbf{x}$. This is equal to performing a single Newton step of $\mathbf{E}_{\boldsymbol{\beta}}(\mathbf{x})$ around the current discrete minimum $\boldsymbol{\psi}_{min}$ which is given by

$$\boldsymbol{\psi}_{opt}(x) = \boldsymbol{\psi}_{min}(x) - \frac{\nabla \mathbf{E}_{\boldsymbol{\beta}}(x)}{\nabla^2 \mathbf{E}_{\boldsymbol{\beta}}(x)} \tag{3.37}$$

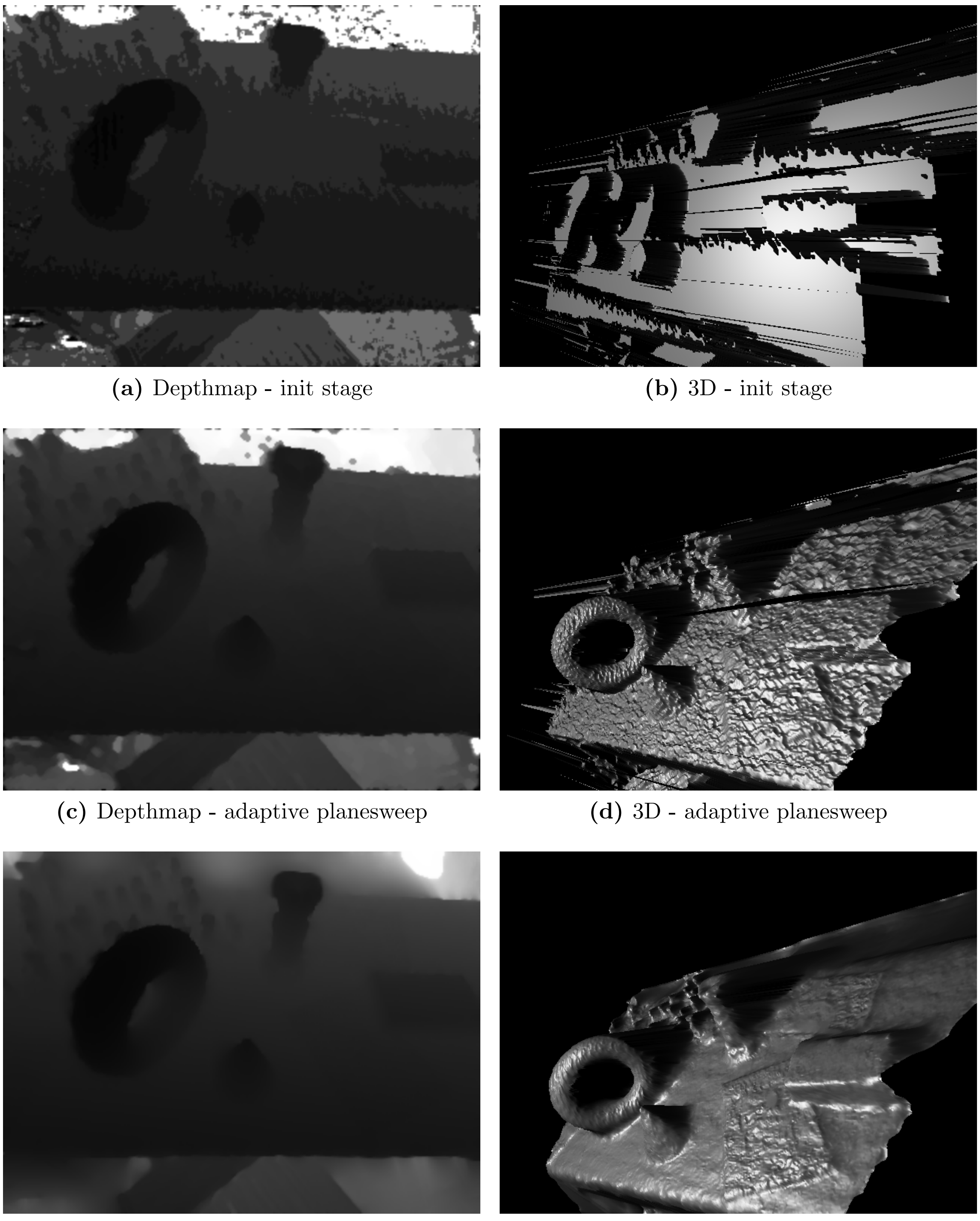
The geometric interpretation of the Newton's method is that at each iteration the original function is approximated around the current argument by a quadratic function and then a step towards the extremum of that quadratic function is taken. Furthermore, if the original function is quadratic, then the exact extremum is found in one step. As $\mathbf{A}(\mathbf{x})$ is modelled well with a parabola around the current discrete minimum, one Newton step is performed.

3.4 Visualization

The proposed reconstruction system is working iteratively. Therefore, we keep the user informed about the current state of the reconstruction by a live visual feedback. Depending on the user's choice, either the depthmap itself or a 3D model, calculated by back-projecting the depthmap into 3D space, is displayed. For the basic depthmap visualization a simple linear interpolation is performed. The near plane $\mathbf{Z}_{near}$ of the matching cost volume is black whereas the far plane $\mathbf{Z}_{far}$ is white. Thus, the resulting visualization is a linearly interpolated grayscale image.
For the 3D visualization, the depthmap first has to be back-projected into 3D space. This is done by using the inverse projection function defined as $\pi^{-1}(\mathbf{x}) = \mathbf{K}^{-1}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ where $\mathbf{x} = [x, y]^{\mathrm{T}} \in \mathbb{R}^2$ are 2D pixel coordinates and $\mathbf{K}$ is the intrinsic camera calibration matrix. The 3D model is now shaded by using the Phong shading model [31] which defines the resulting pixel intensity by

$$\begin{aligned} I_{out} &= I_a k_a + I_{in} k_d \ cos\phi + I_{in} k_s \ cos^n \theta \\ &= I_a k_a + I_{in} \left[k_d (L \cdot N) + k_s (R \cdot V)^n \right] \end{aligned} \tag{3.38}$$

where I_a is the intensity of the ambient light, k_a is the material constant, I_{in} is the intensity of the light source, k_d is the diffuse reflexion constant, k_s is the specular reflexion factor, L is the light direction, N is the surface normal, R is the outgoing light direction and V is the viewing direction. Finally, the 3D model can be also textured using the intensity image of the reference view.



(a) Depthmap - init stage

(b) 3D - init stage

(c) Depthmap - adaptive planesweep

(d) 3D - adaptive planesweep

(e) Depthmap - volume optimization

(f) 3D - volume optimization

Figure 3.16: Depthmap and 3D visualization of the initialization stage, the adaptive planesweep algorithm and the volume optimization algorithm.

4 Implementation

Contents

The implementation details of our system are discussed in this chapter. First, we present our setup followed by our dense tracking implementation. We show two variants of how we store the data in the memory and how the performance critical parts are handled. Afterwards we present the depthmap generation and visualization parts. Finally, the whole system is discussed.

4.1 Setup

For dense tracking and dense depthmap computation we use highly parallelized state-of-the-art algorithms. To utilize all the resources of recent mobile devices, the GPU and the CPU are programmed to run our algorithms. Therefore, we have two major requirements which the device has to meet. First, we need a powerful GPU in order to have an interactive system which is capable of reconstructing the scene from a given viewpoint just within several seconds. Secondly, it must be possible to perform general purpose computations on the GPU (GPGPU). This can be achieved on any modern mobile device using OpenGL

ES[1]. Latest performance benchmarks showed that the Tegra K1[2] is currently one of the most powerful chips available. Therefore, we chose the Nvidia Shield Tablet[3] as our main platform since it is equipped with the K1 chip. Another feature offered by this device is the possibility to perform all GPGPU computations using CUDA[4]. This greatly fits our needs and makes the tablet a state-of-the-art device for mobile graphics computations. We use Android 5.0.1 as a target OS, CUDA for Android 6.0 and the Android Native Development Kit (NDK)[5] r10c for software development.

4.2 Dense Tracking

4.2.1 Memory

The actual dense tracking optimization is expressed in (3.18) and the corresponding calculation of the full Jacobian can be found in (3.12). We implemented two different versions which either handle

- valid points (with a depth > 0)
- all points

in the computation of the Jacobian. Since we are using a weighting scheme during the optimization, invalid points get downweighted and the final result is the same. The difference between those two implementations lies in number of points to be processed and therefore also in the number of memory accesses.
If all points are used, nicely aligned memory is given automatically by CUDA since we are using CUDA textures. In the other case, valid points have to be computed first and then stored in an appropriate memory structure. An illustration is given in Figure 4.1.

In (3.12) it can be seen that the 3D coordinates X, Y, Z, the pixels scaled focal lengths f_x, f_y, the pixel location x, y and the spatial derivatives d_x, d_y are needed. Furthermore we use one field v that signals if the point is valid or not. Since the focal length remains

[1]https://www.khronos.org/opengles/
[2]http://www.nvidia.com/object/tegra-k1-processor.html
[3]http://shield.nvidia.com/
[4]https://developer.nvidia.com/cuda-zone
[5]https://developer.android.com/tools/sdk/ndk/index.html

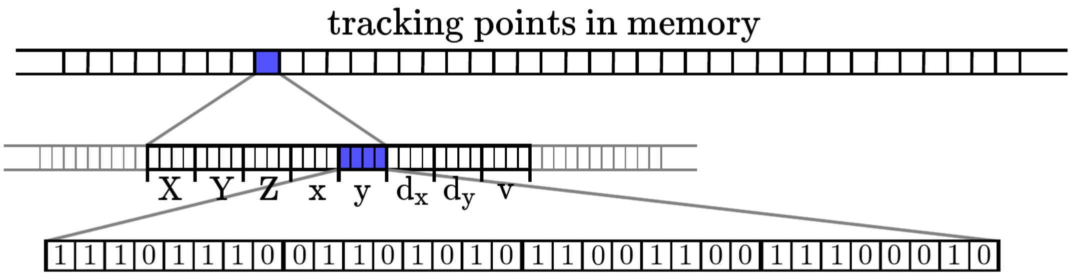


Figure 4.1: Schematic illustration of the aligned memory structure.

the same after calibration we don't have so save it for each point. What remains are 8 fields each taking 4 bytes. In order to reduce memory access we define the structure with a 16 byte memory alignment, i.e. with `__align__(16)`. In that way we can load the whole structure with a two load instruction.

4.2.2 Speed

Our tracking system uses a coarse-to-fine scheme as described in 3.2.2.2. The algorithm starts at the highest level of the pyramid which has the lowest resolution. Afterwards, it uses the current estimate of that level as an initial estimate for the next level. By using that, not only bigger translational and rotational motions can be handled, but also the iterations in lower levels can be reduced which in turn speeds up the process. Furthermore, we found empirically upper bounds for the maximum number of iterations for each pyramid level which can be seen in table 4.1.

Level	#Iterations
3	15
2	8
1	6
0	5

Table 4.1: Maximum number of tracking iterations per pyramid level.

The crucial part of the tracking system is given by (3.18). For n pyramid levels l_i with $i = 0, ..., n-1$ and $t(l_i)$ maximal iterations per level, (3.18) has to be solved $\sum_{i=0}^{n-1} t(l_i)$ times. The computationally costly part is the calculation of $\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}$ since the Jacobian matrix $\mathbf{J} \in \mathbb{R}^{MN\times 6}$ has dimensions $MN \times 6$ where M is the height and N the width of the image. By summing the outer products of the rows of $\mathbf{J}$, as described in (3.19), the computation time can be decreased. Since it is obvious now that this summation is the critical part of the whole tracking system, special attention has to be payed to the implementation. As we are using CUDA we can perform a parallel reduction to compute the sum efficiently. In order to exploit the full potential of the hardware our implementation of the summation includes

- Complete loop unroll
- Template function
- The shuffle down command `__shfl_down()` of the Nvidia Keplar architecture

Since CUDA supports C++ template parameters on both, device and host functions, we know the number of reductions that have to be performed. As template parameters are evaluated at compiletime, a complete loop unroll is possible. Furthermore we also know the upper bound, since the block size is limited by the GPU to 512 local threads. We also use Keplars shuffle down command which saves additional shared memory access, compared to the traditional shared memory implementation. As all instructions are SIMD synchronous within one warp consisting of 32 threads, no further synchronization has to be done. A schematic illustration of `__shfl_down()` with 8 threads can be seen in Figure 4.2.

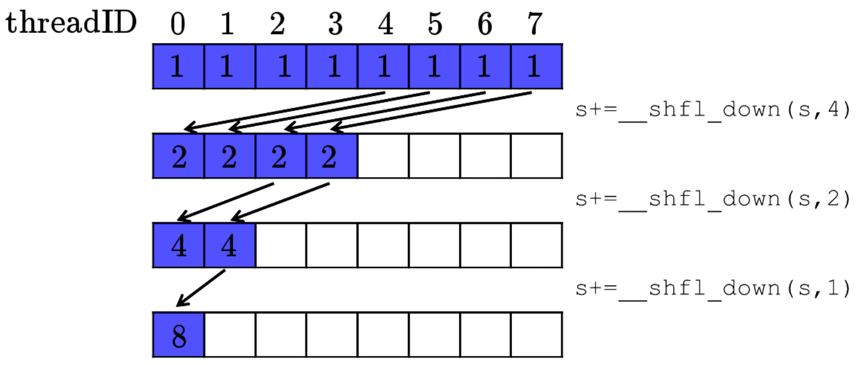


Figure 4.2: Schematic illustration of `__shfl_down()` with 8 threads.

The complete function is given in listing 4.1. Note that the same function is also used when computing $\mathbf{J}^{\mathrm{T}}\mathbf{W}r(\xi_0)$ of (3.18). In that way it is guaranteed that the computationally heavy parts are calculated optimally on the GPU. The resulting system of equations is small and easy to solve. Therefore we use the Sophus[6] library which is computing the

[6]https://github.com/stonier/sophus

solution on the CPU.

Listing 4.1: Local sum with unrolled loops and shuffle down command

```
template<unsigned int blockSize>
__device__ inline float localSumShuffle(volatile float* r, uint tID)
{
  // reduction space - r
  // local thread ID - tID

  float localSum = r[tID];

  // complete loop unroll
  if (blockSize >= 512)
  {
    if (tID < 256) r[tID] = localSum = localSum + r[tID + 256];
    __syncthreads();
  }
  if (blockSize >= 256)
  {
    if (tID < 128) r[tID] = localSum = localSum + r[tID + 128];
    __syncthreads();
  }
  if (blockSize >= 128)
  {
    if (tID < 64) r[tID] = localSum = localSum + r[tID + 64];
    __syncthreads();
  }

  // within one warp (=32 threads) instructions are SIMD synchronous
  // -> __syncthreads() not needed
  // we use new keplar shuffle down commands to save shared memory access
  if (tID < 32)
  {
    localSum += r[tID + 32];

    localSum += __shfl_down(localSum, 16);
    localSum += __shfl_down(localSum, 8);
    localSum += __shfl_down(localSum, 4);
    localSum += __shfl_down(localSum, 2);
    localSum += __shfl_down(localSum, 1);
  }

  __syncthreads();

  return localSum;
}
```

4.3 Depthmap Generation

The computation of the homography, which maps the sensor view onto the plane at depth z is given by (3.25). We perform the computation of the relative rotation and relative translation on the CPU using the Eigen[7] library. Due to the adaptive planesweep algorithm the homographies have to be calculated for each pixel individually since they can adjust their near and far plane of the cost volume individually. This is done in parallel on the GPU. Note that instead of saving the homographies explicitly we use the explained speedup 3.3.1.3 and only save the starting point x_0 and displacement vector v for each pixel. Building up the cost volume by using one on the matching methods from 3.3.1.2 and the depth extraction by performing the WTA is performed completely on the GPU as well.
The cost volume optimization is implemented straight forward. The computation of all relevant parts as explained in 3.3.3 is again performed on the GPU using CUDA.

4.4 Visualization

For all kinds of visual outputs other than the user interface, we are using OpenGL. The visualization of the depthmap is a simple grayscale image interpolated between z_{near} which is black and z_{far} which is white. For the 3D perspective we are back projecting the depthmap into 3D space as explained in 3.4 and create a mesh out of these points. Each pixel in the depthmap corresponds to one 3D point. In that case texture parameters can be set conveniently allowing OpenGL to handle texture interpolation for zooming. Furthermore, we display the camera positions of the sensor views as RGB crosses which can be seen in Figure 4.3.

[7]http://www.eigen.tuxfamily.org/

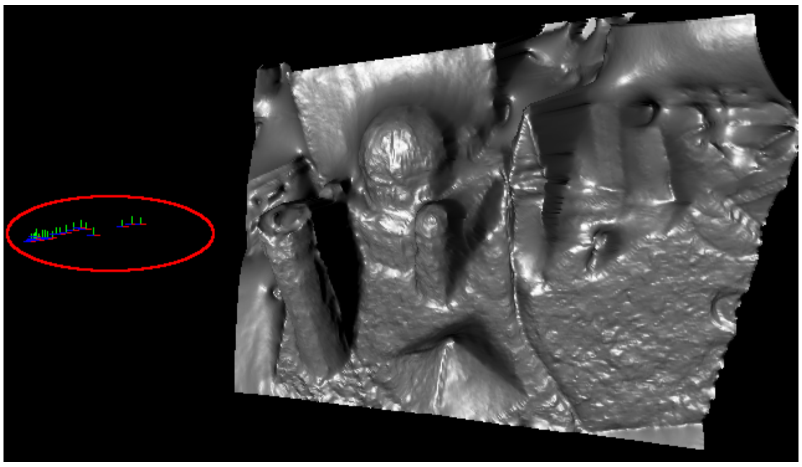


Figure 4.3: Camera positions of sensor views are displayed as RGB crosses.

4.5 System

In this section we illustrate the functionality of the complete system and how the individual parts work together. A screenshot of our developed Android application can be seen in Figure 4.4. By default, the system is using the internal camera to capture frames where the time difference between them can be adjusted by the corresponding slider. In order to reduce lighting changes the user can lock the camera exposure. After clicking *Set KF* the capture process starts. The input can also be changed to a set of precaptured images, which has to be selected in the menu, located in top right corner. After that, the system selects the middle frame as the reference frame and starts tracking the input frames with a homogeneous plane resulting in first rough pose estimates for the sensor images. The user can now select one of the available matching methods 3.3.1.2. The cost volume is built up and the depthmap is calculated using the WTA algorithm 3.3.1. For visualization, either the grayscaled depthmap can be shown or two different 3D views 3.4 are available. In the interactive view, the mesh is recalculated and updated each frame, making the system slow but the reconstruction can be seen live in 3D. The second 3D view is only for displaying the actual depthmap in 3D. Here, tracking and depthmap computation are turned off. Furthermore it is possible to select different viewing options for the 3D view - a phong shaded model, a textured model or a combination of both. The bottom row of buttons does not need any explanation since their names are self-explaining. With the *Auto Params* toggle button the computation stages are changed automatically.

First, the initialization phase is performed and then the system switches automatically into the volume optimization phase where the volume optimization 3.3.3 is performed and quadfitting 3.3.3.6 is activated.

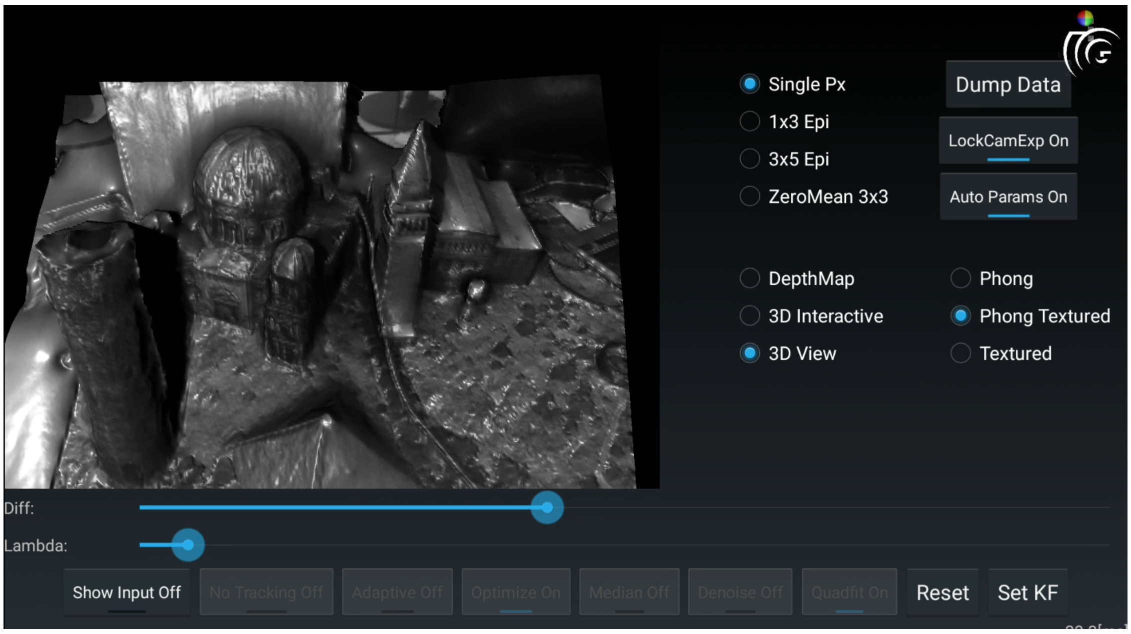


Figure 4.4: Screenshot of the developed Android application.

5 Evaluation

Contents

In this section we present the evaluation of our system. First, we give an explanation of the evaluation setup and the used data. Next, the tracking accuracy of our system is measured relative to ground truth camera poses. Furthermore, we see which parts of the tracking system are computationally expensive through runtime measurements. Finally, we investigate the influence of different reconstruction parameters on the depthmap results.

5.1 Setup

The evaluation of our system is performed on the hardware which was already described in the implementation section 4.1.

In order to get consistent and comparable results, it is crucial that the movement of the tablet and the camera input stream are the same for each run. Without special hardware it would be impossible to perform this by hand. Therefore we are using different sets of precaptured images. In order to evaluate the tracking part of our system separately, we have to have ground truth depthmaps additionally to the precaptured image sets. In that case the depthmap generation phase has no influence on the tracking results since it is bypassed. Since we did not have access to an external tracking system giving us the ground truth camera pose, we are using external datasets.

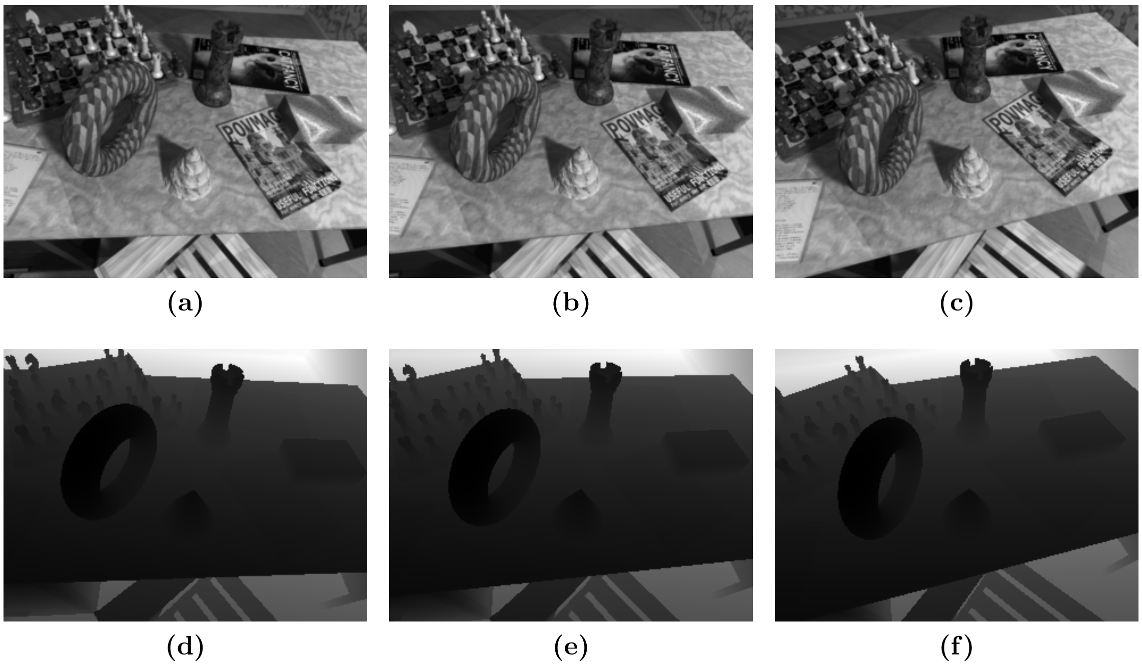


Figure 5.1: (a)-(c): First, reference and last intensity image of the POVRay scene. (d)-(f): Corresponding groundtruth depthmaps.

The first scene was created synthetically using POVRay[1], a high quality raytracing system which is capable of producing photo-realistic renderings. The scene itself was created manually using online available POVRay scene description language models. In order to get more realistic results, the camera trajectory was defined by real user camera motions. Since the gaps between the individual camera motion samples were too big, new motions were interpolated using the POVRay spline subsystem. Due to the enormous computational resources needed by POVRay to simulate focal blur, we are using the normal anti-aliasing method instead. It is also to mention that this set of images lacks visual effects like motion blur due to its synthetic nature.

For the next scene the City of Sights [13] model was used. The images were captured with a Point Grey Dragonfly2 camera where a 2.8mm wide angle lens was mounted. The camera was pre-calibrated and the corresponding internal camera calibration matrix is used. The groundtruth camera poses were computed by PTAM whereas the groundtruth

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

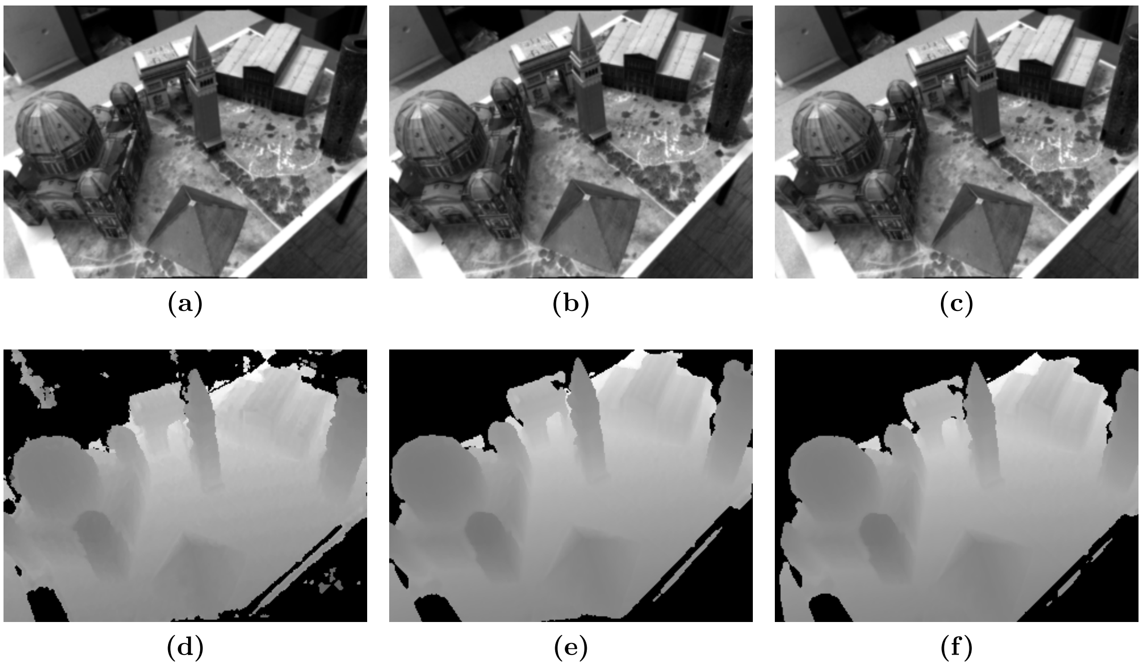


Figure 5.2: (a)-(c): First, reference and last intensity image of the City of Sights scene. (d)-(f): Corresponding groundtruth depthmaps.

depthmaps originate from [11, 12].

The last dataset used for the tracking evaluation is the "Desk" scene of [37]. Here an external tracking system provides groundtruth camera poses alongside the groundtruth depthmaps which come from a Microsoft Kinect. This dataset is especially challenging since the RGB camera of the Kinect is known to be noisy and it is also suffering heavily from motion blur and rolling shutter effects.

The evaluation of the depthmap computation part will be done additionally with a set of pre-captured images originating from the tablets live camera stream. In that way we are simulating perfectly the usual usecase. Since the depthmap computation relies completely on the quality of the tracking output, we are not evaluating the whole system, i.e. tracking and depthmap computation together, again in detail. Instead, we are inspecting the generated depthmap visually and also look at the results in 3D.

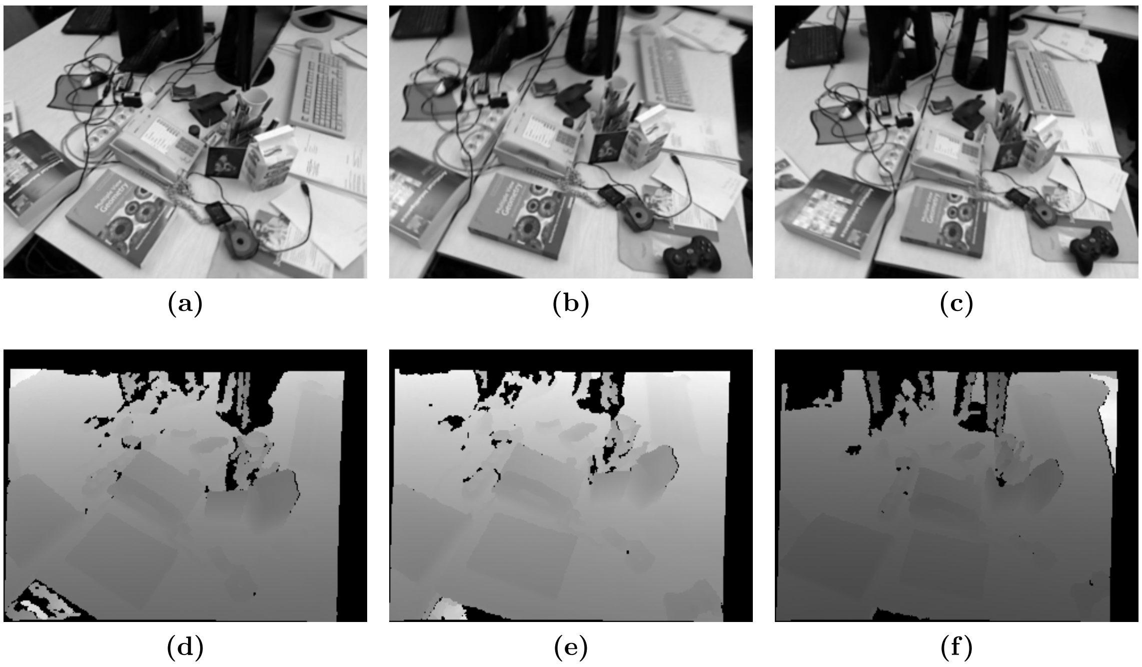


Figure 5.3: (a)-(c): First, reference and last intensity image of the "Desk" scene of [37]. (d)-(f): Corresponding groundtruth depthmaps.

5.2 Dense Tracking

We evaluate the dense tracking part of our system regarding the accuracy of the pose estimates. For this three dataset with different properties are used. Furthermore, we give timings and a correlation between tracking resolution and accuracy.

5.2.1 Accuracy

In this section we present the results of our system regarding accuracy of the dense tracking part. An accurate tracking system is crucial in order to get good reconstruction results. If the pose estimations are just slightly wrong, the epipolar constraint (see 2.2.2) is not fulfilled and wrong stereo matching costs are calculated. Besides of the actual tracking algorithm, the quality of the resulting pose estimates depends heavily on the input frames and the depthmap of the reference frame. The used datasets, explained in 5.1, cover perfect synthetic quality, high quality frames of a good external camera and inferior image quality from a Kinect device.

For evaluation of the accuracy, we compute the relative error between the tracked camera pose and the given groundtruth pose. The error is expressed by two terms, the positional and rotational error. The positional error is computed as the relative positional error in % according to

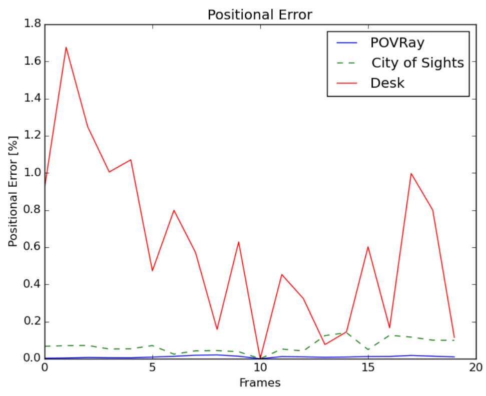
$$e_{pos} = \frac{||\mathbf{p} - \mathbf{p}_{gt}||}{||\mathbf{p}_{gt}||} \cdot 100 \tag{5.1}$$

where $\mathbf{p}$ is the position of the tracked camera and $\mathbf{p}_{gt}$ is the position of the groundtruth in the world coordinate system. The rotational error is computed as

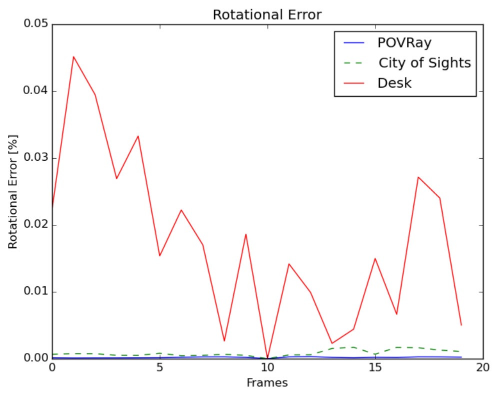
$$e_{rot} = ||\mathbf{I} - \mathbf{R}_{gt}\mathbf{R}^{\mathrm{T}}||_F \tag{5.2}$$

where $\mathbf{I}$ is a 3×3 identity matrix, $\mathbf{R}$ is the rotation of the tracked camera, $\mathbf{R}_{gt}$ is the rotation of the groundtruth and $|| \cdot ||_F$ is the Frobenius norm.

In all three datasets we track 20 frames relative to the reference frame with index 10. Figure 5.4 shows the tracking results in terms of the positional and rotational error, respectively. It is immediately obvious that the accuracy of the pose estimates for the different scenes varies in magnitudes. Note that positional error values below 0.3% are considered very accurate.



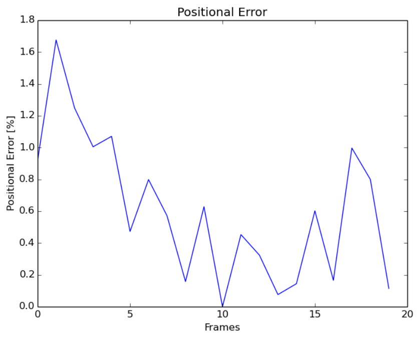
(a)



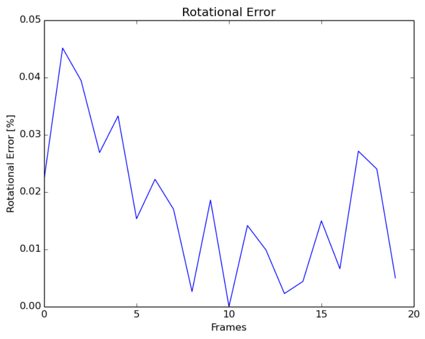
(b)

Figure 5.4: All frames were tracked relative to the reference frame with index 10. (a)-(b) shows the positional and rotational error.

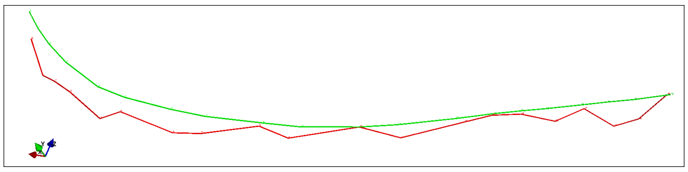
In order to get a better understanding of the individual results, Figure 5.5, 5.6 and 5.7 show detailed results with different scales in the y-axis. Furthermore, the estimated camera trajectories and the groundtruth are shown in 3D.



(a)

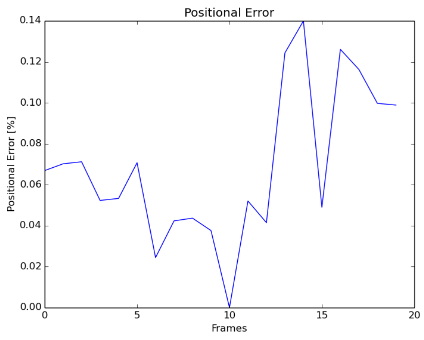


(b)

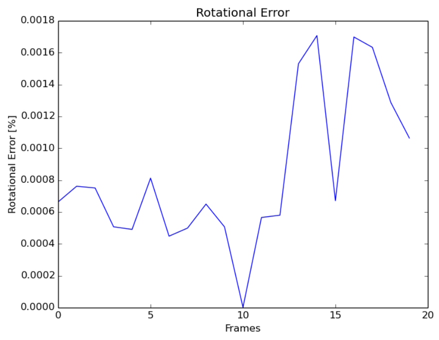


(c)

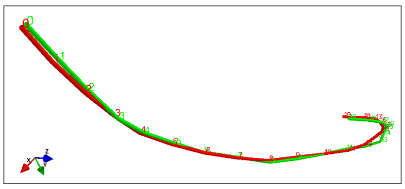
Figure 5.5: "Desk" scene of [37] - all frames were tracked relative to the reference frame with index 10. (a)-(b) shows the positional and rotational error. (c) shows the estimated camera trajectory (red) in comparison to the groundtruth (green).



(a)

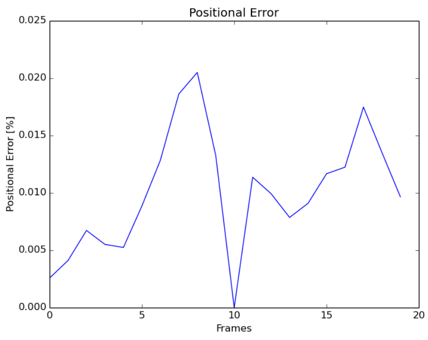


(b)

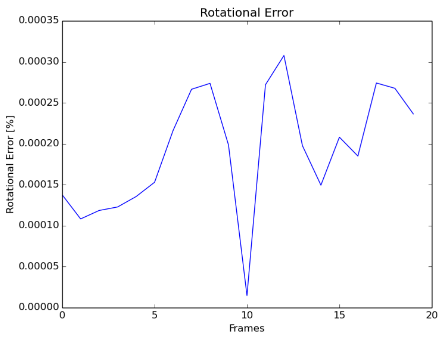


(c)

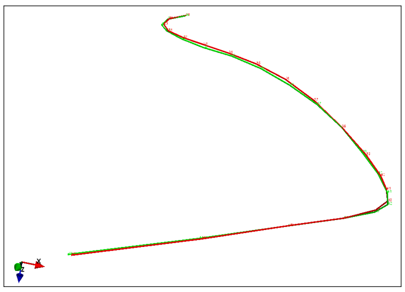
Figure 5.6: City of Sights scene - all frames were tracked relative to the reference frame with index 10. (a)-(b) shows the positional and rotational error. (c) shows the estimated camera trajectory (red) in comparison to the groundtruth (green).



(a)



(b)



(c)

Figure 5.7: POVRay scene - all frames were tracked relative to the reference frame with index 10. (a)-(b) shows the positional and rotational error. (c) shows the estimated camera trajectory (red) in comparison to the groundtruth (green).

The results show the importance of high quality input frames and an appropriate depthmaps used for tracking. As expected, the rolling shutter and motion blur effects of the desk scene decrease the tracking accuracy whereas the synthetic POVRay scene delivers the best results. The tracking results are presented also numerically in table 5.1.

	e_{pos}		e_{rot}	
	avg	RMS	avg	RMS
Desk	0.611	0.753	0.018	0.022
City of Sights	0.069	0.078	< 0.001	< 0.001
POVRay	0.010	0.011	< 0.001	< 0.001

Table 5.1: Average and RMS positional and rotational tracking error in %. Note that values below 0.3% are considered as being very accurate.

5.2.2 Speed

Tracking systems typically have to meet hard realtime requirements and therefore only a timeframe of about 33ms is available (supposing that the system should run at 30Hz). This section shows the timing results of our implementation. For evaluation, we average the results of all three testing datasets where each dataset was tracked several hundred times.

In 4.2.2, table 4.1 shows the upper bounds for the maximum number of iterations for each pyramid level. These values were found as follows. The tracking update step is given by 3.18 and the stopping criteria is defined as $||\Delta\xi|| < 10^{-4}$. Table 5.2 shows the average and maximum number of iterations needed until the convergence criteria was fulfilled.

	#Iterations	
Level	avg	max
3	8.33	60
2	6.12	11
1	4.59	12
0	2.21	12

Table 5.2: Average and maximum number of iterations needed such that the stopping criteria $||\Delta\xi|| < 10^{-4}$ is fulfilled.

It can be seen that in average the algorithm converges after only some iterations even though the maximum number might be a magnitude higher. Furthermore, lower levels need less iterations since higher levels calculate the first rough pose estimates and the lower ones only fine tune the result.

Timings of the individual levels are shown in table 5.3 and depicted in Figure 5.8. Even though we are utilizing the GPU, the hardware resources on the mobile plattforms are still limited. Therefore, the average total tracking time of our implementation is at about 42ms.

Level	Time[ms]
3	12.56
2	9.04
1	8.89
0	11.19
Total	41.68

Table 5.3: Timings of individual pyramid levels.

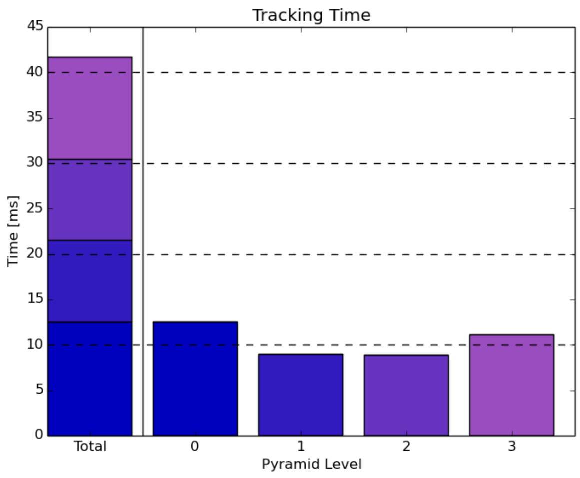
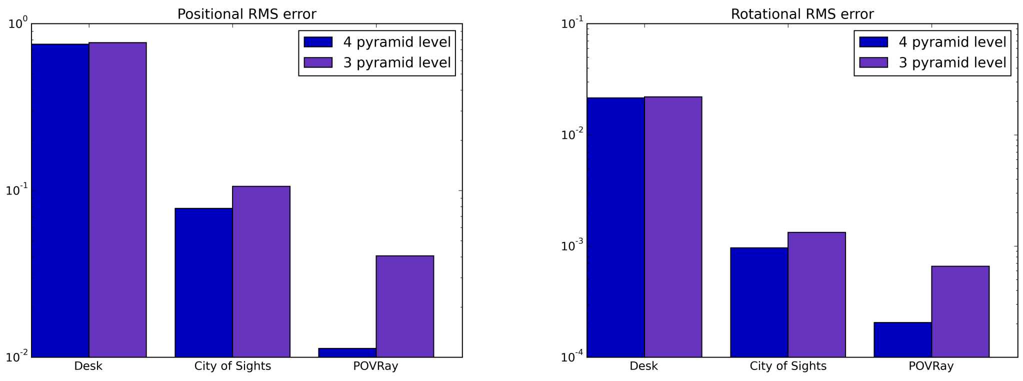


Figure 5.8: Timings of individual pyramid levels compared to total timing.

A common approach to decrease the tracking computation time is to neglect the lowest level and thus reduce the maximum resolution from 320×240 to 160×120. In our case it would be possible to reduce the computation time from 42ms to 30ms. The obvious drawback of this approach is that the tracking accuracy is suffering from the reduced resolution. Figure 5.9 depicts the effects of reducing the pyramid levels (note the logarithmic scale in the y direction).



(a) **(b)**

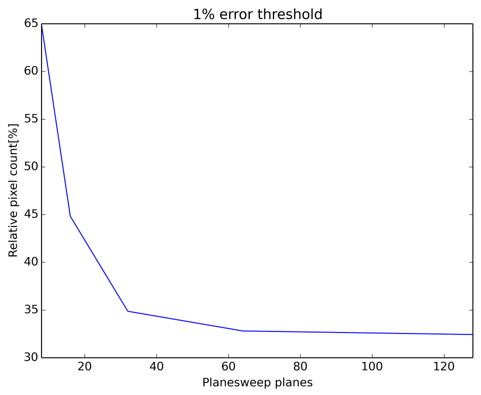
Figure 5.9: Reducing the number of pyramid levels from 4 to 3 reduces the computation time but can drastically decrease the tracking accuracy (note the logarithmic scale in the y direction).

Our implementation shows a computational performance gain of **125%** by reducing the pyramid levels from 4 to 3. At the same time the positional and rotational tracking error increases nearly by **99%** and **87%**, respectively. Therefore, we do not reduce the number of pyramid levels in our implementation but use the full 320 × 240 resolution.

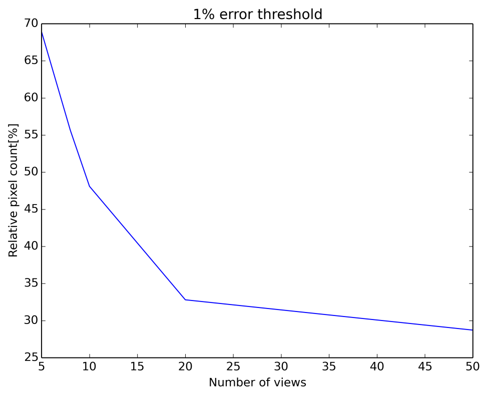
5.3 Depthmap Generation

5.3.1 Planesweep

The two most important parameters of the planesweep algorithm are the depth resolution, i.e. the number of planes within the cost volume defined by Z_{near} and Z_{far} and the number of sensor views used to compute the costs within this volume. A low depth resolution will deliver poor results since the resulting depthmap will be too coarse whereas a high resolution increases the computational time. The number of views behave similar. More views contain more information about the 3D scene structure and thus, we expect that using more views will deliver better results. The results of our experiments are summarized in Figure 5.10. As a measure of precision we took the percentage of pixels for which the depth error is greater than a defined threshold w.r.t. the groundtruth depth. The error threshold was set to 1% of the total depth range.



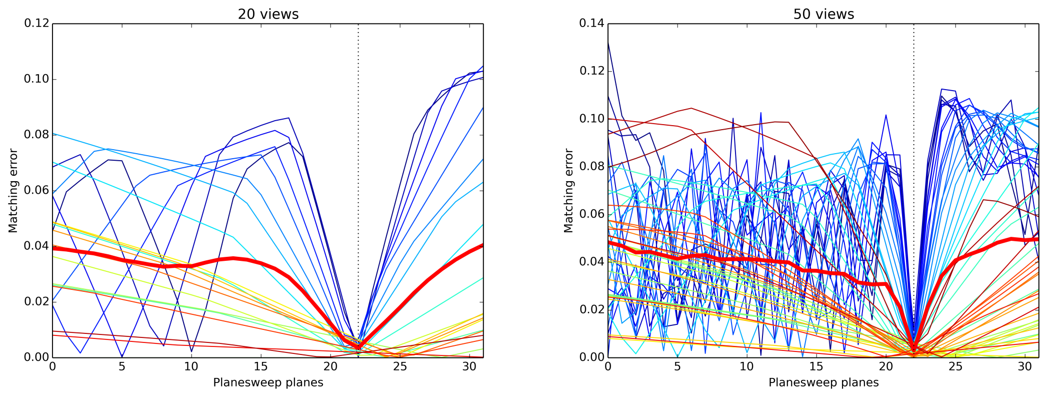
(a) Depth error vs. depth resolution



(b) Depth error vs. number of views

Figure 5.10: Depth error rates for different number of planesweep planes and different number of sensor views, respectively. For (a), the number of views was fixed to 20 whereas for (b), the depth resolution was fixed to 64 steps.

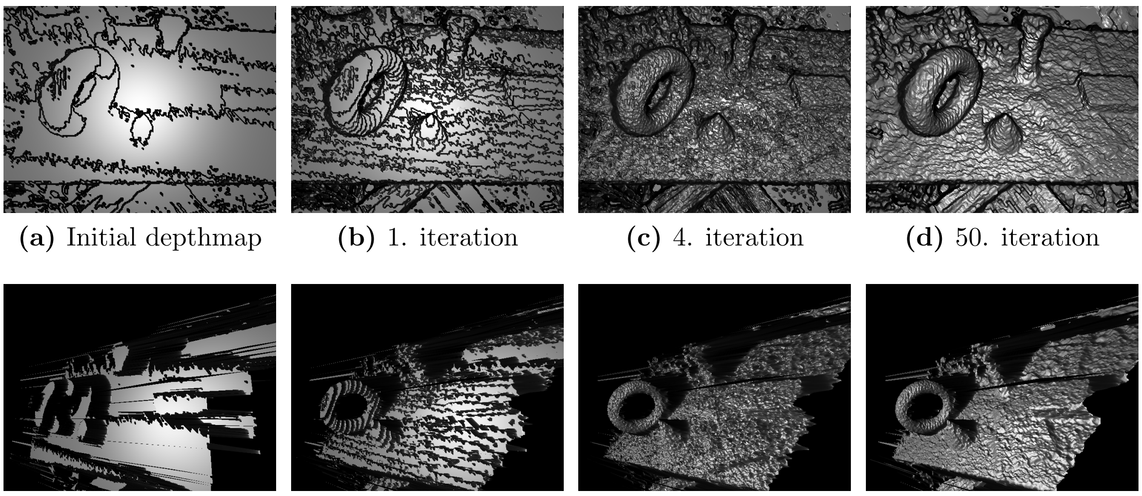
One can see that a resolution below 32 steps delivers poor results whereas increasing the resolution beyond 64 steps does not improve the quality significantly, while the computational time is raised. The results for the number of views behave similar. As expected, more views deliver better results. The matching costs for 20 and 50 views are shown in detail in Figure 5.11. It can be seen that 20 views are already enough to achieve robust results although 50 views provide even more information. These experiments showed us that a configuration of 32 planes and 20 views deliver good results while having an acceptable computational load.



(a) 20 sensor views **(b)** 50 sensor views

Figure 5.11: Matching costs for 20 and 50 number of sensor views, respectively. The cost volume contains 32 planes and the average cost is shown by the thick red line.

In Figure 5.10a we saw that increasing the number of steps for the planesweep algorithm beyond 64 does not significantly increase the accuracy. Therefore we are evaluating the adaptive planesweep only visually. Figure 5.12 shows the evolution of the depthmap in 3D after different numbers of iterations. The first row depicts the depthmap from the reference view whereas the second row shows the same depthmap from another angle. The algorithm is initialized with a depthmap generated by the standard planesweep using 8 planes and 20 sensor views. The following pictures show the depthmap after 1, 4, and 50 iterations, respectively. As mentioned in 3.3.2, we apply a 3×3 median filter on the depthmap in each iteration. On the one hand this is reducing small outliers, i.e. little spikes in the depthmap but on the other hand one can see the results are still not smooth because no global solution is found. The advantage of this approach is though, that the computational load is not increased since a normal planesweep algorithm is calculated in each iteration and only the volume boundaries Z_{near} and Z_{far} are adaptively adjusted for each pixel individually.



(a) Initial depthmap **(b)** 1. iteration **(c)** 4. iteration **(d)** 50. iteration

(e) Initial depthmap **(f)** 1. iteration **(g)** 4. iteration **(h)** 50. iteration

Figure 5.12: Evolution of the depthmap using the adaptive planesweep algorithm. The algorithm is initialized with a depthmap generated by the standard planesweep using 8 planes and 20 sensor views. The first row shows the depthmap from the reference view after 0, 1, 4 and 50 iterations. The second row shows the same depthmap from another angle.

Note that we did not evaluate the different matching methods separately. In our experience, it depends on the scene which one will work the best and the user can interactively change the method. In general the single pixel matching gives the best details but also the highest amount of outliers. The zero mean and other patch based methods have opposite characteristics. Overall, single pixel and zero mean matching works best for most scenes.

5.3.2 Volume Optimization

In this section we will investigate the effects of different parameters on the proposed volume optimization algorithm. The biggest influence on the resulting depthmap have the regularization parameter λ which controls the smoothness of the result, θ which defines the coupling of the depthmap and the auxiliary variable, γ which controls the speed by which θ decreases and finally, the number of iterations.

For our first experiment we were building up the matching cost volume with the standard planesweep algorithm using 32 planes, 20 sensor views and single pixel matching. In order to evaluate the effects of θ, γ and the number of iterations, we fixed $\lambda = 7.2$. The results are summarized in Figure 5.13. The coupling term $A(x) = \frac{1}{2\Theta}(\xi(x) - \beta(x))^2$ in

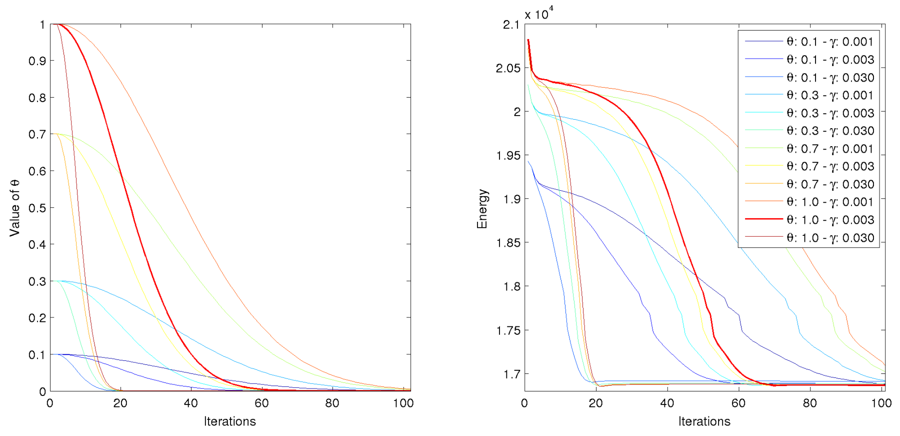
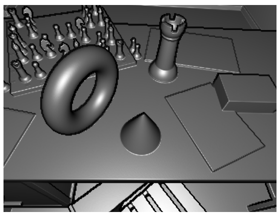
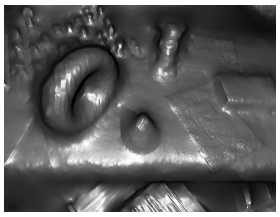


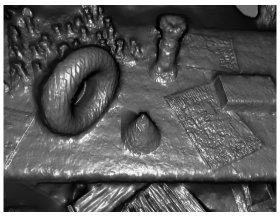
Figure 5.13: Effects of different values of θ and γ on iterations and the resulting energy.

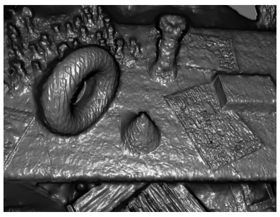
(3.29) is used to drive both, the original and auxiliary variable, together, i.e. that $\xi = \beta$ as $\theta \rightarrow 0$. The lower the starting value of θ, the tighter the coupling and the lower the resulting energy at the beginning. γ is controlling the rate by which θ is decreasing. Here it is other way around, the higher the value, the faster θ is decreasing. However, if the value of γ is too high, θ is decreasing too fast and a certain solution is favored too early resulting in lower quality reconstructions. By setting γ to smaller values, the reconstruction quality is increased but at the same time the number iterations has to be raised for the algorithm to converge. Since the computational resources on mobile devices are limited we are using $\theta = 1$, $\gamma = 0.003$ and 70 iterations (fat line in Figure 5.13). This configuration delivers high quality results with a moderate computational load.

In the second experiment we investigated the effect of λ on the resulting depthmap. For this, we used the same initial depthmap, fixed $\theta = 1$, $\gamma = 0.003$ and used 70 iterations in the calculation. Figure 5.14 depicts the results of different settings of λ. The lower the value, the higher the smoothness of the resulting depthmap. However, by setting λ too low, the regularization term is heavily penalizing deviations from a spatially smooth depthmap resulting in a loss of detail. If the value of λ is too high though, the result is not smooth enough and shows outliers. Therefore, we chose $\lambda = 7.2$ as a standard value for our application.

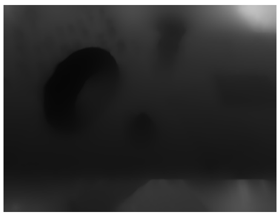

(a) Groundtruth

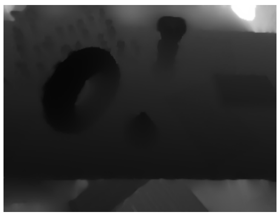

(b) $\lambda = 1.2$


(c) $\lambda = 7.2$


(d) $\lambda = 14.2$

(e) Groundtruth


(f) $\lambda = 1.2$


(g) $\lambda = 7.2$

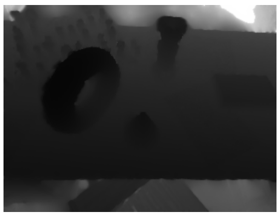

(h) $\lambda = 14.2$

Figure 5.14: Effects of λ on the resulting depthmap. The lower the value, the higher the smoothness of the result. While the first line shows a phong shaded model in 3D, the second line shows the corresponding depthmap.

5.4 System

In this section we first evaluate two different methods of how the system can be bootstrapped, i.e. how the initial depthmap is defined. Afterwards we show an exemplary run of the complete system together with intermediate and final results. Finally, we provide timings of the individual parts and also show a complete list of system parameters.

The bootstrapping is necessary because we do not have any prior knowledge about the scene and we therefore have to assume an initial depthmap. The two main parameters during the initialization phase are the number of planesweep planes and if a homogeneous or random initial depthmap is used. Figure 5.15 summarizes our results. As a fidelity measure we took the number of iterations needed until the depthmap was good enough such that the average positional tracking error was below a threshold w.r.t. the groundtruth camera positions. The positional error threshold was set to 0.5%. For evaluation we used different scenes with 20 sensor views. The homogeneous depthmap is created by taking the mean between Z_{near} and Z_{far}, whereas the random depthmap randomly samples values within that range. The best configuration delivers the homogeneous depthmap together with 8 planesweep steps resulting in about 5 iterations, although it can be seen that the random depthmap needs less iterations in general. Moreover, we encountered the problem

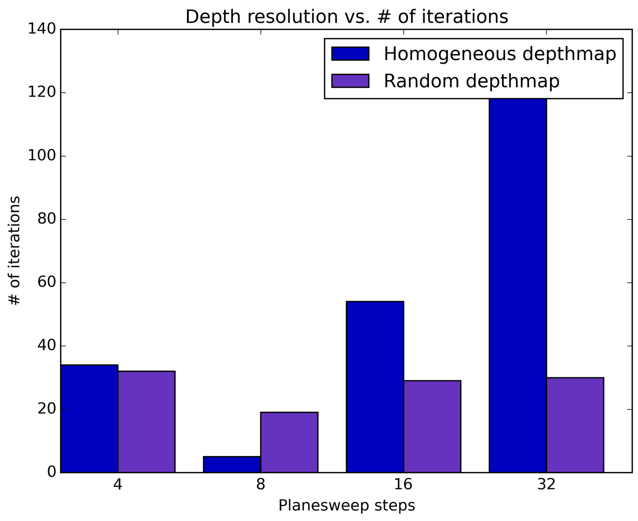


Figure 5.15: Initialization with homogeneous and random depthmap together with different numbers of planesweep steps.

that using a random initial depthmap might result in a tracking ambiguity where the resulting cameras are inversely estimated. The result of this is, that the depthmap is also estimated inversely meaning that near objects are far away and vice versa. Unfortunately we did not find any pattern when this happens. We interpret the results in the way that the coarse depthmap with 8 planes can adjust faster to the real depth in comparison to a finer resolution, i.e. 32 steps. At the same time a depthmap with only 4 steps is too coarse. Therefore, we use the configuration of 8 planesweep planes together with a homogeneous depthmap.

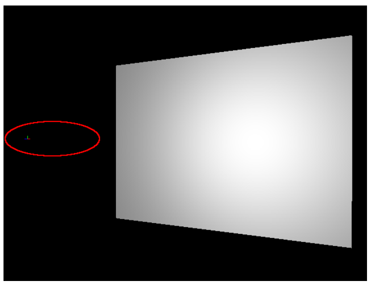
An exemplary run, including timestamps, is depicted in Figure 5.16. The system parameters used for the full system evaluation are shown in table 5.4 and the corresponding performance timings can be found in table 5.5. We again used the City of Sights scene, but this time with the Nvidia Shield tablet. At the beginning, the system captures a set of 20 frames and the middle frame is used as the reference frame. An initial homogeneous depthmap is used to bootstrap the system by tracking all frames with that plane resulting in first rough pose estimates (see Figure 5.16a). A planesweep algorithm with 8 steps is used to calculate coarse dephtmaps during the initial phase to improve the tracking accuracy quickly (see Figure 5.16b, 5.16c, 5.16d). Note that the initial camera poses are at identity in Figure 5.16a and how they adjust during the initialization phase in Figure 5.16d. From this point on, the volume optimization is used to drastically improve the depthmap quality - outliers are removed and a spatially smooth depthmap is created. The results are depicted without the quadratic function fitting in Figure 5.16e and with it in Figure 5.16f. Furthermore, note how the camera positions and orientations also improve during this phase since the tracking is now based on a high quality depthmap. While Figure 5.16g, 5.16h, 5.16i show the grayscaled depthmaps of the corresponding phase, Figure 5.16j, 5.16k, 5.16l depict the final result in various ways.

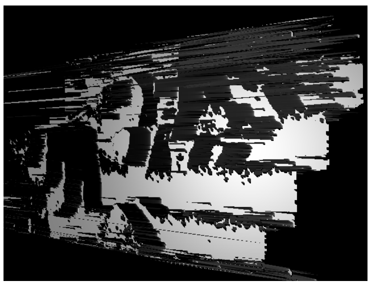
Parameter	Value
Device	Nvidia Shield Tablet
Scene	City of Sights
Image resolution	320×240
# of pyramid levels	4
# of frames	20
Deptmap algorithm	Planesweep
Matching method	Single pixel
Initial depthmap type	Homogeneous plane
# of iterations (initial phase)	5
# of iterations (volume optimization)	70
# of planes (initial phase)	8
# of planes (volume optimization)	32
Volume optimization λ	7.2
Volume optimization θ	1.0
Volume optimization γ	0.003

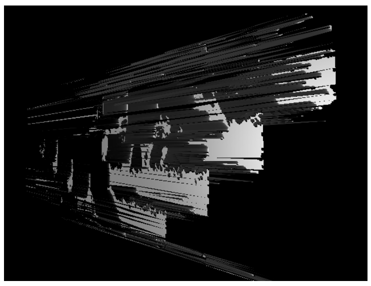
Table 5.4: Full system evaluation parameters

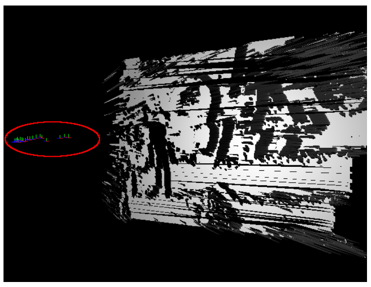
Operation	Time [ms]
Track frame	42
Compute startpoint and displacement vector (planesweep speedup)	19
Planesweep 8 planes (initial phase)	29
Planesweep 32 planes (volume optimization)	90
Volume optimization	515

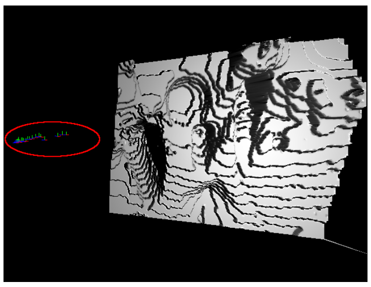
Table 5.5: System timings

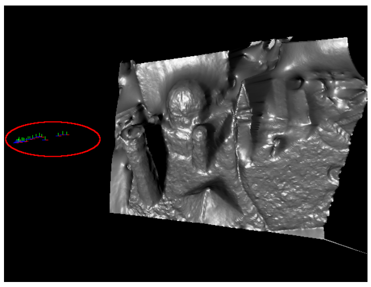

(a) Initial homogeneous plane - 0sec

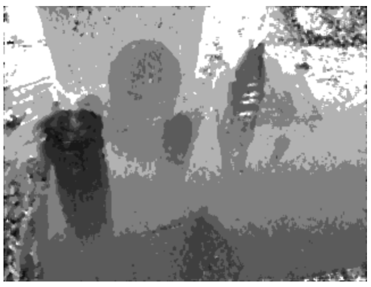

(b) Other perspective of initial phase - 4sec

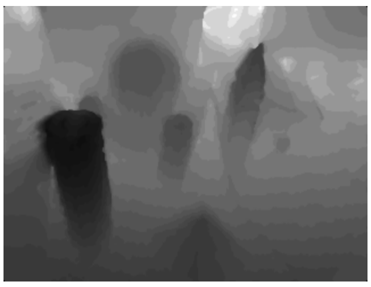

(c) Other perspective of initial phase - 4sec

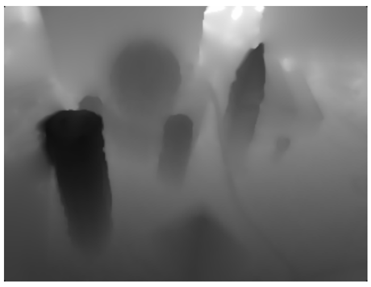

(d) Initial phase - 4sec

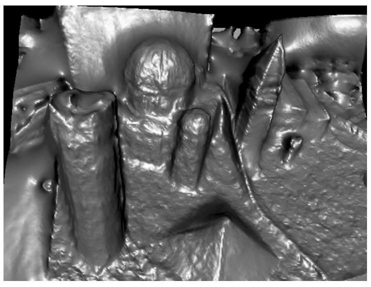

(e) Volume optimization - 7sec

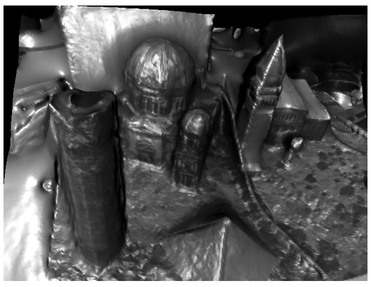

(f) Volume optimization + quadfitting - 7sec


(g) Depthmap initial phase - 4sec


(h) Depthmap volume optimization - 7sec


(i) Depthmap volume optimization + quadfitting - 7sec


(j) Phong shaded result


(k) Phong shaded + textured result

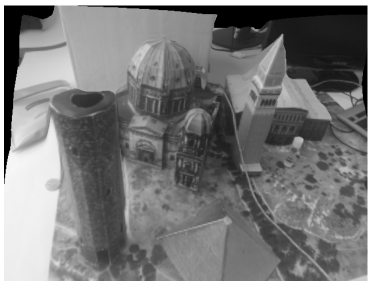

(l) Textured result

Figure 5.16: An exemplary run and results of our system. Note how the camera positions adjust due to the qualitatively increasing depthmap.

6 Conclusion

Contents

6.1 Summary

In this master's thesis a system for interactively generating high quality, dense depthmaps on mobile devices like smartphones and tablets was developed. It has been shown that the system can compute dense depthmaps from arbitrary geometry on-the-fly just within several seconds. The developed tracking system operates directly on images without an intermediate representation like keypoints. The proposed method adapts and combines current state-of-the-art results from various computer vision research areas in a novel way.

Chapter 2 gives an overview and introduction of related work on dense 3D reconstruction on mobile and also desktop platforms since the latter one formed the basic principles.

The methodology of the high quality, dense depthmap computation system on mobile devices is presented in chapter 3. Section 3.2 is devoted to dense tracking, where we describe in detail the theory and each step of the algorithm. Furthermore we show how larger motions can be handled and how the procedure can be sped up.

The generation of depthmaps is addressed in section 3.3. Here, the used multi-view stereo algorithm is presented, alongside with pixel matching methods and a method of speeding up the computation. Furthermore, an adaptive stereo algorithm is presented in section 3.3.2 as well an optimization algorithm that minimizes a global spatially regularized energy functional on the GPU which is described in section 3.3.3.

Chapter 4 gives details about our setup and the implementation of all parts. Special emphasis is placed on the way how the GPU is utilized to accelerate the tracking part of the system.

Finally, the whole system is evaluated in chapter 5. All the different parts of the tracking and depthmap generation system are analyzed separately before a complete system evaluation was performed.

6.2 Outlook

Our system can be improved and extended in several ways. Up to now, we built the fundamental building blocks *dense tracking* and *dense depthmap* computation for a novel mobile SLAM system. The fusion of multiple depthmaps within a single reconstruction volume, in order to get a full 3D model, is still missing though. Moreover are both parts running on the GPU in our implementation. Since the mapping part is typically computationally more expensive, the tracking should be implemented on the CPU in order to have all GPU resources free for the reconstruction.

The initialization procedure can be improved as well. Depending on the scene, both, the homogeneous and random initialization, may need much more iterations until the tracking accuracy improves. Therefore, we might change the initialization procedure to a more sophisticated approach.

Todays smartphones and tablets are equipped with various sensors like accelerometers, gyroscopes or *GPS* receivers. By using the accelerometer and the gyroscope we could estimate the absolute reconstruction scale. The accelerometer also could be used to decrease the tracking drift and to get weak initial estimates for the device translation whereas a gyroscope could give good rotational initial estimates.

The quality of the resulting depthmap depends heavily on the input images. The used planesweep algorithm is fast and delivers good results in information rich image areas but fails in untextured regions. Our system can therefore handle untextured scenes only to a certain degree. Furthermore are reflections a problem since we detect wrong motions in these areas. Latest research results show that reflections can be detected. By neglecting the reflecting parts of the scene we might increase the reconstruction quality as well.

A Dense Tracking Appendix

In reconstruction, visual odometry or generally, SLAM systems, one typically has to work with different coordinate systems and points need to be transformed between them, e.g. from a camera coordinate frame into a global world coordinate system. Rigid body transformations are used to describe the position and rotation of camera poses. Although a broad variety of representations exist for the rotational part it is crucial to choose a minimal representation of the transformation when optimizing the set of transformation parameters, like it is typically done in reconstruction systems. This means that no more parameters than degrees of freedom should be present otherwise inefficient performance or invalid configurations might be the result.

The SE(3) Lie group and $\mathfrak{se}(3)$ algebra are one kind of representation which fulfills that requirement. SE(3) group elements consist of a rotational and translational part and are represented as 3×4 matrices $\mathbf{C} = [\mathbf{R} \mid \mathbf{t}] \in \mathrm{SE}(3)$. This representation has 12 DOF, rotation and translation have only 6, though. Therefore the minimal representation of a rigid body transformation is given as a 6-dimensional vector in the Lie algebra by $\boldsymbol{\xi} = [w_1, w_2, w_3, w_4, w_5, w_6] \in \mathbb{R}^6$. A mapping function, the exponential map, defines the mapping of elements from the algebra (6-dimensional vector) to the manifold (3×4 transformation matrices). The following sections elaborate the transformation and exponential map derivative in detail which are used in the dense tracking optimization. This is summary of the relevant parts of [23].

A.1 Transformation Derivative

A homogeneous point X in 3D space is transformed by the 3×4 matrices $\mathbf{C} = [\mathbf{R} \mid \mathbf{t}] \in \mathrm{SE}(3)$ by

$$\mathbf{X}' = \mathbf{C}\mathbf{X} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11}x_1 + r_{12}x_2 + r_{13}x_3 + t_1 \\ r_{21}x_1 + r_{22}x_2 + r_{23}x_3 + t_2 \\ r_{31}x_1 + r_{32}x_2 + r_{33}x_3 + t_3 \end{bmatrix} \tag{A.1}$$

The same transformation defined in (A.1) can be rewritten. From the point $\mathbf{X}$ a matrix $\mathbf{M} \in \mathbb{R}^{3\times 12}$ is constructed:

$$\mathbf{M} = [x_1\mathbf{I}_3 \mid x_2\mathbf{I}_3 \mid x_3\mathbf{I}_3 \mid \mathbf{I}_3] \tag{A.2}$$

where $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ is the 3×3 identity matrix. For matrices $\mathbf{A} \in \mathbb{R}^{M\times N}$ and $\mathbf{B} \in \mathbb{R}^{P\times Q}$ the Kronecker matrix product denoted by $\mathbf{A} \otimes \mathbf{B}$ is the $MP \times NQ$ block matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \tag{A.3}$$

Using (A.3) the definition of (A.2) can be conveniently written as $\mathbf{M} = \mathbf{X}^{\mathrm{T}} \otimes \mathbf{I}_3$. For matrices the $\mathrm{vec}(\cdot)$ operator produces a vector by stacking the columns on top of each other. This is convenient because derivatives of matrices become now standard Jacobians. The parameter vector $\mathbf{p} = \mathrm{vec}(\mathbf{C}) \in \mathbb{R}^{12}$ is constructed from the transformation matrix $\mathbf{C}$ and finally (A.1) can be rewritten as

$$\mathbf{X}' = \mathbf{M}\mathbf{p} = \begin{bmatrix} x_1 & 0 & 0 & x_2 & 0 & 0 & x_3 & 0 & 0 & 1 & 0 & 0 \\ 0 & x_1 & 0 & 0 & x_2 & 0 & 0 & x_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & 0 & 0 & x_2 & 0 & 0 & x_3 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \\ r_{12} \\ r_{22} \\ r_{32} \\ r_{13} \\ r_{23} \\ r_{33} \\ t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{A.4}$$

Finally, the derivative of the transformation w.r.t. the parameters is given by

$$\frac{\partial \mathbf{X}'}{\partial \mathbf{C}} = \frac{\partial \mathbf{M}\mathbf{p}}{\partial \mathbf{p}} = \mathbf{M} \tag{A.5}$$

A.2 Exponential Map Derivative

The exponential map associates elements of the Lie algebra (6-dimensional vector) to elements of the underlying Lie group (transformation matrix). In the minimization step where the optimal transformation parameters are found, the derivative of the exponential map has to be determined, among others. In order to compute this, several preconditions have to be met. First, the 6 rigid body transformation matrices parameterized by β are specified:

$$x\text{-axis:} \quad \text{translation: } \mathbf{M}_1 = \begin{bmatrix} 1 & 0 & 0 & \beta \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{rotation: } \mathbf{M}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) & 0 \\ 0 & \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$y\text{-axis:} \quad \text{translation: } \mathbf{M}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \beta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{rotation: } \mathbf{M}_5 = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$z\text{-axis:} \quad \text{translation: } \mathbf{M}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \beta \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{rotation: } \mathbf{M}_6 = \begin{bmatrix} \cos(\beta) & \sin(\beta) & 0 & 0 \\ \sin(\beta) & \cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Infinitesimal SE(3) motions are now found by differentiating these matrices w.r.t. β and evaluating them at $\beta = 0$.

$$\mathbf{G}_1 = \frac{\partial \mathbf{M}_1}{\partial \beta}\bigg|_{\beta=0} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{G}_4 = \frac{\partial \mathbf{M}_4}{\partial \beta}\bigg|_{\beta=0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{G}_2 = \frac{\partial \mathbf{M}_2}{\partial \beta}\bigg|_{\beta=0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{G}_5 = \frac{\partial \mathbf{M}_5}{\partial \beta}\bigg|_{\beta=0} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{G}_3 = \frac{\partial \mathbf{M}_3}{\partial \beta}\bigg|_{\beta=0} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{G}_6 = \frac{\partial \mathbf{M}_6}{\partial \beta}\bigg|_{\beta=0} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Elements of the Lie algebra $\mathfrak{se}(3)$ are 6-dimensional vectors and are generated by the *generator matrices* $\mathbf{G}_i, i = 1...6$ by multiplying each vector component with one of them. As mentioned above, the exponential map takes elements of the Lie algebra to the corresponding manifold. The mapping is performed by calculating the matrix exponential, which is defined as

$$e^{\mathbf{X}} = \mathbf{I} + \mathbf{X} + \frac{1}{2}\mathbf{X}^2 + \frac{1}{6}\mathbf{X}^3 + ... = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{X}^k. \tag{A.6}$$

Using (A.6) the exponential map is now defined as

$$\begin{aligned} \exp : \mathfrak{se}(3) &\mapsto \mathrm{SE}(3) \\ \boldsymbol{\xi} &\mapsto e^{\sum_{i=1}^{6} w_i \mathbf{G}_i} \end{aligned} \tag{A.7}$$

where $\boldsymbol{\xi} = [w_1, w_2, w_3, w_4, w_5, w_6] \in \mathbb{R}^6$ is an element of the Lie algebra with motions $w_i, i = 1...6$. By looking at the generator matrices $\mathbf{G}_i$ it is easy to see that $\sum_{i=1}^{6} w_i \mathbf{G}_i$ can be also written in the form

$$\sum_{i=1}^{6} w_i \mathbf{G}_i = \begin{bmatrix} 0 & -w_6 & w_5 & w_1 \\ w_6 & 0 & -w_4 & w_2 \\ -w_5 & w_4 & 0 & w_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{A.8}$$

By using the definition of the skew-symmetric matrix operator $[\cdot]_\times$

$$v_\times = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_\times = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \tag{A.9}$$

(A.8) can be written shorter as

$$\sum_{i=1}^{6} w_i \mathbf{G}_i = \begin{bmatrix} [w_{4...6}]_\times & w_{1...3} \\ 0 & 0 \end{bmatrix}. \tag{A.10}$$

We define $[\boldsymbol{\xi}]_\times = \sum_{i=1}^{6} w_i \mathbf{G}_i$ for notational convenience and finally get $e^{[\boldsymbol{\xi}]_\times}$, the exponential map matrix constructed from the vector in the Lie algebra. For infinitesimal Lie vectors $\boldsymbol{\delta\xi}$ with weights $w_i < 1$ the matrix exponential is approximated well by the linear term

$$e^{[\boldsymbol{\delta\xi}]_\times} = \mathbf{I} + \sum_{i=1}^{6} w_i \mathbf{G}_i = \mathbf{I} + [\boldsymbol{\delta\xi}]_\times. \tag{A.11}$$

The derivative of $e^{[\boldsymbol{\xi}]_\times}$ at the point $\boldsymbol{\xi} = 0$ can be now computed by expanding the matrix using the vec$(\cdot)$ operator and making use of (A.11)

$$\mathrm{vec}(e^{[\boldsymbol{\xi}]_\times}) = \mathrm{vec}(\mathbf{I} + [\boldsymbol{\xi}]_\times) = \mathrm{vec}(\mathbf{I}) + vec([\boldsymbol{\xi}]_\times) \tag{A.12}$$

and write using (A.10)

$$\mathbf{M}\boldsymbol{\xi} = \operatorname{vec}([\boldsymbol{\xi}]_\times)$$

$$\mathbf{M}\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \operatorname{vec}\left(\begin{bmatrix} 0 & -w_6 & w_5 & w_1 \\ w_6 & 0 & -w_4 & w_2 \\ -w_5 & w_4 & 0 & w_3 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ w_6 \\ -w_5 \\ -w_6 \\ 0 \\ w_4 \\ w_5 \\ -w_4 \\ 0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} \tag{A.13}$$

and solving for $\mathbf{M}$ results in

$$\mathbf{M} = \begin{bmatrix} \mathbf{0}_{3\times3} & -[\mathbf{e}_1]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{e}_2]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{e}_3]_\times \\ \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{A.14}$$

where $\mathbf{e}_1 = [1,0,0]^\mathrm{T}$, $\mathbf{e}_2 = [0,1,0]^\mathrm{T}$, $\mathbf{e}_3 = [0,0,1]^\mathrm{T}$. Although $[\boldsymbol{\xi}]_\times$ results in a 4×4 matrix the last row was excluded since it is always 0 by definition.

Finally, the derivative of the exponential map at the origin, i.e. by evaluating it at $\boldsymbol{\xi} = 0$, is given by

$$\left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \frac{\partial(\mathbf{I} + [\boldsymbol{\xi}]_\times)}{\partial \boldsymbol{\xi}} = \frac{\partial [\boldsymbol{\xi}]_\times}{\partial \boldsymbol{\xi}} = \frac{\partial \mathbf{M}\boldsymbol{\xi}}{\partial \boldsymbol{\xi}} = \mathbf{M}. \tag{A.15}$$

By computing the derivative at another point on the manifold it is possible to generalize to arbitrary transformations $e^{[\hat{\boldsymbol{\xi}}]_\times}$ with $\hat{\boldsymbol{\xi}} \neq 0$. Reason for this is the linearity and local euclidean structure of the manifold SE(3). Given an infinitesimal Lie-vector $\boldsymbol{\xi}$ and an arbitrary transformation $\mathbf{T} = e^{[\hat{\boldsymbol{\xi}}]_\times}$ we end up having $e^{[\hat{\boldsymbol{\xi}}]_\times} = e^{[\boldsymbol{\xi}]_\times}\mathbf{T}$. The derivative of the exponential map evaluated at $\hat{\boldsymbol{\xi}}$ is therefore given by the derivative of the composition of $\boldsymbol{\xi}$ and $\mathbf{T}$ evaluated at 0 and is written as

$$\left.\frac{\partial e^{[\hat{\boldsymbol{\xi}}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\hat{\boldsymbol{\xi}}} = \left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}\mathbf{T}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}\mathbf{T}}{\partial e^{[\boldsymbol{\xi}]_\times}}\frac{\partial [\boldsymbol{\xi}]_\times}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} \tag{A.16}$$

Since the zero Lie-vector $\boldsymbol{\xi} = 0$ results in the identity transformation matrix $\mathbf{I}$, i.e. $\left.\partial e^{[\boldsymbol{\xi}]_\times}\right|_{\boldsymbol{\xi}=0} = \mathbf{I}$, (A.16) can be rewritten as

$$\left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}\mathbf{T}}{\partial e^{[\boldsymbol{\xi}]_\times}}\frac{\partial e^{[\boldsymbol{\xi}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} = \left.\frac{\partial \mathbf{QT}}{\partial \mathbf{Q}}\right|_{\mathbf{Q}=e^{[\boldsymbol{\xi}]_\times}=\mathbf{I}} \left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0} \tag{A.17}$$

The first part is the derivative of $\mathbf{QT}$, a composition of transformations, w.r.t. the transformation $\mathbf{Q} = \mathbf{I}$ is given by $\frac{\partial \mathbf{QT}}{\partial \mathbf{Q}} = \mathbf{T}^{\mathrm{T}} \otimes \mathbf{I}_3$. The second term, the derivative of the exponential map, has already been reviewed in eq. (A.11) - (A.14). Finally, the complete derivative of the exponential map with an arbitrary transformation is given by

$$\left.\frac{\partial e^{[\hat{\boldsymbol{\xi}}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\hat{\boldsymbol{\xi}}} = [\mathbf{T}^{\mathrm{T}} \otimes \mathbf{I}_3] \begin{bmatrix} \mathbf{0}_{3\times3} & -[\mathbf{e}_1]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{e}_2]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{e}_3]_\times \\ \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & -[\mathbf{r}_1]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{r}_2]_\times \\ \mathbf{0}_{3\times3} & -[\mathbf{r}_3]_\times \\ \mathbf{I}_{3\times3} & -[\mathbf{t}]_\times \end{bmatrix}, \quad \mathbf{T} = e^{[\hat{\boldsymbol{\xi}}]_\times} \tag{A.18}$$

Summing up, depending on the point of evaluation, the derivative of the exponential map is either given by (A.15) for $\left.\frac{\partial e^{[\boldsymbol{\xi}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=0}$, or by (A.18) for $\left.\frac{\partial e^{[\hat{\boldsymbol{\xi}}]_\times}}{\partial \boldsymbol{\xi}}\right|_{\boldsymbol{\xi}=\hat{\boldsymbol{\xi}}}, \hat{\boldsymbol{\xi}} \neq 0$.

B
List of Acronyms

AR	Augmented Reality
CPU	Central Processing Unit
DOF	Degrees of Freedom
DTAM	Dense Tracking and Mapping
EKF	Extended Kalman Filter
GPGPU	General Purpose Computation on Graphics Processing Unit
GPS	Global Positioning System
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
IRLS	Iteratively Reweighted Least Squares
P3P	Perspective 3-Point
P4P	Perspective 4-Point
PnP	Perspective n-Point
PTAM	Parallel Tracking and Mapping
RANSAC	Random Sample Consesus
SfM	Structure from Motion
SIMD	Single Instruction Multiple Data
SLAM	Simultaneous Location and Mapping
WTA	Winner Takes All

Bibliography

[1] Brown, D. C. (1971). Close-range camera calibration. *PHOTOGRAMMETRIC ENGINEERING*, 37(8):855–866. (page 7)

[2] Chambolle, A. and Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *J. Math. Imaging Vis.*, 40(1):120–145. (page 41, 42)

[3] Collins, R. (1996). A space-sweep approach to true multi-image matching. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pages 358–363. (page 30)

[4] 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. (page 30, 31)

[5] 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. (page 16)

[6] Dissanayake, M., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229–241. (page 15)

[7] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006. (page 14, 15)

[8] Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1449–1456. (page 19)

[9] Faugeras, O. (1993). *Three-dimensional Computer Vision: A Geometric Viewpoint.* MIT Press, Cambridge, MA, USA. (page 9)

[10] 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. (page 13)

[11] Graber, G. (2011). *Realtime 3D Reconstruction*. PhD thesis, Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria. (page 57)

[12] Graber, G., Pock, T., and Bischof, H. (2011). Online 3d reconstruction using convex optimization. In *1st Workshop on Live Dense Reconstruction From Moving Cameras, ICCV 2011*. (page 57)

[13] Gruber, L., Gauglitz, S., Ventura, J., Zollmann, S., Huber, M., Schlegel, M., Klinker, G., Schmalstieg, D., and Hollerer, T. (2010). The city of sights: Design, construction, and measurement of an augmented reality stage set. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 157–163. (page 56)

[14] Guivant, J. and Nebot, E. (2001). Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *Robotics and Automation, IEEE Transactions on*, 17(3):242–257. (page 15)

[15] Hartley, R. and Sturm, P. (1997). Triangulation. *Comput. Vis. Image Underst.*, 68(2):146–157. (page 12)

[16] Hartley, R. I. (1994). An algorithm for self calibration from several views. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 908–912. (page 8, 10, 13)

[17] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition. (page 6, 9, 10, 12, 32)

[18] Kerl, C., Sturm, J., and Cremers, D. (2013). Robust odometry estimation for rgb-d cameras. In *In ICRA*. (page 23, 26, 27)

[19] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan. (page 16, 17, 19)

[20] Klein, G. and Murray, D. (2009). Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. (page 17, 18)

[21] Kolev, K., Tanskanen, P., Speciale, P., and Pollefeys, M. (2014). Turning mobile phones into 3d scanners. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 3946–3953. (page 20)

[22] 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. (page 15)

[23] Ma, Y., Soatto, S., Kosecka, J., and Sastry, S. S. (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag. (page 23, 77)

[24] Martin, P., Marchand, E., Houlier, P., and Marchal, I. (2014). Decoupled mapping and localization for augmented reality on a mobile phone. In *Virtual Reality (VR), 2014 iEEE*, pages 97–98. (page 18)

[25] Matthies, L., Szeliski, R., and Kanade, T. (1988). Incremental estimation of dense depth maps from image sequences. In *Proceedings of Computer Vision and Pattern Recognition (CVPR '98)*, pages 366–374. (page 18, 19)

[26] Maybeck, P. S. (1979). *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. (page 15)

[27] 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. (page 15, 16)

[28] 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*. (page 18)

[29] Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011). Dtam: Dense tracking and mapping in real-time. In *Proceedings of the 2011 International Conference on*

Computer Vision, ICCV '11, pages 2320–2327, Washington, DC, USA. IEEE Computer Society. (page 18, 23, 38, 40)

[30] 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. (page 12)

[31] Phong, B. T. (1975). Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317. (page 44)

[32] Pollefeys, M., Koch, R., and Gool, L. V. (1999). Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. In *INTERNATIONAL JOURNAL OF COMPUTER VISION*, pages 7–25. (page 8, 10, 13)

[33] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. (page 20)

[34] Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Phys. D*, 60(1-4):259–268. (page 41)

[35] Schöps, T., Engel, J., and Cremers, D. (2014). Semi-dense visual odometry for AR on a smartphone. In *ismar*. (page 19)

[36] Steinbrücker, F., Pock, T., and Cremers, D. (2009). Large displacement optical flow computation withoutwarping. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 1609–1614. (page 40)

[37] Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. (page 19, 57, 58, 60)

[38] Tanskanen, P., Kolev, K., Meier, L., Camposeco, F., Saurer, O., and Pollefeys, M. (2013). Live metric 3d reconstruction on mobile phones. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 65–72. (page 20)

[39] 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 J. Rob. Autom.*, 3(4):323–344. (page 9)

[40] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334. (page 9)