Graz University of Technology

Institute for Computer Graphics and Vision , TU Graz

Siemens AG Graz

Master Thesis

Interactive Scene Segmentation using 2D/3D Correspondences

Stefan Wakolbinger

Graz, Austria, July 2013

Thesis supervisors

Dipl. Ing. Dr. Thomas Pock

Dipl. Ing. Dr. Stefan Kluckner

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

EIDESSTATTLICHE ERKLÄRUNG

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

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

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

…… ……..
date (signature)

Abstract

Reconstructing a 3D scene based on a set of images from different vantage points is a fundamental problem in computer vision. However, in most of the methods proposed in literature the complete 3D scene is aimed to be reconstructed. In contrast, the goal of this work is to build a model of only one specific object of interest, which is selected by the user. The problem can therefore be described as a combination of 3D reconstruction and segmentation. The algorithms for an interactive tool are developed, in which the user is asked to mark both object and background regions by placing strokes in some of the images. Additionally, depth maps are computed for each view. The main contribution of this work is the combination of depth information with semantic from color segmentation. The probabilities from the color models are fused with the hypotheses stemming from the depth maps, which is done directly in object space. An optimization step is performed in order to compute the most probable object surface using a smoothness prior, formulated in terms of the total variation framework. Instead of segmenting the object in each of the views and fusing the results to create a 3D model, the optimization is performed directly in object space.

By exploiting the massive parallelism of modern graphics processing units, interactive computation times can be achieved, even for high volumetric resolutions. This allows the user to almost immediately view the results and conveniently interact with the tool by placing additional strokes to refine the results. As shown in various experiments, objects with arbitrary topology can be reconstructed at a high level of detail, with the performance mainly depending on the quality of the user input.

Keywords 3D reconstruction, segmentation, interactive, fusion, semantic & depth, convex optimization, total variation, depth map generation, GPU, parallelism

Kurzfassung

Die Rekonstruktion einer Szene anhand mehrerer Aufnahmen von verschiedenen Blickrichtungen ist eines der fundamentalen Probleme in Computer Vision. Die meisten Algorithmen versuchen die komplette Umgebung zu rekonstruieren, während das Ziel dieser Arbeit die Erstellung eines 3D Modells eines einzelnen, vom Benutzer ausgewählten Objektes ist. Das Problem kann als Kombination von 3D Rekonstruktion und Segmentierung betrachtet werden. Algorithmen für ein interaktives Tool wurden entwickelt, in welchem der Benutzer sowohl das zu rekonstruierende Objekt als auch Regionen des Hintergrundes in einem oder mehreren Bildern markiert. Anhand dieser Markierungen werden Farbmodelle erstellt, welche Objekt- und Hintergrund-Regionen beschreiben. Zusätzlich werden Tiefenbilder berechnet, die mit den Wahrscheinlichkeiten der Farbmodelle im 3D-Raum kombiniert werden. Diese Kombination aus Semantik und Tiefeninformation bildet den Kern dieser Arbeit. Das Problem wird als Minimierung einer konvexen Energiefunktion interpretiert, wobei unter Verwendung des Total Variation Ansatzes die wahrscheinlichste Oberfläche berechnet wird. Das Optimierungsproblem wird direkt im 3D-Raum gelöst, anstatt jedes Bild einzeln zu segmentieren und die Ergebnisse zu kombinieren.

In der Implementierung des Tools werden die teils rechenintensiven Algorithmen parallelisiert und auf die dafür optimierte Grafikkarte ausgelagert. Dadurch werden interaktive Laufzeiten erreicht, auch für hohe Auflösungen. Der Benutzer kann damit direkt auf die Ergebnisse reagieren und zusätzliche Markierungen platzieren, falls dies notwendig ist. Wie in verschiedenen Experimenten gezeigt wird, können Objekte mit beliebiger Topologie detailgetreu rekonstruiert werden, wobei die Qualität großteils vom Benutzer-Input abhängig ist.

Schlagwörter 3D Rekonstruktion, Segmentierung, interaktiv, Fusionierung, Semantik & Tiefeninformation, Konvexe Optimierung, Total Variation, Tiefenkarten, GPU, Parallelisierung

Acknowledgments

I would like to express my gratitude to all the people who helped me in whatever way they did during my work on this thesis. First, I would like to thank my advisors Dr. Thomas Pock and Dr. Stefan Kluckner, who both always had an open ear for my questions and concerns. Their guidance was of great help in all the time of project work and writing this thesis. Without their ideas and immense knowledge this thesis would not have been possible. Furthermore, I am grateful to Dr. Manfred Klopschitz for his help, especially at the beginning of this project.

Also, I would like to thank my parents for giving me the opportunity to study and for encouraging and supporting me in everything I do. During the past years they provided me with the basic requirements needed to study here in Graz.

Finally I am deeply thankful to my friends, for always being there when I need them and for giving me the distraction I sometimes needed, especially during the process of writing this thesis.

Contents

Chapter 1

Introduction

Contents

1.1 Motivation

Reconstructing a 3D Scene from multiple images is a typical inverse problem ([1]). Instead of modeling the camera projection based on a known scene to approximate the resulting images, the inverse problem is solved. Given a set of images, one seeks to find the most probable 3D surface giving rise to these observations, given the calibrated cameras. Since depth information is lost during the projection process, inferring 3D information from 2D images is an ill-posed problem. This means that multiple 3D models can project to the same images and a unique solution can therefore not be guaranteed.

In this work, the task is to reconstruct only one single object of interest while all other parts of the scene need to be discarded. This leads to a binary segmentation problem, where volumetric elements are classified as either being part of the object of interest, or background. Note that in the remainder of this work, the term 'background' will refer to any part of the scene except the object to be modeled.

Since various well-established techniques for camera calibration based on multiple images exist, this task is not part of this work. This means that the camera parameters and pose are assumed to be given. The main contribution of this work is to reconstruct

one meaningful object within a scene, while several methods for recovering a scene as a whole already exist, with the ability to achieve very good results. The difficult problem tackled in this work is the semantic interpretation of the scene, i.e. the identification of one object of interest. This object is densely reconstructed and separated from the rest of the scene, which is typically not done in most structure-from-motion algorithms and can be very useful in many applications.

One approach to solve the 3D reconstruction and segmentation problem is to try to identify the object boundaries in each of the views and intersect their projections in 3D space. However, this approach is usually not feasible, since it heavily depends on the quality of the 2D segmentation - Erroneous results in only one of the images lead to distortions in the 3D model. Instead, a probabilistic approach was chosen, which uses a discrete voxel grid. An optimization step in scene space computes the most probable surface based on the image observations. Probabilities for each voxel are assigned, which are based on the color values of their projections as well as depth information. Obviously, some user input needs to be given, indicating which object is to be segmented. This step is performed by asking the user to place a few strokes on up to four of the input images, allowing the system to learn properties of object- and background regions based on the marked color values. The images presented to the user are chosen such that the object is shown from directions as different as possible, such that any part can be marked.

There are several important applications for 3D reconstruction of objects based on multiple views. Arguably, the most important one is augmented reality, where the 3D model can be used to interact with the real world. As an example, human organs can be modeled based on CT images, allowing doctors to inspect a dense 3D model instead of single images. Other applications are 3D face recognition or the generation of geometric 3D models for movie industry, games or virtual environments, just to name a few.

1.2 Related Work

Retrieving an accurate dense 3D model from a set of multiple views (also referred to as "Structure from Motion") is an active field of research, and many different algorithms have been proposed. A review of different techniques together with a quantitative evaluation was carried out by Seitz et al. ([2]). According to their work, multiview stereo algorithms can be characterized by their fundamental properties, i.e. scene representation, photo-consistency measure, shape prior, visibility model and reconstruction algorithm. Many of the most successful methods work by estimating the distance between each pixel's

corresponding 3D point and the camera center. This yields a so-called depth map (or range image) for each of the views. Fusing the individual range images usually results in a dense 3D model, if the object is covered sufficiently by the camera views. Examples of such approaches can be found in [3], [4] and [5].

Other popular methods use space carving to remove voxels from an initially solid 3D grid in case they are not photo-consistent in their projections ([6]). Similarly, instead of carving out voxels which are likely to belong to the background, the object can be reconstructed by adding solid elements to an initially empty voxel space at consistent positions. Such photo-consistency based techniques recover the photo hull of the object, which is the tightest bound on the true 3D scene that yields photo-consistent views in its projections. Another approach, so-called Voxel Coloring, views the task of scene reconstruction as a *color reconstruction* problem ([7]).

However, the vast majority of multiview reconstruction algorithms attempt to recover the scene as a whole instead of only one object of interest, which in contrast is the aim of this work.

The earliest approaches in binary 3D scene segmentation, i.e. identifying and reconstructing a single object in a scene, used silhouette-based methods ([8], [9]). Boundaries of the object are identified in each of the views and combined by intersecting the viewing rays going through boundary pixels. While these methods are quite stable, able to recover homogeneous regions and computationally efficient, they suffer from a few drawbacks. Object concavities can not be reconstructed, since they do not affect the silhouettes. Furthermore, the result heavily depends on the computation of single view segmentations, which is a difficult task by itself.

In order to select the object of interest and to find properties of both the object and the rest of the scene, user interaction is typically required. A popular approach is to let the user place strokes on both object and background image regions, such that the system can learn color properties based on this input. Such methods are well-known from 2D segmentation problems, e.g. when using Graph Cuts ([10], [11]).

Kolev et al. ([12]) presented an interactive scene segmentation approach using a probabilistic voxel grid and a total variation optimization step. These core ideas also form the basis of this thesis. However, their method is restricted to uniformly colored objects and does not account for different illumination conditions among the views. Furthermore, depth information is not including, which is why only the visual hull of the object can be computed. Their GPU-based implementation, in which computations for each voxel are

executed independently on parallel threads, serves as a model for the implementation in this work.

Reinbacher et al. ([13]) use a very similar approach as in [12] in order to identify an object in multiple views. Their method is based on backprojection of spatial constraints and is able to achieve very good results. However, the object itself is not reconstructed, but it's silhouettes in the input images. The proposed method, in contrast, tries to reconstruct the object directly in object space, since the fusion of silhouettes only yields the visual hull.

1.3 Contribution

A novel approach to 3D reconstruction and segmentation is presented in this work, which combines a probabilistic volumetric model with the fusion of depth maps. While most interactive scene segmentation algorithms use Mixtures of Gaussians based on color values to train a model for regression ([12], [13]), the concept of Random Forests was used in this work, as done for example in [14]. Some adaptations to the original concept were made to make this regression model suitable for the task of image segmentation. For the computation of range images, a multiview plane sweep algorithm was used. A core feature of the implementation is that random forest training and testing, computation and fusion of depth maps as well as a final convex optimization step are performed on programmable graphics hardware. Therefore, a high degree of parallelism can be achieved, leading to a major decrease in computational time compared to a CPU-based implementation. While being able to almost immediately showing results to the user, the 3D model can be reconstructed at a high level of detail and without major distortions, as long as plenty of views are present and the quality of the user input is sufficiently high.

1.4 Thesis Overview

The remainder of this thesis is organized as follows: Chapter 2 explains the most relevant theoretical concepts for 3D reconstruction and segmentation, as well as the basic algorithms and building blocks used in this work. After introducing fundamental principles such as color spaces and the pinhole camera model, a brief overview over various image segmentation methods is given. How to recover depth information from multiple calibrated views is also subject of this section, together with methods for recovering 3D structure. Subsequently, the concept of random forests for classification and regression is

explained, followed by an introduction to total variation based global optimization.

Chapter 3 introduces the multiview scene segmentation tool implemented in this work, where each building block is examined in detail. It is closely related to Chapter 2, since it shows the application of the algorithms introduced there. References to corresponding sections are therefore present at adequate points.

Chapter 4 briefly discusses how to implement the algorithms using programmable graphics hardware in order to achieve an impressive performance gain in terms of execution time. Evaluation using visual and quantitative results is performed in Chapter 5. Finally, Chapter 6 gives a conclusion together with suggestions for future work.

Chapter 2

Methodology

Contents

Outline: *This chapter describes the underlying principles and algorithms that form the building blocks of the 3D segmentation algorithm. First, in Section 2.1, different color models are reviewed, including the CIE Lab model used in this work. In Section 2.2, the Pinhole Camera Model is introduced, leading to the derivation of the camera projection matrix P and ways of estimating it. Furthermore, is is shown how camera parameters and properties can be derived from P. Section 2.3 gives an overview of image segmentation methods. The subsequent Section 2.4 describes the so-called 'Plane Sweep Algorithm', which is used for computing depth maps based on multiple views. In Section 2.5, methods for 3D reconstruction of a scene are explained. The segmentation algorithm proposed in this work uses a regression model called 'Random Forest' to assign background/object-probabilities based on color values. The basic principle behind Random Forests is described in Section 2.6. The method of Total Variation in Image Segmentation is discussed in Section 2.7, together with an iterative algorithm to solve the resulting optimization problem.*

2.1 Color Models

Visible light is electromagnetic radiation with wavelengths between approximately 380 nm and 750 nm. Most light sources emit light at many different wavelengths, but different objects absorb certain parts of the spectrum. If, for example, an object reflects only electromagnetic waves with wavelengths between 640 nm and 700nm and mostly absorbs other frequencies, this object appears to be red to a human observer. In order to facilitate the specification of colors in some standard, accepted way, various color models have been proposed. A color model is usually a specification of a coordinate system (typically 3- or 4-dimensional) and a subspace within that system, where each color is represented by a coordinate tuple. While it is impossible to describe every possible perceived color in such a coordinate system, color models aim to cover the significant part of the human vision color space sufficiently. In the remainder of this section, several different color spaces are briefly introduced. See [37] for an extensive review.

2.1.1 RGB Color Model

The RGB Color Model is the most popular and widely used model in computer vision. It uses additive color mixing of the primary colors red, green and blue (therefore its acronym RGB). The human eye has three types of color receptors which are stimulated by electromagnetic waves in the spectrum of these three colors. If media transmits light, it uses any combination of these colors to produce a large part of the human color space. Mixing all primary colors at full intensity yields white, while zero intensity defines black. Adding two of the three primary colors produces the secondary colors cyan, magenta and yellow.

Some variations of the RGB model exist, such as sRGB or Adobe RGB.

2.1.2 CMYK Color Model

In contrast to the RGB model, CMYK is a subtractive color model. It is an acronym for the secondary colors cyan, magenta and yellow, together with black (denoted by K). These colors are subtracted from white to produce other colors. Printers use this model by applying pigments of ink in these secondary colors to a white surface to subtract some color from it. Subtracting two of the secondary colors cyan, magenta and yellow from white produces the primary colors red, green and blue. Figure 2.1 shows how the primary colors can be produced from the secondary ones and vice versa.

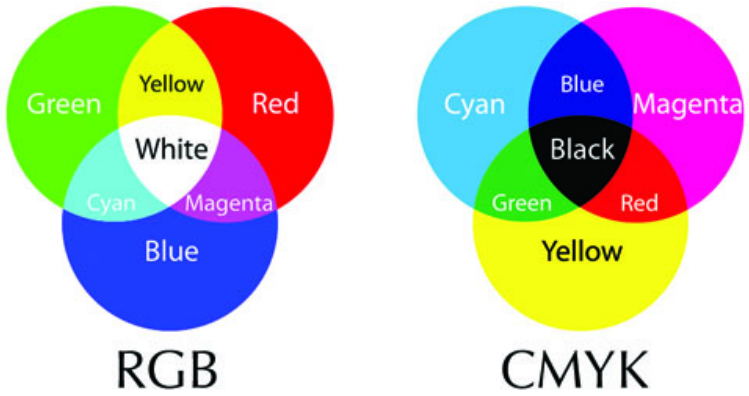

Figure 2.1: Additive and Subtractive Color Mixing. Subtracting two of the Secondary Colors from White yields the Primary Colors. Image taken from `http://picsbox.biz/key/%20Color%20Systems`

2.1.3 HSV and HLS Color Models

HSV (**h**ue, **s**aturation, **v**alue), also known as HSB (where the 'B' stands for brightness), rearranges the geometry of the RGB color space in order to achieve a more intuitive representation. It is very similar to the HLS (**h**ue, **l**ightness, **s**aturation) model. In both models, colors can be manipulated more intuitively. While hue defines the color itself, saturation defines how much the hue differs from the neutral gray. The intensity component (value or brightness) indicates the illumination level. Every color value from the RGB model can be uniquely transformed to any of these two models via transformation equations, and vice versa. See [37] for details.

Even though these two models are often more intuitive or suitable for image processing applications, they are also criticized for not separating the attributes adequately as well as for not being perceptually uniform.

2.1.4 CIE XYZ Model

The XYZ color model was developed by the CIE (Commission Internationale de l'Éclairage, International Commission on Illumination). It is based on three primaries X, Y and Z, which, in fact, are only hypothetical and do not correspond to any real light wavelengths. The Y-component approximately matches to luminance, while X and Z define the color itself. However, arbitrary combinations of the three components don't necessarily yield visible colors. The main advantage of the CIE XYZ model and its derivations is the fact that it is device-independent.

2.1.5 CIE Lab and CIE Luv Model

The CIE Luv and CIE Lab color models are very similar and both derivations of the CIE XYZ space. Their main advantage is that they are considered to be perceptually uniform. This means that the Euclidean distance between two points in the color space approximates the distance a human would sense. Similar colors therefore lie close together in the coordinate system, while perceptually different ones lie far apart. This property can be of advantage in many image processing and computer vision systems. In this work, color values are classified based on their proximity to other, already labeled, values. Since perceptually similar colors should be assigned the same label, the CIE Lab (or alternatively, the CIE Luv) model is the model of choice.

Furthermore, the L-component directly indicates the lightness, similar as the Y-component in the XYZ model. However, in comparison, it approximates visual differences better. The other two components hold information about the color itself. RGB and CMYK only contain parts of the Lab and Luv space, as indicated in Figure 2.2. It shows the representable colors by the XYZ model (and its derivations Lab and Luv) compared with sRGB (inside the black triangle) and CMYK (inside the gray polygon).

Figure 2.2: CIE XYZ Chromaticity Space compared with sRGB (black) and CMYK (gray). Image taken from `http://www.posterlia.at/service/softproof.html`

2.2 Pinhole Camera Model

An image is a 2D projection of a 3D real-world scene. While modeling this projection mathematically can be rather complex, the Pinhole Camera Model often suffices in describing these relations by making some idealizing assumptions.

2.2.1 Derivation of the Camera Matrix P

Figure 2.3 shows the geometry of the Pinhole Camera Model. A 3D Point $\mathbf{X} = (X, Y, Z)$ is projected onto the image plane by intersecting that plane with the viewing ray passing through $\mathbf{X}$ and originating at the camera center $\mathbf{C}$. Any real-world point lying on this viewing ray projects onto the same point in the image plane. Therefore, given an image point and the corresponding camera calibration data, the actual 3D position of the point cannot be recovered, only the viewing ray it lies on.

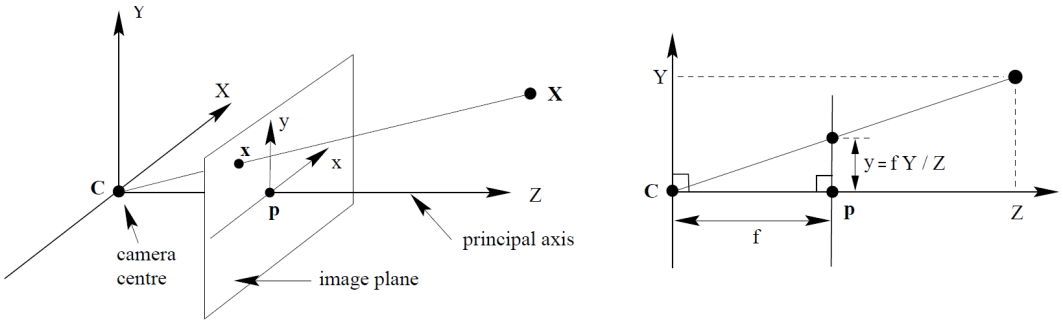

Figure 2.3: Geometry of Pinhole Camera Model. Left: Intersection of a Viewing Ray through the Point $\mathbf{X}$ with the Image Plane yields Image Point $\mathbf{x}$. Right: Computation of $\mathbf{x}$ using Similar Triangles. Images taken from [22]

Assume both the 3D point $\mathbf{X}$ and its projection $\mathbf{x} = (x, y, z)$ are described in terms of the camera coordinate system with its origin in $\mathbf{C}$ and its Z-axis perpendicular to the image plane, as shown in Figure 2.3. Then, by using similar triangles, the following relations can be obtained:

$$\frac{x}{X} = \frac{f}{Z}, \quad \frac{y}{Y} = \frac{f}{Z} \quad \Rightarrow \quad x = \frac{f\,X}{Z}, \quad y = \frac{f\,Y}{Z} \tag{2.1}$$

where f is called the *focal length* of the camera.

Using homogeneous coordinates allows for a convenient representation in

matrix form:

$$\begin{bmatrix} x\,w \\ y\,w \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.2}$$

This equation only holds if the origin of the image coordinate system is at the principal point **P**, which is defined as the intersection of the image plane with the Z-axis of the camera coordinate system (see Fig. 2.3). However, this assumption does generally not hold, since the origin is usually defined to lie at the top-left or bottom-left corner of an image, therefore allowing only for positive pixel coordinates. In order to shift the image coordinate origin to the desired position, two translation parameters t_x and t_y are introduced:

$$\begin{bmatrix} x\,w \\ y\,w \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & t_x \\ 0 & f & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.3}$$

While Equation (2.3) computes continuous 2D coordinates in the same measurement unit as the real-world 3D point (e.g. mm or inches), one is typically interested in computing discrete pixel coordinate values. Therefore, the pixel resolutions m_x and m_y are introduced (e.g. in pixels/mm or pixels/inch). For square pixels, as usually used, m_x equals m_y. Also, an additional skew parameter s can be included, which accounts for non-orthogonal image coordinate axes.

This leads to the projection equation

$$\begin{bmatrix} x\,w \\ y\,w \\ w \end{bmatrix} = \begin{bmatrix} m_x\,f & s & m_x\,t_x \\ 0 & m_y\,f & m_y\,t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{K}_{3\times3}\,\mathbf{X} \tag{2.4}$$

The matrix **K** is called the intrinsic- or calibration matrix of the camera. Alternatively, it can be written as

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

where α_x and α_y are the focal length in image pixels, s the skew parameter and $[-x_0, -y_0]^\top$ the principal point in pixel coordinates. Usually, $\alpha_x = \alpha_y$ and $s = 1$.

A camera typically also shows non-linear lens distortion, which can also be modeled using a non-linear distortion function of the projected image coordinates [22]. Typically, a quadratic function is used, which is applied separately to the linear projection matrix $\mathbf{P}$ to undistort an image before further processing.

The calibration matrix projects 3D world coordinates to 2D pixel coordinates. However, the 3D coordinates are assumed to be defined in terms of the camera coordinate system, where the Z-axis is perpendicular to the image plane. This is not the case for many scenarios, since usually a more convenient coordinate system is used to describe the scene, e.g. one where the Z-axis is perpendicular to the ground plane. Also, in a multiview setup, each camera has its own coordinate system, while the scene coordinate system is common to each of them. In order to be able to use the projection given by Equation (2.9), the world coordinate system has to be rotated and translated such that its axis coincide with the ones of the camera coordinate system. Mathematically, this can be described by multiplying a world coordinate vector with a rotation matrix $\mathbf{R}$ and adding a translation vector $\mathbf{t}$. This process is depicted in Figure 2.4.

$\mathbf{R}$ is a 3x3 rotation matrix. It is therefore orthogonal and has the following properties:

$$\mathbf{R}^{-1} = \mathbf{R}^\top, \quad |\mathbf{R}| = 1, \quad \mathbf{R}^\top\mathbf{R} = \mathbf{I} \tag{2.6}$$

The translation vector $\mathbf{t}$ can be interpreted as the position of the world coordinate origin described by the camera coordinate system. Together with the rotation matrix $\mathbf{R}$ it forms the extrinsic parameter matrix $[\mathbf{R}|\mathbf{t}]$, which is multiplied with the intrinsic matrix $\mathbf{K}$ to finally yield the 3x4 projection matrix $\mathbf{P}$:

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R} \mid \mathbf{t}] \tag{2.7}$$

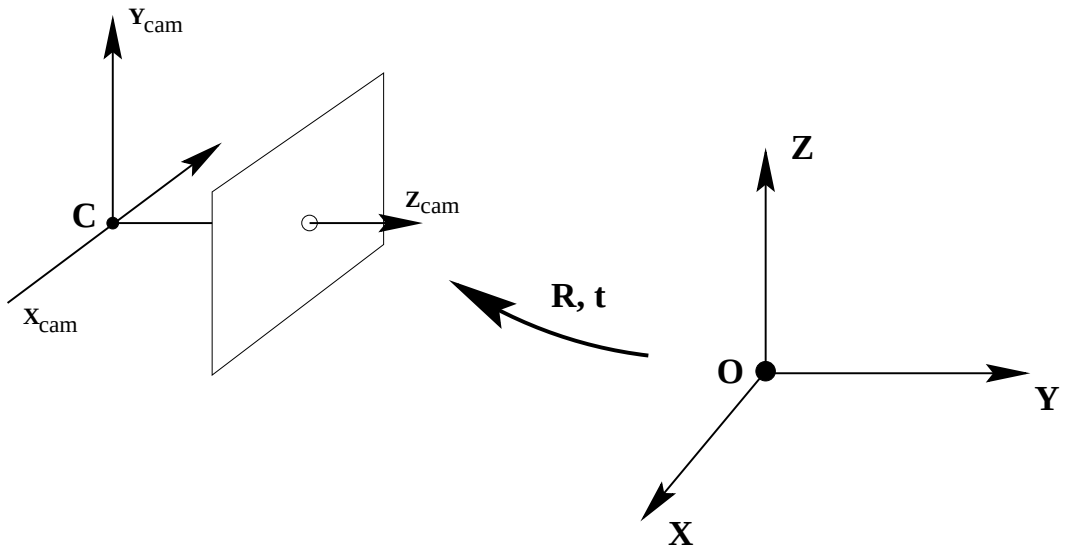

Figure 2.4: Transformation from World- to Camera Coordinate System via a Rotation **R** and a Translation **t**. Image taken from [22]

Alternatively, the extrinsic matrix can be formulated in terms of the camera center in world coordinates, **T**:

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R} \mid -\mathbf{R}\mathbf{T}] \tag{2.8}$$

Using the projection matrix **P**, any 3D scene point can be projected onto the image plane, yielding 2D pixel coordinates:

$$\begin{bmatrix} x\,w \\ y\,w \\ w \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{K}_{3\times 3}\,\mathbf{X} \tag{2.9}$$

The actual discrete pixel coordinates can be obtained by rounding x and y to the nearest integer value.

2.2.2 Camera Resectioning

Usually, the intrinsic and/or extrinsic parameters of the camera are not given and need to be estimated from given images or information about the scene geometry. This

process is called camera resectioning. Often the term "camera calibration" is used instead. However, this might be misleading, since camera calibration can also refer to color mapping, i.e. applying color transformations to images.

Coordinates-based calibration methods use observations of 3D scene points and their corresponding projections in the image. A special calibration pattern is used in classical methods to estimate the projection matrix. Often, a checkerboard pattern is used, since its geometry is simple and corresponding corner points can be found easily. The 3D coordinates of these points can be measured, while their correspondences in the images can be identified using simple corner detection methods. Figure 2.5 shows an example of such a calibration object.

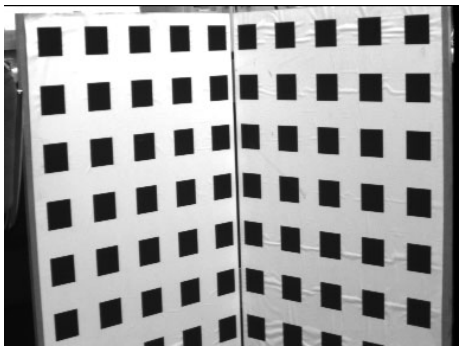

Figure 2.5: Sample Calibration Object. Image taken from [22]

In general, the 3x4 projection matrix $\mathbf{P}$ has 11 unknown parameters, since it is only defined up to scale. The projection of a 3D scene point $\mathbf{X} = [X, Y, Z]^\top$ to image pixel coordinates $\mathbf{x} = [u, v]^\top$ can be written as

$$\begin{bmatrix} u\,w \\ v\,w \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.10}$$

Expanding the right-hand side and dividing by w yields

$$u = \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33}Z + 1} \tag{2.11}$$

$$v = \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33}Z + 1} \tag{2.12}$$

Since each point correspondence leads to two equations, at least 6 correspondences are needed to solve for the 11 unknown (in fact only one of the two coordinates from the 6^{th} correspondence has to be known, providing the 11^{th} equation). If this minimum number of point matches is given, the solution is exact. However, if available, more than $n \geq 6$ correspondences are used, since these observations are often noisy. This leads to an over-determined system of linear equations. Expanding Equations 2.11 and 2.12 and joining the resulting $2n \geq 12$ equations together into a matrix yields the following representation for n correspondences:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 \\ \vdots & & & & & & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \end{bmatrix} \tag{2.13}$$

This is an over-determined system of linear equations of the form $\mathbf{A}\,\mathbf{x} = \mathbf{b}$. In order to find the "best" of the infinite set of solutions, this matrix equation can be solved with respect to an error measure. Minimizing the mean-squared error yields to following optimization problem:

$$\min_x \|\mathbf{A}\,\mathbf{x} - \mathbf{b}\|^2 \tag{2.14}$$

The minimum mean-squared error solution can be obtained using the pseudo-inverse of $\mathbf{A}$ as

$$\mathbf{x} = \mathbf{A}^{+}\mathbf{b} = \left(\mathbf{A}^{\top}\mathbf{A}\right)^{-1}\mathbf{A}^{\top}\mathbf{b} \tag{2.15}$$

An alternative way of writing Equation 2.13 is

$$\begin{bmatrix} \mathbf{0}^\top & -w_i\mathbf{X}_i^\top & y_i\mathbf{X}_i^\top \\ w_i\mathbf{X}_i^\top & \mathbf{0}^\top & -x_i\mathbf{X}_i^\top \end{bmatrix} \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix} = \mathbf{0} \tag{2.16}$$

where $\mathbf{X_i}$ and $\mathbf{x_i}$ are the n point correspondences in homogeneous coordinates and $\mathbf{p}^{i\top}$ is the i-th row of $\mathbf{P}$. The matrix $\mathbf{A}$ is obtained by stacking up these equations. Then, the set of equations $\mathbf{Ap} = \mathbf{0}$ is solved. Since in an overdetermined system there is no exact solution, the problem can be interpreted as minimization of $\|\mathbf{Ap}\|$. In order to avoid the obvious solution $\mathbf{p} = \mathbf{0}$, an additional constraint on the norm has to be used, e.g. $\|\mathbf{p}\| = 1$. The solution can be computed using the Direct Linear Transformation (DLT) Algorithm ([22]), where the Singular Value Decomposition (SVD) of the matrix $\mathbf{A}$ is computed. The solution for $\mathbf{p}$ is the unit singular vector corresponding to the smallest singular value. This linear solution then serves as a starting point for minimizing the geometric error by using an iterative technique, such as the Levenberg-Marquardt Algorithm ([25], [26]).

In many cases linear models do not suffice in estimating the projection matrix properly, since radial distortion is present. More sophisticated methods also models this property. In [32], a step-wise method is presented that models radial distortion, but assumes some camera parameters are provided by the manufacturer. For example, the principal point is not among the estimated parameters but assumed to lie in the middle of the image (it was included in a later formulation of the algorithm, see [33]). Another popular method is the one presented in [34]. It uses a planar calibration pattern to estimate the homography between the image plane and the calibration target. An advantage of this approach is that either the camera or the pattern can move freely, which makes it more flexible compared to methods where the 3D points need to be known precisely.

Another class of calibration algorithms is self-calibration. Here, no calibration object is used. The internal camera parameters are estimated by finding correspondences in a set of uncalibrated images, making a Euclidean reconstruction possible. The first multiview self-calibration method was introduced in [27]. A variety of algorithms have been developed since then, see [29], [30] or [31] for examples.

2.2.3 Decomposing the Camera Matrix

In this work, multiple images of a 3D scene are given, together with corresponding camera projection matrices $\mathbf{P}$. This section describes how $\mathbf{P}$ can be decomposed into its intrinsic and extrinsic components. This is a necessary step, for example for the computation of depth maps, where the camera parameters and poses need to be known.

The projection matrix $\mathbf{P}$ can be written as a 3×3 sub-matrix $\mathbf{P}_{3\times3}$ and a 3×1 column vector $\mathbf{p}_4$ in the following form:

$$\mathbf{P}_{3\times4} = [\mathbf{P}_{3\times3} \mid \mathbf{p}_4] \tag{2.17}$$

The matrix $\mathbf{P}_{3\times3}$ itself is computed as the product of the intrinsic matrix $\mathbf{K}$ and the rotation matrix $\mathbf{R}$:

$$\mathbf{P}_{3\times x3} = \mathbf{K}\,\mathbf{R} \tag{2.18}$$

The decomposition of $\mathbf{P}_{3\times3}$ into $\mathbf{K}$ and $\mathbf{R}$ can be obtained using the RQ-Decomposition of a matrix, which decomposes a square matrix into the product of an upper triangular matrix and a rotation matrix. For details on the RQ-Decomposition of a matrix see [22]. The position of the camera center in world coordinates (vector $\mathbf{T}$ in Equation (2.8)) can easily be computed from $\mathbf{P}$:

$$\mathbf{P} = [\mathbf{KR} \mid -\mathbf{KR}\ \mathbf{T}] = [\mathbf{P}_{3\times3} \mid \mathbf{p}_4] \tag{2.19}$$

$$\rightarrow \mathbf{p}_4 = -\mathbf{P}_{3\times3}\ \mathbf{T} \tag{2.20}$$

$$\rightarrow \mathbf{T} = -\mathbf{P}_{3\times3}^{-1}\ \mathbf{p}_4 \tag{2.21}$$

Note that the translation vector $\mathbf{t}$ can easily be obtained from $\mathbf{T}$ using (from Equation (2.7) and Equation (2.8))

$$\mathbf{t} = -\mathbf{R}\ \mathbf{T} \tag{2.22}$$

Some other properties of the camera and the scene can be computed from the camera matrix $\mathbf{P} = [\mathbf{P}_{3\times3} \mid \mathbf{p}_4]$. See [22] for a more detailed and complete summary. Some of the most useful relations are the following:

- The column vectors of $\mathbf{P}_{3\times3}$ are the vanishing points in the image, corresponding to the X, Y and Z axes, respectively.
- The last row of $\mathbf{P}$ parameterizes the principal plane of the camera.
- The principal point $\mathbf{x}_0$ in the image is computed as $\mathbf{x}_0 = \mathbf{P}_{3\times3}\overline{\mathbf{p}}_3$, where $\overline{\mathbf{p}}_3^\top$ is the last row of $\mathbf{P}_{3\times3}$.
- The principal axis vector of the camera is determined as $\mathbf{v} = \det(\mathbf{P}_{3\times3})\,\overline{\mathbf{p}}_3$.
- A viewing ray from the camera center through an image pixel $\mathbf{x}$ can be computed using $\mathbf{X}(\lambda) = \mathbf{P}^+\mathbf{x} + \lambda\mathbf{T}$, where $\mathbf{P}^+$ is the pseudo-inverse of $\mathbf{P}$, s.t. $\mathbf{P}\mathbf{P}^+ = \mathbf{I}$, $\mathbf{T}$ is the camera center in world coordinates and λ parameterizes any point on the ray.

2.3 Image Segmentation

Image segmentation divides an image into two ore more distinct parts, i.e. connected pixel regions which can usually be interpreted as meaningful parts of a scene. In binary segmentation, as used in this work, an object in an image is classified as such, while the rest is interpreted as background. In multi-label segmentation, several objects are identified. Image segmentation can either be fully automatic or interactive. In interactive segmentation methods some user input is provided, which is usually the number of distinct regions and prior information about them, i.e. color or texture properties. A valid segmentation of an image I into N distinct regions S_i has the following properties:

- $I = \bigcup_{i=1}^{N} S_i$
- $S_i \cup S_j = \emptyset$
- $H(S_i) = \text{True } \forall i$
- $H(S_i \cup S_j) = \text{False } \forall \text{ neighboring } S_i \text{ and } S_j,\ i \neq j$

where $H(\cdot)$ is a homogeneity criterion, indicating that a region cannot be split any further with respect to some criterion. Figure 2.6 shows an example of a valid segmentation into meaningful regions. It already illustrates some of the difficulties in image segmentation – objects often don't have uniform color values, boundaries between them are not always clearly identifiable and fine details and textures complicate the process.

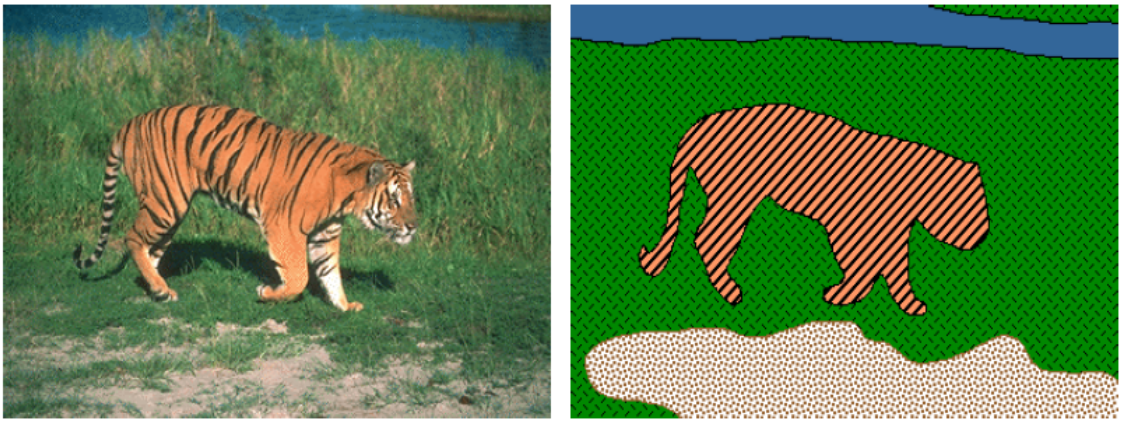

Figure 2.6: An Example of Image Segmentation. Left: Original Image. Right: Division into meaningful Segments. Images taken from `http://graphics.cs.cmu.edu/courses/15-463/2004_fall/www/Lectures/BlobProcessing.pdf`

A wide variety of image segmentation algorithms have been developed over the years, but there is not a single method that performs superior to all others on different kinds of images. Even selecting a method for a known type of images can be a difficult task. Basically, most algorithms work by either finding discontinuities in images (possible segment boundaries) or similarities (pixel regions with similar properties, e.g. textures, intensities or color values). In the remainder of this section, several of the most popular segmentation methods are briefly described. More detailed surveys about image segmentation methods can be found in most textbooks on image processing (e.g. [36], [37], [35]).

2.3.1 Overview of Image Segmentation Methods

The simplest of all image segmentation methods is **thresholding**. It operates on single-channel images and often gives sufficient results when there is a dark object on bright background or vice versa. Each pixel is compared with a threshold and labeled either as background or foreground, depending if it is greater than the threshold or not. In order to account for changes in illumination, the optimal threshold can be determined in a local neighborhood instead of globally. However, thresholding cannot be applied to color images and does not take spatial characteristics of the scene into account, but works only pixel-based. Also, in many scenes a meaningful segmentation only based on thresholding cannot be obtained.

Another, more sophisticated method is **region growing**. Here, the idea is to group pixels into regions based on pre-defined criteria, such as color similarity. Starting with

a seed pixel, neighboring pixels are added to the region if they are similar with respect to these criteria. The region properties are then updated by including the newly added pixels and the procedure is repeated, until no more pixels can be added. However, region growing is very sensitive to noise and does not incorporate global information.

While Region Growing is a typical bottom-up method for image segmentation, starting at single seed points and growing regions around them, **Split and Merge** is a top-down approach. Starting point is the image as a whole. Again, some homogeneity criterion is defined. If this criterion is not fulfilled for the whole region, it is divided into four subregions. These are again tested for homogeneity and further subdivided into quadrants, until the criterion is fulfilled. At this point, adjacent regions can have identical properties. An additional merging stage connects all these regions leading to the final segmentation result, which is obtained when no further merging is possible. Split and Merge usually yields similar results as region growing and suffers from the same drawbacks.

Another popular class of segmentation algorithms is **edge based segmenation**, which can be a powerful tool if adjacent regions in the image have sharp transitions in intensity values. Such segmentation methods work by identifying edges in an image, which serve as estimates for the region borders. A refinement step is usually needed to link edges such that closed boundaries are formed, as well as to remove meaningless edges stemming from noise. More sophisticated methods interpret the border detection problem in terms of graph theory, where finding the optimal path corresponds to identifying the most likely border. A cost function has to be defined, which is usually based on edge strength, curvature, proximity etc. – paths with minimum cost within the graph are then determined. Another approach uses the Hough transform, which works especially well if the objects to be detected can be parameterized, e.g. if they are similar to circles or ellipses.

Watershed Segmentation uses the idea of interpreting a graylevel image as a topological surface. It grows regions around local minima of the gradient image, and can therefore be interpreted as a combination of region-based and edge-based methods. It always produces closed region boundaries and is often more stable then the methods introduced so far. On the downside, over-segmentation often occurs, which can only be prevented by manually defining seed points or the number of regions.

Segmentation based on **Active Contours** also works by detecting boundaries in an image. A contour, which is usually initialized by user interaction, is attracted to the final solution by imaginary forces from the image data and possibly from user interaction. Three popular related approaches are **Snakes** ([42]), **Intelligent Scissors** ([43]) and **Level**

Set methods. Snakes are based on the minimization of an energy functional, which is a combination of an internal- an image- and an external energy. Snakes have several advantages, for example their usefulness for object tracking, where the segmentation of the last frame is used as initialization of the current one. Also, there is a lot of flexibility in how the energy is defined and weighted. However, the result heavily depends on a decent initialization and therefore a high-quality user interaction is important. Also, prior information about the image is often needed in order to set the parameters of the energy function well. Intelligent Scissors are an interactive tool for image segmentation. While the evolution of the contour is unpredictable when using Snakes, Intelligent Scissors allow the user to control the contour by movements of the mouse, while it evolves in real-time. In contrast to these two methods, the Level Set method does not suffer from the limitation that the curve around the objects of interest must be parametrized. Instead, another representation for the closed contours is used, so-called level sets, where the zero-crossings of a higher dimensional function define the curve. This underlying function is evolved instead of the contour itself.

Graph based segmentation methods treat an image as a weighted graph, where each image pixel is a node in the graph. Nodes within a certain neighborhood are connected by edges, with costs defined for each edge based on a similarity measure. The idea is to break the graph into segments along edges of minimal cost, i.e. minimal similarity. Similar pixels should be in the same segments, while dissimilar pixels should be in different ones. The cost of a cut in a graph is defined as the sum of all edge weights deleted by the cut. Several fast algorithms for finding minimum cuts exist, an experimental study can be found in [45]. Some algorithms make use of the *max-flow min-cut theorem*, which states that the maximum possible flow through a graph from a source to a sink is equal to the cost of the minimum cut. Boykov & Kolmogorov ([46]) proposed an efficient way to compute the maximum flow for graphs in computer vision. While graph cuts are a popular alternative to other methods that try to find contours between objects, their extensive use of memory for large images is one of their disadvantages.

Two popular approaches to image segmentation will be covered in more detail in the subsequent sections. The first one is **clustering based segmentation**, which groups image pixels into clusters of pixels with similar properties. The second one is variational methods, where additional smoothness priors are used to obtain a meaningful solution, and the problem is formulated as minimization of an energy functional. The algorithms in this work use both ideas, which is why they are covered in more detail.

2.3.2 Clustering Based Segmentation

Image segmentation can be viewed as the problem of clustering pixels with similar properties into groups. Each pixel in an image is described by a feature vector, usually including information such as color value and position. Different clustering algorithms exist, e.g. K-Means Clustering, Mean Shift Clustering or the EM Algorithm. These three algorithms are briefly described in the remainder of this section.

K-Means Clustering

The K-Means Algorithm ([16])assigns each feature vector to one of K clusters, where the number of clusters needs to be known in advance. This can be a major limitation – however, methods that estimate this number have been proposed ([38], [39]). K-Means minimizes the inner-cluster variance, which is a measure of compactness. The algorithm works as follows:

1. Choose K initial cluster centers. These can for example be randomly selected feature vectors. It has been shown that selecting feature vectors that lie far apart from each other can be advantageous.

2. Assign each feature vector to the cluster with the closest cluster center

3. Re-compute the cluster centers as the mean vector of all feature points assigned with that cluster

4. Go to step 2 and repeat until convergence

Figure 2.7 shows a demonstrative example of how the K-Means Algorithm works. Here, the initial cluster centers are chosen to be at random locations in space and do not coincide with any feature vector. The purple line indicates the decision boundary, which contains all points with equal distance to both cluster centers. The figure is is adapted from one in [41], which is an excellent resource for information on clustering techniques.
While K-Means is a very simple method, it also has some disadvantages, e.g. its sensitivity to noise and outliers. Also, the initial locations of the cluster centers can affect the result. In addition, since feature vectors are assigned to the cluster center which is closest, clusters are likely to turn out to be somewhat spherical, which is not always the desired behavior.

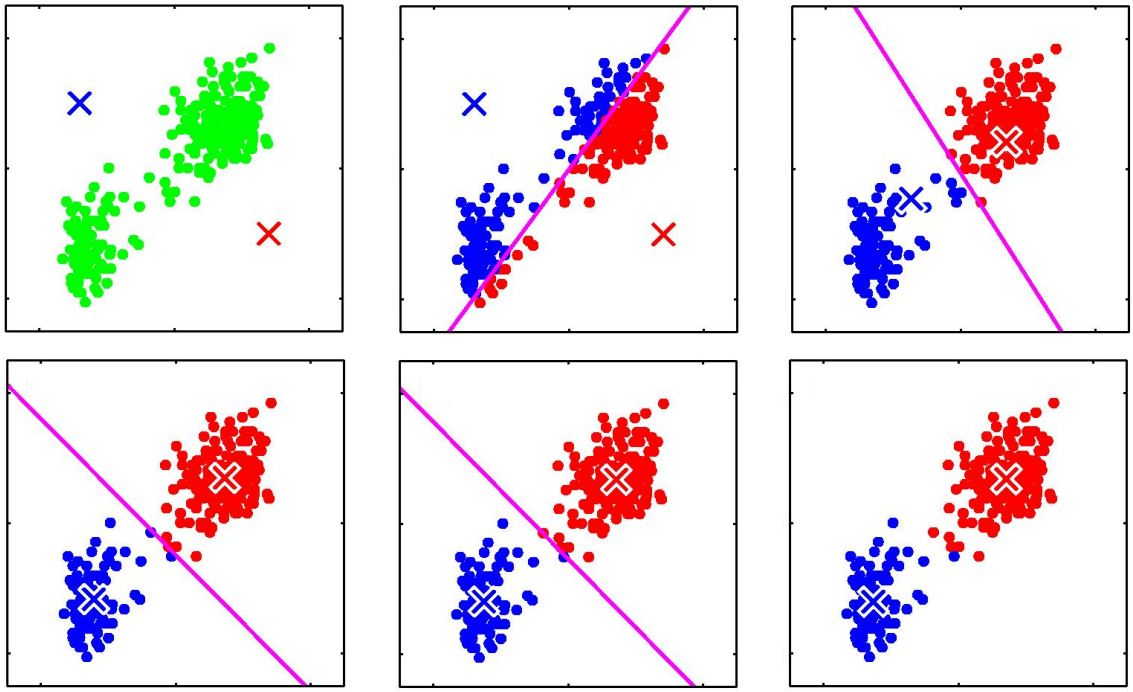

Figure 2.7: Illustration of the K-Means Clustering Procedure. First, Cluster Centers are chosen at random and Data is labeled based on which Center is nearest. Cluster Centers are then re-estimated as Mean Value of corresponding Data Points. Procedure is repeated until Convergence. Images taken from [41]

Mean Shift Clustering

In contrast to the K-Means Algorithm, Mean Shift clustering ([17]) does not require prior knowledge about the number of clusters. Also, it does not favor any kind of their shapes over others. The data points to be clustered are interpreted as samples of an underlying probability density function. For each point, the local maximum of this function is found – these maxima represent the cluster centers. The algorithm works as follows:

- Define a window around each data point
- Compute the mean of all data points within this window
- Move the window to this location
- Repeat until convergence

For each data point, the window converges to a location where the probability density is locally maximal. All data points that converge to the same point are assigned the same cluster. Local maxima that lie very close together can be joined in order to avoid over-segmentation. Figure 2.8 shows how for different data points the corresponding modes of the probability density function are found.

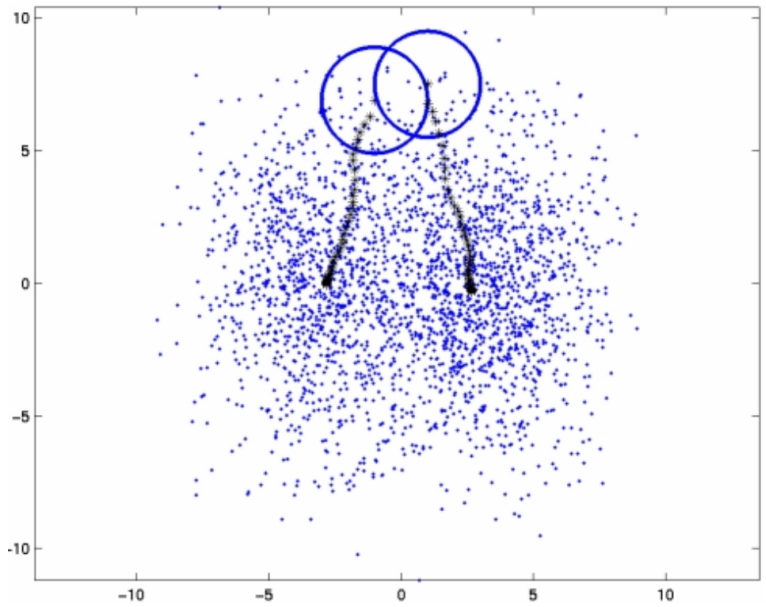

Figure 2.8: Illustration of Mode Finding in Mean Shift Algorithm. Window around Data Point (indicated by Circle) is moved towards highest Density until Convergence. Image taken from `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TUZEL1/MeanShift.pdf`

One major advantage over the K-Means algorithm is that certain shapes of clusters are not more likely than others. While in K-Means clusters are likely to be spherical, Mean Shift allows for arbitrary cluster shapes, as illustrated in Figure 2.9. Disadvantages of Mean Shift are its higher computational complexity and that the results depend on the size of the window around the data points.

EM-Algorithm

In contrast to K-Means, the *Expectation-Maximization (EM)* algorithm partially assigns data points to different clusters, instead of only to one. Each cluster is modeled using a probabilistic distribution. A data point is associated with each of

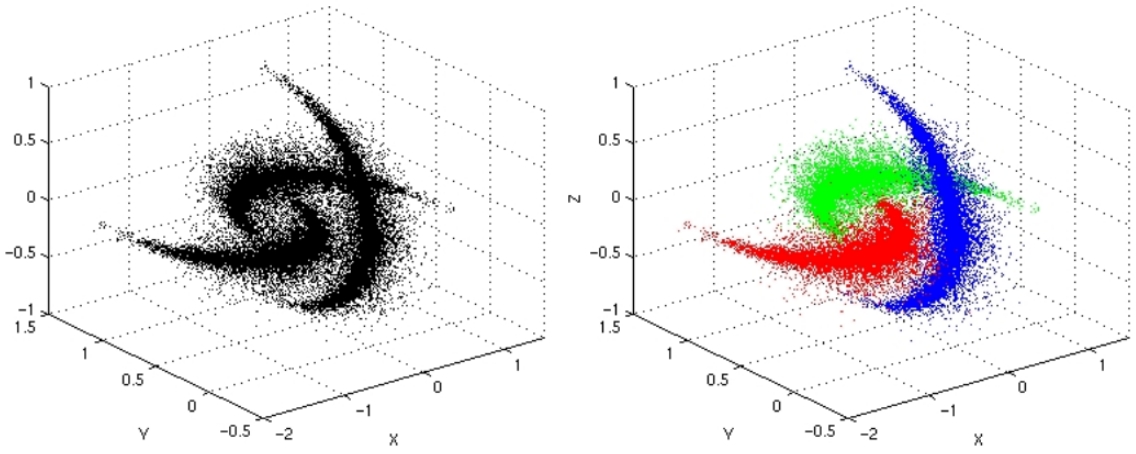

Figure 2.9: Example of Mean Shift Algorithm on non-linearly separable Clusters. Mean Shift can cluster Data with arbitrary Distribution. Image taken from `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TUZEL1/MeanShift.pdf`

them with a certain probability and finally assigned to the cluster with the highest one. Often, Gaussian Mixture Models (GMMs) are used to model the underlying probability distributions. This section briefly introduces the EM algorithm for GMMs – further information, EM for other distributions and a general view of the algorithm can be found in [41].

The GMM is a weighted sum of Gaussian distributions, with the weights summing up to 1:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2.23}$$

where π_k is the weight of the distribution for cluster k, $\boldsymbol{\mu}_k$ the mean vector and $\boldsymbol{\Sigma}_k$ the covariance matrix. The *likelihood* of a model given a set of data points is the product of the single probabilities for each point. Since this value tends to be very small and therefore underflows are likely to happen on a computer system, the so-called *log-likelihood* is used instead. Calculating the logarithm of the single probabilities and summing them up does not change the position of the maximum in parameter-space, but only the value of the likelihood. The EM-Algorithm maximizes this *log-likelihood* using the following iterative procedure:

1. Initialize $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and π_k and evaluate the initial value of the log-likelihood

$$\ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{2.24}$$

2. E-Step: Evaluate the so-called responsibilities using the current parameter values:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2.25}$$

3. M-Step: Re-estimate the parameters using the current responsibilities:

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{2.26}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})^\top \tag{2.27}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{2.28}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{2.29}$$

4. Evaluate the log-likelihood (Equation (2.24)) and return to step 2 until either the parameters or the log-likelihood have converged.

The EM-algorithm finds a maximum likelihood solution based on the given input data. However, the number of clusters needs to be known in advance. EM is closely related to the K-Means algorithm, which can be interpreted as a variant of EM. Using the EM Algorithm for GMM usually gives a better fit to the data than K-Means, since clusters can be modeled more accurately by using different covariance matrices.

2.3.3 Variational Segmentation Methods

In variational segmentation methods an energy functional is derived from some *a priori* mathematical model. This functional is minimized over all possible segmentations. It consists of terms describing the likelihood of the segmentation without any global assumptions on the solutions as well as a regularization on the model, which is usually described by constraints on the smoothness of the solution. Minimizing such a composite energy is a similar idea as used in graph based methods or active contours. Section 2.7 gives a general introduction of the Total Variation approach together with ways how to minimize such energy functionals. The Mumford-Shah

model is a well-known variational model for image segmentation and forms the basis of many other algorithms. It will therefore be explained in the remainder of this section.

The Mumford-Shah Model

The Mumford-Shah functional is used for segmenting an image into smooth sub-regions. On one hand, the model penalizes the distance between the input image and the piecewise smooth image, i.e. the segmentation result, based on the intensity values. On the other hand, both non-smoothness of the pixel values inside a region as well as the length of the boundaries between regions are penalized. The energy functional can be expressed as

$$E(u,\Gamma) = \alpha \int_{\Omega} (u-f)^2 dx + \beta \int_{\Omega\setminus\Gamma} |\nabla u|^2 dx + \nu \mathcal{H}^1(\Gamma) \tag{2.30}$$

where u is the approximated piecewise smooth image, Γ the set of boundaries and Ω the image domain. The parameters α and β are arbitrary weight factors which control the influence of the different terms. The 1-dimensional *Hausdorff Measure* $\mathcal{H}^1(\Gamma)$ describes the length of the edge set Γ. Minimizing Equation (2.30) is a difficult task, due to the last term, which is describing the length of the boundaries. The functional is often solved using the level set framework, but also graph-based methods are popular. However, most approaches can only find local minima and therefore depend on the initialization. A popular method for global minimization of the functional was proposed by Amrosio and Tortorelli in [48]. Here, the edge set is replaced by a continuous 2-dimensional function. Another approach proposed by Pock et al. ([47]) is based on convex relaxation and is also able to find a globally optimal solution.

A widely used variation of the model is the piecewise constant Mumford-Shah functional, which is defined as

$$E(u,\Gamma_u) = \alpha \int_{\Omega} (u-f)^2 dx + \nu \mathcal{H}^1(\Gamma_u) \tag{2.31}$$

where u is piecewise constant and Γ_u is the jump set of u. This problem is related to the Potts/Ising Model ([49]) and can also be solved by convex relaxation techniques (e.g. [50]). Another variation of the Mumford-Shah functional is the *Active Contour Without Edges (ACWE) Model*, presented by Chan and Vese ([51]). It makes a connection to the framework of active contours and is solved using level sets.

2.4 Depth Map Generation

A depth map (or range image) of a camera view contains a depth value for every pixel of the frame, giving the distance to the nearest surface point projecting to that pixel. This distance is defined along the Z-axis of the camera coordinate system, meaning that it is the distance to the image plane. Figure 2.10 shows an example of a depth map.

Figure 2.10: Example of a Depth Map. Left: Original Image. Right: Depth Map, where high Intensity indicates large Distance from the Viewing Plane. Images taken from `http://devernay.free.fr/vision/focus/office/`

A wide variety of algorithms for computing depth maps from stereo images exists (see [15]). Most of them compute the disparity between two corresponding image points, which is inversely proportional to the depth of the corresponding 3D point.

Active methods can be used to ease the process of finding correspondences, as for example used in Microsoft's Kinect hardware. Here, a special pattern is projected onto the scene, which makes it possible to find correspondences even in homogeneous regions. However, the depth range of such methods is usually very limited, and they are not suited for outdoor environments.

Passive methods, in contrast, try to solve the correspondence problems by finding features in the images, as for example corners or blobs.

For the algorithm described in this work, however, more than two images of the same 3D object are given. Thus, multiple images can be taken into account for a more robust computation of range images. A simple approach would be to combine the results of the stereo method using any combination of two camera views. This is for example described in [21]. The main drawback, however, is the high complexity of $\mathcal{O}(n^2)$. Probably the first algorithm that tackled the problem of computing range images using multiple views with

complexity $\mathcal{O}(n)$ is the well known Plane Sweep Algorithm [20], which is used in this work. It is subject of the remainder of this section.

2.4.1 Plane Sweep Algorithm

As its name suggests, the algorithm works by sweeping a plane through 3D space. This plane is in parallel to a key- or reference view, i.e. the image plane of the camera for which the depth map is to be computed. It is placed at an arbitrary number of discrete depths with respect to the image plane. Figure 2.11 illustrates this process.

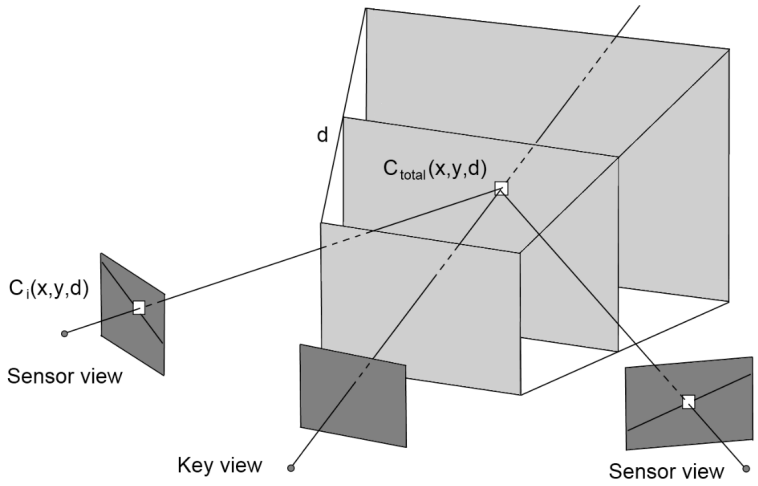

Figure 2.11: Setup of Plane Sweep Algorithm. A Plane parallel to the Reference View is moved along the View Vector at discrete Steps. Each Plane represents a Depth Hypotheses, which is evaluated by computing Costs for each Sensor View based on a Similarity Measure. Image taken from [40]

For each pixel in the key view, the corresponding viewing ray is intersected with the plane located at depth d. This point in 3D space is then projected onto an arbitrary number of other available camera views (sensor views). The basic idea is that the viewing rays through corresponding image features (nearly) intersect in 3D space. Therefore, under the assumption of Lambertian surfaces, the color value of the pixel in the reference view matches the resulting ones in the sensor views. Or, in other words, if the plane passes through the surface of the object, the resulting pixel values in the different views match.

The corresponding pixel coordinates in the different sensor images can be efficiently computed using homographies. The mapping of pixels from the reference view to a sensor view is planar and can therefore be described by a homography induced by the plane at depth d. This homography can be computed as follows:

The camera matrix $\mathbf{P}$ can be described as the product of an intrinsic matrix $\mathbf{K}$ and an extrinsic matrix $\mathbf{C} = [\mathbf{R} \mid \mathbf{t}]$ (see Section 2.2). Assume that the world coordinate system is aligned with the coordinate system of the reference camera, i.e. $\mathbf{C}_{ref} = [\mathbf{I} \mid \mathbf{0}]$. A viewing ray through a point $\mathbf{x} = [x, y, z]^\top$ can then be described by all 3D points $\mathbf{X} = (\mathbf{x}^\top, \lambda)^\top$. An arbitrary plane $\boldsymbol{\pi}$ is characterized by its normal vector, i.e. $\boldsymbol{\pi} = (\mathbf{n}^\top, 1)^\top$, and any point lying on $\boldsymbol{\pi}$ must satisfy $\boldsymbol{\pi}^\top \mathbf{X} = 0$. The intersection of the viewing ray through $\mathbf{x}$ with the plane $\boldsymbol{\pi}$ can therefore be computed as

$$\boldsymbol{\pi}^\top \mathbf{X} = \left[\mathbf{n}^\top, 1\right] \left[\mathbf{x}^\top, \lambda\right]^\top = 0 \Rightarrow \mathbf{X} = \left[\mathbf{x}^\top, -\mathbf{n}^\top \mathbf{x}\right]^\top \tag{2.32}$$

The intersection point $\mathbf{X}$ is then transformed into the second camera coordinate system, described by the extrinsic matrix $\mathbf{C}_2 = [\mathbf{R} \mid \mathbf{t}]$, yielding

$$\mathbf{x}' = [\mathbf{R} \mid \mathbf{t}]\, \mathbf{X} = \mathbf{R}\mathbf{x} - \mathbf{t}\mathbf{n}^\top \mathbf{x} = (\mathbf{R} - \mathbf{t}\mathbf{n}^\top)\mathbf{x} \tag{2.33}$$

The homography induced by a general plane $\boldsymbol{\pi}$ and two camera views is thus given as

$$\mathbf{H} = \left(\mathbf{R} - \mathbf{t}\mathbf{n}^\top\right) \tag{2.34}$$

where $\mathbf{R}$ and $\mathbf{t}$ are the rotation and translation, resp., of the second camera view with respect to the reference view.

Since one is interested in computing correspondences in pixel coordinates instead of camera coordinates, each coordinate vector has to be multiplied with the inverse of the corresponding intrinsic camera matrix $\mathbf{K}$. Therefore, the homography when using pixel coordinates is

$$\mathbf{H} = \mathbf{K}_{sens} \left(\mathbf{R} - \mathbf{t}\mathbf{n}^\top\right) \mathbf{K}_{ref}^{-1} \tag{2.35}$$

In the generic case, the camera coordinate system of the reference view does not coincide with the world coordinate system. The extrinsic matrices are given with respect to the world coordinate system, but the relative rotation and translation between two cameras

can easily be computed using

$$\mathbf{R}_{rel} = \mathbf{R}_{sens}\mathbf{R}_{ref}^{-1} \tag{2.36}$$

$$\mathbf{t}_{rel} = \mathbf{t}_{sens} - \mathbf{R}_{rel}\mathbf{t}_{ref} \tag{2.37}$$

As mentioned above, the Plane Sweep Algorithm uses homographies induced by planes in parallel to the reference view at discrete depths d. Therefore, the parametrization of a plane $\boldsymbol{\pi}$ is

$$\boldsymbol{\pi}(d) = \left[\mathbf{n}^\top, -d\right]^\top = \left[\frac{\mathbf{n}^\top}{-d}, 1\right] \tag{2.38}$$

Therefore, the final homography evaluates to

$$\mathbf{H} = \mathbf{K}_{sens}\left(\mathbf{R}_{rel} + \frac{\mathbf{t}_{rel}\mathbf{n}^\top}{d}\right)\mathbf{K}_{ref}^{-1} \tag{2.39}$$

In many cases, as in this work, the same camera is used to capture the different views, therefore $\mathbf{K}_{sens} = \mathbf{K}_{ref}$.

As depicted in Figure 2.11, a cost value C_i is computed for each of the reference views at a certain depth d. These costs are aggregated yielding the total cost C_{total} for a pixel in the depth map and a plane at depth d. The plane that yields the lowest value of C_{total} is indicating the most probable depth at the corresponding pixel position. The question arises how to select a proper dissimilarity measure for computing the costs C_i. A simple approach is the compute the squared error between the pixel value in the reference view and the corresponding ones in the sensor views. The single costs are summed up to give the total cost C_{total}:

$$C_{total} = \sum_{i=1}^{N}\left((R_i - R_r)^2 + (G_i - G_r)^2 + (B_i - B_r)^2\right) \tag{2.40}$$

In order to account for occlusions, a threshold on the maximum radius of influence in RGB-space can be introduced, limiting the influence of outliers:

$$C_{total} = \sum_{i=1}^{N} \min \left(C_{i,max}, (R_i - R_r)^2 + (G_i - G_r)^2 + (B_i - B_r)^2 \right) \tag{2.41}$$

In addition to this implicit occlusion handling, several optimization steps can be performed to increase the quality of the resulting range image, e.g. post-filtering or refinement by making the algorithm hierarchical in depth resolution. See [19] for an overview of some possible optimization steps.

In practice, a window approach is usually used in order to reduce noise and to give better estimates in low-textured regions. Typically, a square window around the center pixel is defined, under which the average dissimilarity is computed. Often, either the sum of absolute differences (SAD) or the sum of squared distances (SSD) is used, which usually give similar results. The SAD or SSD, resp., between two views can be computed using the following equations:

$$SAD = \sum_{i,j \in W} |I_1(x+i, y+j) - I_2(x'+i, y'+j)| \tag{2.42}$$

$$SSD = \sum_{i,j \in W} \left(I_1(x+i, y+j) - I_2(x'+i, y'+j) \right)^2, \tag{2.43}$$

where W is the window around the pixel of interest and $\mathbf{x}' = (x', y')$ is the corresponding position in the second view of the pixel at $\mathbf{x} = (x, y)$ in the first view. For multiple views, the SAD or SSD values are averaged, as in eq. (2.40). Truncation can also be used to limit the impact of outliers. However, several other methods exist to tackle this problem. Popular approaches are shiftable windows, where the window is moved to several locations around the center pixel to find the best match, or adaptive weights ([70]), where pixels with similar intensity/color as the center pixel are assigned higher weights, as well as pixels which are close to the center. These methods usually also significantly improve the results at depth discontinuities. Without any of these steps, the effect of "foreground fattening" often occurs, which makes objects appear thicker than they really are.

Using the sum of squared errors as similarity measure is likely to perform poorly if the lighting conditions change from one view to another, which is often the case in real-world scenarios. An error measure which is insensitive to these changes is the normalized cross-correlation, given by

$$C_{total} = \sum_{x,y \in W \;\; x',y' \in W'} \frac{\left(I_1(x,y) - \overline{I}_1\right)\left(I_2(x',y') - \overline{I}_2\right)}{\sigma_1 \sigma_2}, \tag{2.44}$$

where $\overline{I}_1, \overline{I}_2, \sigma_1, \sigma_2$ are the mean and standard deviation of pixel values in the window W. In general, a large number of reference images yields better results and makes the algorithm more robust. However, it is crucial that the different views taken into account show the 3D scene from angles not too far apart from each other, otherwise occlusions are likely to decrease the quality of the result. Again, the dissimilarity measure is computed for each of the sensor views, and the average value is typically used as the final cost for the corresponding depth hypothesis. However, one can also use only the best NCC value, or the median of all results. Typically, the results are quite similar.

Another dissimilarity measure which is robust again radiometric changes is the *Census Transform* ([69]). Here, each pixel value within a window is compared with the value at the center. Based on the intensity difference, bit values are assigned to each position within the Window. In the simplest case, only one bit indicating that the pixel is brighter or darker than the center is used. The resulting bit strings for each view are compared using the Hamming-Distance, which directly serves as the cost measure.

Other methods exist, for example *Mutual Information* ([71]). An evaluation of the performance of different dissimilarity measures can be found in [15].

2.5 Multiview 3D Reconstruction

Reconstructing a 3D scene from multiple images is one of the core problems in computer vision. The problem is a lot harder if the camera parameters are unknown and need to be estimated as well. This is for example the case in Simultaneous Location and Mapping (SLAM) systems, which were originally designed for mobile robots learning a map of the environment. Here, the map has to be deduced from images alone. In many algorithms the optical flow is an important source of information about the motion of the camera and the distance of the scene points. It describes the perceived relative motion of objects as the observer moves.

Since in this work it is assumed that projection matrices are given (i.e. camera calibration data is present), the focus will be on 3D reconstruction based on multiple views from calibrated cameras. The problem of multiview 3D reconstruction (also referred to as

Structure from Motion) is illustrated in Figure 2.12.

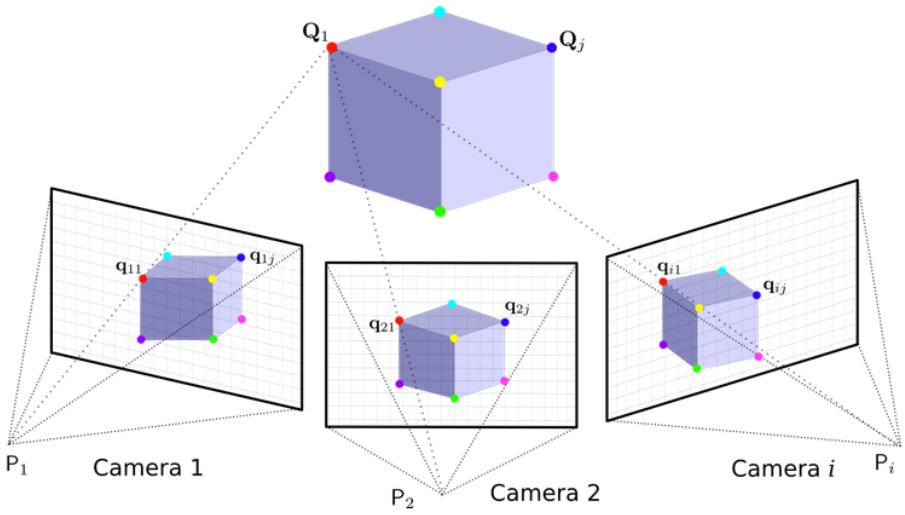

Figure 2.12: Illustration of the Multiview 3D Reconstruction Problem. In the ideal Case, Viewing Rays view corresponding Image Features intersect in 3D Space. Image taken from `http://michot.julien.free.fr/drupal/?q=content/research`

2.5.1 Reconstruction by Feature Matching

Most reconstruction algorithms work by finding point correspondences between the different views, as indicated by the corner points of the cube in Figure 2.12. Selecting features in an image that can be matched in others is not a trivial task. The most popular algorithm which tries to find such features is the Scale-invariant Feature Transform (SIFT, [55]).

The epipolar geometry of two views describes important relationships to compute where feature correspondances can be located in the second view. Matching between multiple views works by finding that feature pair-wise in other images, which is a trivial extension to the 2-view case. Figure 2.13 shows the epipolar geometry of two views. A 3D point $\mathbf{X}$ is seen by two cameras. The projections of the camera centers are called the **epipoles** e_L and e_R. The camera centers $\mathbf{O}_L$ and $\mathbf{O}_R$ together with the point $\mathbf{X}$ form the **epipolar plane**, shown in green in Figure 2.13. The intersections of this plane with the image planes are the so-called **epipolar lines** – the one for the right view is indicated in red. If a feature in one of the images is selected, the 3D point it originated from can lie at any position on the viewing ray through that feature pixel. All these possible 3D points project onto the epipolar line of the second view. Therefore, the correspondence only

needs to be searched for on that line, making the task a lot easier compared to scanning the whole image.

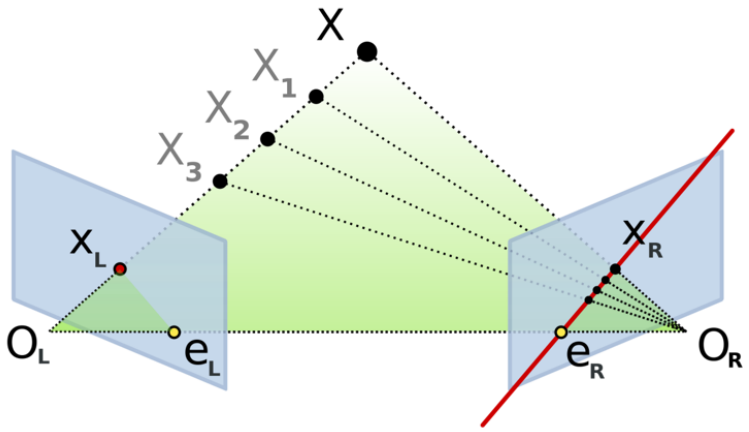

Figure 2.13: Epipolar Geometry. A Point in the first Image corresponds to a Point lying on the corresponding Epipolar Line in the second Image. The Epipolar Line can be computed using the Fundamental Matrix **F**. Image taken from `http://en.wikipedia.org/wiki/Epipolar_geometry`

The question arises how to compute the corresponding epipolar line in the second view of an image feature in the first one. This can be achieved using the 3x3 rank 2 homogeneous **Fundamental Matrix F**:

$$\mathbf{l}' = \mathbf{F}\mathbf{x} \tag{2.45}$$

where $\mathbf{l}'$ is the epipolar line and $\mathbf{x}$ the pixel coordinates of the image feature in the first view. The fundamental matrix is defined solely by the geometry of the scene and therefore only needs to be computed once. For all corresponding points, $\mathbf{F}$ satisfies

$$\mathbf{x}'^{\top}\mathbf{F}\mathbf{x} = 0 \tag{2.46}$$

The epipoles $\mathbf{e}$ can also be computed using via the fundamental matrix:

$$\mathbf{F}\mathbf{e} = \mathbf{0} \tag{2.47}$$

Note that for a fundamental matrix $\mathbf{F}$ of a pair of cameras defined by their projection matrices $\mathbf{P}$ and $\mathbf{P}'$, $\mathbf{F}^{\top}$ is the fundamental matrix of this pair in opposite order.

Computing the fundamental matrix is straightforward once the two projection matrices are known:

$$\mathbf{F} = [\mathbf{e}'_{\times}]\mathbf{P}'\mathbf{P}^{+} \tag{2.48}$$

where $\mathbf{P}^{+}$ is the pseudo-inverse of $\mathbf{P}$ and $\mathbf{e}' = \mathbf{P}'\mathbf{C}$ is the epipole in the second view, with $\mathbf{C}$ being the camera center of the first view defined by $\mathbf{PC} = \mathbf{0}$. The notation $[\mathbf{a}]_{\times}$ defines a skew-symmetric matrix corresponding to $\mathbf{a} = (a_1, a_2, a_3)^{\top}$ as

$$[\mathbf{a}]_{\times} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{2.49}$$

which is related to the cross-product of two vectors as

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}_{\times}]\mathbf{b} = \left(\mathbf{a}^{\top}[\mathbf{b}_{\times}]\right) \tag{2.50}$$

In order to simplify stereo matching, the two images are often rectified. Figure 2.14 illustrates this process. The advantage is that correspondences don't need to be searched on epipolar lines with arbitrary orientation, but on horizontal ones. This makes the process a lot easier, since instead of computing the pixel coordinates on the line, the vertical coordinate can be fixed while the other is checked from the lowest to the highest possible value. Rectification of a pair of images is achieved by rotating and translating the cameras such that they are co-planar. Several algorithms that compute the rectification have been proposed, e.g. [52], [53] or [54].

In theory, once corresponding features have been detected, finding the 3D point they originated from is trivial –the 3D point is simply the intersection of the viewing rays through the feature points. In practice, however, these rays usually don't intersect due to noise and inaccuracy of measurements. The problem of finding the most likely 3D point is referred to as *triangulation*. Several methods for solving the triangulation problem can be found in [22].

2.5.2 Volume Based Reconstruction

Reconstruction of a 3D scene based on image features in multiple views has some disadvantages. First of all, finding image features is computationally expensive. Even after the images have been rectified, identifying features and finding their correspondences in the other views is a difficult task. Furthermore, and often more importantly, these methods only lead to a sparse reconstruction – the number of reconstructed 3D points equals the

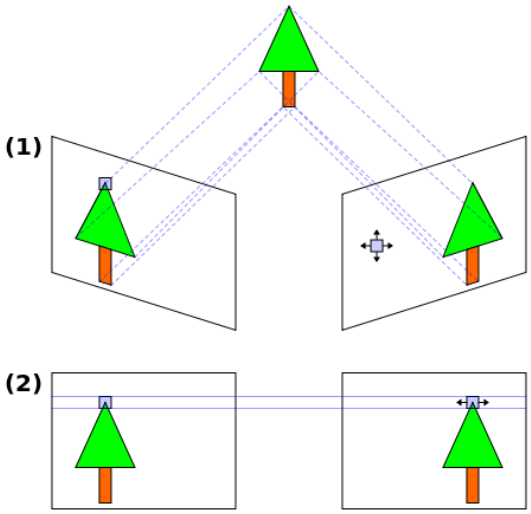

Figure 2.14: Image Rectification – Search Space before (1) and after (2) Rectification. After Rectification, Epipolar Lines are horizontal. Image taken from `http://en.wikipedia.org/wiki/Image_rectification`

number of features. Interpolating a dense structure from sparse point clouds does not always yield satisfying results.

Another approach, which computes a dense reconstruction, tackles the problem directly in scene space. A discrete voxel grid is initialized and during the computation process voxels are either marked as being transparent or opaque. Instead of finding features in the images and computing the corresponding 3D points they originated from, these methods work by projecting each voxel onto the images and checking if the resulting color values are consistent. If they are, the voxel probably lies on the surface of a solid object. A similar idea is used in the *Plane Sweep Algorithm* to compute depth maps (see Section 2.4.1). Instead of comparing single color values, more sophisticated methods can be used, e.g. texture descriptors or correlation windows. Two general approaches can be followed – One can either initialize the voxel grid as transparent and gather all points with high photo consistency, or initialize it as solid and carve away non-consistent parts. Often the latter method is used, which is referred to as *space carving* ([6]).

Voxel based methods work well when objects are textured and photo consistency measures therefore yield clear maxima. In homogeneous regions, however, the 3D shape cannot be uniquely determined – several 3D models are consistent with the images. In these cases, prior information such as smoothness constraints can be included, leading to regularized models. Implementations can use techniques such as graph cuts or variational methods to find the most probable 3D surface.

Advantages of volume based reconstruction methods are that they are easy to implement, that explicit feature correspondences don't need to be found and that they can handle arbitrary complex topologies. However, it is necessary that the number of camera views is sufficiently high and the whole scene is covered from each side. Figure 2.15 shows that space carving methods actually do not compute the exact 3D model, but the so-called *photo hull* of the object, which is the union of all photo consistent scenes in the voxel grid.

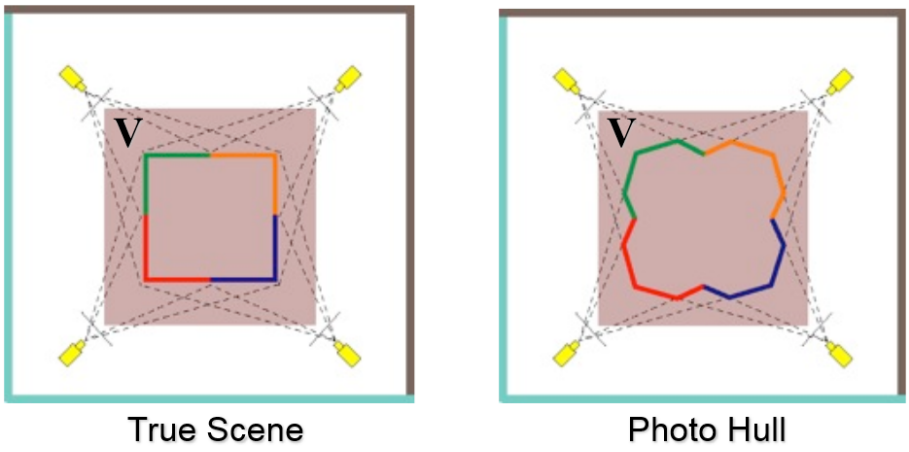

Figure 2.15: Space Carving – True Scene and Reconstruction Result. Only the Photo Hull of the Object can be reconstructed. Image taken from `http://en.wikipedia.org/wiki/Image_rectification`

2.5.2.1 Probabilistic Volume Intersection

Kolev et al. proposed a volumetric approach for segmenting and reconstructing the 3D model of an object based on multiple views ([12]). Their ideas form the basis of the reconstruction algorithm in this work. Based on strokes placed in one of the input images, color models are computed for both the object and the background of the scene, using the EM algorithm described in Section 2.3.2. While this step is performed differently in this work, and also depth information is included into the model, the basic idea of how to combine single view probabilities for object and background is basically the same.

Let these probabilities for object and background, resp., be denoted as

$$P(I_i(\pi_i(x)) \mid x \in R_{obj}) \tag{2.51}$$

$$P(I_i(\pi_i(x)) \mid x \in R_{bck}) \tag{2.52}$$

where $\pi_i(x)$ projects a voxel x into the image I_i. The set of voxels being part of the

background is R_{bck}, while R_{obj} is the interior of the object. These single view probabilities need to be fused to form a probabilistic volume model that incorporates probabilities from all views. Let these joint probabilities be defined as

$$P_{obj}(x) := P(\{I_l(\pi_l(x))\}_{l=1,\dots,n} \mid x \in R_{obj}) \tag{2.53}$$

$$P_{bck}(x) := P(\{I_l(\pi_l(x))\}_{l=1,\dots,n} \mid x \in R_{bck}) \tag{2.54}$$

To simplify the model, independence of the single image observations is assumed. Then, the probability that a voxel belongs to the object is simply the probability that all cameras observe this voxel as part of the object, i.e. the product of the single view probabilities. For the joint background probability, however, this is not the case. A voxel which belongs to the background can be occluded by the object in up to n-1 views. Therefore, $P_{bck}(x)$ describes the probability that the voxel belongs to the background in at least one of the views. Following this train of thoughts, the following mathematical formulation of the joint probabilities can be obtained:

$$P_{obj}(x) = \prod_{i=1}^{n} P(I_i(\pi_i(x)) \mid x \in R_{obj}) \tag{2.55}$$

$$P_{bck}(x) = 1 - \prod_{i=1}^{n} [1 - P(I_i(\pi_i(x)) \mid x \in R_{obj})] \tag{2.56}$$

Note that this model implicitly handles occlusions. Also, it assumes that the object itself is not occluded by other obstacles, i.e. it is completely visible in all images. However, this formulation contains a bias with respect to the number of images, n. For $n \to \infty$, $P_{obj}(x)$ would typically tend to zero, while $P_{bck}(x)$ tends to one. Hence, instead, the geometric mean of the single view contributions is used, yielding

$$P_{obj}(x) = \sqrt[n]{\prod_{i=1}^{n} P(I_i(\pi_i(x)) \mid x \in R_{obj})} \tag{2.57}$$

$$P_{bck}(x) = 1 - \sqrt[n]{\prod_{i=1}^{n} [1 - P(I_i(\pi_i(x)) \mid x \in R_{obj})]} \tag{2.58}$$

Simply computing these two probabilities for each voxel and making an assignment according to the maximum does usually not lead to a tight, smooth object surface due to noise and outliers. Therefore, a regularization step has to be performed. This is done via a variational formulation leading to an energy minimization problem. The general

principle of this approach is explained in Section 2.7. Without any regularization, the most probable surface is the one that maximizes the sum of the product of all P_{bck} for background voxels and the product of all P_{obj} for foreground voxels. Since these products usually lead to extremely small numbers, causing underflows, the sum of the negative logarithms of the single probabilities is used instead. The position of the minimum then equals the position of the maximum in the original formulation, since the logarithm is a strictly monotonically increasing function. Regularization is introduced by adding a term that minimizes the total surface area, thus favoring smooth objects. The final energy minimization problem then reads

$$E(S) = -\int_{R_{obj}} \log P_{obj}(x)dx - \int_{R_{bck}} \log P_{bck}(x)dx + \nu|S| \tag{2.59}$$

where S is the surface of the object and $|S|$ is its area. The weight factor ν controls how strictly smooth surfaces are enforced.

As a first step to minimize this energy functional, the surface S is represented by the characteristic function $u : V \to 0, 1$ of R_{obj}, i.e. $u = \mathbf{1}_{R_{obj}}$ and $1 - u = \mathbf{1}_{R_{bck}}$. Then, the energy minimization problem can be expressed as

$$E(u) = \int_V \log \frac{P_{bck}(x)}{P_{obj}(x)} u(x)dx + \nu \int_V |\nabla u|dx \tag{2.60}$$

with $u \in \{0, 1\}$. Since u is a non-convex set of binary functions, the optimization problem is also not convex. However, relaxing the set of binary labeling functions to $u \in [0, 1]$ circumvents this difficulty, since thresholding the global minimizer within the interval $(0, 1)$ gives a global minimizer of the original problem ([64]).

Finally, to simplify the formulation, the constant part not depending on u can be summarized, yielding

$$E(u) = \int_V fu \; dx + \nu \int_V |\nabla u|dx \tag{2.61}$$

with

$$f := \log \frac{P_{bck}(x)}{P_{obj}(x)} \tag{2.62}$$

Using the dual norm, which is defined for a finite-dimensional Hilbert space $\mathcal{H}$ over $\mathbb{R}$ or $\mathbb{C}$ as

$$|x| = \sup_{w \in \mathcal{H}} \{|\langle x, w\rangle| : |w| \leq 1\}, \tag{2.63}$$

Equation (2.61) can be re-written using an auxiliary variable ξ as

$$E(u) = \int_V fu \; dx + \nu \left(\sup_{|\xi| \leq 1|} \int_V \langle \xi, \nabla u \rangle dx \right) \tag{2.64}$$

This leads to a new energy functional depending on both u and ξ:

$$E(u, \xi) = \int_V fu \; dx + \nu \int_V \langle \xi, \nabla u \rangle dx \tag{2.65}$$

This functional should be minimized with respect to u and maximized with respect to ξ under the constraints $u \in [0, 1]$ and $|\xi| \leq 1$, which is a typical saddle-point problem. It can be solved by applying the primal-dual method explained in Section 2.7.2.1 and proposed by Chambolle and Pock in [61].

2.5.3 Reconstruction by Depth Map Fusion

In this work, depth maps are computed to infer information about the 3D shape of the object to segment. Fusing depth maps in 3D space is not a trivial problem and still an active field of research. In theory, if a sufficient number of views is present, a dense 3D model can be obtained from depth maps by projecting each pixel to its corresponding depth in the scene space. For a homogeneous pixel coordinate $\mathbf{x} = [x, y, 1]^\top$ the projected scene point can be computed as

$$\mathbf{X} = \mathbf{C}^{-1} \, z \, \mathbf{K}^{-1} \, \mathbf{x} \tag{2.66}$$

where $\mathbf{C} = [\mathbf{R} \,|\, \mathbf{t}]$ is the 3x4 camera pose matrix containing the extrinsic parameters, i.e. rotation and translation. Its inverse can easily be obtained by $\mathbf{C}^{-1} = [\mathbf{R}^{-1} | - \mathbf{R}^{-1}\mathbf{t}]$. Since $\mathbf{R}$ is a rotation matrix, the inverse is equal to its transpose, $\mathbf{R}^{-1} = \mathbf{R}^\top$. The scalar z in Equation (2.66) denotes the corresponding depth of the pixel at coordinate $\mathbf{x}$. The 3x3 matrix $\mathbf{K}$ contains the intrinsic parameters of the camera. See Section 2.2 for more information about intrinsic and extrinsic camera parameters as well as how to derive them from a given projection matrix $\mathbf{P}$.

In a realistic scenario, depth maps usually contain noise and outliers. Also, multiple depth maps are often not consistent. Therefore, a fusion step needs to be performed, in which information from different vantage points is combined to yield a probable 3D model. This is a challenging task, since outliers, occlusions and missing depth data, especially in textureless regions, are problems that need to be handled at once. Many algorithms transform the depth maps to so-called *distance functions* in voxel space. These distance functions are then combined by averaging to yield the dense 3D model. Curless and Levoy

proposed one of the earliest algorithms that use this method ([56]).

Simple averaging without any regularization usually causes inconsistent surfaces, since depth maps typically contain inaccurate values. Therefore, a regularization step is included in order to favor smooth surfaces. A popular approach is to penalize the surface of the resulting 3D model, which can be implemented using graph-cut algorithms ([57]) or variational techniques ([58]).

Zach et al. proposed a variational method for robust total variation range image integration using truncated signed distance functions ([59]). Such a function is defined as $\Omega \to [-1, 1]$, where $\Omega \subseteq \mathbb{R}^3$. It assigns a scalar value to every voxel, representing its signed distance to the true surface. This value is truncated to lie in the interval $[-1, 1]$ such that -1 means that the voxel is part of the solid object while $+1$ indicates that the voxel is in free space. The value 0 indicates that it exactly lies of the surface, i.e. that the pixel in the depth map projects to that voxel in 3D space. Therefore, the surface can be interpreted as the zero level set of the truncated signed distance function. Figure 2.16 shows how a depth map is converted to a signed distance field. All voxels along the line of sight between the camera and the surface are assigned positive values, indicating that these voxels are probably not part of any solid object. The positive value decreases near the object surface, making up for uncertainty and inaccuracy of the depth value. The parameter δ controls the width of this uncertain near-surface region. Every voxel on the line of sight that lies behind the estimated surface point $\mathbf{X}$ is assigned a negative value, indicating that it is probably part of the object. However, one cannot make any assumptions about what lies behind the surface, i.e. the thickness of the object. Therefore, the parameter η is introduced. It controls the distance behind the surface within which solid voxels are assumed to be present.

The variational approach seeks to find a regularized field $u : \Omega \to [-1, 1]$ which simultaneously optimizes over all K input distance fields f_i. The following TV-L^1 energy functional has been proposed:

$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i=1}^{K} |u(x) - f_i(x)| \right\} dx \tag{2.67}$$

which is minimized with respect to u. The first term is the total variation of u, leading to a smooth solution (see Section 2.7). The second term includes data fidelity with a weight factor λ, which models the likelihood of the model in terms of accordance with the distance fields. The robust L^1 norm is used to measure the distance of u and all f_i. One limitation of this model is the high memory consumption, since K distance fields need to be stored

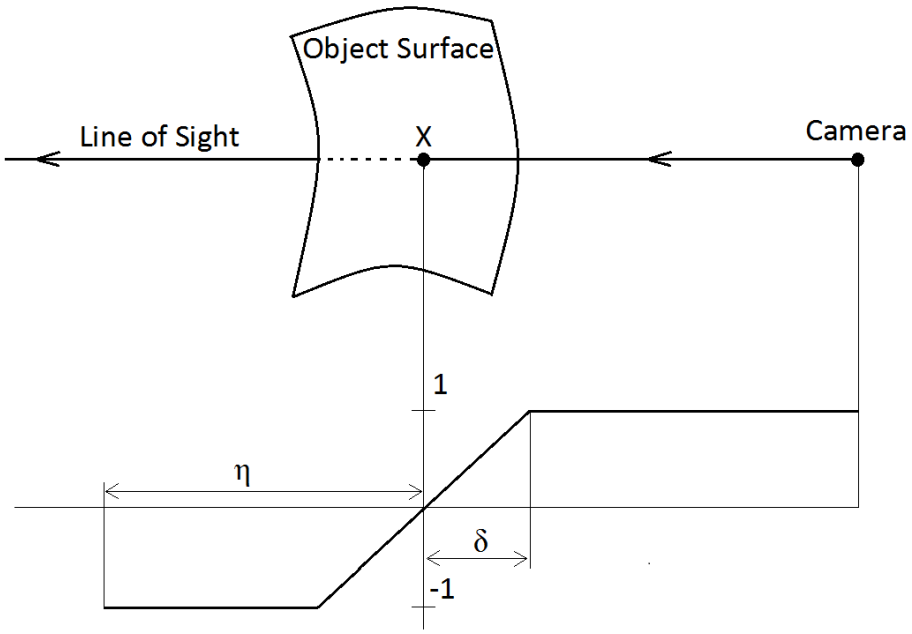

Figure 2.16: Truncated Signed Distance Function from Depth Map. Negative Values indicate that the 3D Point is likely to be Part of the Object, positive Values indicate Free Space

separately. An alternative method with memory complexity of $\mathcal{O}(1)$ was proposed in [60]. Here, the signed distance function is sampled at N discrete steps. An arbitrary number of distance fields can then be stored using a volume histogram at each voxel, where $h(x, i)$ denotes the histogram count of bin i at voxel x, i.e. how often the value d_i of the distance function occurs at voxel x. The model then reads

$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i=1}^{N} h(x,i)|u(x) - d_i| \right\} dx \tag{2.68}$$

This problem is convex, which means that a global optimum is found, regardless of the initialization of u. For minimization, the first-order primal-dual algorithm of Chambolle and Pock is used ([61]), which is briefly explained in Section 2.7.

As mentioned above, depth map fusion only tackles the reconstruction problem, but does not make any distinction between the object of interest and other objects in the scene possible. Therefore, additional information needs to be given in order to describe the object and the background regions. In this work, color models are learned for that purpose, using a so-called Random Forest. The concept of Random Forests is subject of the following section.

2.6 Random Forests

A Random Forest is an ensemble model which can be used for both classification and regression, and has first been introduced in [23]. The accumulation of an ensemble of weak learners forms a stronger classifier/regressor. In the framework of Random Forests, these weak learners are binary decision trees. A Random Forest is used as a regression model in this work, even though finding the final segmentation is a typical classification problem. However, the Random Forest only assigns probabilities to voxels, indicating how likely these are being part of the object or background. A subsequent global optimization method together with incorporating depth information yields the final binary segmentation.

2.6.1 Binary Decision Trees

A Random Forest is an ensemble of binary decision trees. A decision tree is a predictive model that computes target values based on a set of binary rules. These rules are learned based on a training set with given target values. Figure 2.17 shows a sample decision tree with 2-dimensional input data. Starting at the root of the tree, a binary condition is validated at each node. Based on the outcome, the training vector is traversed through the tree until a leaf is reached. In a regression problem, each leaf contains probabilities for each of the possible classes, while in a classification tree it only contains the label of the most probable class.

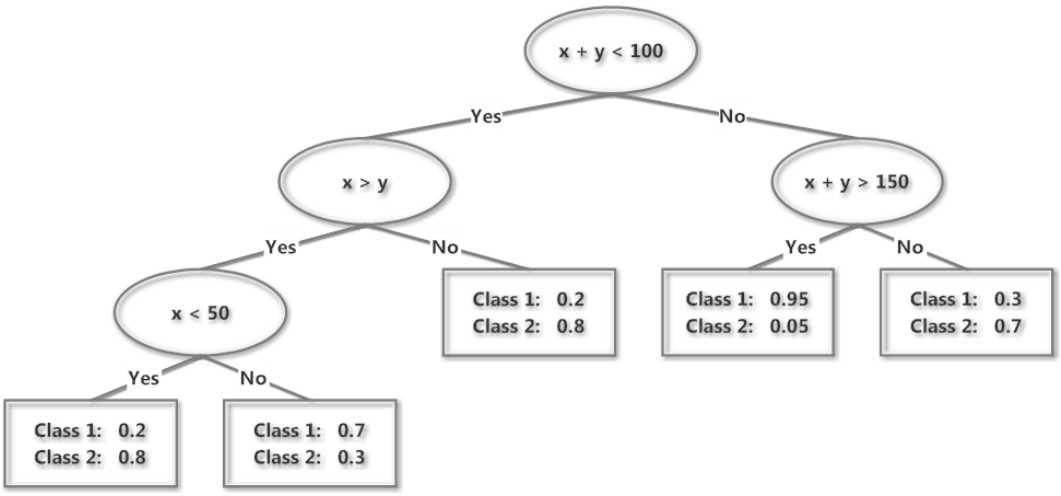

Figure 2.17: Sample Decision Tree - Binary Tree with 2-dimensional Input Data.

Training a Decision Tree

Typically, a training set is given, containing data vectors and their corresponding target classes. In an image segmentation framework, this training data could be obtained by user interaction, e.g. a user marking some image pixels as either belonging to one of two or more regions. Training a decision tree is the task of finding optimal split functions for the nodes of the tree. Usually, a so-called greedy algorithm is used. Starting at the root of the tree, a locally optimal split function is found at each level, being a typical top-down approach. Such an optimal function divides the training set into two smaller subsets, which both are as homogeneous as possible. Afterwards, for each subset of the training data, the optimal split function at the next level is determined. This process is repeated until each subset only contains data vectors of a single class, or splitting the data no longer adds value to the prediction. A measure for the homogeneity of a set of data points is the Shannon entropy. For discrete class labels, as used in image segmentation, it is defined as

$$H(S) = -\sum_{c \in C} p(c) \log\left(p(c)\right) \tag{2.69}$$

where $p(c)$ is the fraction of data points belonging to class c in the set S. An optimal split function maximizes the information gain, which is defined as

$$I = H(S) - \sum_{i \in L,R} \frac{|S_i|}{|S|} H(S_i) \tag{2.70}$$

where L, R are the subsets of the input data assigned with the left and right child of the parent node.

Figure 2.18 illustrates the process of training a decision tree. All figures in this section are taken from [24], which also gives an extensive explanation of decision trees and random forests. In this example, four different class labels are present. The two-dimensional training data points are depicted by the colored dots, where each color represents a different class. The gray dots represent test data points that are unknown at the training stage. The right figure illustrates a fully grown regression tree together with sample probability distributions of the training data at different levels of the tree. While the data classes are equally distributed at the root node, the entropy increases at lower levels, reaching its maximum at the leaves.

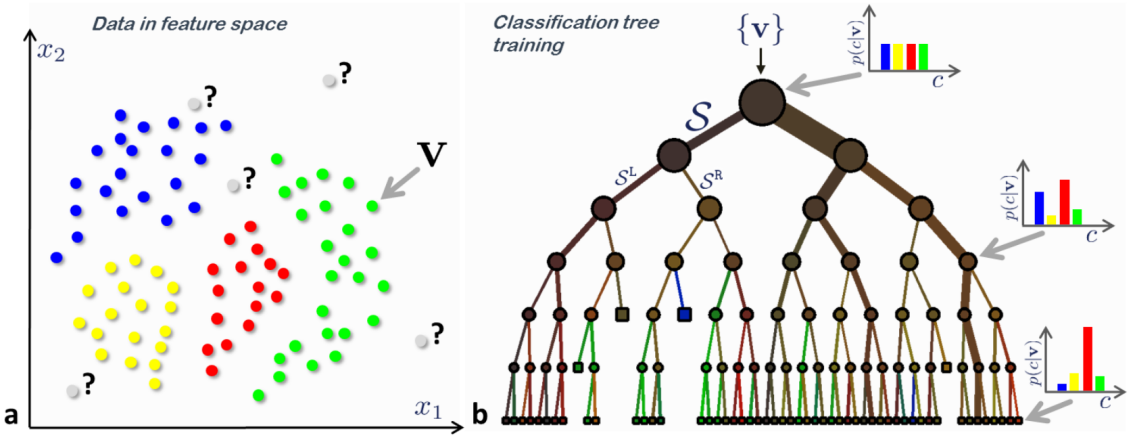

Figure 2.18: Decision Tree Training. Left: Labeled Training Data in Feature Space and unlabeled Test Data Points. Right: Resulting Decision Tree learned from Training Data - Entropy decreases at each Node. Image taken from [24]

In practice, a certain model for split functions is chosen, typically axis-aligned or linear functions in the data dimensions. Fig. 2.19 shows how training data is separated by using a linear, axis-aligned or conic weak learner. Note that by increasing the complexity of the chosen model, the computation time for training the trees also increases. Since there are infinitely many possible split functions (e.g. infinitely many line positions and orientations), a certain set of functions is chosen and tested for each node. The function which yields the highest information gain is chosen and the data is split among the two child nodes according to the split criterion.

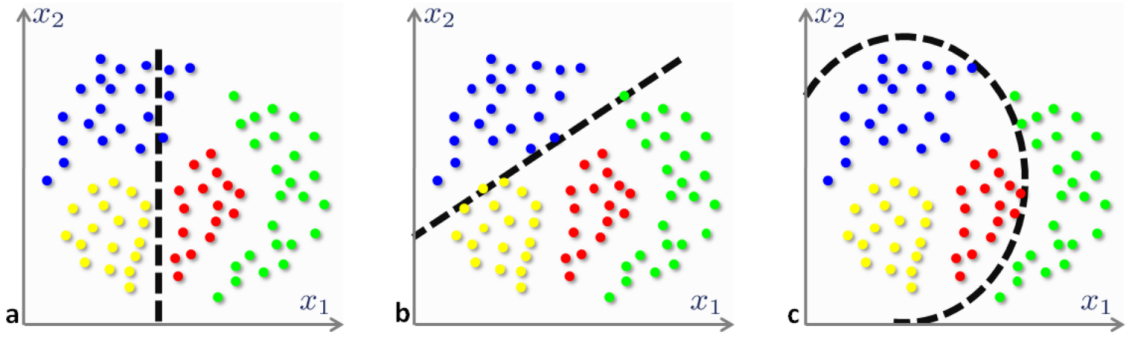

Figure 2.19: Linear, Axis-Aligned and Conic Weak Learners. Image taken from [24]

A decision tree is fully grown when only leaves remain at the bottom level. Then, assigning probabilities or class labels, resp., to new test data can be done very efficiently.

Starting at the root node, a test vector is propagated through the tree until a leaf is reached. The class label stored in that leaf (or class probabilities, if regression trees are used) is assigned to the test data vector.

Decision trees have several advantages over other methods, which makes them a powerful tool. The decision rules are easy to interpret, testing is very fast and they are quite robust against outliers. However, they tend to overfit the data, leading to poor results when applied to a newly introduced test data set. The problem of overfitting can be alleviated by pruning the tree once fully grown. Nonetheless, it can often not be eliminated to an acceptable level.

Random Forests combine the results from different decision trees, usually yielding better results than the individual trees and avoiding overfitting.

2.6.2 Random Forests as an Ensemble of Decision Trees

A Random Forest contains many individual decision trees. Test data is classified by testing every decision tree and choosing the label with the maximum number of votes. In a regression problem, the average of the probabilities obtained by the single trees is assigned. Fig. 2.20 illustrates the process for three regression trees. The resulting probability distributions in the leaves a test data point ends up in are averaged to obtain the final distribution.

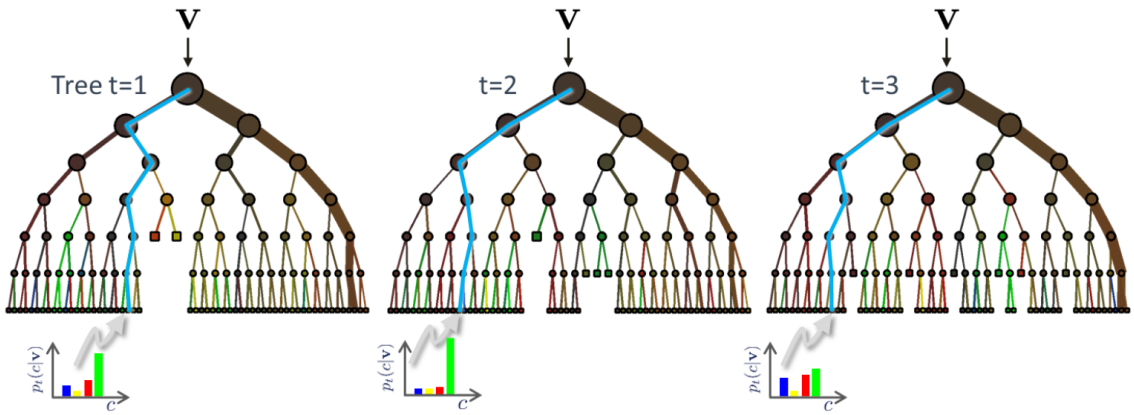

Figure 2.20: Decision Tree Testing. Unlabeld Data Points are propagated through the Trees and resulting Label Probabilities are averaged. Image taken from [24]

This procedure alone does not necessarily avoid overfitting – as the name suggests, randomness has to be introduced to the model. This can be achieved by limiting the

number of possible split functions by choosing a random set of function parameters at each node. By doing this, the predictive power of a single tree decreases. However, using a lot of different weak learners like this, the overall predictive power is usually very high, while overfitting is drastically reduced by averaging over the single results. Another source of randomness can be introduced by using only a randomly chosen subset of the training data (a so-called bootstrap sample) for finding the split functions. This subset is different for every tree, and therefore this approach is also helping to avoid overfitting and to reduce the impact of outliers.

Fig. 2.21 shows how choosing different models for the split functions affects the performance of a random forest. Four different colors indicate the four possible class labels, while the color intensities indicate the probabilities that the test data point at this location belongs to that class. The left figure shows a sample result when axis-aligned split functions are used. While each training data point is correctly classified, the model does not fit the data distribution very well. Due to only two possible orientations of the decision lines, block artifacts are visible. Also, data points that are located far from any training data point are often assigned very high probability of belonging to that class, even though they do not coincide with the training data. This problem is also present in the second example (middle), where linear split functions are used. However, due to the variety of possible line orientations, the training data is modeled more accurately. Better results are achieved when using conic weak learners (right figure). Here, the model describes the data best, and also the probabilities decrease for test points located far away from any training data. On the downside, the computation time for training the forest is usually much higher than for simpler split criteria.

One advantage of using Random Forests is that overfitting caused by training too many trees can usually not occur. While training a lot of trees obviously increases the computation time and storage requirements, it does not degrade the performance on test data. However, since each tree has only limited prediction capabilities due to the incorporation of randomness, a minimum number of trees in the forest is required to assure satisfactory performance. Fig. 2.22 shows how the number of trees affects the result. Figure (a) shows the training data, which is easy to separate even by axis-aligned split functions. In Figure (b), two sample trees and the resulting class probabilities are shown. Only one split criterion at the root node is enough to separate the training data, leading to trees with

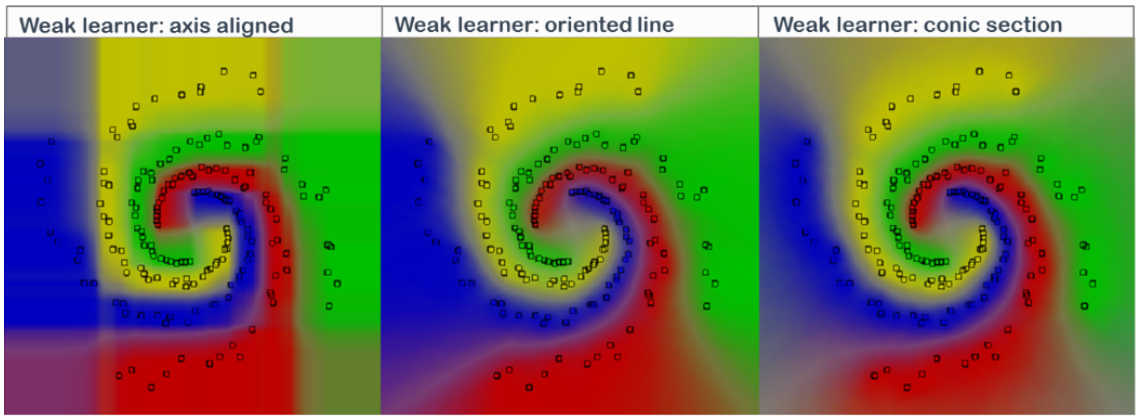

Figure 2.21: Effect of Different Split Functions. Left: Axis aligned Functions are easy to parametrize, but yield blocky Artifacts. Middle: Oriented Lines reduce Artifacts, but don't model the Data Distribution intuitively. Right: Concic Learners yield best Model, but are harder to parametrize. Image taken from [24]

only one level and two leaves. Figure (c) shows the results obtained by combining several of these trees. The more trees are used, the smoother the probability transitions are and the better the data is modeled.

While training a random forest can be complex and time consuming (depending on the training data and the complexity of the split functions), testing is very efficient. Also, storing a random forest for future use on test data is efficient, since only the different nodes together with the split functions need to be saved, while training data can be discarded. Furthermore, the optimal parameter range for introducing randomness is usually quite wide, which makes parameter estimation a simpler task than in many other models.

So far, the reconstruction problem and the segmentation problem have been treated separately. However, the main contribution of this thesis is the fusion of both methods. Both the random forest probabilities as well as the depth maps indicate surface regions, but the different hypotheses are not necessarily consistent. Therefore, the most probable surface based on the input observations is tried to be found. The problem is formulated as the minimization of an energy functional, which incorporates both probabilities depending on the input data, as well as a smoothness prior in order to favor compact and smooth surfaces. In order to find the solution minimizing this energy, an optimization problem is formulated. The following section gives an introduction to optimization problems, with emphasis on problems similar to the one to be solved in this work.

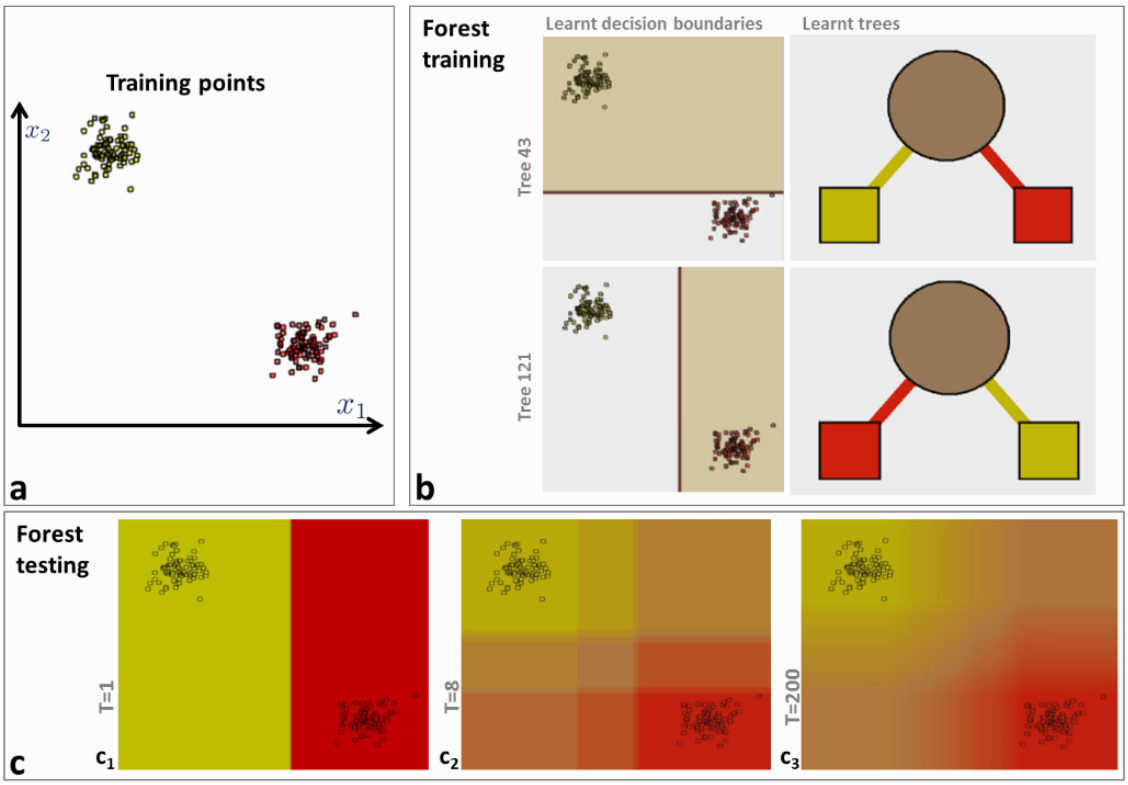

Figure 2.22: Effect of Number of Trees. Transitions become smoother if Number of Trees is increased. Image taken from [24]

2.7 Convex Optimization and Variational Models

Ill-posed inverse problems often occur in image processing and computer vision. Given an observation, which is usually (noisy) image data, some problem needs to be solved for which no uniquely defined solution exists. This could for example be image denoising, 3D reconstruction or image segmentation. In 1963, Tikhonov ([62]) proposed the usage of an optimization problem as a combination of prior knowledge about the desired solution and data fidelity based on the observation:

$$\min_u \{\mathcal{R}(u) + \lambda \mathcal{D}(u, f)\} \tag{2.71}$$

where u is the model and f the observation. The first term incorporates prior knowledge while the second term is the data fidelity, weighted with a factor λ that controls the trade-off between the regularization and the data fidelity. Often, the regularization term imposes a smoothness constraint on the solution. In image denoising, for example

, the constraint is used to adjust pixel values that don't match with their neighborhood. In image segmentation and 3D reconstruction, the regularization term favors tight objects with small boundary lengths or surface areas, respectively. Many inverse problems make use of a model described by Equation (2.71). For finding the best solution, i.e. the minimum over all u, convexity of the model is an important criterion. If the problem is convex, computing the first derivative, setting it to zero and solving for u yields the global minimum, while in non-convex problems this is not necessarily the case. Usually, iterative methods such as gradient descent are used to find the solution, since it can usually not be computed directly in closed form.

Optimization models such as Equation (2.71) can also be formulated in terms of conditional probabilities, as

$$\max_u p(u|f) \tag{2.72}$$

which aims to find the solution u which is most probable based on the observation f, also called the *maximum a posteriori* estimation, where $p(u|f)$ is the *posterior probability*. Using Bayes' Theorem, this probability can be expressed as

$$p(u|f) = \frac{p(f|u)p(u)}{p(f)} \tag{2.73}$$

where, again, $p(u)$ models prior information about u while $p(u|f)$ measures data fidelity. The denominator is constant for every solution u and is therefore irrelevant for finding the optimum.

2.7.1 Variational Models in Image Denoising

As an introduction to variational optimization methods, its application in image denoising is described in this section. Assume the true image u^* is degraded by additive noise n, leading to a noisy image $f = u^* + n$. The task of image denoising is to find an image u that approximates the original image u^* in an optimal way.

In the **Tikhonov Model** ([62]), the following representation is used:

$$\min_u \left\{ \frac{1}{2} \int_\Omega |\nabla u|^2 + \frac{\lambda}{2} \int_\Omega (u - f)^2 dx \right\} \tag{2.74}$$

which corresponds to the solution in the MAP sense for additive, white Gaussian noise. The regularization term penalizes high image gradients, which leads to smooth image regions. However, it also favors smooth transitions at boundaries, which is why the resulting

image is often quite blurry. The popular **ROF Model** ([63]) preserves discontinuities better, while still effectively removing noise. It is very similar to the Tikhonov model, but replaces the quadratic regularization term by an $L1$ norm:

$$\min_u \left\{ \int_\Omega |\nabla u| dx + \frac{\lambda}{2} \int_\Omega (u - f)^2 dx \right\} \tag{2.75}$$

This regularization term is referred to as the **Total Variation** of u, i.e. the sum over all absolute image gradients:

$$TV(u) = \int_\Omega |\nabla u| dx = \int_\Omega \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2} dx \tag{2.76}$$

In the **TV L1** model, the $L1$ norm is not only used for the regularizer, but also for the data term:

$$\min_u \left\{ \int_\Omega |\nabla u| dx + \frac{\lambda}{2} \int_\Omega |u - f| dx \right\} \tag{2.77}$$

However, this makes the problem being not strictly convex anymore, which means there is no unique solution. On the other hand, the TV-$L1$ model outperforms the ROF model for certain types of noise, and the effect on blurring and contrast loss is smaller.

2.7.2 Convex Optimization

Solving variational problems, e.g. those presented in Section 2.7.1, is a broad field of studies and a sufficient review of the math behind it would be beyond the scope of this thesis. An extensive introduction to convex optimization can be found in [65]. In this thesis, only the main ideas will be presented, with a more detailed explanation of the primal-dual algorithm used for solving the optimization problem in this work.

By definition, a set C in a vector space is called a **convex set**, if the following condition holds:

$$t\mathbf{x} + (1 - t)\mathbf{y} \in C, \quad \forall \mathbf{x}, \mathbf{y} \in C, \ t \in [0, 1] \tag{2.78}$$

This means that every point that lies on the line segment connecting two points in the convex set also belongs to that set.

In the simplest case, an unconstrained convex energy is to be minimized, i.e.

$$\min_u E(u) \tag{2.79}$$

where $E : \mathbb{R} \to \mathbb{R}^n$ is a convex energy and $u \in \mathbb{R}^n$. Usually, a closed-form solution that allows to compute the energy E directly does not exist. Instead, iterative gradient-based methods are used. The gradient $\frac{dE(u)}{du}$ gives the direction of the largest increase of the energy, therefore an update of u in the direction of the negative gradient approaches the minimum. The step size of the update needs to be chosen carefully to effectively reach and converge to the minimum. If the energy to be minimized is complex, this minimum is the global minimum and therefore the only point where the gradient equals zero.

Gradient descent methods have some drawbacks - they tend to be slow in flat regions and do not converge to the minimum directly when the step size is too large. Other approaches have been proposed to tackle these problems, e.g. Gauss-Seidel Iterations or Successive Over-Relaxation. However, discontinuities and additional constraints on the energy are problems that still remain.

So-called interior point methods allow the incorporation of both equality and inequality constraints. See [65] for a review of such methods. Another approach is using the duality principle in convex optimization. Any primal optimization problem can be transformed to a dual representation, which can be solved instead. Examples are the Fixed-Point Algorithm ([66]) or the Projected Gradient Descent Method ([67]). The following section discusses the primal-dual algorithm proposed in [61], which is used in this work.

2.7.2.1 A general Primal-Dual Algorithm

Chambolle and Pock ([61]) proposed an algorithm to solve non-smooth saddle point problems. A saddle point problem is an optimization problem that minimizes with respect to one variable and maximizes with respect to another. Problems that can be solved using this method are of the general form

$$\min_{x \in X} \max_{y \in Y} \{\langle Kx, y \rangle + G(x) - F^*(y)\} \tag{2.80}$$

where X and Y are real vector spaces and the K is a linear operator $K : X \to Y$. The two proper, convex, lower-semicontinuous functions G and F^* map from X and Y, resp., to the set of real numbers including infinity, i.e. $G : X \to \mathbb{R} \cup \{\infty\}$, $F^* : Y \to \mathbb{R} \cup \{\infty\}$.

This saddle point problem is a primal-dual formulation of the primal problem

$$\min_{x \in X} F(Kx) + G(x) \tag{2.81}$$

The primal-dual algorithm proposed in [61] which is used to solve such a problem is

summarized in Algorithm 1.

Algorithm 1 General Primal-Dual Algorithm

Initialization: Choose $\tau, \sigma > 0,\ \Theta \in [0,1],\ (x^0, y^0) \in X \times Y,$ set $\bar{x}^0 = x^0$

Iterations (n $\geq$ 0): Update until Convergence:

$$y^{n+1} = (I + \sigma \partial F^*)^{-1}(y^n + \sigma K \bar{x}^n)$$
$$x^{n+1} = (I + \tau \partial G)^{-1}(x^n + \tau K^* y^{n+1})$$
$$\bar{x}^{n+1} = x^{n+1} + \Theta(x^{n+1} - x^n)$$

The resolvent operators are defined as

$$x = (I + \tau \partial F)^{-1}(y) = \arg\min_x \left\{ \frac{\|x - y\|^2}{2\tau} + F(x) \right\} \tag{2.82}$$

The time steps τ and σ are chosen such that $\tau\sigma L^2 < 1$, with the Lipschitz constant $L^2 = \|K\|^2$. For this general algorithm, a convergence rate of $\mathcal{O}(1/n)$ has been proven.

The optimization problem in this thesis is of the form (see Section 2.5.2.1)

$$E(u) = \int_V fu \; dx + \nu \int_V |\nabla u| dx \tag{2.83}$$

which belongs to a class of optimization problems of the more general form

$$\min_{x \in C} \max_{y \in K} \langle Ax, y \rangle + \langle g, x \rangle - \langle h, y \rangle \tag{2.84}$$

The general primal-dual algorithm for this special class of convex optimization problems with pointwise linear terms $\langle g, x \rangle$ and $\langle h, y \rangle$ is shown in Algorithm 2.

Algorithm 2 Primal-Dual Algorithm for Problems with pointwise linear Terms

Initialization: Choose $\tau, \sigma > 0,\ (x^0, y^0) \in X \times Y,$ set $\bar{x}^0 = x^0$

Iterations (n $\geq$ 0): Update until Convergence:

$$y^{n+1} = \textstyle\prod_Y (y^n + \sigma(A\bar{x}^n - h))$$
$$x^{n+1} = \textstyle\prod_X (x^n + \tau(A^* y^{n+1} + g))$$
$$\bar{x}^{n+1} = 2x^{n+1} - x^n$$

The resolvent operators in this case are projections onto the domains X and Y and the parameter Θ was set to 1. One can easily see that Equation (2.83) is of the same form as Equation (2.84) with $h = 0$ and A being the differential operator. Note that the adjoint

of this operator, i.e. A^*, is the the negative divergence operator, therefore fulfilling

$$\langle \nabla u, \mathbf{v} \rangle_Y = -\langle u, \mathrm{div}(\mathbf{v}) \rangle_X \tag{2.85}$$

Chapter 3

3D Scene Segmentation Tool

Contents

Outline: *This section presents the multiview 3D segmentation method implemented in this work. First, an overview of the system is given, together with a detailed problem formulation and requirements for the method to work. Subsequently, the different parts of the system are described in detail. The theoretical background which is needed to understand how these parts work has been introduced in Chapter 2. References to the corresponding sections explaining the underlying concepts will be given at appropriate points.*

3.1 System Overview

Based on multiple views from calibrated cameras, the tool should reconstruct a single object marked by a user. Not only the reconstruction quality is important, but also the performance in terms of execution time, which should not be higher than a view

seconds. The user interacts with the system by using the mouse to draw strokes on both object- and background regions in at least one of the images. Pixel values under these strokes are collected and analyzed in order to find descriptive properties of both regions. Therefore, it is essential that the user intelligently places these labels in order to cover the essential colors and intensities describing these regions. However, there should not be any limitations on how many and what kind of strokes the user has to draw. After finishing this process of interaction, the algorithm should compute a dense 3D model of the object marked as such. The segmentation process, i.e. distinguishing the object from its background, should be aided by the fact that multiple images from different views are present, all showing the same object. The 3D model should then be presented to the user. Optionally, there should be a possibility to place additional strokes or to start over the process again in case the results are not satisfying. In the end, the model should be converted to a polygon file format and saved for further usage.

To summarize, the basic requirements for the tool are the following:

- Detailed dense 3D Reconstruction of one Object of Interest
- Object is chosen by the User
- Accurate Reconstruction after placing only a few Strokes
- Execution Time in the order of a few Seconds
- Conversion of the Model to Polygon File Format

As a starting point, it is assumed that multiple images from the scene are given together with their camera projection matrices $\mathbf{P}$, as defined in Section 2.2. Intrinsic and extrinsic parameters do not need to be given, since they can be computed from $\mathbf{P}$. Furthermore, a rough bounding box around the object of interest in 3D space is assumed to be present. However, the bounding box could also be computed by finding the common field of view of the images, but only if the object is completely present in all of them.

Figures 3.1 and 3.2 give an overview of how the tool works. While Figure 3.1 shows a flow chart from the user's perspective, Figure 3.2 describes the underlying computations from an algorithmic point of view. One of the core features of the tool is that highly parallelizable tasks, as computations based on pixel or voxel elements usually are, are

executed on the computer's graphics processing unit (GPU). Since the GPU is optimized for parallel computations, this leads to a large gain in performance. Figure 3.2 also shows which parts of the system are executed on the GPU.

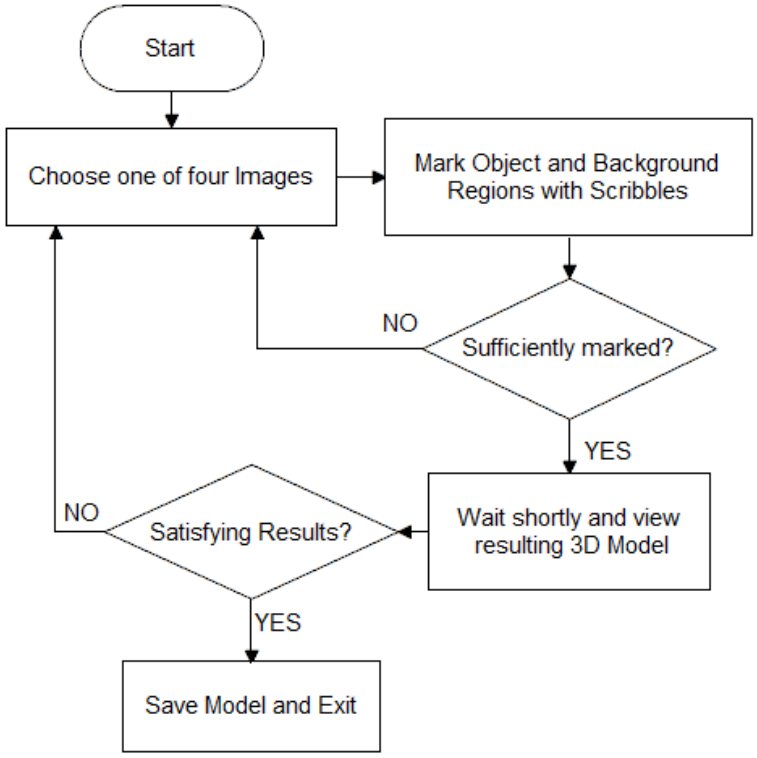

Figure 3.1: Flow Chart for Segmentation Tool from User's Perspective

3.2 Image Pre-Processing

3.2.1 Image Denoising

Images are typically distorted by camera noise to some extent. While these distortions are often negligibly small, noise removal before performing any further steps on the images turns out to increase performance in some cases. Especially in real-world scenarios, where objects usually contain very fine structures and small details, image smoothing can be helpful. However, applying a mean filter or Gaussian smoothing also blurs the edges, which decreases the accuracy of the resulting 3D model. A more suitable alternative is the median filter, which performs better at removing noise whilst preserving edges. A computationally more expensive, but even better technique is the Bilateral Filter. It performs Gaussian filtering based on both geometrical distances and differences in color

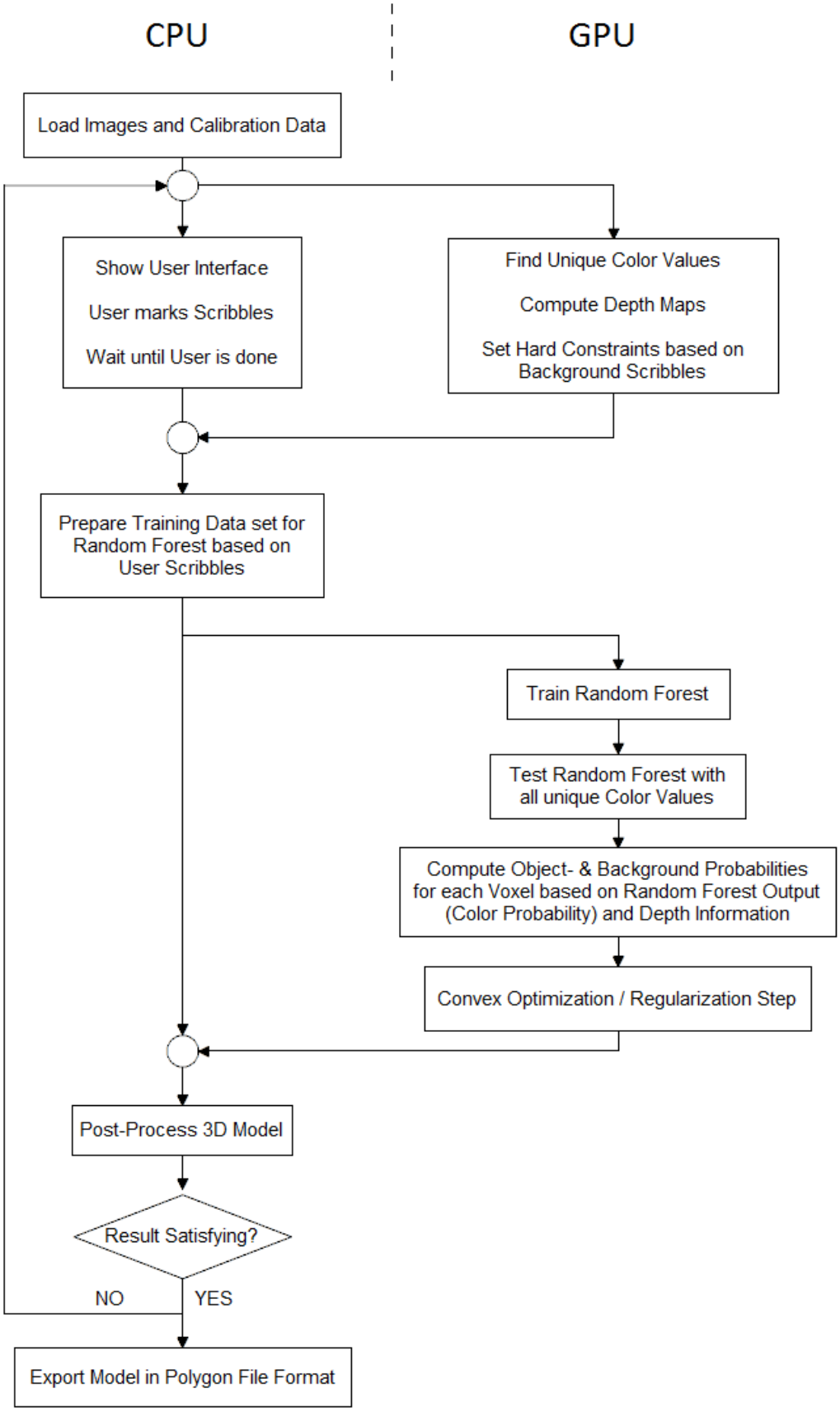

Figure 3.2: Detailed Flow Chart for Segmentation Tool. Left Part: CPU Implementation. Right Part: Parallel GPU Implementation

values. The intensity of a pixel is computed as

$$I_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q|) G_{\sigma_r}(|I_p - I_q|) I_q \tag{3.1}$$

with

$$W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q|) G_{\sigma_r}(|I_p - I_q|) \tag{3.2}$$

where S is a neighborhood around a pixel p and $G_{\sigma_s}, G_{\sigma_r}$ denote Gaussian filters with standard deviation σ_s and σ_r, respectively. The main advantage of the bilateral filter is that it preserves edges well while still smoothing the homogeneous regions of the image, resulting in a comic-like image. In many cases, segmentation of such an image is easier than when the original one is used. While a bilateral filter was used in this work, one has to keep in mind that it might not always be a helpful step or even be counterproductive in some scenarios. Therefore, image pre-filtering is an optional technique in the segmentation tool implemented in this work.

3.2.2 Transformation to CIE Lab Color Representation

Typical image formats represent color values by using the RGB model (see Section 2.1. However, it turned out to be very helpful to first transform the input images to a representation where luminance information is directly represented by one of the channels, as it is the case in the CIE Lab model. The CIE Luv model performs equally well and could also be used instead. The advantages in representing color values using CIE Lab are the following:

- Because of varying lightning conditions, a color segmentation method should not overvalue differences in the luminance. Since color information is separated from brightness, higher importance can be placed on the color itself than on the brightness, without any additional computational effort. This can also be done for finding point correspondences, e.g. in the Plane Sweep Algorithm for computing range images (Section 2.4.1).

- Euclidean distances between two color values approximate perceptual differences. This is very important when assigning pixels to one of multiple color models representing the classes, which is usually based on distances in color space.

3.3 Random Forests for Image Segmentation

Finding a way to model region properties of both object and background based on scribbles drawn by the user is the most crucial part of the 3D segmentation tool. The concept of random forests turned out to be suited well for that task and yielded better results than similar regression models, e.g. the EM-Algorithm or K-Means clustering. As explained in Section 2.6, using a random forest has several advantages over other techniques, e.g. it does not make assumptions about the shape of the clusters and performs very well in terms of computation time. The remainder of this section explains how training data for random forests is collected via user interaction, how the model is trained, what aspects need to be taken care of and how region probabilities are computed for test data using the trained model. Also, some sample results of applying the regression model to images without any regularization are shown. These should serve to show how the random forest alone already forms a very good basis for further segmentation and reconstruction steps.

3.3.1 Obtaining Training Data from User Interaction

The random forest regressor needs to be trained on a set of input data points. In this work, only color values are used for this purpose, while including more sophisticated features into the model is also possible. Such additional features could for example be texture descriptors. The subjects to be discussed in this section are how to collect these data points, how to reduce eventual noise, as well as how to thin out the training data to only use a representative subset, which is done for performance reasons.

3.3.1.1 Interactive Region Labeling using Mouse drawn Strokes

Since the goal of the segmentation tool is to reconstruct one specific object from a scene, some user input needs to be given in order to know which object to extract. Also, prior information about the properties of the object and background need to be identified, since reconstruction from just selecting a seed point inside the object without any other knowledge is a tremendously difficult task. Therefore, the user is asked to draw strokes in one ore more of the images, which mark the two distinct regions. The color values of the marked pixels are collected, analyzed and used to train a regression model, which serves to compute object/background likelihoods for other pixels in the images. Figure 3.3 illustrates the user interaction process. The red strokes indicated background regions, while the blue ones are used to label the object. Additionally, the user can draw a bounding box around

the object, in case several similar ones are present and might falsely be reconstructed instead.

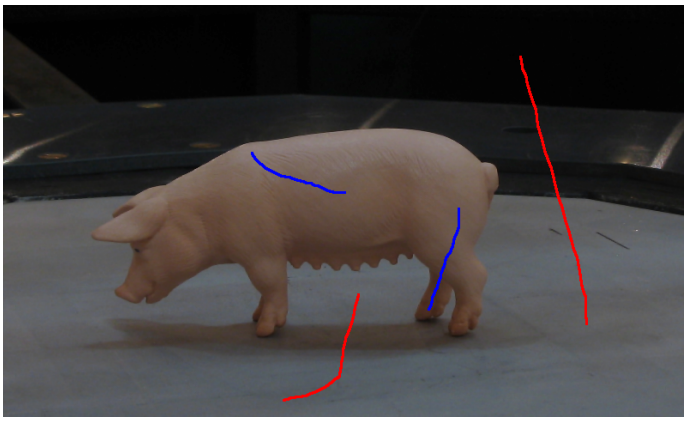

Figure 3.3: User Interaction - Strokes mark Regions

Viewing an object from only one perspective for user interaction might not be sufficient, since colors and textures might not be the same when viewed from the other side. Also, lightning conditions may be different. In order to account for this problem, the user is presented four images of the set of available views. The first image is selected randomly. Then, the other three images are selected by subsequently choosing the one that maximizes the minimum angle to the already selected images. This angle is defined as the angle between the viewing vectors, which are pointing from the camera centers towards the scene, perpendicular to the image planes. If views from all directions are present, this approach should guarantee that the user can mark any significant area on the surface of the 3D object. It is up to the user in how many images regions are marked, but the results heavily rely on sufficiently covering all the differently colored areas.

3.3.1.2 Outlier Reduction

If all of the user defined seed points are used, outliers due to noise might be present. These data points are likely to decrease the performance of the system. While pre-filtering the image can successfully reduce the effect of noise (see Section 3.2.1), an additional measure to counteract outliers was implemented. Instead of only using the single marked pixels, a neighborhood (typically 3x3 pixels) around that point is taken into account. However, computing an average color value in that neighborhood and using this as a training point can be counterproductive. This is because mixing existing colors is likely to produce new colors, which are actually not present in the image. As an example, consider a graylevel

image where the object consists of black and white areas, while the background is medium gray. At borders between the interior object regions, the average also produces a medium gray. Thus, the model is falsely assuming that this color is present in both regions. A better approach, which is used in this work, is to use a *median* color value of the pixel neighborhood. In a single-channel image, the median is defined by the pixel with intensity such that half of the pixels in the neighborhood are darker and half are brighter. This results in only actually existing pixel values being used as training data. However, the median is not unambiguously defined for color images, since vectors cannot be sorted such as scalars. However, the median generally minimizes the L1-norm. This means, that the pixel within the neighborhood is used which has the lowest sum of distances to the other points, i.e.

$$\mathbf{m} = arg \min_{\mathbf{y} \in \mathcal{N}} \sum_{i=1}^{|\mathcal{N}|} \|\mathbf{x_i} - \mathbf{y}\|_2 \tag{3.3}$$

where $\mathbf{x}$ and $\mathbf{y}$ are pixel color vectors in the neighborhood $\mathcal{N}$ around the center pixel, which was marked by the user. This minimization problem can be solved using an iterative subgradient descent algorithm.

3.3.1.3 Reducing the Number of Training Data Points

Training a random forest on a lot of data points can be computationally expensive. At each node of each decision tree, several features and thresholds need to be tested, the data needs to be partitioned and the information gain needs to be computed (see Section 2.6 for details). If all of the collected data points are used, the forest has to be trained on typically several hundreds or even thousands of training points. In this work, instead, the training data set is reduced to a set of representative data points. First, multiple occurrences of color values are deleted, resulting in a reduced set of unique colors. Then, a maximum number of color values is defined, typically 100-200 points per class. Then, for all the collected points in one class, one of the data points is chosen at random. A second point is added to the subset, which has the maximum Euclidean distance to the first point. Subsequently, points are added which have the maximum minimal distance to any of the points in the subset. This is repeated until the maximum number of colors is reached. Figure 3.4 illustrates the result of thinning out color values on a sample test data set. The left 3D plot shows the collected color values, while the right one shows the resulting representative subset.

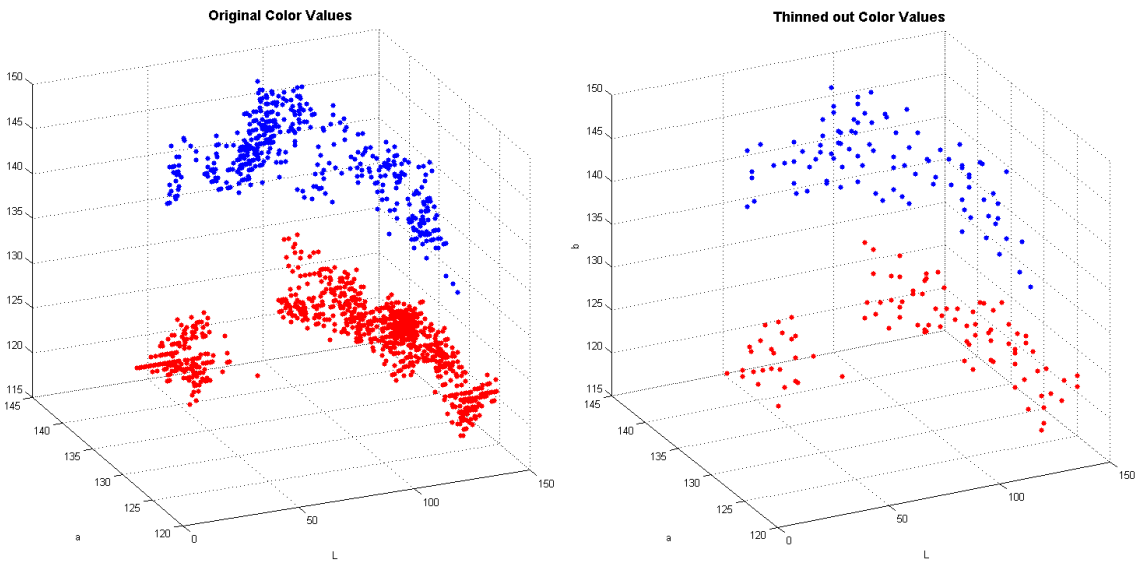

Figure 3.4: Process of reducing Number of Color Values - Data is thinned out, but Distribution Properties are preserved

The training algorithm which produces the random forest assumes that the density of data points is about the same for object and background regions. This is not a critical criterion if the clusters can be separated, but it is if certain colors occur in both regions. Then, the random forest training algorithm would assign slightly higher probabilities for the class which has a higher density of data points in the cluster representing that color. If the number and size of clusters is significantly different for the two regions, simply reducing the collected color values to the same number of points leads to different densities, as shown in the example of Figure 3.5. Here, the background consists of only one small cluster, representing the gray ground plane in the image. The object, however, consists of several distinct colors with different brightness, leading to more and wider clusters. The red cluster (background) would be much denser if the same number of data points is allowed as for the foreground. Therefore, a slightly adapted approach is used: First, the number of color values is reduced to the maximum allowed number of points in each region. Then, the minimum distances of the last point added to the thinned out subset in each class are compared. The class for which this distance is smaller, i.e. the one with the higher density, is again thinned out from the beginning. However, this time only until the minimum distance between any two points in the subset reaches the minimum distance from the other class. This leads to more or less equally dense point clouds, which improves the quality of the random forest regression in some scenarios.

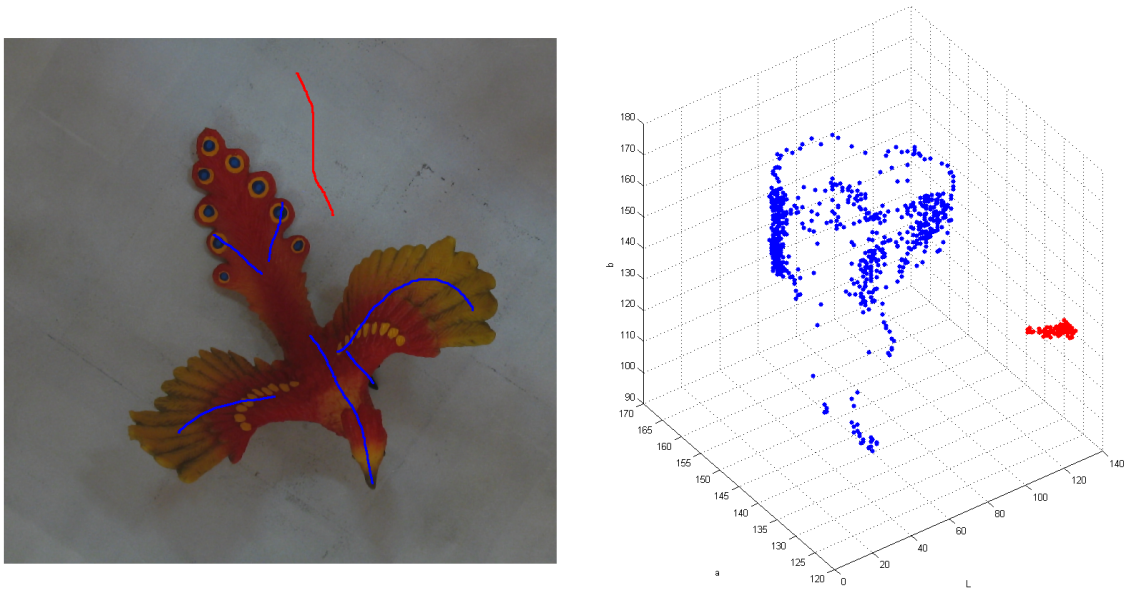

Figure 3.5: Example of Collected Color Values in Image (left) and Lab-Space (right)

Note that a major assumption was made when deleting double occurrences of color values as well as thinning out the point cloud to obtain a more uniform density: The amount of pixels marked by the user in certain regions does not imply any prior information about the size of that region. This means, if for example 90% of the collected pixels are red, while 10% are blue, the algorithms in this work do not assume that red points are more likely than blue points. Some other proposed methods do not make this assumption and directly use the input data as is. Figure 3.6 shows two situations - in the left figure, the number of collected data points gives information about the likelihood of their occurrence. The right figure shows a scenario where these implications would be completely mistaken, while still being a realistic user input. Therefore, the conclusion that not interpreting the user input in such a way is safer seems to be reasonable.

3.3.2 Training the Model

Based on the training data, a random forest is built which should satisfy the following conditions for assigning corresponding probabilities to a test data point:

1. If the test color vector is similar to a color being present in the object region, but not being present in the background: Assign high object- and low background probability (and vice versa)

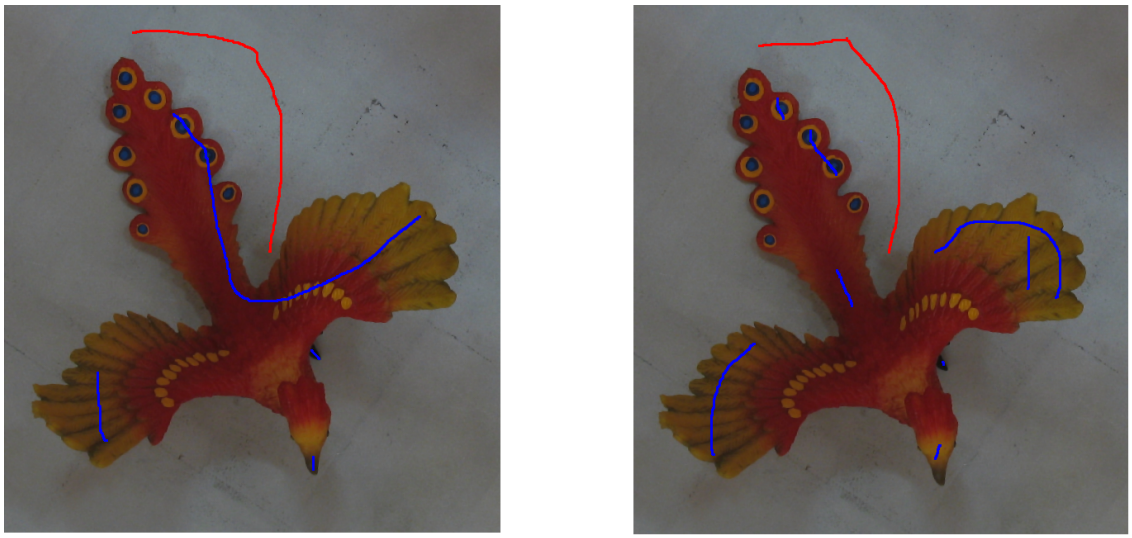

Figure 3.6: Left: Strokes correctly imply Color Distribution Right: Not the Case

2. If the test color is similar to a color being present in both of the classes: Assign equal probabilities

3. If the test color is not similar to any of the colors in either region: Also assign equal probabilities

A random forest is built by training an ensemble of binary decision trees, which are weak learners in a sense that they introduce randomness in several ways (again, see Section 2.6 for details). Choosing adequate split functions at the nodes of each tree as well as finding suitable thresholds to partition the data is crucial. However, standard implementations, such as using axis-aligned or linear split functions with random feature and threshold selection, do not suffice the needs of image segmentation in this work. Certain considerations, which will be explained in the remainder of this section, need to be made in order to fulfill all three conditions above.

3.3.2.1 Choosing a Split Function

As discussed in Section 2.6, linear split functions are not suited well for the purpose of interactive segmentation, since the third condition above is not fulfilled. For example, imagine a situation where each class consists of one tight cluster, with the two clusters being linearly separable. Any split function would be a hyperplane somewhere between these clusters, separating the data by determining on which side of the plane a test point

lies. Any test point outside of both clusters, but not between them, would be assigned highest probability of belonging to the nearest cluster, no matter of it's distance to it.

To overcome this problem, conic split functions can be used. These are much more complicated to parametrize though, and finding parameters that form conics which are actually separating the data requires a lot of computational effort. In this work, conic split functions in form of ellipsoids are used, instead of allowing for any arbitrary parametrization. This makes them a lot easier to handle, while still bounding the clusters in all directions. Furthermore, these ellipsoids have fixed orientations and a fixed ratio between the length of the principal axis. These constraints do not pose noteworthy limitations in terms of performance, since the randomness introduced in each tree averages out when using enough trees, thus allowing for any arbitrary cluster shape.

As a starting point, consider a circular split function in the 3-dimensional Lab color space, i.e. a sphere:

$$r^2 = L^2 + a^2 + b^2 \tag{3.4}$$

where L, a and b define the midpoint of the sphere, while r is the radius. Wether another point lies inside or outside that circle can be determined using

$$(L - L_i)^2 + (a - a_i)^2 + (b - b_i)^2 \leq r^2 \tag{3.5}$$

where the subscript denotes the data point to be checked. If the inequality holds, the point lies within the circle.

As already pointed out in Section 3.2.2, the L-value in the CIE Lab color model describes the brightness, which can be used advantageously. When performing segmentation of color images, differences of brightness can originate from different illumination conditions in different parts of the same-colored object and should not be over-interpreted by the segmentation method. Instead, more influence should be placed on the color information itself, i.e. the a- and b-value. This idea can be included into the split functions of the random forest. The spherical split functions are "stretched" in the direction of the L-component, thus making differences in this component weigh less heavily. In practice, a factor of 3-4 gave the best results for several different data sets. A factor of 4 for example means that a difference of 1 in the a- or b-components of two color values is interpreted as the same distance as a difference of 4 in the L-components. Some segmentation algorithms completely omit the L-component, which in addition makes the problem computationally less complex. However, this turned out to not lead to acceptable results - already the

fact that black, white and any gray level in between are then described by the same color vector makes this method not practicable.

Theoretically, the a- and b-component can also be assigned different importance. However, in practice there is not really any reason that justifies doing that. The general model for the split functions are then inequalities of the form

$$\frac{(L - L_i)^2}{C_L^2} + \frac{(a - a_i)^2}{C_a^2} + \frac{(b - b_i)^2}{C_b^2} \leq r^2 \tag{3.6}$$

where C_L, C_a and C_b are constant weight factors. Geometrically interpreted, a data point fulfilling this condition lies inside an ellipsoid centered at $[L, a, b]^\top$. The principal axes are of length r times the corresponding weight factors.

To summarize, elliptical split functions turned out to be a good trade-off between complexity of the model and computational simplicity. They are easy to interpret geometrically with only a few parameters while still allowing for bounded clusters.

3.3.2.2 Finding the best Split at each Node

A good split function partitions the data into two sub-regions, which are both more homogeneous than the original one. The *information gain* is used to measure the quality of a split function. Details on the definition of this measure can be found in Section 2.6. Obviously, infinitely many possible split functions exist, and they cannot all be tested. Also, the concept of random forests requires that only a small number of functions is tested at each node, introducing randomness in each tree. In order to effectively limit the search space, the centers of the ellipsoids are chosen to be random training data points assigned to the corresponding node for which the split is to be performed. Also, clusters tend to become tighter and modeled more accurately if ellipsoids centered at data points are used, instead of choosing the centers randomly in Lab space. In the remainder of this work, the terms *"feature"* and *"ellipsoid center"* are used interchangeably. Similarly, since no information about cluster size is present, the thresholds, i.e. the parameters r in Equation (3.6), are chosen to be the distance to another randomly chosen data point multiplied with a random factor between 0.5 and 1.5. This leads to reasonable sizes of the ellipsoids and gives better results for the same number of thresholds tried compared to using arbitrary random numbers. As an additional measure, a maximum for the possible thresholds is introduced, which was set to approximately $r^2 = 300$ in this work. This also helps modeling clusters more accurately and preventing merging several smaller clusters into a single large one.

For each ellipsoid center, several thresholds are chosen and the data is partitioned according to the resulting split function. The information gain is computed for each center and threshold, and the function yielding the best split is chosen. Some considerations about how to determine this optimal split function need to be made though. The left plot in Figure 3.7 illustrates how choosing an ellipse around the red cluster in the 2D scenario immediately perfectly separates the data. If enough split functions are tried, most of the trees in the forest will perform a similar split. This leads to a poor modeling of the blue clusters, as well as "neutral" regions being interpreted as likely being part of the blue data class.

In order to tackle this problem, the following idea was used: Instead of randomly choosing the center point of the ellipsoids from the set of feature points, two "best" split functions are searched - one that embeds mostly data points of the first class, and one for the other. Then, the weaker one of these two functions is used. This leads to a subsequent modeling of the smaller clusters at each level of the tree, before finding the final split. If both functions yield the same information gain, any of them is chosen at random. This is often the case at the last split, when the data can be separated perfectly. The second and third plot in Figure 3.7 illustrate how clusters are subsequently modeled and the last split can embed either mostly points from the first or the second class. Since each of the two possibilities is chosen at random, test points actually not being close to any training point are assigned 0.5 probability on average, which is the desired behavior.

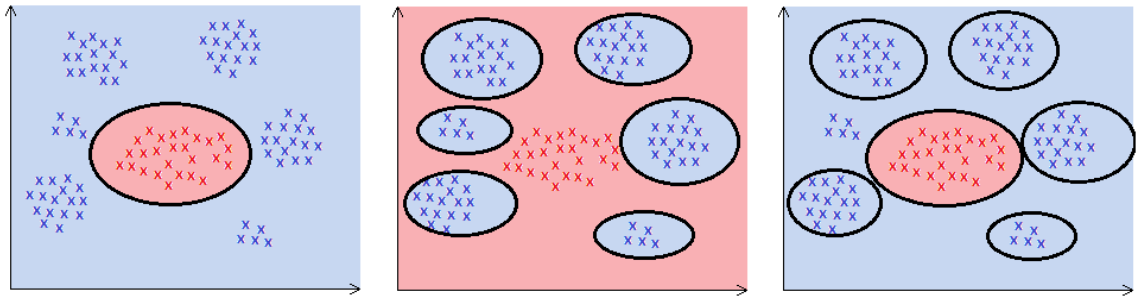

Figure 3.7: Possible Partitions depending on Split Functions. Last performed Split is critical for Probabilities of Points in Feature Space outside of any Region (should be 0.5 after Averaging)

In a realistic scenario, clusters are not as distinct and separable from each other as the ones in Figure 3.7. Despite the measures explained above, in some cases the neutral regions are still favored to belong to one of the two classes. Therefore, an additional technique was introduced. The region outside of any cluster is defined by the probability distribution

of the rightmost leaf in a decision tree, since a node's right child always holds the data belonging to the exterior of the ellipsoid. Instead of using this probability distribution, an additional post-processing step is performed in each tree, which explicitly analyzes the exterior region. The remaining data points in that region are fused into clusters until no more points remain. Then, all other regions are assigned exactly probabilities of 0.5.

Algorithm 3 describes the process of training the random forest, as it was used in the implementation of this work.

3.3.3 Applying the Random Forest

While training a random forest requires some computational effort, applying it to a test set is very efficient. For each tree in the forest, the test data point is traversed through the that tree according to the split criterion at each node. When a leaf is reached, the probability values stored in that leaf are assigned to the test data point. With the extension presented in the previous section, i.e. the explicit treatment of the exterior region, special care needs to be taken if the data point ends up in the rightmost leaf. In this case, it is checked against the separately stored split functions for that region. If the point lies inside of any of these ellipsoids, the corresponding probabilities are assigned. Otherwise, equal probabilities of 0.5 are chosen, since in that case the test data point does not lie within any cluster.

The resulting probabilities from each tree are averaged in order to yield the final probability values assigned with that point. Algorithm 4 illustrates the process of applying the random forest to test data.

3.3.4 Sample Results and Influence of Parameters

This section serves to show some results of applying the random forest, which was learned based on strokes from the user interaction, to an image. In these experiments, each image pixel was tested independently using the trained forest. The resulting probabilities for a pixel to be part of the object are illustrated as a graylevel image, where high gray levels depict high probabilities. The probability for a pixel of belonging to the background region is simply that converse probability of the object region. In the final 3D segmentation tool, the user can place strokes in up to four images. However, since reconstruction is not of interest in these experiments, only one image was used for evaluation.

Figure 3.8 shows a sample result, which illustrates that the desired properties of the regression model are fulfilled. Colors that appear only in one of the two regions lead to

Algorithm 3 Random Forest Training

Initialization: choose num_trees, $features\_per\_node$, $thresholds\_per\_feature$, max_tree_depth, C_L, C_a, C_b, $bootstrap\_ratio \in (0, 1]$

for $tree \leftarrow 1$ **to** num_trees **do**
- select random subset of data points, fraction of data points defined by $bootstrap\_ratio$
- **while** $depth \leq max\_tree\_depth$ **do**
  - **for** $class\_label \leftarrow \{object, background\}$ **do**
    - **for** $f \leftarrow 1$ **to** $num\_features$ **do**
      - **for** $t \leftarrow 1$ **to** $thresholds\_per\_feature$ **do**
        - choose ellipsoid center as random data point of class *{class_label}*
        - choose random threshold (radius)
        - compute resulting information gain of split function applied to the data
        - save best split function and information gain of class *{class_label}*
    - **if** *at least one information gain > 0* **then**
      - compare information gain of resulting two best split functions
      - choose the one with **lower** information gain, but > 0
      - if equal, choose one function at random partition data using corresponding split function and save partitions in two child nodes
    - **else**
      - make node a leaf
      - compute probability distribution from remaining data points and assign with leaf
      - exit **while**
  - repeat procedure for each child node, depth $\leftarrow$ depth + 1
- find rightmost leaf, describing region outside of any cluster
- **while** *data in rightmost leaf present* **do**
  - **for** $f \leftarrow 1$ **to** $num\_features$ **do**
    - **for** $t \leftarrow 1$ **to** $thresholds\_per\_feature$ **do**
      - choose random feature and threshold within remaining data
      - find split function yielding highest information gain
  - assign data inside split ellipsoid to one cluster
  - save probability distribution and split function
  - mark these points as clustered and remove from leaf
- set probabilities in rightmost leaf to 0.5 each

Algorithm 4 Random Forest Testing

```
for tree ← 1 to num_trees do
    node ← root of tree
    while node is split, not leaf do
        check split function
        if data point lies inside split ellipsoid then
            node ← left child node
        else
            node ← right child node
    if node is leaf, but not rightmost leaf then
        assign probabilities stored in leaf for that tree
    else if node is rightmost leaf then
        for cluster ← 1 to num_outside_clusters do
            if data point lies inside cluster then
                assign probabilities stored with that cluster for that tree
                next tree
        not in any cluster, assign probabilities of 0.5 for that tree
return average of single tree probabilities
```

clear probabilities near 0 or 1. This is the case for the red and yellow tones of the bird, as well as for the light gray of the ground plane. Colors that are present in both object and background yield medium gray, indicating probabilities of around 0.5. See for example the claws of the bird or the far background. Finally, colors that have not been marked by the user, e.g. the bright reflections of the light in the background, also get assigned equal probabilities.

Of course the performance heavily relies on the quality of the user input. If colors which are only present in one region are not marked, a probability of 0.5 will be assigned, which eventually might still lead to acceptable results. If however a color is present in both regions, but only marked in one (e.g. if the dark gray of the claws is not marked), probabilities of around 0 or 1, resp., will be assigned, usually degrading the resulting 3D model.

3.3.4.1 Influence of different Channel Weights

If each channel in the CIE Lab color model is assigned equal importance, the split functions are spheres in $\mathbb{R}^3$. However, it turns out to be advantageous to treat the brightness information differently, which is explicitly given by the L-value. While the other two channels have equal importance, since they both model the color information itself, the

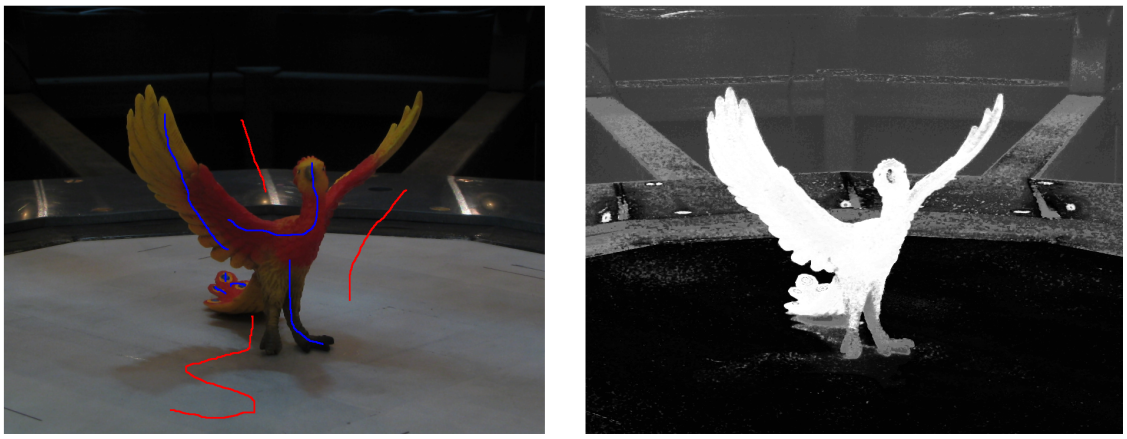

Figure 3.8: Left: User-placed Strokes Right: Resulting Pixel Probabilities for Object Region

brightness of the color is only a weaker criterion. The reasoning behind this is that different brightness of the same color does not necessarily indicate different regions, but often originates from varying lightning conditions. Mathematically, this can be achieved by setting the constant C_L in Equation (3.6) to a higher value than C_a and C_b. However, brightness information should not be neglected completely, since it is still a necessary distinguishing feature in most cases. Therefore, a trade-off has to be found. In several experiments on different kinds of data sets, a reasonable choice for C_L was approximately $3\,C_a \leq C_L \leq 4\,C_a$ with $C_a = C_b$.

Figure 3.9 shows the effect of choosing the factor C_L equal to C_a and C_b as well as for choosing it to be 4 times higher. While the object has uniform color and should therefore be easy to classify as such, the rough surface structure causes variations in brightness. While the results are quite unsatisfactory if all channels have the same influence (middle figure), the random forest performs well if the brightness values are weighed less heavily (right figure). Note that, as in all experiments analyzing the influence of choosing different parameter values, the same set of strokes was used to make a meaningful comparison possible.

3.3.4.2 Influence of Number of Features and Thresholds

As discussed in Section 2.6, limiting the the number of split functions to be tested at each node introduces randomness, which is very important for learning a good regression model. In this work, several features, i.e. centers of the ellipsoids which split the data, are tested, together with corresponding thresholds determining the size of these ellipsoids.

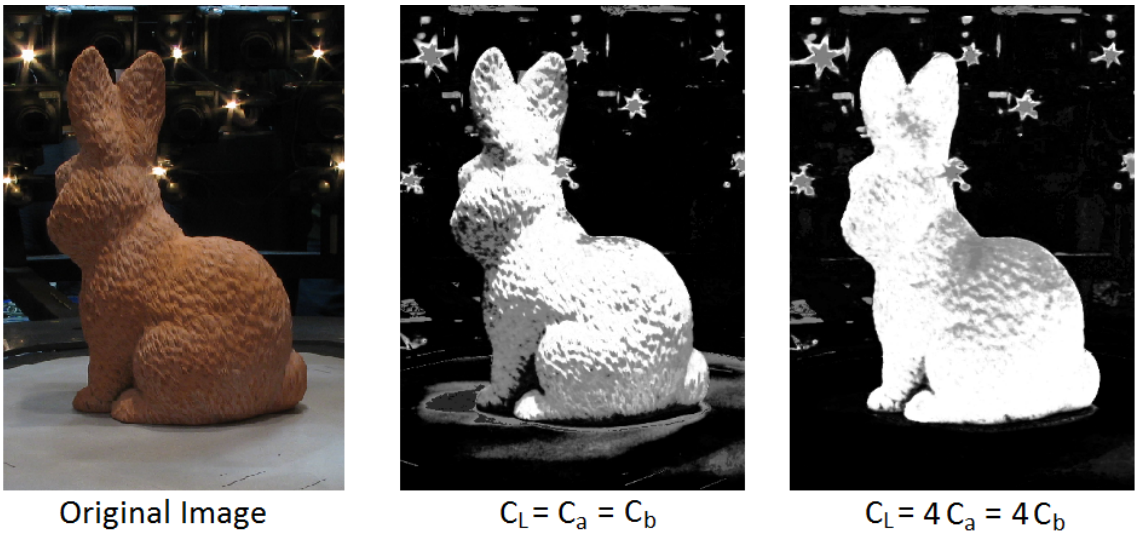

Figure 3.9: Effect of different Weights on L-Channel. If less weight is put on Brightness Differences, self-shadowing is less critical

Experiments have shown that using only very few features, but testing them for several thresholds, usually leads to the best results. Randomness can most effectively be imposed by reducing the number of features, while reducing the number of thresholds for each features can easily make the algorithm fail finding meaningful clusters. As a guidance value, the number of features is suggested to be set to about 3-5 per class, while the number of thresholds is usually chosen to be between 10 and 20. Note that in this work, around 50-200 data points for each class are usually used as training data.

Figure 3.10 compares the results for using only 1 feature per class, tested with 2 thresholds each, against using 10 features with 20 thresholds each. While the differences are not drastic, one can see that using more features and thresholds leads to a more decisive result. In contrast, testing less split function at the nodes produces wider clusters with less confidence, leading to less distinctive probabilities. However, using too many split functions reduces the amount of randomness, which can turn out to be disadvantageous. In this example, the shadow of the bird is modeled better in the left image, as an area of medium gray, whereas in the second image it has very high object confidence at some spots.

3.3.4.3 Influence of Bootstrap Ratio

Another source of randomness when using random forests is achieved by using only a subset of the input training data for each tree, a so-called *bootstrap sample*. This subset

Figure 3.10: Effect of different Number of Features and Thresholds. More Features and Thresholds lead to more distinct Probabilities

is randomly chosen and different for each tree. The choice of the bootstrap ratio, i.e. how much of the training data is used, controls the randomness. This technique can also help to reduce the effect of outliers, since single, isolated data points are left out in many trees. In this work, however, choosing a low bootstrap ratio did usually not increase the quality of the segmentation result. Since the user is assumed to place strokes intelligently and images are pre-filtered to remove noise, outliers do normally not pose a problem. Also, randomness is already achieved by using less features and thresholds as well as by thinning out the input data, which is similar to using a bootstrap sample. Therefore, the bootstrap ratio is set to 0.8-1.0 throughout this work. The visual results for using a low vs. a high ratio are very similar to the ones in Figure 3.10 for using few vs. many split functions. Figure 3.11 shows sample results for using a ratio of 0.4 and another one of 0.9. Again, a lower ratio, i.e. more randomness, decreases the confidence of the probabilities, leading to more values around 0.5. However, if the user input is not sufficiently accurate or clusters are unable to be separated, a low bootstrap ratio may be advantageous. In practice, using fewer split functions instead of changing the bootstrap ratio also works well in these cases.

3.3.4.4 Influence of other Parameters

While the most important parameters have been analyzed above, there are some more ways to influence the behavior of the random forest. One of them is to change the **maximal number of training points** when applying the procedure of thinning out the collected color values. However, the differences are marginal, which is why a relatively small number of points is used for performance reasons. Also, using less data slightly increases randomness, which can be of advantage too.

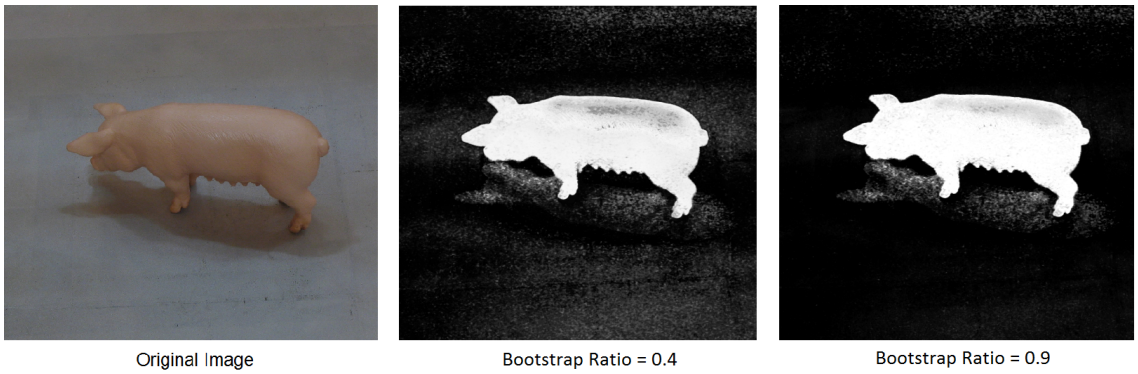

Figure 3.11: Effect of different Bootstrap Ratios

Another parameter is the **number of trees** of in the forest. One major advantage of using random forests is that overfitting by using too many trees cannot occur. Therefore, at some point, further increasing the number of trees does not change the performance anymore, but only increases computational time. In this work, a number of around 100 trees was found to be sufficient.

Not only the number of trees, but also the **maximal depth of each tree** can be adjusted. If the maximal depth is reached at the training stage, the node is turned into a leaf, without further splitting. This is another way to increase randomness. A maximal depth of 5-6 turned out to be a good trade-off between limiting computational time and memory requirements by allowing trees not to grow arbitrary deep, but still having enough split nodes to cluster the data sufficiently.

3.3.4.5 Performance in Real-World Examples

Real-World scenarios often pose hard challenges to the segmentation algorithm. Especially in nature photographs, very small details and shadows make the classification very difficult. Here pre-smoothing of the image can be helpful, in order to remove details like single blades of grass and their shadows. Another problem is that colors often occur in both the object to be segmented and the background region. When using the methods in this work, this leads to regions with equal object- and background probabilities, making classification without other knowledge an almost impossible task.

Figure 3.12 shows an example of a real-world photograph. Here, the clock tower is the object of interest. The major challenge is that many similar color values are present and occur in both regions. Also, many small details and shadows complicate the process. However, if the user places strokes intelligently by marking only regions that actually

matter, acceptable results can be achieved. In scenarios like this, the color information alone is typically still not enough for reconstruction and segmentation of the object. This is why it is important to incorporate depth information to aid the process. In this example, trees with a maximal depth of 6 were used, the bootstrap ratio was set to 0.8 and the forest consisted of 100 trees.

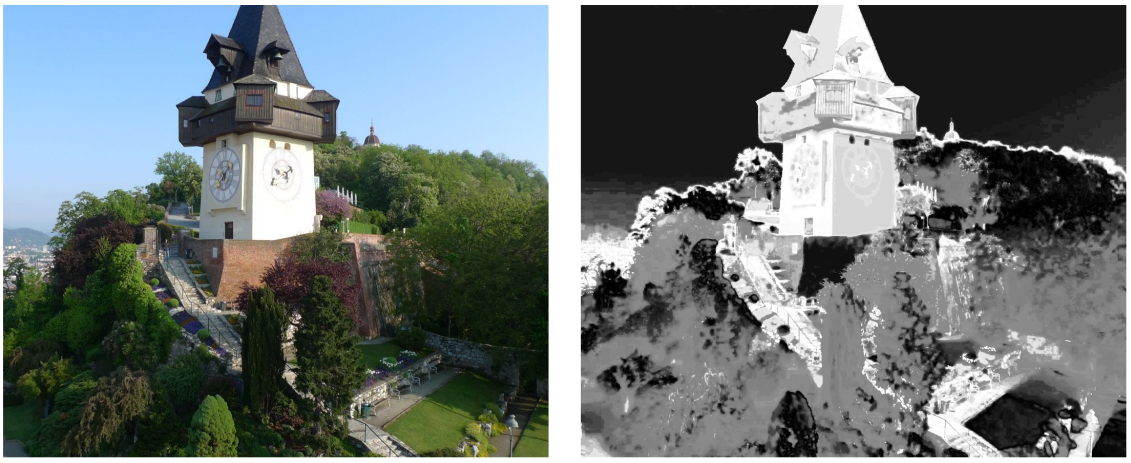

Figure 3.12: Example of a Real World Scenario

3.4 Depth Map Generation

Because of its comparatively low computational complexity and its ability to be parallelized efficiently, the Plane Sweep Algorithm introduced in Section 2.4.1 is used to compute the range image for each camera view. The corresponding depth values for the image pixels are combined with the output of the random forest in order to yield a reliable and accurate reconstruction of the object of interest in 3D. While this section concentrates on the process of computing depth maps, the subsequent Section 3.5 explains how color- and depth information are combined to form a probabilistic model used as input for the final optimization process.

As explained in Section 2.4.1, the Plane Sweep Algorithm works by projecting each pixel of a key view into all sensor views using a homography induced by a plane at a certain depth parallel to the key view. The depth of the plane which yields the most similar color values is assigned as the depth at that pixel. As a similarity measure, several methods with different complexity can be implemented. In the most simple case, the sum of Euclidean distances between all the corresponding pixel values is used. As an extension to reduce

the impact of outliers, a threshold on the maximum value of the single distances can be incorporated. More reliable results which are less affected by noise are typically obtained when using a windowed approach, where the dissimilarity is computed and accumulated for each pixel in a window around the pixel of interest. A more sophisticated approach than the sum of Euclidean distances uses the normalized cross-correlation in a small window around the pixel. While this usually leads to more reliable results which are less affected by different lighting conditions, the computational effort is much higher. If a sufficient amount of memory is available, integral structures can be used to accelerate the algorithm. An even more expensive method in terms of memory consumption and execution time is to not just apply a winner-takes-all scheme for choosing the best depth, but to store all cost values in a 3D probability volume. The most probable surface can then be estimated by finding the minimum cost surface, including a smoothness constraint.

In this work, a low-complexity algorithm was chosen over a more sophisticated approach. Since keeping the cost values and solving the 3-dimensional optimization problem in each view is computationally infeasible, a simple winner-takes-all scheme was implemented. As a similarity measure, a simple sum of squared differences approach was used instead of a cross-correlation solution, mainly for the following two reasons:

- Depth maps are computed on the GPU while the user is placing strokes in order to capture the scene's color properties. This process might only take a few seconds, and the number of depth maps available at the moment of finishing user interaction is used for further processing. It turned out that a higher number of range images yields better results than investing more computation time to get more reliable depth values, but for a smaller number of views.

- Since brightness information is directly available as the L-value of the CIE Lab model, less weight can be put on this channel when computing the geometric distances, in order to account for different lighting conditions. Thus, good results can be achieved without the necessity of a correlation-based error measure. Since this approach yielded similar results to more sophisticated measures, the simple sum of squared differences was used in this work. The L-channel was divided by a value of 4 in order to achieve a better insensitivity to changes in illumination.

Figure 3.13 shows two example results of applying the Plane Sweep Algorithm with a simple sum of squared differences and winner-takes-all scheme to one of the input views. Dark pixels indicate near objects, while high pixel values indicate objects that are farther

away. The depth range was set such that only depth values inside a tight bounding box around the object are allowed. This leads to wrong estimates in regions which are further away or closer to the camera center. Also, homogeneous regions, such as the ground plane, can cause the algorithm to fail.

It is possible to discard unreliable depth values and only keep to ones with high confidence. Not only the ones which have a high dissimilarity value are considered to be unreliable, but also the ones which do not show a clear minimum in the cost function among the different plane hypotheses. The latter is often the case in homogeneous regions, where the dissimilarity might be low, but very similar for different depths. However, discarding unreliable depth values yielded worse results in most of the experiments, since for some regions in the scene no depth information was available at all in the set of sparse depth maps. It turned out to be advantageous to use all values in the depth maps and let the robust fusion step discard inconsistent values, which is subject of the subsequent section.

3.5 Fusion of Color and Depth Information in Voxel Space

Scene segmentation from color information is not always possible, since very similar colors can appear in both regions, or the user input does not sufficiently cover the object's properties. Therefore, depth information is used in this work to aid the process. In theory, a 3D scene can be completely reconstructed from range images, as long as the views capture all parts of the object sufficiently. In practice, however, depth maps contain outliers and are not fully consistent among the views. Also, when using the Plane Sweep Algorithm, depth values are discretized to a certain resolution and limited to a certain range, which makes reconstruction of far-away objects infeasible. Moreover, occlusions and the difficulty of recovering depth information in homogeneous regions are typical problems that complicate the reconstruction process.

While reconstructing a 3D scene from high-quality depth maps from a multitude of views is often a manageable problem, scene segmentation from solely depth information is not possible without additional user input, since the algorithm alone cannot know which object to segment and what parts to discard. This is why depth maps are used in combination with the probabilities coming from the random forest, i.e. the color information stemming from user input. The idea is to use the depth map fusion algorithm introduced in [60] and described in Section 2.5.3, and incorporate the probabilities from the random forest into this model. Recall that the individual depth maps are fused by computing corresponding truncated signed distance functions (TSDFs) in scene space. The TSDF values

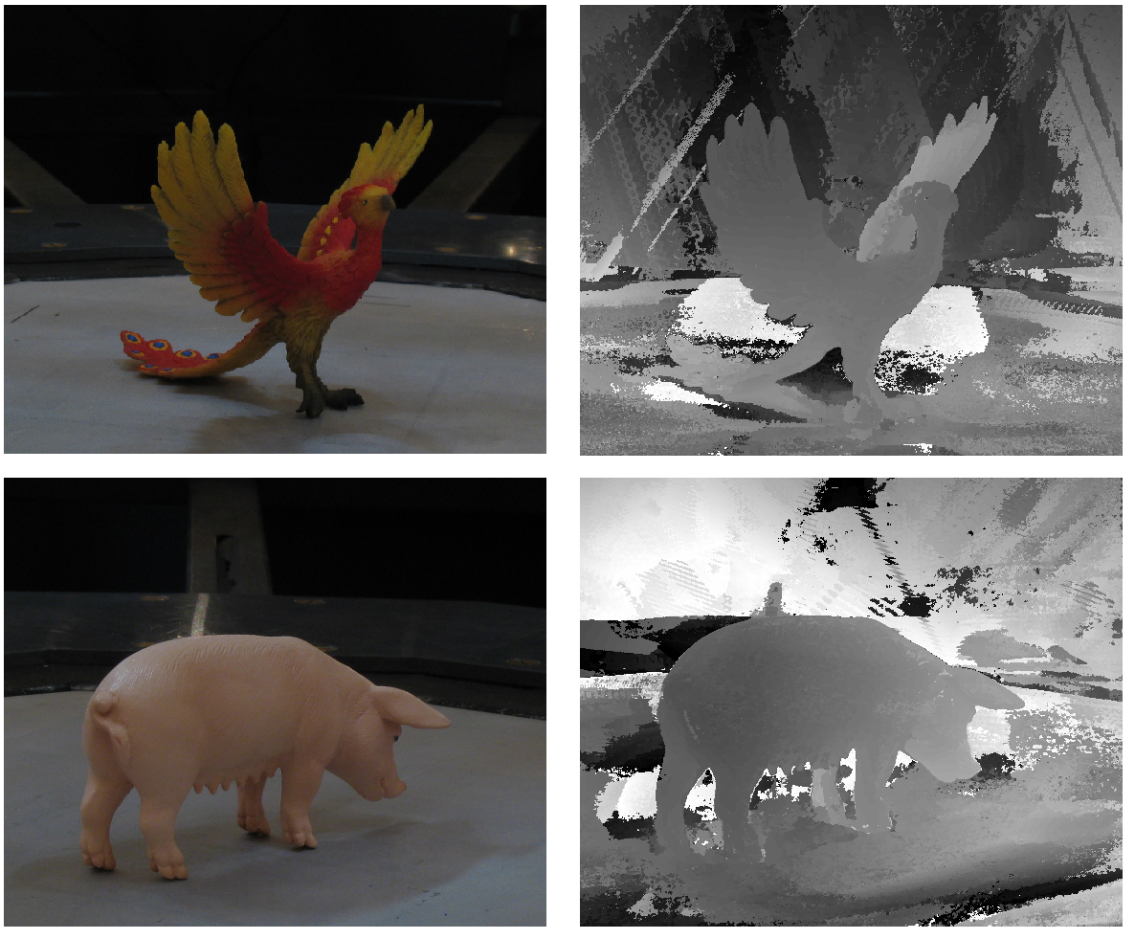

Figure 3.13: Examples of Plane Sweep Results. Left: Original Images. Right: Depth Maps

are discretized and instead of averaging the values at each scene location, a histogram containing all discretized values is computed for each voxel. Histogram bins corresponding to negative values indicate object probabilities, while positive values indicate free space. In Section 2.5.2.1, a probabilistic approach for fusing color probabilities from different views to a common 3D model was introduced. Here, the data term in the variational formulation was characterized by the term

$$f(x) := \log \frac{P_{bck}(x)}{P_{obj}(x)} \tag{3.7}$$

with

$$P_{obj}(x) = \sqrt[n]{\prod_{i=1}^{n} P(I_i(\pi_i(x)) \mid x \in R_{obj})} \tag{3.8}$$

$$P_{bck}(x) = 1 - \sqrt[n]{\prod_{i=1}^{n} [1 - P(I_i(\pi_i(x)) \mid x \in R_{obj})]} \tag{3.9}$$

The values P_{obj} and P_{bck} are the probabilities that a voxel is part of the object or free space, resp., based on the colors of its projections into the camera views. The single-view probabilities under the square root are computed directly from the random forest.

As it is the case for the TSDF values, negative values for f indicate object probabilities, while positive values indicate free space. Therefore, the color term f can be incorporated into the histogram representation. This is done by adding an extra bin at location f with bin count β. The value for β controls the impact of color information compared to depth information. If β is zero, the extra bin does not influence to probability distribution represented by the histogram. However, if β is very large, the opposite is the case, and the TSDF values do not have significant impact. As mentioned in Section 2.5.3, the primal-dual algorithm (see Section 2.7.2.1) is used to solve the resulting optimization problem. The application of the algorithm to this specific problem is explained in the following section.

3.6 Convex Optimization

The resulting optimization problem is very similar to the one introduced and solved in [5], where depth maps are fused in a robust way using discretized truncated signed distance functions, as suggested in [60]. The only modification in this work is the introduction of the additional histogram bin accounting for the probabilities based on the color distributions, as explained in Section 3.5. The energy minimization problem when fusing depth maps without adding color probabilities, as introduced in [5], is formulated as

$$\min_u \left\{ \int_\Omega |\nabla u| + \lambda \int_\Omega \sum_{i=1}^{N} h(x,i)|u(x) - d_i| dx \right\} \tag{3.10}$$

The i^{th} histogram bin at voxel x is denoted by $h(x,i)$, and the corresponding discretized TSDF value by d_i. The surface of the object is represented by the zero level set of u. Negative values of u indicate solid voxels, while positive values indicate free space. Introducing the additional histogram bin accounting for the color probabilities leads to the following extension of the minimization problem:

$$\min_u \left\{ \int_\Omega |\nabla u| + \lambda \left(\int_\Omega \sum_{i=1}^{N} h(x,i)|u(x) - d_i| dx + \beta \int_\Omega |u - f| dx \right) \right\} \tag{3.11}$$

The impact of the color probabilities compared to the depth values is controlled by the factor β. The probabilities themselves are modeled by f, as defined in Equation (3.7).

In order to solve the minimization problem, the first-order primal-dual algorithm by Chambolle & Pock is used (see [61] for details). The primal-dual formulation of Equation (3.11) is given by

$$\min_u \max_{\|p\|_\infty \leq 1} \left\{ -\int_\Omega u \operatorname{div} p + \lambda \left(\int_\Omega \sum_{i=1}^{N} h(x,i)|u(x) - d_i| dx + \beta \int_\Omega |u - f| dx \right) \right\} \tag{3.12}$$

where $p : \Omega \to \mathbb{R}^3$ is the dual variable. The global optimum is found by performing alternating gradient descend steps in u and gradient ascend steps in p, which are given by

$$\begin{cases} u^{n+1} = \operatorname{prox}_{\text{hist}}(u^n - \tau(-\operatorname{div} p^n)) \\ p^{n+1} = \operatorname{prox}_{\|p\|_\infty \leq 1}(p^n + \sigma \nabla(2u^{n+1} - u^n)) \end{cases} \tag{3.13}$$

In order to assure convergence, the time steps τ and σ are chosen to fulfill the criterion $\tau\sigma\|\text{div}\|^2 < 1$. For the dual variable p, the proximal operator reduces to a projection of the form

$$p = \operatorname{prox}_{\|\tilde{p}\|_\infty \leq 1} \quad \Leftrightarrow \quad p_{ijk} = \frac{\tilde{p}_{ijk}}{\max\{1, |\tilde{p}_{ijk}|\}} \tag{3.14}$$

The proximal operator for the primal variable u is computed as the solution of the optimization problem

$$\operatorname{prox}_{\text{hist}}(\tilde{u}(x)) = \arg\min_u \left\{ \frac{\|u - \tilde{u}(x)\|^2}{2\tau} + \lambda \left(\sum_{i=1}^{N} h(x,i)|u - d_i| + \beta|u - f| \right) \right\} \tag{3.15}$$

The solution is given as ([5])

$$\operatorname{prox}_{\text{hist}}(\tilde{u}) = \text{median}\{d_1, \ldots, d_N, f, p_0, \ldots, p_{N+1}\} \tag{3.16}$$

where d_i are the distances related to the according histogram bin i and f models the color probabilities, as described above. Let v denote the updated histogram after inserting the additional bin at location f with height β. The values for p_i in Equation (3.16) are then computed from the $N + 1$ histogram bins of v as

$$p_i = \tilde{u} + \tau\lambda W_i \ , \quad W_i = -\sum_{j=1}^{i} v(x,j) + \sum_{j=i+1}^{N+1} v(x,j) \tag{3.17}$$

Again, see [5] for details.

The ∇-operator as well as the div-operator need to be discretized for the usage in a three-dimensional voxel grid. This grid of size $M \times N \times K$ is defined as

$$\{(i,j,k) : 1 \leq i \leq M, 1 \leq j \leq N, 1 \leq k \leq K\} \tag{3.18}$$

Let $X = \mathbb{R}^{MNK}$ and $Y = \mathbb{R}^{MNK} \times \mathbb{R}^{MNK} \times \mathbb{R}^{MNK} = \mathbb{R}^{3MNK}$ be finite dimensional vector spaces. The differential operator $\nabla : X \to Y$ is then defined using finite differences and Neumann boundary conditions as

$$(\nabla v)_{i,j,k} = [(\delta_x^+ v)_{i,j,k},\ (\delta_y^+ v)_{i,j,k},\ (\delta_z^+ v)_{i,j,k}]^\top \tag{3.19}$$

where

$$\begin{aligned}
(\delta_x^+ v)_{i,j,k} &= \begin{cases} v_{i+1,j,k} - v_{i,j,k} & \text{if } i < M \\ 0, & \text{if } i = M \end{cases} \\
(\delta_y^+ v)_{i,j,k} &= \begin{cases} v_{i,j+1,k} - v_{i,j,k} & \text{if } j < N \\ 0, & \text{if } j = N \end{cases} \\
(\delta_z^+ v)_{i,j,k} &= \begin{cases} v_{i,j,k+1} - v_{i,j,k} & \text{if } k < K \\ 0, & \text{if } k = K \end{cases}
\end{aligned} \tag{3.20}$$

While the differential operator is defined in terms of forward differences, backward differences are used for the div-operator. The discrete version $\text{div} : Y \to X$ is defined as

$$(\text{div } \mathbf{p})_{i,j,k} = (\delta_x^- p^x)_{i,j,k} + (\delta_y^- p^y)_{i,j,k} + (\delta_z^- p^z)_{i,j,k} \tag{3.21}$$

with

$$
\begin{aligned}
(\delta_x^- p^x)_{i,j,k} &= \begin{cases} 0 & \text{if } i = 0 \\ p^x_{i,j,k} - p^x_{i-1,j,k} & \text{if } 0 < i < M \\ -p^x_{i-1,j,k} & \text{if } i = M \end{cases} \\
(\delta_y^- p^y)_{i,j,k} &= \begin{cases} 0 & \text{if } j = 0 \\ p^y_{i,j,k} - p^y_{i,j-1,k} & \text{if } 0 < j < N \\ -p^y_{i,j-1,k} & \text{if } j = N \end{cases} \\
(\delta_z^- p^z)_{i,j,k} &= \begin{cases} 0 & \text{if } k = 0 \\ p^z_{i,j,k} - p^z_{i,j,k-1} & \text{if } 0 < k < K \\ -p^z_{i,j,k-1} & \text{if } k = K \end{cases}
\end{aligned}
\tag{3.22}
$$

The parameter λ in Equation (3.11) controls the balance between data fidelity and smoothness. Since the value of the histogram bins and therfore the optimal value for λ depends on the number of input images, normalization by dividing through the number of views is performed, as

$$\lambda = \frac{\lambda'}{N} \tag{3.23}$$

where N is the number of images. In most experiments, a value of λ' between 1 and 10 lead to good results. Note that a lower value makes the surface smoother and more compact, but might also cause certain features to be eliminated. Especially when the random forest as well as the depth maps do not give clear results, long and thin objects may vanish due to the high cost of their surface compared with the relatively low data fidelity.

The parameter β also depends on the number of views. The color probabilities dominate the optimization algorithm if the according histogram bin is higher than the ones from the depth maps. The parameter can e.g. be set to between 0.1 and 1 times the number of views, depending on how much weight one wants to put on the color probabilities.

3.7 Post-Processing of the 3D Model

After the 3D model of the object of interest is reconstructed, an additional post-processing step is necessary in some cases. This is because multiple objects can be present. In contrast, the goal of the algorithm is to reconstruct only one object. If, however, multiple similar objects are present in the scene, these will probably be reconstructed as well. This

can be prevented by defining a tight bounding box in 3D space around the object, but the coordinates might not always be known in advance.

The following algorithm tries to eliminate any unwanted additional solid objects in the reconstructed scene. First, a seed point needs to be defined, which is part of the object of interest. This could be calculated based on projection of the placed strokes to 3D space, but this method does not necessarily give correct seed points. An easier method, which was used in this work, is to assume the midpoint of the 3D bounding box as an initial estimate. If this point is not part of the reconstructed object, it's neighboring voxels are checked, and again their neighbors, if none of them is marked as solid. Therefore, with increasing distances from the midpoint, the object is searched for. Once it is found, a filling algorithm is used to subsequently mark all connected voxels. It basically works by fixing the z-coordinate and solving the 2D filling problem for that slice. Then, the z-coordinate is increased by one. For every x- and y-coordinate which was filled in the previous slice, the new slice is filled using this voxel as seed point. After two iterations in both directions, the 3D object has successfully been filled. In the end, only voxels which have been marked are kept, yielding one final object only. This algorithm is computationally quite expensive, such that it might be left out in certain applications, where multiple objects do not pose a problem.

Another problem that can occur in some cases is the possible presence of holes inside the solid objects due to falsely marking these voxels as background during regularization. This problem can also be counteracted by using the same 3D filling algorithm as described above. This time, however, and inverse version is used, which fills everything from the outside which belongs to the background. Then, everything that has not been filled is part of the final object.

3.8 Visualization

After the 3D model has successfully been computed, the question arises how to represent and store this model. The algorithms in this work use a volumetric approach, meaning that the 3D object is described by a set of voxels. In practice, polygonal file formats are often preferred instead. Several algorithms for converting voxel grids to polygon meshes have been proposed in the literature. In this work, a standalone implementation of the well-known Marching Cubes algorithm was used*. For a detailed explanation of the algorithm see [68].

*http://www.paulbourke.net/geometry/polygonise

The tool itself uses a simple ray casting algorithm to render the resulting voxel grid in one of the input images. It simply computes viewing rays from the camera center through each of the image pixels, follows them until the object is hit and approximately determines the angle of impact. The intensity of the pixel is then proportional to this angle, with a maximum if the viewing ray is perpendicular to the surface.

Chapter 4

Parallel Implementation on GPU

Contents

Outline: *Most of the algorithms used in this work are designed such that they can be parallelized. Since a volumetric representation is used, certain computations can be executed independently for each voxel, and therefore in parallel. This usually leads to a large performance gain compared to a CPU-based implementation. The computer's graphics processing unit (GPU) is optimized for parallel computations, which is why these algorithms are outsourced to the GPU. Section 4.1 describes the hardware architecture of GPUs, together with a brief history of the modern graphics processor. In Section 4.2, Nvidia's parallel computing platform and programming model CUDA is introduced, which is used to implement the GPU-supported algorithms in this work.*

4.1 Hardware

In many image processing tasks, as well as in this work, the computations are highly parallelizable, since each pixel or voxel, resp., undergoes the same operations. Graphics hardware is optimized to render highly complex 3D scenes in real-time. Therefore, high throughput and parallelism are crucial. Compared to Central Processing Units (CPUs), which are optimized for serial programs running only on up to a few cores, Graphics Processing Units (GPUs) contain thousands of smaller, more efficient cores designed for parallel performance. Transistors on a GPU are devoted to data processing rather than flow

control and data caching. Historically, the GPU's computational powers where optimized for efficient scene rendering only. The *graphics pipeline* describes the stages necessary for processing information about a 3D scene with the result of obtaining the projection of the scene as a 2D image. These stages used to be fixed in the early years of GPUs, it was not possible to use the pipeline for general purpose tasks. Later, parts of the graphics pipeline became programmable by the user. In order to access the graphics pipeline, special APIs such as OpenGL or Direct3D are used. Figure 4.1 shows a typical programmable graphics pipeline. At this stage, special knowledge about shaders and GPU architecture was required in order to use the GPU for general purpose computations. Later, languages such as CG or GLSL were developed to facilitate accessing the graphics card's computational capabilities. Recently, several languages have been developed to enable writing general purpose parallel code for GPUs without requiring detailed knowledge about the graphics card's architecture. Examples are Brook, OpenCL or Nvidia's CUDA, with the latter one being used in this work and therefore described in the subsequent section. Latest models of graphics cards use as so-called *unified shader architecture*, where all programmable units (i.e. the shaders) are combined to one general purpose processing unit.

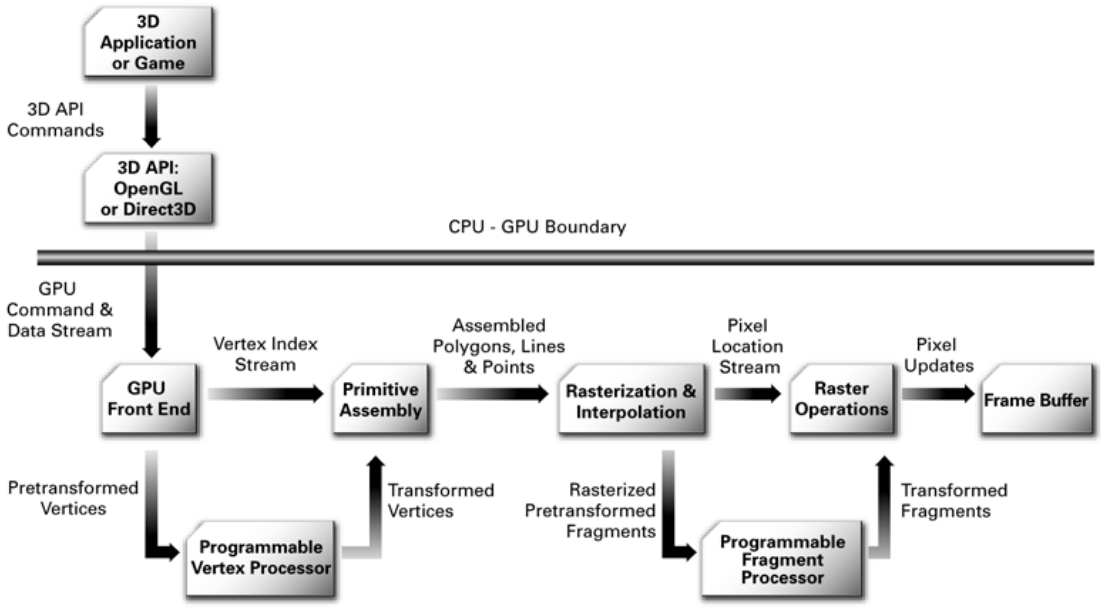

Figure 4.1: A Standard Programmable Graphics Pipeline. Image taken from `http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html`

4.2 Nvidia's CUDA

The GPU's tremendous power in parallel processing became increasingly interesting for applications in science. However, until a few years ago it was still necessary for programmers to have knowledge about OpenGL or Direct3D in order to use this power efficiently. To overcome this limitation, Nvidia released the CUDA programming model together with the GeForce 8 Series of graphics cards. CUDA makes it possible for developers to use the virtual instruction set and memory of a GPU with a comparable ease as programming a CPU. It is accessible via libraries (such as cuBLAS, cuFFT, ...), compiler directives as well as extensions to standard programming languages. In this work, CUDA C/C++ is used, which can be compiled using Nvidia's compiler *nvcc*. It should be noted that CUDA only works with Nvidia GPUs from the GeForce 8 series onwards, other cards are not supported.

When writing CUDA Code, the user can launch *kernels* from the CPU, which execute a portion of code on up to several thousands of threads on the GPU. The same piece of code is executed in each thread, but on different data, e.g. different image pixels. These threads are organized in blocks, which can have up to three dimensions, where each thread in a block is running on the same physical processor core. The blocks themselves are organized in grids, which can also be up to three dimensional. The user has no control over the order of execution of the different threads, but different data can be accessed using the position of a thread within a block. In a simple example, the position of a thread in a two-dimensional block can correspond to the position of a pixel in an image. Therefore, the thread can access the pixel by using it's own coordinates within a block, which is available at runtime. Arithmetically expensive operations on independent data elements usually yield the highest increase of performance compared with a serial implementation on a CPU, since data transfer between the Host and the Device is quite slow, as well as reads and writes from and to global memory on the GPU. Figure 4.2 illustrates the CUDA programming model as well as the memory architecture.

First, data is usually transferred from the Host (CPU) to device memory, also called global memory. Data in global memory is available from within any thread, but memory operations are slow. Each block of threads has a smaller amount of shared memory, which is much faster than global memory and can only be accessed from threads within the same block. In addition, each thread has a number of very fast registers, where for example local intermediate variables computed by the kernel are stored. Much slower local memory is also available on a per-thread basis, which is for example used to store local arrays of

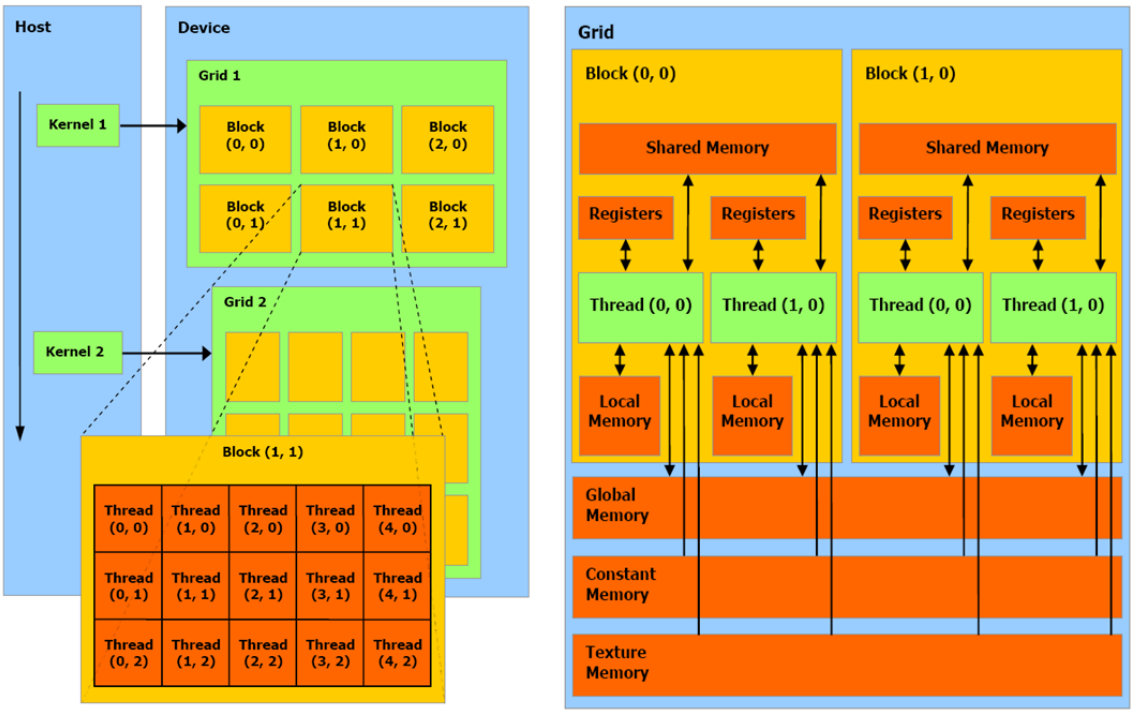

Figure 4.2: The CUDA Memory Model. Division into Grids, Blocks and Threads. Image taken from `http://www2.engr.arizona.edu/~yangsong/gpu.htm`

data and is about as slow as global memory. Furthermore, there is a portion of read-only constant memory, which is accessible by every thread and much faster than global memory. Finally, texture memory can also be used as a faster alternative to global memory, since it is available to all threads. However, it is read-only, while global memory can also be written.

In order to achieve a high performance gain compared to a serial implementation on the CPU, the code has to be designed such that a maximum number of threads can run in parallel. This number is limited by several different factors. Based on the so-called *compute capability* of a graphics card, different limits for the amount of registers and the shared memory per block exist. Kernels should therefore be designed to be as memory efficient as possible, in order to allow a lot of threads to run in parallel without reaching the memory limits. It is up to the user how to design the grid of blocks and threads, but there might be major implications on the performance. Since threads are physically executed in *warps* of typically 16 or 32 threads on a multiprocessor, the total number of threads in a block should be a multiple of this number. Also, memory acquisition should be organized in a way to allow simultaneously requested data to be accessed coalesced.

This can be achieved if consecutive threads access consecutive regions of memory. Another important consideration is to make threads as independent from each other as possible, since thread synchronization is expensive. Shared memory should be used if all threads in a block use the same data, since it is much faster than global memory. Copying data from global to shared memory also takes some amount of time, but this is often negligible compared to the performance increase using shared memory. However, newer graphics cards can cache global memory, which makes using shared memory a less crucial factor compared to older graphics cards, where global memory is not cached and therefore very slow.

Chapter 5

Results

Contents

Outline: *In this section, the performance of the 3D segmentation tool is analyzed as well as its components, i.e. the random forest output probabilities and the quality of the depth maps. First, the evaluation setup is briefly explained. In the subsequent section, some general quantitative measures are introduced, which allow a performance analysis when a ground-truth model is present. Then, the data sets used for evaluation are explained, with special emphasis on how they differ from each other and why they are chosen to be representative for showing the performance. As in most segmentation algorithms, visual analysis of the results is important, since weaknesses of the algorithms can sometimes better be detected than by analyzing quantitative measures. Therefore, visual interpretation will be discussed as well as quantitative results.*

5.1 Evaluation Setup

In order to allow for a meaningful comparison of results obtained by using different parameters, the same circumstances need to be given each time. It is hardly possible for the user to place exactly the same scribbles on the same views each time - especially since with every execution of the tool four randomly chosen input images are presented, which are most probably different each time. Therefore, for evaluation purposes, the color values which were marked by the user are saved to a file, which is loaded for subsequent runs with different parameters.

While visual results can be analyzed by either visualizing the voxel grid itself or its rendered projection onto the input images, a ground-truth model needs to be present for quantitative results in terms of error rates. For some of the data sets used in this work, a ground-truth is given in image space, i.e. the silhouette of the object in each of the views. Figure 5.1 shows an example of an input image together with its silhouette. Since shape from silhouette algorithms only recover the visual hull of the 3D object, a variety of views from different directions needs to be given in order to accurately reconstruct the object. This is the case in these data sets, where typically dozens of views are available. The ground-truth model was simply obtained by running the algorithm on the silhouette images without including depth information. Since there is no ambiguity in the object- and background probabilities, regularization was also not included.

Figure 5.1: Input Image and corresponding Silhouette

The silhouette images also make quantitative evaluation of the random forest possible, since it can be tested on the pixels of each input view and compared with the ground-truth, i.e. the silhouettes.

However, there was no ground-truth available for depth maps, which is why they were mostly analyzed visually and by fusing them without including additional information from the color distributions.

5.2 Error Measures

Three different error measures have been used in this work, which capture different aspects of the segmentation quality. In the following definitions, X denotes the resulting binary segmentation, while Y is the ground-truth model.

The *hit rate* measures the fraction of correctly classified object voxels:

$$HR(X,Y) = \frac{|X \cap Y|}{|Y|} \tag{5.1}$$

The *false alarm rate* measures the amount of voxels incorrectly classified as belonging to the object:

$$FAR(X,Y) = \frac{|X \backslash Y|}{|X|} \tag{5.2}$$

Finally, the *dice similarity coefficient* measures the mutual overlap between the segmentation result and the ground-truth:

$$DSC(X,Y) = \frac{2\,|X \cap Y|}{|X| + |Y|} \tag{5.3}$$

5.3 Data Sets

The segmentation tool was evaluated mainly on data sets from the Internet. The algorithms have been tested extensively on the data set provided by the Computer Vision Group at the University of Technology in Munich ([18])*, as well as the well-known Middlebury Multiview Dataset ([15])†. Each data set from TU Munich consists of typically 20-40 views of an object, together with silhouette images and full camera calibration parameters. The Middlebury Dataset consists of two different scenes, each captured from over 300 viewpoints in the hemisphere above the object. The different objects from these data sets pose different challenges to the segmentation tool. While some of them have

*vision.in.tum.de/data/datasets/3dreconstruction
†http://vision.middlebury.edu/mview/data/

a uniformly colored surface, others consist of a multitude of different colors. Also, some of them contain detailed and fine structures. Additionally, in several data sets the background colors are similar to some of the colors in the object, which are very challenging circumstances. Self-shadowing also poses a major challenge to the color segmentation algorithm. Figure 5.2 shows sample images of different data sets from TU Munich, while Figure 5.3 shows samples of the Middlebury data set. The number of views and the image resolution are shown in Table 5.1.

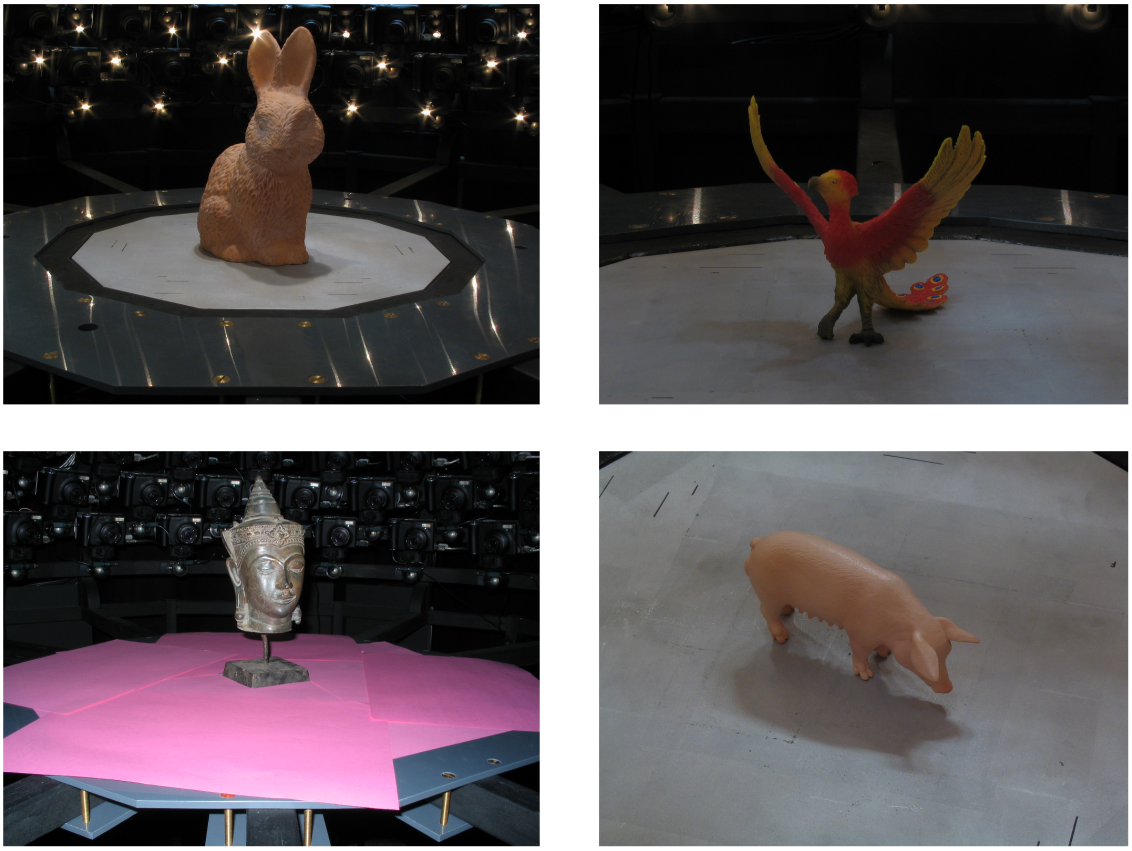

Figure 5.2: Sample Images from TU Munich Data Sets

Also, a natural scene has been captured to evaluate the performance. However, no ground-truth is present, which makes only visual evaluation possible. Figure 5.4 shows an example image of the real-world data set used in this work.


Figure 5.3: Sample Images from Middlebury Data Sets

Data Set	No. of Images	Resolution
Pig	27	1024x768
Bunny	36	1024x768
Bird	21	1024x768
Head	33	1024x768
Beethoven	33	1024x768
Dino Munich	36	720x576
Temple	312	640x480
Dino Middlebury	363	640x480

Table 5.1: Number of Images and Resolution of TU Munich and Middlebury Data Sets

Figure 5.4: Sample Image from Real-World Data Set

5.4 Visual Results

As in most computer vision problems, visual interpretation of the results is crucial and can often give more insight into the strengths and weaknesses of an algorithm than quantitative measures. For example, the segmentation and reconstruction algorithm's behavior at discontinuities and in regions with fine details can best be evaluated by interpreting visual results. This section gives some selected typical and meaningful outcomes. First, the two core modules, i.e. the random forest regressor and the computation of depth maps, are analyzed. Then, the resulting 3D models are shown for some selected data sets and typical user inputs. All parameters have been chosen appropriately - a detailed discussion on the influence of the single parameters is given in the respective sections in Chapter 3. The resolution in voxel space was set to about 7 million voxels in a bounding box around the object of interest.

5.4.1 Random Forest Classification

Different aspects of the random forest regression model were already explained extensively in Section 3.3. Also, sample results discussing the effect of different parameters, the user input and the occurrence of similar colors in both regions are given in that section.

If the parameters are chosen appropriately, the random forest yields very good results if the color distributions permit. This means that even the best regression model cannot distinguish object and background in images if both regions have very similar color distributions. In the following, three sample results are given to show this fact. Note that, again, the gray value represents the probability that a pixel belongs to the object, i.e. white indicates object certainty, while black represents pixels that have maximal background probability. Next to the figure illustrating these probabilities, the original image with the user input is shown.

In the experiment illustrated in Figure 5.5, the results are very satisfying, since the object differs strongly from the background. In Figure 5.6, the results are not as clear, since deviations in the brightness are present, which are not sufficiently covered by the user input. Note how the shadow of the pig is marked by the user, but the similar bottom region of the pig is not. This is why parts of the model have high background probability. However, the results are still significantly better than if RGB values are used and brightness is not treated independently.

Finally, Figure 5.7 shows an example where the random forest fails to compute meaningful probability values is some of the regions. This is due to the user input, which does

not sufficiently cover the different colors (see the claws of the bird), and can therefore not be interpreted as a weakness in the performance of the random forest .

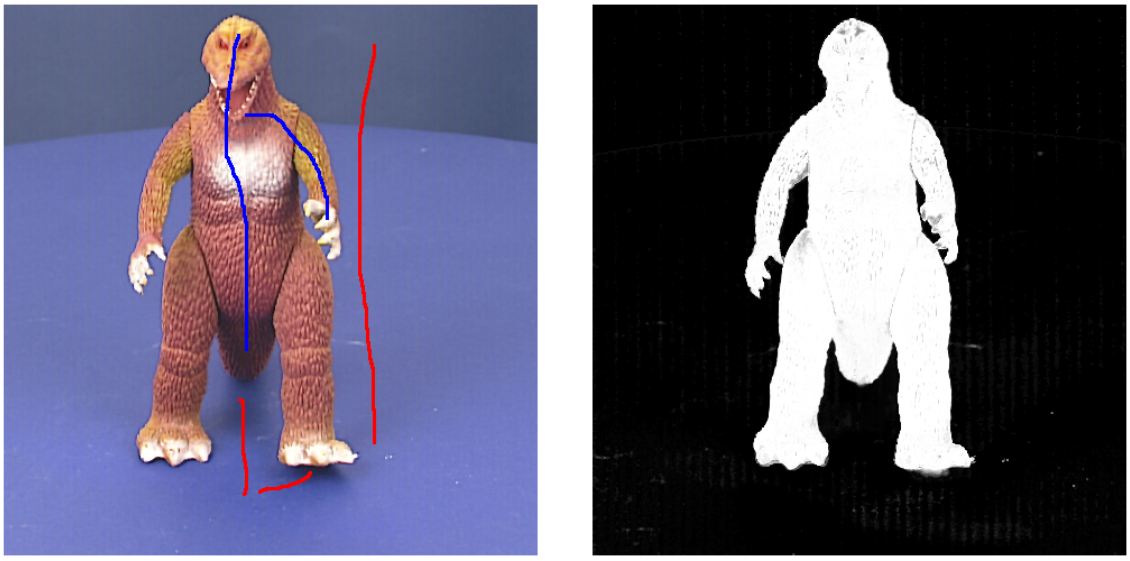

Figure 5.5: Satisfying Random Forest Result. Left: Original Image and User Input Right: Object Probabilities from Random Forest

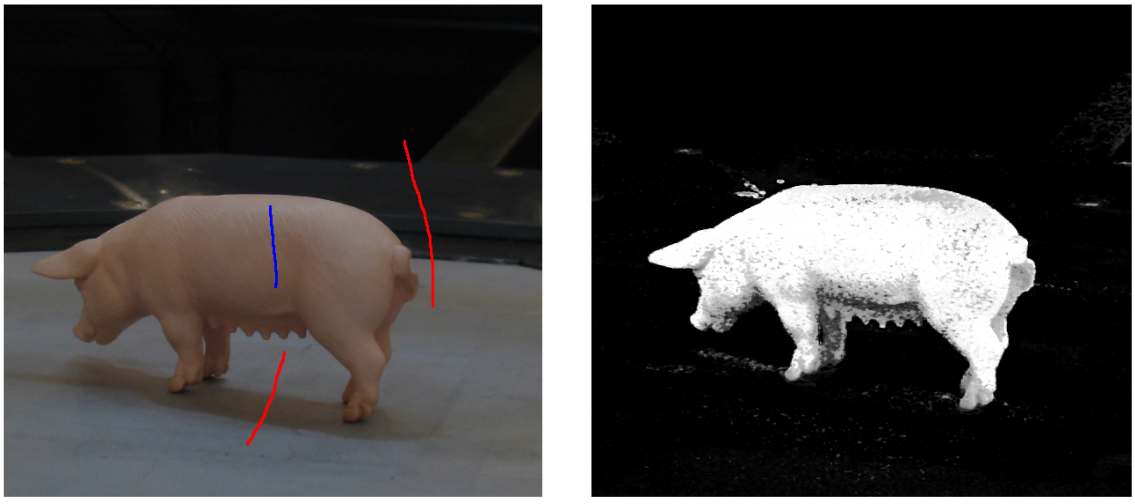

Figure 5.6: Random Forest works as desired, but User Input suboptimal. Left: Original Image and User Input Right: Object Probabilities from Random Forest

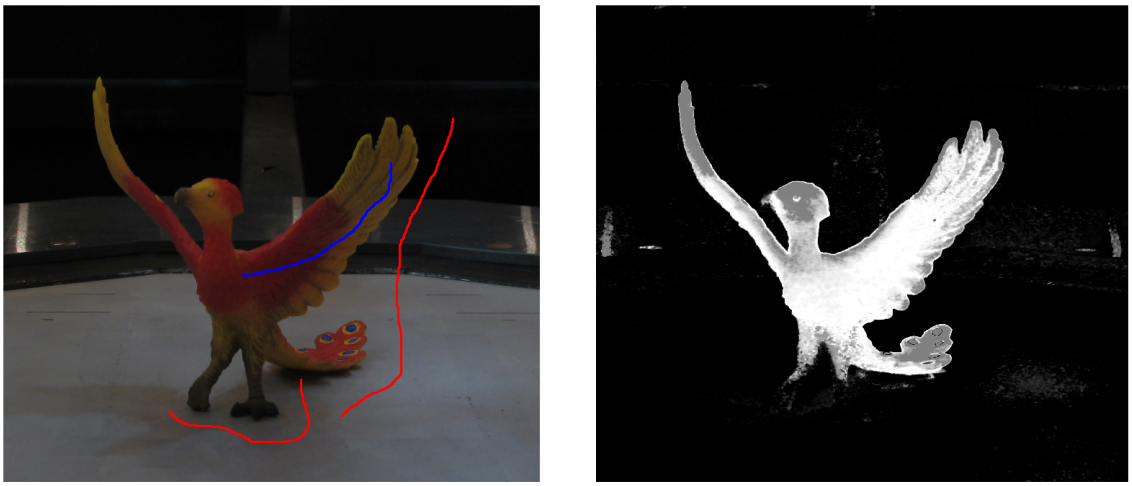

Figure 5.7: Random Forest fails to identify all Object Regions due to insufficient User Input. Left: Original Image and User Input Right: Object Probabilities from Random Forest

5.4.2 Depth Map Generation

When generating depth maps to aid the process of 3D scene segmentation, a compromise between quality and computational effort needs to be found. Since dense, reliable and outlier-free depth maps typically require a computationally intensive similarity measure and a subsequent optimization step, this approach turned out to be quite infeasible. Instead, a simpler similarity measure is used and the window size is kept reasonably small. Especially if many views are present, the additional redundancy from different depth maps is more valuable than only a few, but more reliable range images. It turned out that the quality of single depth maps is not crucial to a certain extent if enough views are present and if the fusion step is robust in order to filter out wrong depth values. Figure 5.8 shows some results for depth map estimation using the plane sweep algorithm. Recall that a set of planes is swept through space at discrete steps between a minimal and a maximal depth value, and the best plane hypotheses indicates the correct depth value. However, since the range is limited, objects in the far distance cannot be reconstructed and yield erroneous depth values. Also, in homogeneous regions the algorithm tends to fail.

The depth range was chosen such that a tight bounding box around the object of interest is covered. Therefore, the depth maps do not provide meaningful information in regions in front of or behind the object, as visible in Figure 5.8. This does usually not pose a problem, since corresponding erroneous depth values among different views are most likely inconsistent and therefore do not imply a high probability for any object

surface region. In contrast, in object regions the results are quite pleasing, considering the fact that a low-cost solution was implemented. For all data sets tested in this work, the depth maps yielded good reconstruction results after applying the robust fusion algorithm introduced in Section 2.5.3. In order to improve the performance and to reduce outliers, a median filter was applied to the final depth maps.

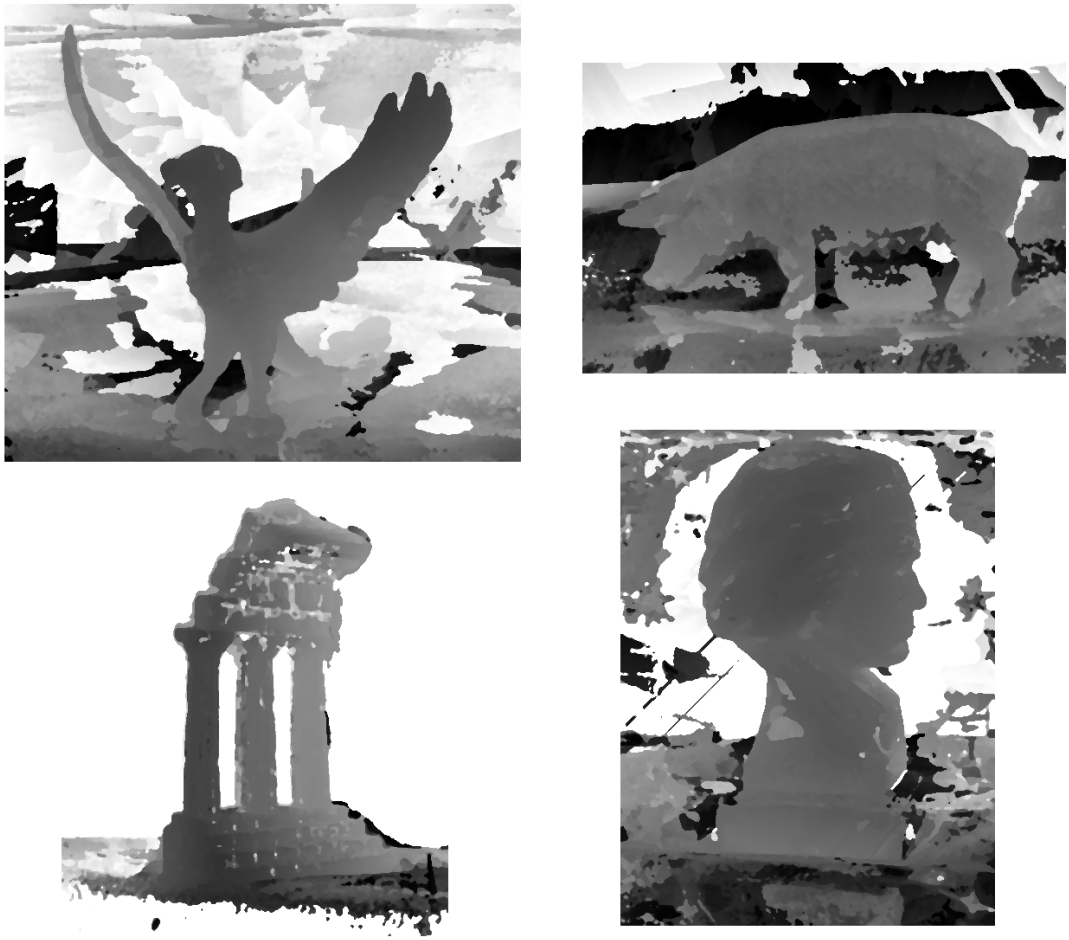

Figure 5.8: Examples of resulting Depth Maps

5.4.3 Depth Map Fusion

Most 3D reconstruction algorithms work by fusing depth maps from different views into a common 3D model. However, this does only tackle the reconstruction problem, but is not suitable by itself for object segmentation. Nonetheless, the performance of the segmentation tool can heavily depend on the quality of this fusion step, especially if the

color distributions do not suffice to distinguish the regions, or if the views do not cover the object from sufficiently many vantage points, such that the visual hull does not adequately model the object. This section give some results of depth map fusion, i.e. the reconstructed scenes. Note that, however, the scene is reconstructed as a whole, and from the resulting models itself it is not possible to distinguish the object of interest from the rest of the scene.

Figure 5.9 shows two sample results of the depth map fusion step. Note that the quality of the range images is limited by the fact that a sufficient number of them must be computed in the order of only a few seconds. Nonetheless, the results are quite accurate. However, as depicted in the two examples, parts of the scene that do not belong to the object of interest are reconstructed as well (the ground plane in this case). This is where color information needs to be incorporated in order to identify the object of interest.

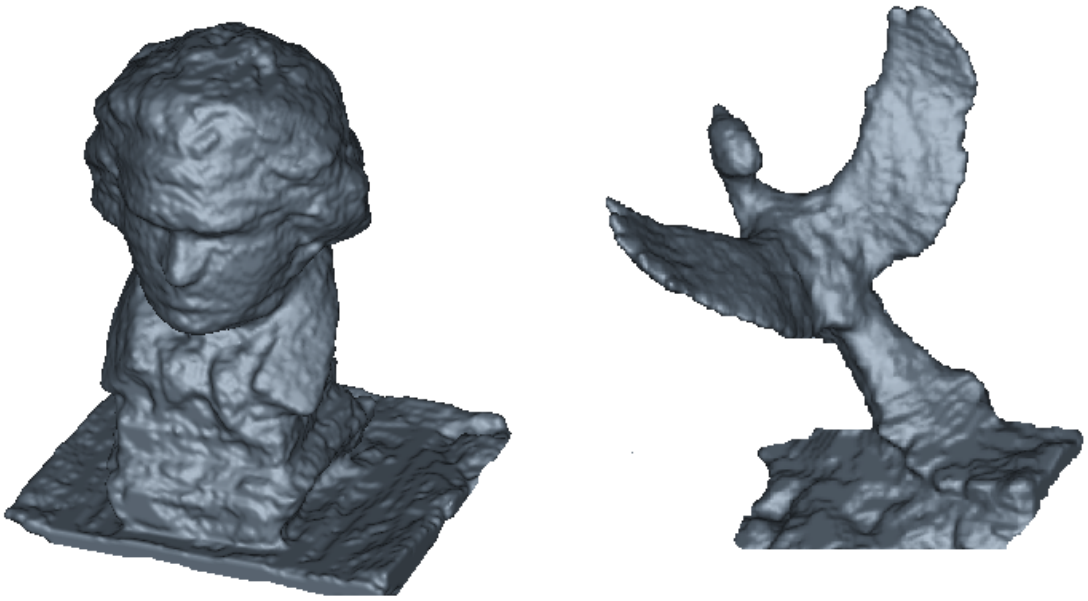

Figure 5.9: Sample Results of Depth Map Fusion (Beethoven and Bird Data Set). Object is accurately reconstructed, but also unwanted Parts of the Scene

5.4.4 3D Segmentation Results

Some typical examples of the resulting final 3D models are shown in this section. First, the results for the TU Munich data sets are discussed. Figure 5.10 shows very good results for the pig data set, which could be reproduced most of the time with different user inputs

and parameters. The tool is able to successfully handle difficulties such as the shadow and the varying brightness in different parts of the object.

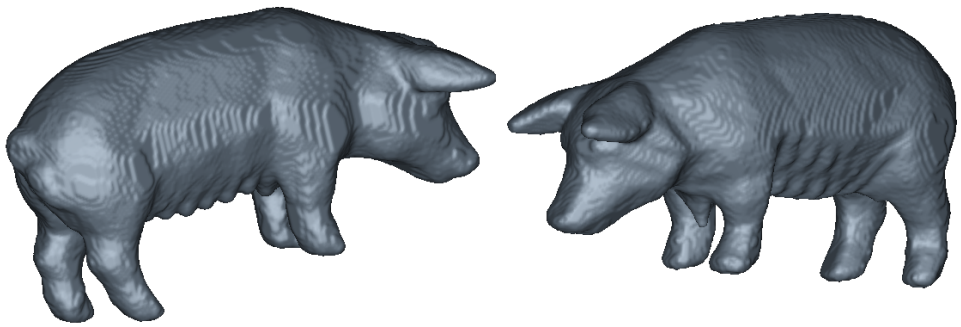

Figure 5.10: Resulting 3D Model (Pig)

Figure 5.11 shows a sample result of the bunny data set, which is very similar to the pig data set in terms of color values. However, the surface of the model has significant irregularities and self-shadows, which makes it a bit more challenging. However, the tool was able to reconstruct the model appropriately in almost all experiments.

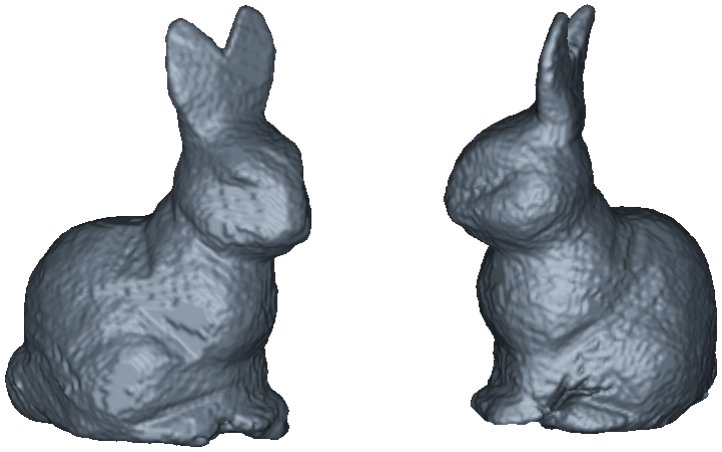

Figure 5.11: Resulting 3D Model (Bunny)

Probably the simplest model tested in this work is the dinosaur data set, which has very distinctive colors in object- and background. Low brightness at some parts of the body is the only mentionable difficulty, but did not affect the quality of the result in the vast majority of the experiments. A sample result is show in Figure 5.12.

As mentioned above, the main challenge with the bird data set is that very similar color values appear in both the object and the background. This is for example the case at

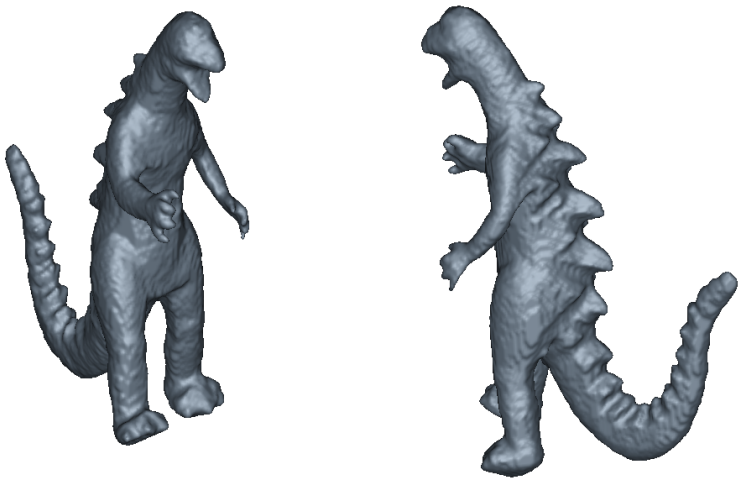

Figure 5.12: Resulting 3D Model (Dinosaur)

the claws and the beak of the animal, which are about as dark as the background. Since these are protruding parts of the model, the convex optimization step tends to discard them in order to achieve a smaller surface area, if the model is solely based on color probabilities. However, including depth information usually solves the problem, and the bird can successfully be reconstructed, as shown in Figure 5.13.

Figure 5.13: Resulting 3D Model (Bird)

The head- and Beethoven data sets (Figures 5.14 and 5.15, resp.) show examples where different gray levels pose a major challenge to the random forest based segmentation model, since color information is limited. Especially self-shadowing is present due to the surface-irregularities and the non-ambient light source. Figure 5.16 illustrates these challenging conditions by showing three different views. Note that some spots are very bright, while others can hardly be distinguished from the gray values of the ground plane.

In these experiments, applying a bilateral filter to smooth the image improved the results significantly, since small shadows and irregularities could be reduced. Additionally, care needs to be taken to sufficiently mark both dark- and bright parts of the object. Fusing depth maps to aid the process is crucial here, since the color models can hardly describe the scene by themselves.

When looking at the model from the head data set, one can see that there is a very thin connecting piece between the head an the platform it is mounted to. If one wants to reconstruct the whole object, the smoothness parameter in the optimization step needs to be quite low. This is because the surface area would decrease significantly by cutting the model off right underneath the head, yielding a smaller energy if the data fidelity term is weighing not strongly enough.

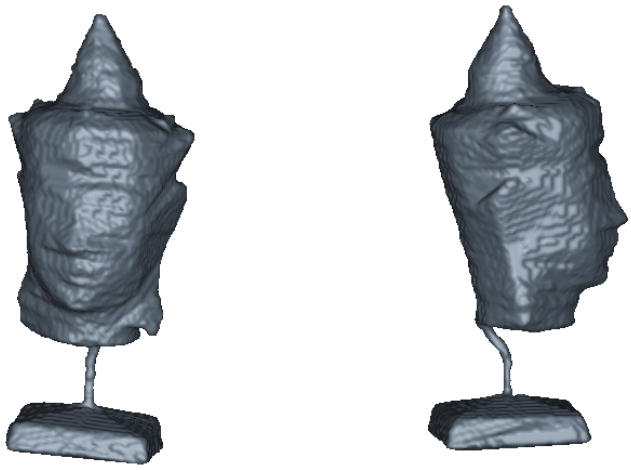

Figure 5.14: Resulting 3D Model (Head)

In contrast to the TU Munich data sets presented above, the Middlebury data sets consist of a higher number of views. This yields a higher redundancy, which is why one can expect good results. However, these data sets are not very suitable for 3D segmentation, since only one object is present, while every other part of each image is just dark background. Nonetheless, the performance of the tool is shown in terms of 3D reconstruction. Figure 5.17 shows a resulting model for the temple data set. The results are very pleasing, but care needs to be taken when labeling regions, since shadows at the surface of the temple need to be modeled as well. A sample result for the dinosaur data set is shown in Figure 5.18, which is of similar quality as the result for the temple. In both cases more weight was placed on the depth information than on the colors, since dark spots caused by shadows are a challenging problem for color segmentation.

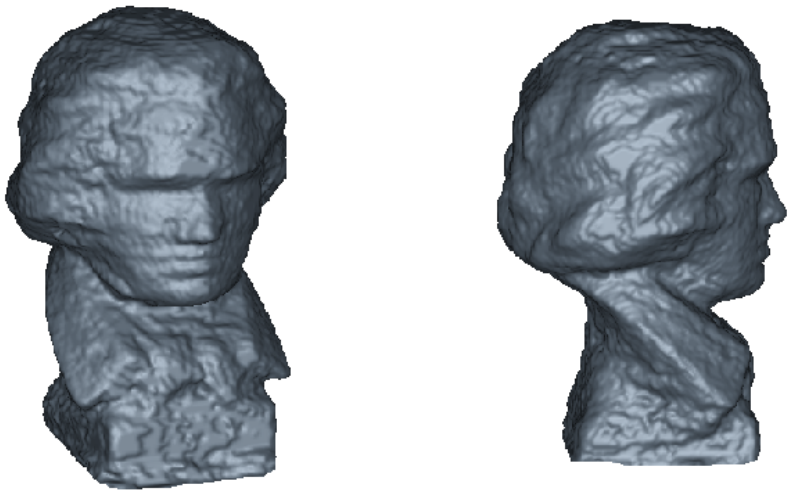

Figure 5.15: Resulting 3D Model (Beethoven)

Figure 5.16: Varying Brightness within Data Set (Beethoven)

5.5 Real World Example and Importance of including Depth Information

This section shows an example of applying the algorithms to a real world example, i.e. the clock tower shown in Figure 5.4. The problem with this data set is that only a few views are present, which is why the visual hull does not accurately model the object of interest, i.e. the clock tower. Hand-labeling the tower in all of the input images and fusing these silhouettes yields the result shown in the left image in Figure 5.19. If depth information is not included and therefore only color models are used to compute the 3D model, this is the result one would obtain for perfect segmentation and without using global optimization.

Figure 5.17: Sample Result for Temple Data Set

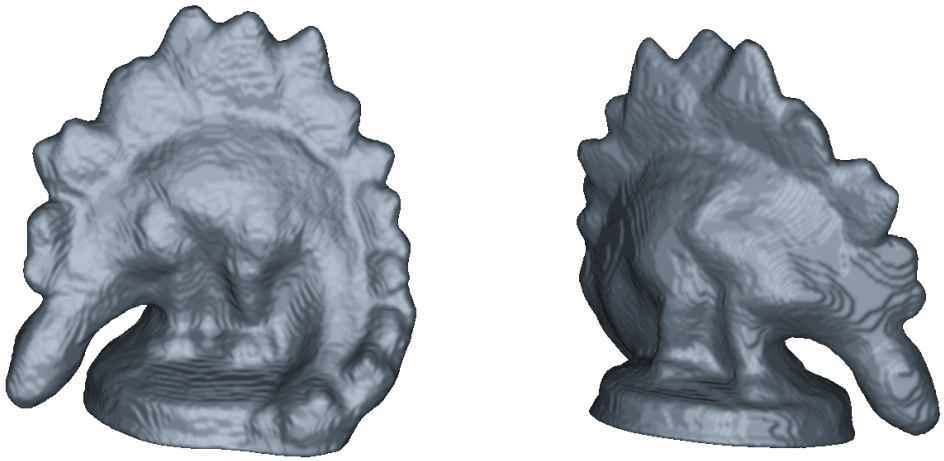

Figure 5.18: Sample Result for Dino Data Set

Including depth information, the actual shape of the clock tower can be recovered, as shown in the right figure, which shows the output of the segmentation tool. Here, voxels belonging to free space in front of the tower have been carved away, since the depth maps indicate the correct labeling. However, since color information does not recover the correct surface, the result heavily depends on the quality of the depth maps.

Figure 5.20 shows how the results improve when fusion color probabilities with depth information. In Figure (a), the result after performing depth map fusion is shown. Note

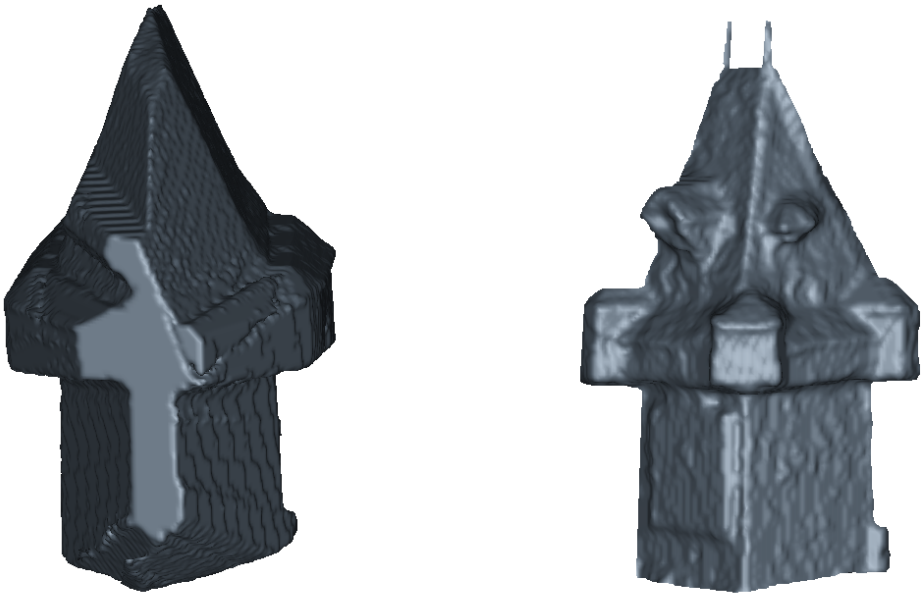

Figure 5.19: Sample Result for Clock Tower Real World Data Set. Left: Visual Hull from Silhouette Fusion. Right: Result after including Depth Information

how the scene is reconstructed as a whole, including the ground plane, which is not the desired result. Figure (b) shows the result if only color probabilities are used. Since there is only a limited number of views available, some parts of the object are not carved away sufficiently. The result after applying the proposed method is shown in Figure (c). Here, the ground plane is not part of the reconstruction, since its color values clearly indicate background. Also, the object itself is accurately modeled, since the depth maps make reconstruction of object concavities possible.

5.6 Quantitative Results

Due to the lack of data sets with ground truth models, performance evaluation in terms of quantitative measures is often not possible. Also, the user input influences the results, which is why the numbers presented in this section can only give a rough insight. The following procedure was followed in order to achieve meaningful results:

- As a ground truth model, the tool was applied to the binary segmentation masks that come with some of the data sets. The optimization step was omitted when generating the model, since the data fidelity is maximal in silhouette images

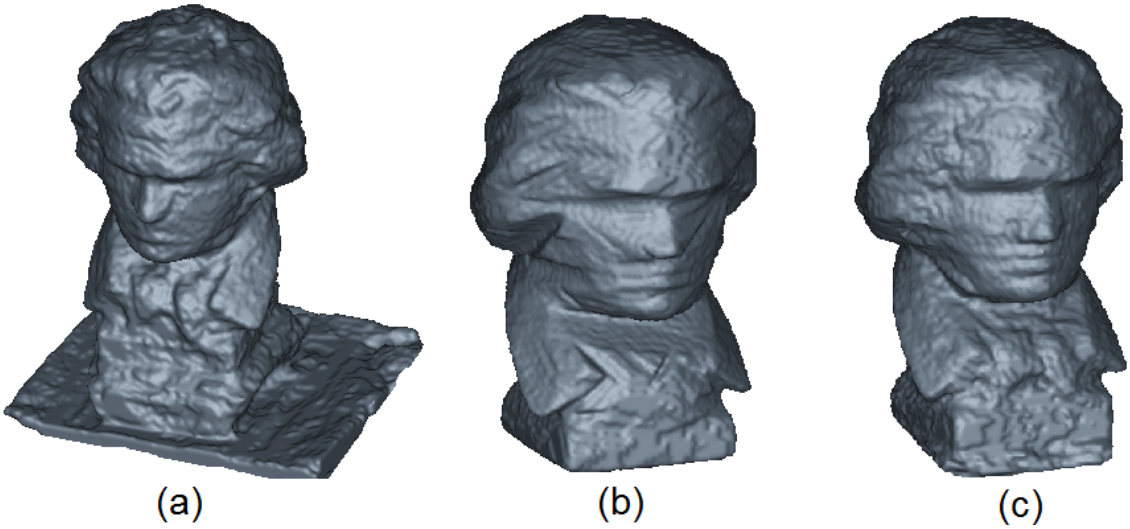

Figure 5.20: Comparison of Depth Map Fusion and Color-based Segmentation. (a) Depth Map Fusion Result (b) Visual Hull from Silhouette Fusion (c) Result using proposed Method

- All experiments were carried out at least 3-5 times, with results being averaged
- User Input was tried to be placed intuitively, but also intelligently
- Parameters and Voxel Resolution were kept unchanged throughout the different experiments

As performance measures, the *hit rate* (HR), the *false alarm rate* (FAR) and the *dice similarity coefficient* (DSC) were implemented, which are explained in Section 5.2.

Table 5.2 shows the resulting error values, averaged over several runs. Coinciding with the visual results, the pig and bunny data sets yield the best performance. The *dice similarity coefficient* of the bird data set is quite low, which is because of the missing body parts in the reconstructed model. However, also the *false alarm rate* is higher than in the other examples, indicating that this was the most challenging data set. What is conspicuous is the fact that the *hit rate* almost equals one in all of the examples. This indicates that almost all of the voxels being part of the real model (ground truth) are correctly classified as such. Thus, most errors occur when voxels in background regions are incorrectly classified as being part of the object.

Data Set	DSC	FAR	HR
Pig	0.96	0.09	1.00
Bunny	0.95	0.07	0.99
Bird	0.94	0.04	0.93
Head	0.94	0.05	0.98
Beethoven	0.96	0.05	0.98

Table 5.2: Quantitative Results - Different Error Measures

Data Set	DSC	DSC w/o Depth	FAR	FAR w/o Depth	HR	HR w/o Depth
Pig	0.96	0.96	0.09	0.09	1.00	1.00
Bunny	0.95	0.94	0.07	0.07	0.99	0.99
Bird	0.94	0.89	0.04	0.09	0.93	0.90
Head	0.94	0.92	0.05	0.09	0.98	0.95
Beethoven	0.96	0.92	0.05	0.8	0.98	0.96

Table 5.3: Comparison of Error Measures with and without using Depth Information

5.6.1 Effect of including Depth Information

As explained above, depth information turns out to be very useful and in some cases crucial for a meaningful reconstruction and segmentation of the desired object. Especially when similar colors appear in both background and object regions, or when not enough views are present, it is very important to fuse color probabilities with the probabilities coming from the depth maps. In the data sets with ground truth segmentations used in this section, these problems are not severe. Nonetheless, including depth information lead to improved results compared to the outcome when only color probabilities are used. Table 5.4 shows the DSC, FAR and HR values for the same setting as in Table 5.2, but also without including depth information. Throughout all experiments, not including depth maps into the probabilistic model decreased the quality of the results, or at least did not effect it in a positive way. Only in the pig and bunny data sets the improvement was negligible. This is because the random forests already performed very well, since the colors in object and background regions are quite distinctive.

5.6.2 Comparison with State of the Art

An important part of evaluating the performance of an algorithm is comparing the results with other state of the art methods. However, only very few 3D segmentation methods with quantitative evaluation of the results exist. Since this work is based on the method by Kolev et al. ([12]), their results are compared to the ones in this work. Since they

Data Set	This Work			Kolev et al.			Reinbacher et. al		
	DSC	FAR	HR	DSC	FAR	HR	DSC	FAR	HR
Pig	**0.96**	0.09	**1.00**	0.93	0.08	0.94	0.95	0.08	0.98
Bunny	0.95	0.07	0.99	0.95	0.09	0.99	0.98	**0.01**	0.99
Bird	0.94	0.04	**0.93**	0.90	0.08	0.89	0.95	**0.01**	0.93
Head	0.94	0.05	**0.98**	0.87	0.13	0.88	0.94	0.04	0.93
Beethoven	0.96	0.05	0.98	0.95	0.08	0.98	0.97	0.04	0.99

Table 5.4: Comparison of Performance with State of the Art Methods

use a simpler color model as well as no depth map computations, the results in this work should outperform theirs. Additionally, the results are compared to the ones obtained by Reinbacher et al. ([13]), who use backprojection of spatial constraints to obtain silhouettes. In their work they compare error rates with the ones from a re-implementation of the algorithm by Kolev et al. These values are used for comparison in this work.

The results show that the method by Kolev et al. performs worse in most experiments than the method presented in this work. This agrees with the fact that they use a simpler color model and no depth information. When comparing with the algorithm by Reinbacher et al., their values for the FAR could not be reached. However, in terms of the other two error measures, the performance is similar. For the bird data set, the tool produces higher error rates than the other two methods. This is probably because their algorithms do not carve away the claws of the bird, which have very similar color as the background.

5.7 Computational Time

As mentioned above, the parallel implementation on the graphics processing unit yields very low computational times, even for high voxel resolutions. The only parts being dependent of the resolution of the voxel grid are the assignment of probabilities, the subsequent optimization step and the post-processing algorithm. The computation time for training the random forest only depends on the forest parameters. Applying the model to obtain color probabilities depends on the number of unique color values in the set of input images. For each voxel, solely its projected color values need to be determined and the corresponding probabilities need to be read from the pre-computed results. This is very efficient, which makes the complex optimization step the much more expensive task in terms of computational time. Table 5.5 shows average computational times for different voxel resolutions, where MV stands for megavoxel. The experiments were performed on a

Nvidia GeForce GTX 660 graphics processing device and an Intel i5-3470 CPU. As input data, the 27 images of the pig data set at a resolution of 1024x768 pixels were used.

Even for very high resolutions in voxel space, the computational times are in the range of several hundred milliseconds. As shown in the table, the effort for post-processing is higher than for computing the initial 3D model, while often being a redundant step. Therefore, post-processing can be left out in many scenarios.

In all experiments, the random forest consisted of 100 trees with 50 possible split functions at each node. Increasing the number of trees, while also being unnecessary, did not significantly increase the computational time, since on modern GPUs several hundred or thousand threads can run in parallel. On the system used in this work, typical total computation times for training and testing the random forest were at around 250-400 ms.

The computation of depth maps is carried out while the user is placing scribbles to mark the regions. In around 5 seconds, depth maps for all of the input views could be computed successfully.

The rest of the system does not have significant influence in terms of execution time, only pre-filtering the images can take up to a few hundred milliseconds.

	1 MV	2 MV	5 MV	9 MV	15 MV
Assign Voxel Prob.	10 ms	15 ms	30 ms	65 ms	95 ms
Optimization Step	250 ms	330 ms	560 ms	850 ms	1300 ms
Post-Processing	70 ms	135 ms	350 ms	670 ms	1200 ms

Table 5.5: Computational Times depending on Voxel Resolution

Chapter 6

Conclusion

Contents

6.1 Summary

In this work, a tool for reconstructing a single object of interest from multiple views was developed. In order to learn probability models for the object- and background regions, as well as to determine which object to reconstruct, additional user input is required. The user is asked to draw strokes in one or more of the images, marking object and background. Based on the collected pixel values, a random forest regressor is trained as shown in Section 3.3, which is able to assign object and background probabilities to test color values. The classification is supported by the use of depth information, which is recovered using a multiview plane sweep algorithm, as described in Section 3.4. Using truncated signed distance functions, a 3-dimensional probability grid is computed, where the object surface is represented by the zero level set of the function. These probabilities are fused with color-based probabilities from the random forest, which are based on the projections onto the input views. This fusion step is described in Section 3.5. Section 3.6 shows how a total variation based regularizer is used to determine the most probable surface based on the resulting fused signed distance fields and a smoothness constraint. After some optional post-processing, which is only required in certain scenarios (see Section 3.7), a watertight, dense 3D model is finally obtained.

The core computations of the 3D segmentation tool are carried out on programmable

graphics hardware, exploiting its massive parallelism. Very low computation times could therefore be achieved, even for high voxel resolutions and many input views. In most scenarios, the segmentation tool yielded satisfying results, with reconstructions including a high level of detail. Chapter 5 shows sample results for different data sets. Only in cases where the object has very similar properties as the background, the algorithms sometimes failed to compute a proper 3D model. Also, the quality of the user input is an important factor. If the user fails to collect pixel values representing the basic color distributions, the random forest cannot train a meaningful model.

6.2 Outlook

While the 3D segmentation tool already works well in many scenarios, it can still be optimized in various ways. The implementation is structured into different building blocks, which can all be easily replaced by improved versions. Regression models different to the concept of random forests could for example be used. In this work, the only alternative tried were Gaussian mixture models. Support Vector Machines are an example of models that are still to be evaluated. A main point of possible improvements is the computation of depth maps. If high quality dense depth maps can be computed efficiently, the probabilistic model based on color values becomes a less critical factor for the performance of the overall system.

One of the drawbacks of the current implementation is that the resulting 3D model is presented to the user after finishing the interaction process. However, it would be desirable to show intermediate results after each scribble is placed, such that the user knows at every moment if the input is already sufficient or not. At the moment, the user can add scribbles or re-start the process after viewing the results, but live results during the interaction process would be of advantage. Currently, this feature is prevented by slightly too slow execution times for satisfyingly high voxel resolutions. In the current version of the tool, the random forest would need to be re-trained each time before showing an updated result. Using an incremental version would be helpful in order to achieve lower computational times. However, the development of more powerful GPUs will obviously also ease this task.

Bibliography

[1] A. Mohammad-Djafari, "Inverse problems in imaging and computer vision - from regularization theory to bayesian inference," in *International Conference on Computer Vision Theory and Applications*, 2010.

[2] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, 2006.

[3] P. Merrell, A. Akbarzadeh, L. Wang, J.-M. Frahm, and R. Y. D. Nister, "Real-time visibility-based fusion of depth maps," in *In International Conference on Computer Vision and Pattern Recognition*, 2007.

[4] C. Zach, M. Sormann, and K. Karner, "High-performance multi-view reconstruction," in *In International Symposium on 3D Data Processing, Visualization and Transmission*, 2006.

[5] G. Graber, T. Pock, and H. Bischof, "Online 3d reconstruction using convex optimization," in *International Conference on Computer Vision, Workshops*, 2011.

[6] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision*, vol. 38, 2000.

[7] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," in *International Journal of Computer Vision*, 1997.

[8] B. Baumgart, *Geometric Modeling for Computer Vision*. 1974.

[9] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, 1994.

[10] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images," in *In International Conference on Computer Vision*, vol. 1, 2001.

[11] C. Rother, V. Kolmogorov, and A. Blake, ""grabcut": interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics*, vol. 23, 2004.

[12] K. Kolev, T. Brox, and D. Cremers, "Fast joint estimation of silhouettes and dense 3d geometry from multiple images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[13] C. Reinbacher, M. RÃ$\frac{1}{4}$ther, and H. Bischof, "Fast variational multi-view segmentation through backprojection of spatial constraints," *Image and Vision Computing*, 2012.

[14] J. Santner, T. Pock, and H. Bischof, "Interactive multi-label segmentation," in *Proceedings of the 10th Asian Conference on Computer Vision - Volume Part I*, 2011.

[15] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, 2002.

[16] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[17] D. Comaniciu, P. Meer, and S. Member, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, 2002.

[18] K. Kolev, M. Klodt, T. Brox, S. Esedoglu, and D. Cremers, "Continuous global optimization in multiview 3d reconstruction," in *In International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2007.

[19] N. Cornelis and L. J. V. Gool, "Real-time connectivity constrained depth map computation using programmable graphics hardware.," in *Conference on Computer Vision and Pattern Recognition*, 2005.

[20] R. T. Collins, "A space-sweep approach to true multi-image matching.," in *Conference on Computer Vision and Pattern Recognition*, 1996.

[21] M. Okutomi and T. Kanade, "A multiple-baseline stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, 1993.

[22] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. 2004.

[23] L. B. Statistics and L. Breiman, "Random forests," in *Machine Learning*, 2001.

[24] A. Criminisi and J. Shotton, *Decision Forests for Computer Vision and Medical Image Analysis*. Advances in Computer Vision and Pattern Recognition Series, 2013.

[25] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *The Quarterly of Applied Mathematics*, vol. 2, 1944.

[26] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal on Applied Mathematics*, vol. 11, 1963.

[27] S. J. Maybank and O. D. Faugeras, "A theory of self-calibration of a moving camera.," *International Journal of Computer Vision*, vol. 8, 1992.

[28] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*. 2007.

[29] R. I. Hartley, "Self-calibration from multiple views with a rotating camera," 1994.

[30] Q.-T. Luong and O. D. Faugeras, "Self-calibration of a moving camera from pointcorrespondences and fundamental matrices," *International Journal of Computer Vision*, vol. 22, 1997.

[31] T. Thormählen, H. Broszio, and P. Mikulastik, "Robust linear auto-calibration of a moving camera from image sequences," in *Proceedings of the 7th Asian Conference on Computer Vision - Volume Part II*, 2006.

[32] R. Y. Tsai, "An efficient and accurate camera calibration technique for 3D machine vision," in *Computer Vision and Pattern Recognition*, 1986.

[33] R. Lenz and R. Y. Tsai, "Techniques for calibration of the scale factor and image center for high accuracy 3-d machine vision metrology.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, 1988.

[34] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, 2000.

[35] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. 2002.

[36] R. Szeliski, *Computer Vision: Algorithms and Applications*. 2010.

[37] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. 2006.

[38] A. M. Dan Pelleg, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[39] G. Hamerly and C. Elkan, "Learning the k in k-means," in *In Neural Information Processing Systems*, 2003.

[40] A. Irschara, M. Rumpler, P. Meixner, T. Pock, and H. Bischof, "Efficient and globally optimal multi view dense matching for aerial images," *in Proceedings of the 22th Congress of the ISPRS 2012*, 2012.

[41] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006.

[42] M. Kass, A. P. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, 1988.

[43] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 1995.

[44] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International Journal of Computer Vision*, vol. 22, 1997.

[45] C. S. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein, "Experimental study of minimum cut algorithms," in *Proceedings of the Eigth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[46] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, 2004.

[47] T. Pock, D. Cremers, H. Bischof, and A. Chambolle, "An algorithm for minimizing the mumford-shah functional," in *International Conference on Computer Vision*, 2009.

[48] L. Ambrosio and V. M. Tortorelli, "Approximation of functional depending on jumps by elliptic functional via t-convergence," *Communications on Pure and Applied Mathematics*, vol. 43, 1990.

[49] R. B. Potts, "Some generalized order-disorder transformations," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 48, 1952.

[50] T. Pock, A. Chambolle, D. Cremers, and H. Bischof, "A convex relaxation approach for computing minimal partitions.," in *Conference on Computer Vision and Pattern Recognition*, 2009.

[51] T. Chan and L. Vese, "An active contour model without edges," in *International Conference on Scale-Space Theories in Computer Vision*, 1999.

[52] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, 2000.

[53] D. Oram, "Rectification for any epipolar geometry," in *Proceedings of the British Machine Vision Conference*, 2001.

[54] M. Pollefeys, R. Koch, and L. J. V. Gool, "A simple and efficient rectification method for general motion," in *International Conference on Computer Vision*, 1999.

[55] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, International Conference on Computer Vision, 1999.

[56] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual Conference on Computer Graphics and Interactive Techniques*, 1996.

[57] G. Vogiatzis, C. Hernández Esteban, P. H. S. Torr, and R. Cipolla, "Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, 2007.

[58] K. Kolev, M. Klodt, T. Brox, and D. Cremers, "Continuous global optimization in multiview 3d reconstruction," *International Journal of Computer Vision*, vol. 84, 2009.

[59] C. Zach, T. Pock, and H. Bischof, "A globally optimal algorithm for robust tv-l1 range image integration.," in *International Conference on Computer Vision*, 2007.

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

[61] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of Mathematical Imaging and Vision*, vol. 40, 2011.

[62] A. N. Tikhonov, "Regularization of incorrectly posed problems," *Soviet Mathematics Doklady*, 1963.

[63] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, 1992.

[64] T. F. Chan, S. E. Glu, and M. Nikolova, "Algorithms for finding global minimizers of image segmentation and denoising models," tech. rep., SIAM Journal on Applied Mathematics, 2004.

[65] S. Boyd and L. Vandenberghe, *Convex Optimization*. 2004.

[66] A. Chambolle, "An algorithm for total variation minimization and applications," *Journal of Mathematical Imaging and Vision*, vol. 20, 2004.

[67] A. Chambolle, "Total variation minimization and a class of binary mrf models," in *Proceedings of the Fifth International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*, 2005.

[68] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, vol. 21, 1987.

[69] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Proceedings of the third European conference on Computer Vision - Volume Part II*, 1994.

[70] K.-J. Yoon and I.-S. Kweon, "Locally adaptive support-weight approach for visual correspondence search," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[71] G. Egnal, "Mutual Information as a Stereo Correspondence Measure," 2000.