UNIVERSITÄT
KOBLENZ · LANDAU

Towards Believable Augmented Reality: Combining the Real and Virtual Worlds

DISSERTATION

von

Anna Katharina Hebborn, M.Sc.

Koblenz, März 2018

Vom Promotionsausschuss des Fachbereichs 4: Informatik der Universität Koblenz-Landau zur Verleihung des akademischen Grades **Doktor der Naturwissenschaften (Dr. rer. nat.)** genehmigte Dissertation.

Vorsitzender des Promotionsausschusses: Prof. Dr. Ralf Lämmel
Vorsitzender der Promotionskommission: Prof. Dr. Harald F.O. von Korflesch
1. Berichterstatter: Prof. Dr.-Ing. Stefan Müller
2. Berichterstatter: Prof. Dr.-Ing. Didier Stricker

Datum der Einreichung: 21.03.2018
Datum der wissenschaftlichen Aussprache: 03.02.2021

Abstract

Augmented reality (AR) applications typically extend the user's view of the real world with virtual objects. In recent years, AR has gained increasing popularity and attention, which has led to improvements in the required technologies. AR has become available to almost everyone.

Researchers have made great progress towards the goal of believable AR, in which the real and virtual worlds are combined seamlessly. They mainly focus on issues like tracking, display technologies and user interaction, and give little attention to visual and physical coherence when real and virtual objects are combined. For example, virtual objects should not only respond to the user's input; they should also interact with real objects. Generally, AR becomes more believable and realistic if virtual objects appear fixed or anchored in the real scene, appear indistinguishable from the real scene, and response to any changes within it.

This thesis examines on three challenges in the field of computer vision to meet the goal of a believable combined world in which virtual objects appear and behave like real objects.

Firstly, the thesis concentrates on the well-known tracking and registration problem. The tracking and registration challenge is discussed and an approach is presented to estimate the position and viewpoint of the user so that virtual objects appear fixed in the real world. Appearance-based line models, which keep only relevant edges for tracking purposes, enable absolute registration in the real world and provide robust tracking. On the one hand, there is no need to spend much time creating suitable models manually. On the other hand, the tracking can deal with changes within the object or the scene to be tracked. Experiments have shown that the use of appearance-based line models improves the robustness, accuracy and re-initialization speed of the tracking process.

Secondly, the thesis deals with the subject of reconstructing the surface of a real environment and presents an algorithm to optimize an ongoing surface reconstruction. A complete 3D surface reconstruction of the target scene offers new possibilities for creating more realistic AR applications. Several interactions between real and virtual objects, such as collision and occlusions, can be handled with physical correctness. Whereas previous methods focused on improving surface reconstructions offline after a capturing step, the presented method de-noises, extends and fills holes during the capturing process. Thus, users can explore an unknown environment without any preparation tasks such as moving around and scanning the scene, and without having to deal with the underlying technology in advance. In experiments, the approach provided realistic results where known surfaces were extended and filled in plausibly for different surface types.

Finally, the thesis focuses on handling occlusions between the real and virtual worlds more realistically, by re-interpreting the occlusion challenge as an alpha matting problem. The presented method overcomes limitations in state-of-the-art methods by estimating a blending coefficient per pixel of the rendered virtual scene, instead of calculating only their visibility. In several experiments and comparisons with other methods, occlusion handling through alpha matting worked robustly and overcame limitations of low-cost sensor data; it also outperformed previous work in terms of quality, realism and practical applicability. The method can deal with noisy depth data and yields realistic results in regions where foreground and background are not strictly separable (e.g. caused by fuzzy objects or motion blur).

Zusammenfassung

Typischerweise erweitern Augmented Reality (AR)-Anwendungen die Sicht des Benutzers auf die reale Welt um virtuelle Objekte. In den letzten Jahren hat AR zunehmend an Popularität und Aufmerksamkeit gewonnen. Dies hat zu Verbesserungen der benötigten Technologien geführt. AR ist dadurch für fast jeden zugänglich geworden.

Forscher sind dem Ziel einer glaubwürdigen AR, in der reale und virtuelle Welten nahtlos miteinander verbunden sind, einen großen Schritt näher gekommen. Sie konzentrieren sich hauptsächlich auf Themen wie Tracking, Anzeige-Technologien und Benutzerinteraktion und schenken der visuellen und physischen Kohärenz bei der Kombination realer und virtueller Objekte wenig Aufmerksamkeit. Beispielsweise sollen virtuelle Objekte nicht nur auf die Eingaben des Benutzers reagieren, sondern auch mit realen Objekten interagieren. Generell wird AR glaubwürdiger und realistischer, wenn virtuelle Objekte fixiert oder verankert in der realen Szene erscheinen, sich nicht von der realen Szene unterscheiden und auf Veränderungen dieser Szene reagieren.

Diese Arbeit untersucht drei Herausforderungen im Bereich Maschinelles Sehen um dem Ziel einer glaubwürdig kombinierten Welt näher zu kommen, in der virtuelle Objekte wie reale erscheinen und sich ebenso verhalten.

Diese Dissertation konzentriert sich als erstes auf das bekannte Tracking- und Registrierungsproblem. Hierzu wird die Herausforderung von Tracking und Registrierung diskutiert und ein Ansatz vorgestellt, um die Position und den Blickpunkt des Benutzers zu schätzen, so dass virtuelle Objekte in der realen Welt fest verankert erscheinen. Linienmodelle, die dem Erscheinungsbild entsprechen und nur für Trackingzwecke relevante Kanten beinhalten, ermöglichen eine absolute Registrierung in der realen Welt und ein robustes Tracking. Einerseits ist es nicht notwendig, viel Zeit in die manuelle Erstellung geeigneter Modelle zu investieren, andererseits ist das Tracking in der Lage mit Änderungen innerhalb des zu verfolgenden Objekts oder Szene umzugehen. Versuche haben gezeigt, dass die Verwendung von solchen Linienmodellen die Robustheit, Genauigkeit und Reinitialisierungsgeschwindigkeit des Tracking-Prozesses verbessert haben.

Zweitens beschäftigt sich diese Dissertation mit dem Thema der Oberflächenrekonstruktion einer realen Umgebung und präsentiert einen Algorithmus zur Optimierung einer laufenden Oberflächenrekonstruktion. Vollständige 3D-Oberflächenrekonstruktionen einer Szene eröffnen neue Möglichkeiten um realistischere AR-Anwendungen zu erstellen. Verschiedene Interaktionen zwischen realen und virtuellen Objekten, wie Kollisionen und Verdeckungen, können physikalisch korrekt behandelt werden. Während sich die bisherigen Methoden darauf konzentrierten die Oberflächenrekonstruktionen nach einem Aufnahmeschritt zu verbessern, wird die Rekonstruktion während der Aufnahme erweitert, Löcher werden geschlossen und Rauschen wird reduziert. Um eine unbekannte Umgebung zu erkunden muss der Benutzer keine Vorbereitungen treffen. Das Scannen der Szene oder eine vorhergehende Auseinandersetzung mit der zugrundeliegenden Technologie ist somit nicht notwendig. In Experimenten lieferte der Ansatz realistische Ergebnisse, bei denen bekannte Oberflächen für verschiedene Oberflächentypen erweitert und Löcher plausibel gefüllt wurden.

Anschließend konzentriert sich diese Dissertation auf die Behandlung von realistischen Verdeckungen zwischen realer und virtueller Welt. Hierzu wird die Herausforderung der Verdeckung als Alpha Matting Problem formuliert. Die vorgestellte Methode überwindet die Grenzen moderner Methoden, indem ein Überblendungskoeffizienten pro Pixel der gerenderten virtuellen Szene schätzt wird, anstatt nur deren Sichtbarkeit zu berechnen. In mehreren Experimenten und Vergleichen mit anderen Methoden hat sich die Verdeckungsbehandlung durch Alpha Matting als robust erwiesen und kann mit Daten, die durch preiswerte Sensoren aufgenommen wurden, umgehen. Hinsichtlich der Qualität, des Realismus und der praktischen Anwendbarkeit übertrifft die Methode die Ergebnisse von bisherigen Ansätzen. Des Weiteren kann die Methode mit verrauschten Tiefendaten umgehen und liefert realistische Ergebnisse in Regionen, in denen Vorder- und Hintergrund nicht strikt voneinander trennbar sind (z.B. bei Objekten mit einer undeutlichen Kontur oder durch Bewegungsunschärfe).

Contents

1 Introduction 1

1.1 Augmented Reality and Challenges 1

1.2 Overview and Contributions 4

2 Robust Model-based Tracking using Appearance-based Line Models 7

2.1 Registration and Tracking Challenge 8

2.2 Background of Model-based Tracking 10

2.2.1 Taxonomy Model, Line and Edge 11

2.2.2 Edge-based Methods and Challenges 12

2.3 Related Work on Line Model Generation 15

2.4 Appearance-based Line Models based on Mapping and Refinement 22

2.5 Method Overview 24

2.6 Line Model 24

2.6.1 Keyposes 25

2.6.2 Lines and Confidences 27

2.7 Line Management 27
2.7.1 Line states 28
2.7.2 Re-initialization 28
2.8 Edge-based Tracking 29
2.9 Line Mapping and Refinement 31
2.9.1 Extracting Lines 33
2.9.2 Establishing Correspondences 33
2.9.3 Adding New Lines and Updating Confidences 34
2.10 Experimental Results 35
2.10.1 Synthetic Image Sequences 35
2.10.2 Real Image Sequences 38
2.11 Conclusion and Future Work 40

3 Optimizing 3D Reconstructions 43

3.1 3D Reconstruction Challenge 44
3.2 Background to KinectFusion 47
3.2.1 Truncated Signed Distance Function 47
3.2.2 Overview of the Pipeline 49
3.2.3 Limitations and Improvements 51
3.3 Related Work on Optimizing 3D Surface Reconstructions 52
3.3.1 Plane Segmentation and Fitting 53

3.3.2 Hole Filling, De-noising and Propagating Surfaces 57

3.4 Propagating Surfaces, Hole Filling and De-noising during the 3D Reconstruction 59

3.5 Method Overview 60

3.6 TSDF Volume 63

3.7 Local Plane Prior Estimation 64

3.8 Planar Surface Propagation from Local Plane Priors 66

3.9 De-noising and Refinement through Local Plane Priors 68

3.10 Non-Planar Surface Propagation and Hole Filling via Diffusion 69

3.11 Volume Integration 71

3.12 Collisions and Occlusions with the TSDF 72

3.12.1 Collision Detection and Response 73

3.12.2 Occlusion Handling 75

3.13 Experimental Results 75

3.13.1 Surface Propagation without using Plane Priors 76

3.13.2 Surface Propagation using Plane Priors 77

3.13.3 Implementation Details and Runtime Performance 80

3.14 Conclusion and Future Work 82

4 Occlusion Matting: Realistic Occlusion Handling 85

4.1 Occlusion Handling Challenge 86

4.2 Related Work on Occlusion Handling 87
  4.2.1 Object-based Occlusion Handling 87
  4.2.2 Model-based Occlusion Handling 89
  4.2.3 Depth-based Occlusion Handling 92
4.3 Occlusion Matting 94
  4.3.1 Idea and Motivation 94
  4.3.2 Occlusion as an Alpha Matting Problem 95
4.4 Related Work on Alpha Matting 97
  4.4.1 Natural Image Matting 97
  4.4.2 Real-time Natural Image Matting and Video Matting 99
  4.4.3 Depth-based Trimap Generation and Matting 100
4.5 Method Overview 102
4.6 Adaptive Trimap Generation 102
  4.6.1 Coarse Segmentation 104
  4.6.2 Labeling of Unknown Regions 105
  4.6.3 Adaptive Dilation 106
4.7 Foreground and Background Propagation 107
4.8 Alpha Estimation 110
4.9 Experimental Results 111
  4.9.1 Implementation Details and Runtime Performance 112

4.9.2 Evaluation of the Algorithm 113

4.9.3 Comparison with Edge-aware Filtering Methods 114

4.9.4 Comparison with Edge Snapping-Based Method 117

4.9.5 Comparison of Runtime Performance 120

4.10 Limitations 120

4.11 Optimizing Sensor Data with 3D Reconstruction 121

4.12 Conclusion and Future Work 122

5 Conclusion **125**

5.1 Summary and Achievements 125

5.2 Future Work 127

Own Publications **135**

Bibliography **139**

Chapter 1

Introduction

This chapter introduces the concept of augmented reality (AR). Fundamental challenges are discussed in Section 1.1. Section 1.2 presents an overview and explains the contribution of this thesis.

1.1 Augmented Reality and Challenges

AR extends the real environment with virtual information. Typically, the user's view or a camera image is extended with virtual objects. AR supports the user's interaction with the real world and helps them to perform real-world tasks [Azu97]. Thus, AR holds potential for several areas and applications, such as medicine (guidance and training for surgery), education, design and entertainment. For example, AR can assist users during maintenance or repair tasks. Instructions to complete a task or to identify a component can be overlaid on the real world so that the user can perform without looking at a manual. Because users do not have to switch between the task and an instruction document, they can focus on completing the task. AR applied to interior design can help to evaluate how a new piece of furniture would fit with the existing interior decor and find an appealing configuration. AR systems also help people to fulfill or solve tasks collaboratively. For example, a co-located user can see the real environment in which another user interacts. Thus, the co-located user can help the user on site with additional information by annotating the user's view of the real environment. An example would be showing a route through a city (as shown in a game-based scenario in [HDD$^+$15]).

Azuma [Azu97] provides a commonly used and widely accepted definition of an AR system. The following three characteristics are identified:

1. **Combination of real and virtual content.** The real scene is augmented with virtual objects.
2. **Interactivity.** Virtual content reacts to the user's input and to changes in the environment in real-time.
3. **3D registration.** Virtual content appears fixed in the real world, even if the user or the camera changes the point of view.

Ideally, the combination must be as realistic as possible to give the user the illusion that the virtual and real worlds co-exist in the same space [Azu97]. Therefore, virtual objects should be seamlessly integrated into the real scene. They should be fixed in their placement and appear indistinguishable from the real surrounding. Moreover, virtual objects should interact with the real scene in a physically plausible manner: they should collide with real objects, they should occlude or be occluded by real objects, and they should cast shadows on real objects and vice versa [Kli00].

In recent years, AR has succeeded remarkably and has become more readily available to all people. Technologies take advantages of this trend and much progress has been made to fulfill the vision of AR as a system where virtual and real worlds co-exist. However, several technical challenges remain and some of the to current solutions could be optimized. Fundamental computer vision tasks entail 1) tracking the user to fix virtual objects in space; 2) realistic rendering of virtual objects with correct illumination, shadows and occlusions; and 3) the 3D reconstruction of the real environment.

Registration and Tracking is an essential problem in AR. To register virtual objects in the user's surroundings, the position and orientation of the user's head or camera with respect to the real world need to be continuously estimated and tracked. In addition these extrinsic data, intrinsic data of the camera are needed to project virtual objects realistically. The intrinsic data can be obtained through a calibration procedure before the tracking process starts. Ideally, the registration and tracking system should fulfill the following requirements: 1) the is accurate and robust even if the user moves rapidly, 2) no interventions are required in the real environment. Even small errors like jitter or lags between real and virtual objects, or placing markers as references reduce the illusion of co-existence.

Illumination and Shadows are also fundamentally important to seamlessly combine virtual objects with the real environment. The illumination between real and virtual world should be consistent. Real objects should cast shadows on virtual objects and vice versa. Shadows improve the impression that virtual objects appear as part of the real scene and give the user an idea of correct distances between objects. In addition, the shading of virtual objects in relation to the real world is essential for achieving consistent illumination [NS09].

Occlusions in AR have always posed a challenge. Sometimes virtual objects are partially or totally occluded by opaque or even transparent real objects. Currently, many AR applications suffer from inadequate occlusion handling, such that virtual objects are always in front of real objects regardless of the spatial relationship. Incorrect and inaccurate occlusion handling not only breaks the illusion of co-existence between the real and virtual worlds, it also results in wrong depth perception. User interactions, such as grabbing a virtual object, result in misunderstanding and confusion.

3D Surface Reconstruction of the real environment is fundamental to many AR applications. The geometry of the real scene must be known to simulate interaction between real and virtual components. For example, moving virtual objects should not move through real objects; they should collide instead. If a surface representation (also called *world model*) is available, it can be registered to the real scene and used to handle collisions through standard computer graphic algorithms. The same representation can be used to solve the occlusion problem in static scenes. Ideally, the model should be as complete and detailed as possible. Such a model can be modeled manually, which is highly labor intensive and only useful for static and known scenes. Therefore, the common solution is to reconstruct the scene in an offline step before the application starts or the reconstruction is generated during the runtime of the application. Generally, the reconstruction of arbitrary scenes is an important issue and many approaches have already been proposed. The results of these methods are mostly dense and accurate but the reconstructions remain incomplete. This is due to the fact that parts of the scene are non-visible during the capturing process (e.g. occluded by another object). Moreover, precise models of cluttered and dynamic scenes are hard to generate.

Furthermore, collision handling and occlusion handling both depend on the registration and tracking process. Accurate registration and tracking allow exact alignment of the model with the real world, resulting in realistic collisions and occlusions.

Shading and shadows are not the focus of this thesis, but accurate reconstruction as well as registration and tracking are helpful to solve this illumination problem. The model can be used to cast shadows on virtual objects [HDH03] or to recover the illumination distribution of a scene [SSI03]. The reconstruction can be extended with reflectance and illumination information to enable a realistic and seamless integration of correctly illuminated synthetic objects in the real world [GHH01].

1.2 Overview and Contributions

This thesis focuses on three challenges in computer vision to meet the goal of co-existence in AR, namely: registration and tracking, complete 3D reconstruction, and occlusions.

As discussed in Section 1.1, these tasks are fundamental to realize geometrical consistency and to create physically plausible AR applications in which virtual objects act like real objects.

The main contributions of this thesis can be summarized as follows:

- An approach to optimizing registration and tracking using significant and appearance-based edges. The algorithm estimates the camera pose in relation to the real world, and extends a 3D line model of the real environment or object. The algorithm adds new lines to the model and considers their appearance and the current view of the camera, to decide which lines are more likely to be visible in the camera image.
- An algorithm to de-noise and complete a 3D surface reconstruction of the real environment. It fills holes and gaps in the actual observed surfaces of the real environment and extends the surface during the capturing process. By considering the structure of the surface (flat or curved) the algorithm can obtain appealing results for several types of surfaces and overcomes limitations of previous approaches.
- A new depth-based approach, named *occlusion matting*, to handle occlusions between real and virtual worlds more realistically. This algorithm effectively solves the occlusion problem by formulating it as an *alpha matting* problem. It overcomes limitations of previous work on occlusion handling and can deal with different types of occlusions that have sharp and smooth transitions.
- A novel algorithm for high-quality alpha matting to solve the occlusion matting in real-time. It automatically estimates a trimap based on a depth sensor image and the virtual objects, and effectively calculates an alpha value for each unknown pixel.
- An approach to extend the obtained depth images by considering the 3D surface reconstruction to obtain complete depth maps of the real scene. This approach improves the quality of depth maps from low-cost depth sensors, thereby ensuring a precise occlusion matting. Holes and gaps which occur because of well-known

problems like reflections or limited sensor ranges are likely to be filled in with the reconstruction data.

This thesis is structured as follows. Chapter 2, discusses the theoretical background of model-based tracking and describes the approach for improving tracking results by using appearance-based line models. Chapter 3 starts with a discussion of 3D surface reconstruction using implicit surfaces. Thereafter, the algorithm to de-noise and extend an ongoing reconstruction and its results for several surfaces are presented. Chapter 4 introduces and presents the approach to handle occlusions as an alpha matting problem. Each chapter gives a short introduction, summarizes recent work on this topic, and presents an approach to overcome state-of-the-art problems. The thesis is concluded in Chapter 5.

Chapter 2

Robust Model-based Tracking using Appearance-based Line Models

This chapter focuses on the registration and tracking challenge and presents an approach to overcome limitations of commonly used model-based tracking methods. Most results presented in this chapter were peer-reviewed and presented at a conference [HEM15].

Section 2.1 discusses the registration and tracking problem in the context of AR and provides definitions and related taxonomy in this field. Section 2.2 describes the theoretical background to model-based tracking using edge features and discusses the benefits of model-based tracking in AR, and examines fundamental problems in the common methods. Section 2.3 provides an overview of existing work and highlights the differences from the presented method. Section 2.4 describes the idea of the proposed approach and Section 2.5 gives an overview. The following sections (Sections 2.6 to 2.9) explain the method in detail, and Section 2.10 presents the experimental results. The final section (Section 2.11) summarizes the contribution of this work and discusses future possibilities.

2.1 Registration and Tracking Challenge

To register virtual objects in the surroundings of the user, the position and orientation of the user's head or camera (called pose) with respect to the real world or an object in the real environment which serves as an anchor must be known [BCL15]. Billinghurst *et al.* [BCL15] describe this process as comprising one or two stages: 1) a registration stage which determines the pose in relation to the anchor; and 2) a tracking stage which updates the pose relative to the previously estimated pose. In this thesis, the term *tracking* refers to both stages.

Tracking with respect to an object or an environment is a well-studied and widely discussed task in several computer vision applications (e.g. in robotics or virtual reality). Thus, several solutions exist to solve the tracking problem, which use various types of sensors such as cameras, mechanical encoders, magnetometers, and gyroscopes.

For AR applications a high-quality tracking is required. Virtual content is precisely aligned with the real world only if the tracking is robust and accurate. Almost all AR applications currently use visual tracking approaches, sometimes in combination with other sensors like gyroscopes and magnetometers. Generally, visual tracking enables robust registration without the need for any other sensor, simply by using a cheap video camera. Typically, the camera of a smart phone or a camera attached to a head-mounted display is used. To estimate the camera pose in relation to the real world, natural or pre-placed features are localized, identified and tracked in the incoming camera stream.

A simple and still commonly used strategy is to place artificial markers in the scene which serve as anchors. Specific characteristics of these markers, such as geometry or color, enable the use of simple and potentially more robust tracking algorithms [SH15]. However, placing markers to simplify the tracking task requires interventions in the natural environment and sometimes is not possible [LF05]. Marker-based tracking in large environments like corridors, living rooms or factory halls requires the use of multiple markers, with known relations among the markers. In this scenario the use of markers is impractical and requires a long setup time. In addition, markers might have annoying effects for the end users [LF05] and can reduce the illusion of co-existence.

This thesis focuses on *markerless tracking* sometimes called *natural feature tracking* which exclusively uses existing natural features in the scene. Markerless approaches can be classified into two categories: model-based and model-free tracking [SH15]. In model-based methods the environment or the object to be tracked is known. These techniques use a 3D model of the object or scene to be tracked. In contrast, model-free methods do not require any knowledge. The model to be tracked is created during runtime and extended while the user moves within the scene. This challenge is known as simultaneous localization and mapping (SLAM) and is well-studied in computer vision. SLAM is fundamental for a wide range of applications, including robotics and autonomous driving. These algorithms build a map of an unknown environment and simultaneously estimate the pose of the camera related to the map. SLAM systems are not limited to the visibility of a known model. However, a significant drawback of these systems and their use in AR applications is they cannot provide an absolute reference. Virtual objects have to be placed by the user spontaneously [SH15]. Model-based tracking systems, by contrast, virtual content to be registered in the coordinate system of the known model. Recent systems combine the advantages of both model-based and model-free tracking [SH15] by using the model to initialize the tracking procedure. Generally, most SLAM techniques (e.g. Klein *et al.* [KM07]) use landmarks (such as point features) to create a sparse representation of the environment as a point cloud. The point cloud representation is useful to estimate the camera pose, but is not useful to handle occlusions or collisions between the real and virtual worlds. To handle such interactions between real and virtual components the geometrical structure of the scene is needed. In model-based systems, this is often given by a surface model (like a polygonal representation). It can also be generated rapidly by using dense SLAM techniques. This reconstruction challenge is discussed in Chapter 3.

2.2 Background of Model-based Tracking

Model-based tracking methods can be categorized by the features that are used. Edge-based methods match projections of objects' 3D lines to sharp edges in the intensity image. Other techniques use textural information and estimate the pose of the camera through optical flow, template matching algorithms, or keypoints [LF05; CC10]. Many approaches use edges because they provide computationally efficiency and simple implementation. Furthermore, edges are naturally stable despite lighting changes [LF05; CC10]. Keypoints do not work well if the difference among viewpoints between consecutive frames is too large; hence, edges are more stable despite viewpoint changes. It is true that keypoints can deal with illumination, orientation, scale, and partial viewpoint changes as well. However, the description of features to obtain local textural or orientation information around stable points is very expensive [CC10]. Generally, edges work well in poorly textured scenes and they are able to deal with objects having specular materials [LF05]. Thus, they are useful in urban environments or industrial scenarios where not many planar and textured objects can be observed, and other techniques such as keypoint-based methods fail.

Model-based tracking methods can be categorized into two classes [LSF+10]. These classes are recursive tracking, in which the previous camera pose is used to estimate the current pose; and tracking-by-detection, in which allows the estimation of the camera pose without a previous pose estimation. In contrast to recursive methods, which follow a frame-to-frame tracking, tracking-by-detection allows an automatic initialization and re-initialization if the tracking algorithm fails. However, tracking-by-detection methods suffer from jitter and less accuracy [LSF+10]. In the context of AR, current techniques retain the best elements of these two approaches by using the tracking-by-detection to initialize a recursive tracking procedure (e.g. see [CC10]). This thesis focuses on recursive tracking by using edge features. Further information about methods based on keypoints or tracking-by-detection is available in the literature [LSF+10; LF05].

2.2.1 Taxonomy Model, Line and Edge

Various definitions exist for the terms *line* and *edge*. In the context of model-based tracking, *line* is often used to define line segments of the model to be tracked. Thus, lines are geometrically defined by two endpoints, either in 3D space or projected in 2D space. The term *edge* is often used to define significant discontinuities in an image. Edges can be detected by calculating derivatives (as described in [RN17]) and lines or line segments can be extracted using algorithms (e.g. [GJM$^+$10; HHM14]).

Edges arise as consequence of the geometrical and photometric conditions within a scene [KA08]. The term edge is also used in the field of computer graphics to define line segments of a polygon mesh. Each edge connects two vertices of a polygon [FDF$^+$90].

Generally, edges can be categorized by their causes, which are specific properties in the scene. In the context of model-based tracking, edges are of interest which occur because of the specific properties of the object to be tracked. According to the classification systems in [Pan11; KA08], edges are categorized into four classes with regard to the object to be tracked (approximately labeled in Figure 2.1):

1. **Orientation edges** (sometimes called *object edges* or *crease edges*) arise because of strong discontinuities between surface orientations. These edges are independent of the view; the corresponding line segments can be calculated in a pre-processing step from a surface representation.

2. **Texture and material edges** (also called *reflectance edges*) occur because of discontinuity of the surface material, such as textural characteristics of the model.

3. **Silhouette and depth edges** (also called *step edges* or *occlusion edges*) are the boundaries between the object and the background or the object itself. They connect parts of the surface that face toward the viewpoint and parts that face away from the viewpoint. Such edges can be calculated by identifying front and back faces

based on the surface normal and viewing direction of the user. These types of edges are view dependent and must be calculated during runtime.

4. **Non-physical edges** are unstable in their appearance and location; they might have a negative effect on tracking results. They cannot be associated with a physical property of an object. They occur because of illumination, such as shadow edges or specular edges.

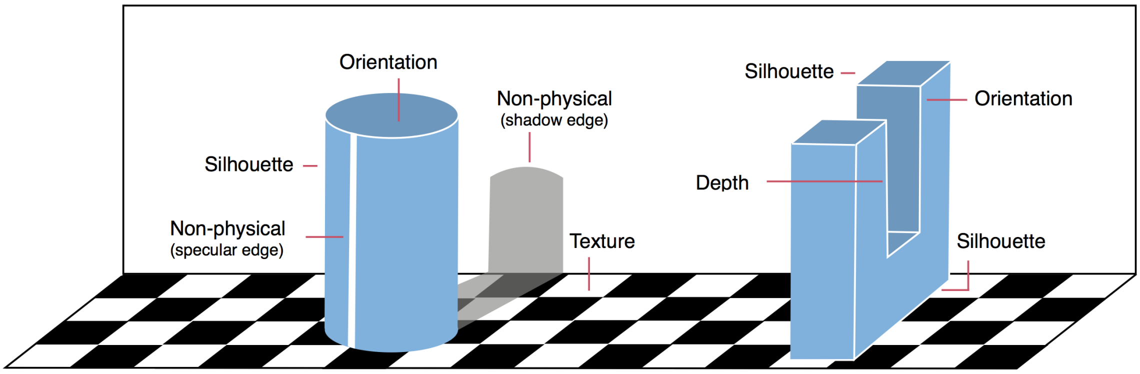


Figure 2.1: **Classes of edges.** Different types of edges are shown; they arise because of the illumination or specific model properties and structures. (Image modified and extended from [KA08].)

2.2.2 Edge-based Methods and Challenges

Historically, edge-based approaches estimate the camera pose by registering the 3D line model onto the image edges. The model lines can be registered onto the edges with or without explicit line extraction [LF05]. Early approaches using explicit extraction [Low92; KDN93] gathered straight 2D line segments from the image gradient (defined by two endpoints) and minimized the distance between observed and projected lines of the model to adjust the camera pose. Given the fast extraction of lines from image edges, the challenge is to correctly match lines of the model with lines extracted from the image. Edges are less differentiable than point features [GM06] and offer only few number of description and matching techniques.

Hence most model-based methods avoid complex or computationally expensive explicit extraction, description, and matching of edges. They use information from prior frames and follow a recursive scheme, known as frame-to-frame tracking.

Real-time attitude and position determination (RAPiD) [HS90; Har93] is an early and well-known 3D model tracker that runs in real-time. The idea is to simplify the lines of the model to a set of sample points, called control points. To obtain 2D-3D point correspondences, image space is searched for each projected control point. Finally, the camera pose is estimated by a distance minimization between observed correspondences.

Generally, RAPiD-like methods are fast, relatively simple to implement and provide good results [LVT^{+}03]. Because of this simplicity and low computation effort, several improvements of the algorithm already exist [DC02; WVS05]. Today, improved variants can run in real-time even on relatively slow computers such as mobile devices. Hence, they are often used in industrial environments where significant edges are available [LVT^{+}03].

After an initialization step, a RAPiD-like system performs the following steps for each frame. First, it uses a predicted pose (which can be the estimated pose of the previous frame) to decide which lines or control points are visible and to find their location in image space. Control points can be obtained by sampling the 3D lines of the model; or they can also be generated sampling the projected 2D lines. After the projection procedure, a search for significant image gradients along the perpendicular direction for each control point is performed. A gradient can be accepted as a match if, for example, its magnitude is larger than a predefined threshold. Lastly, the pose of the camera is calculated by projecting the 3D control points and minimizing the distance to their corresponding 2D image points, using the least squares method (often called re-projection error).

Generally, outliers tend to occur because edges in the scene are close to model edges and are thus used during pose estimation. Significant edges in the background, non-physical edges or aspect changes lead to incorrect correspondences between edges [LVT^{+}03]. Outliers reduce the quality of the tracking procedure significantly. Hence recent approaches

[DC02; WVS05] minimize the distances between correspondences using the iterative reweighted least squares method. This allows the use of an M-estimator, which is more robust to outliers than a simple least square procedure. In this thesis the tracking approach [WVS05] is used; further details are provided in Section 2.8.

An assumption of this approach is that the camera does not move fast and thus a change between two consecutive poses is considered as small. The tracking works only if the projection of the line model is very close to the edges in the camera image. Reitmayr and Drummond [RD06], for example, reduced failure under fast motion by using inertial sensors to predict a camera pose. Alternatively, motion models that describe plausible movements of the camera or of the object to be tracked between subsequent frames can be used to predict a pose (as described in [MKS$^+$11; KMB$^+$14].

Most RAPiD-like systems require manual initialization by the user. The user must overlay the model to be tracked with its real counterpart. This is time-consuming and means that no automatic re-initialization is possible if the tracking is lost. Manually re-initialization is needed. In addition, RAPiD-like methods and their various improvements remain sensitive to interference such as background clutter or texture on the object itself.

It is worth mentioning that tracking failures occur more often when very simple objects are tracked, compared with relatively complex ones. Complex objects offer more control points so that points corrupted by noise can easily be ignored by a robust estimator [LVT$^+$03]. Accordingly, it is advantageous if models of simple objects (e.g. rectangular boxes) possess not only geometrical edges but also textural edges.

A significant drawback can be identified in that all these approaches need a precise and complete line model, which, at best, contains only relevant lines for the tracking process. Such models are often generated manually in an offline step. The manual creation of well-designed models is time-consuming and the modeler should understand the tracking algorithm to select suitable edges, in an optimum manner.

2.3 Related Work on Line Model Generation

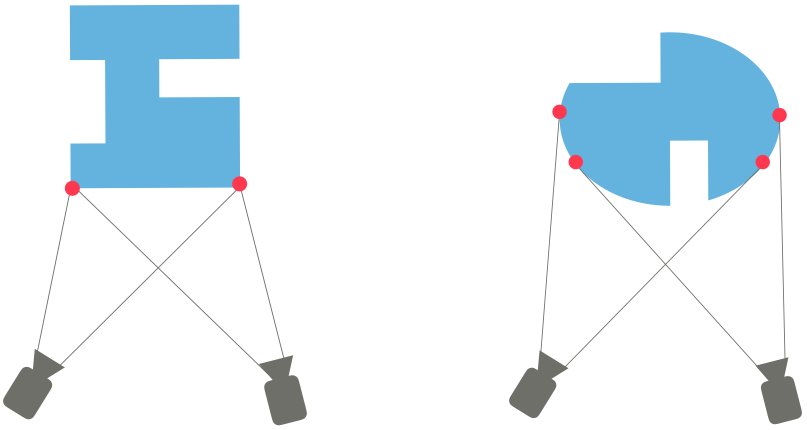
At present, only a few approaches deal with the automatic generation of suitable line models. The main topic of research in this area remains the improvement of the tracking process. Because most objects in man-made environments are manufactured such as furniture, polygonal models are available from the production procedure [CC10] and can be used to generate a line model. The easiest method is to extract lines directly from a polygonal model of the object to be tracked.

Choi *et al.* [CC10] used an object-space method to create a line model in an offline procedure. Their approach identifies sharp edges from the triangle mesh representation. The angle between the surface normals of attached faces is used as an indicator for sharpness. In other words, the algorithm analyzes the mesh and identifies sharp edges by a simple thresholding of the inner product of two normals. During runtime, extracted lines are used in a RAPiD-like tracking approach.

A disadvantage of object-space methods is that the generated models contain no texture, depth and silhouette edges. Object-space silhouettes, however, can be obtained during runtime by identifying edges of the model that connect a front-face with a back-face. Moreover, if the object has a complex 3D structure or the user moves around the object and observes it from different sides it is likely that some lines will be occluded by the object itself. That is why visible parts of the line model should first be identified. Otherwise the pose can be heavily affected by incorrect control points on non-visible lines. So called hidden-line removal is performed by a rendering a binary space partition tree [DC02] or more easily by using OpenGL occlusion queries [CC10].

Platonov and Langer [PL07] pursued an image-space method instead; their method creates a contour model by a rendering procedure. The polygonal model is rendered from different viewpoints and under different lighting conditions. The most stable edges in terms of illumination invariance and view independence are extracted. For each rendering, a 2D edge detection is performed, edge points are back-projected in 3D space, and visibility statistics are saved. The back-projection uses color-coded rendering of the polygonal model, in which each face of the model is shown in an unique color. After the rendering, the algorithm identifies the corresponding face for each 2D edge point based on the color. The 3D points are then estimated by calculating the intersection between the identified face and a geometric ray (from the center of the camera through the edge point). Finally, all 3D points that are visible under different perspectives and illuminations are merged to contours, approximated either by a B-Spline or a sequence of line segments.

Clearly, the quality of contours depends strongly on the quality of the model and the renderings; the renderings should represent the object as realistically as possible. Processing time of up to several hours is needed to generate a suitable model, but this processing is performed only once. However, curved objects are problematic to describe using this algorithm. Silhouette and depth edges of non-curved objects do not change their 3D position if they are obtained from multiple similar viewpoints. In contrast, silhouette and depth edges of curved objects change their 3D positions if the viewpoint is even sightly changed. An example is shown in Figure 2.2. This limitation means the algorithm is not capable of describing curved objects well.



(a) Object without curved structures (b) Object with curved structures

Figure 2.2: **Problem of silhouette edges.** (a) Silhouette edges of non-curved objects do not change their position. (b) Silhouette edges of curved objects change their positions even if the viewpoint is changed.

Alternatively, view-dependent 3D lines or control points can be generated during the tracking process using an image-space strategy. For each frame, control points are obtained by performing a rendering of the polygonal model, followed by detection of 2D edges and a back-projection.

Reitmayr and Drummond [RD06] generated suitable line models for outdoor urban environments. First, their system renders a coarse textured polygonal model of the object to track, based on a prediction of the pose. Using a standard edge detector *edgels* (pixels with a significant gradient magnitude) are extracted which are used as control points in a RAPID-like tracking approach. To calculate the 3D coordinates for each control point, the system uses the depth information of the rendered model.

The rendering of textured models automatically culls details so that only edges are detected that are likely to be visible at the current scale. Pure and complex line models, in contrast, work well in regions nearby the object being tracked. In large outdoor environments where the user can move far away from the object to tracked, such as a building, a projection of detailed line models results in clusters (e.g. dense clusters of facade edges). Edges in clusters are no longer distinguishable and this leads to incorrect correspondences and incorrect pose estimation.

Wuest *et al.* [WS07] presented a similar approach which is additionally able to deal with untextured surface models. If no or incorrect material properties are given, their algorithm generates a line model exclusively based on the geometric properties of the object. Therefore, it renders the model with a predicted pose into the depth-buffer and normal-buffer. By applying edge detection on these rendered maps different types of edges are obtained. Silhouette and depth edges are extracted by identifying discontinuities in the depth map. Orientation edges are detected in the normal map as they presented an edge between surfaces having different orientations.

Generally, the rendering of polygonal models enables the extraction of view-dependent line models. The benefits of a view-dependent line model can be summarized as follows:

1) it does not contain occluded line segments and thus avoids the task of hidden-line removal; 2) it contains silhouette edges (also at the boundaries of curved objects where offline object-space methods fail) and; 3) it always consists of lines with an appropriate level of detail [WS07].

However, these methods depend strongly on the quality of the model. The texture of coarse models (as in Reitmayr and Drummond [RD06]) must be as accurate as possible to generate well-positioned lines. Inaccuracy in the position of a line reduces the quality of the pose estimation. With untextured models it is the same: if the geometry is inaccurate or incorrect, incorrect lines are produced, and the pose estimation is no longer accurate.

Alternatively, the line model can be generated without rendering the 3D model. The approach, presented by **Neubert *et al.*** [NPD07], creates an appearance-based edge model directly from a recorded image sequence of the object to be tracked. To obtain the camera pose for each image, the approach passes them through a SLAM system. The user then indicates planar regions, for example, by drawing polygons around them. After reconstructing of the polygons and identifying the images in which the polygons are visible, the algorithm creates so called *edge frames*. Each edge frame contains a list of edgels extracted from the region of the projected polygon. Lastly, the system selects a small subset of edge frames and uses them as keyframes in a real-time tracking approach. Generally, this approach creates useful line models for objects having significant planar regions, but does not work well for objects with complex 3D structures or curved surfaces. Furthermore, user input for the pre-selection of planar regions is required.

Shimizu *et al.* [NSS^{+}16] proposed another two phase method: offline model generation and online tracking. The model generation phase is used to obtain the 3D geometry of a textureless scene, represented as line segments. Therefore, the system uses a set of RGB-D image sequences as input. For each frame, the system detects 2D line segments represented by two endpoints using a fast detector algorithm. In the next step, the system back-projects the endpoints of the line segments into the 3D space via the corresponding depth value from the depth image. Depth images captured by low-cost sensors are mostly noisy, especially at the boundaries of objects, and often do not register well with the RGB data. To overcome this problem the approach performs a plane segmentation and fit line segments onto planar structures. A 2D line segment matching between two frames is later achieved using a line descriptor to create 2D-3D line segment correspondences. The correspondences are then used to solve the perspective-n-lines (PnL) problem to estimate the camera pose.

The online tracking phase works only on RGB data. To track the camera online, the system detects 2D lines and extracts the corresponding line descriptors. Matching these descriptors against the database from the previous model generation, the system can identify correspondences to estimate the current camera pose.

As mentioned before, methods that generate a line model in an offline procedure are unable to deal with curved objects and dynamic scenes in which the objects are movable or non-rigid. Moreover, all methods presented so far work only if the model to be tracked is visible.

Alternative methods reconstruct a model while the user explores the scene, as SLAM approaches do. In the context of AR, it is helpful to use a known and static object as an anchor point such as a marker or a model of an object and extend the model while the camera moves in the scene. Methods that do so can be called *extensible model-based tracking* [KM07]. Problems of SLAM approaches and their use in AR applications, like unknown scale or location of the coordinate system are simply solved by using the model as an anchor.

The approach, presented by **Bleser *et al.*** [BWS06], uses a model to initialize the tracking. In the tracking stage, the user can move freely and is not restricted to the known part of the scene because the algorithm recovers the structure of the scene automatically. The tracking is realized by a detection and tracking of point features from frame-to-frame.

However, most of these approaches (e.g. [DMM03; SMG05; BWS06]) use point features and are thus often unsuitable in industrial or man-made environments. **Gee *et al.*** [GM06] presented a model-based approach to vision-based SLAM that uses edges features as landmarks. Starting from a simple model, new lines are extracted and automatically added to the model during a SLAM mapping process. These lines are used within a RAPiD-like model-based tracking approach. Because new lines continually extend the model, silhouette edges of curved surfaces or non-physical edges (like shadow edges) may be added to the model. To eliminate such lines from the model, a RANSAC procedure (see Section 3.3.1) is used during the pose estimation. Each edge keep a count of how often it was detected as an inlier during the past frames. Lines which were consistently detected as an outlier are removed from the model. Additionally, the counter is used to optimize the camera pose estimation using a weighting factor. Generally, this approach does not offer information about the surface structure available, so that hidden-line removal cannot be performed. For the sake of completeness, it should be noted that SLAM-based approaches exist which use edge features and work without a reference for initialization, as in traditional SLAM techniques. Whereas point features are often used in this research area, edge features are not often considered as landmarks. However, the use of edge features offers significant advantages. Edge features can build richer maps with a high degree of geometric information, and can overcome the problem of insufficient point features available for tracking [ED06]. Moreover, edges features are more robust under fast motions and can deal with motion blur [KM08]. Nevertheless, edges are difficult to use in a SLAM context because of their selection, observation, initialization, and data association [ED06].

Hirose and Saito [HS12] also used line segments as landmarks in a monocular SLAM approach. They presented a line descriptor called *line-based eight-directional histogram feature* (LEHF), which offers a fast and efficient way of describing features of detected line segments. They demonstrated that line segment matching was more robust than point matching with regard to partial occlusion and viewpoint changes.

Gomez-Ojeda *et al.* [GMS+17] proposed a stereo visual SLAM system, called PL-SLAM, that combines points and line segments to work robustly in different scenarios. Their system is especially useful for scenarios in which point features are scarce or not well-distributed. While the system tracks point and line segments along sequences of stereo frames, it builds a map to keep detected landmarks. Line segments are handled like point features and stored in a list with the most representative descriptors to enable frame-to-frame tracking.

Alternative approaches, described below, do not try to define edges by a patch around a detected line segment; they save edge points directly. **Eade and Drummond** [ED06] used edge features as landmarks and called them *edgelets*. An edgelet represents a local straight segment of an edge having a significant intensity change in the image. In this way curved objects can be represented by linear segments. An edgelet is defined by its center and a unit vector representing the direction of the edgelet in the 3D scene.

To find an edgelet in an image, the system predicts its 2D location and detected all edgels in a region around the predicted edgelet. Then edgels having a similar gradient direction to that of the edgelet are identified. Identified edgels that formed a straight edge segment are merged and used as possible observations of the edgelet. Because edgelets are described only by the direction of their intensity change (from low to high), there might be several possible observations in the considered image region. Hence the system processes a maximum-likelihood data association to choose the most likely hypothesis and to be robust against outliers. (For more details see [ED06].) Generally, Eade and Drummond showed that edgelets can be successfully tracked and mapped. However, the computational cost of locating an edglet in an image is very high.

Klein and Murray [KM08] extended a point feature and keyframe-based SLAM approach with edge features. They used the edglet representation from Eade and Drummond [ED06] to improve the tracking results during fast motions.

Bose and Richards [BR16] presented an edge-feature-based SLAM using an RGB-D sensor. Their approach builds a 3D point cloud which represented the edge structure of the target scene. Each pixel belonging to both the depth or color edges of an observed RGB-D image is back-projected using the depth component. The registration between two sets of edge points is achieved using the well-known *iterative closest point* (ICP)[1] algorithm. Point clouds of edges consist of significantly fewer points than usual for point clouds because they describe only the main structure of the scene, such as geometric discontinuities and discard points on flat surfaces. Thus, the registration process is less computationally expensive.

2.4 Appearance-based Line Models based on Mapping and Refinement

Several solutions exists to generate a line model automatically, as presented in Section 2.3. Some approaches generate the line model based on a surface model in an offline procedure, whereas others use the surface model to create a view-dependent line model during the tracking process. Overall, the results of these methods depend strongly on the quality of the input model. Extracted line models are not always suitable for the realization of a robust and precise edge-based tracking algorithm. At worst, line models contain lines that are not detected in the camera stream or are not well positioned, for example, because of impreciseness in the texture.

Ideally, the line model contains only lines that are robust and relevant for the tracking procedure. Good lines to track should result in sharp and significant edges (edgels) in the camera frame, so that, the corresponding image edges are easy to detect by standard edge detection algorithms. Lines should appear continuously and should be stable in their position. Moreover, only lines that are likely to be visible in the current view should be

---

[1] The ICP algorithm is explained in Section 3.2.2.

used for tracking. To decide which lines are suitable, it is helpful to follow a SLAM-like procedure and update the model generate view-dependent and appearance-based lines.

To fulfill the above requirements, the presented approach in this thesis creates a suitable line model by mapping new lines to the model and refining existing ones. The method works with relatively few 3D lines and utilizes a surface model (ideally a surface model of the complete scene) as input. Starting from the coarse line model, the model is extended and refined by a mapping procedure as the user moves around the object. New lines are generated from the incoming camera frame and thus it is likely that they will appear in subsequent frames.

To overcome the problems of previous methods the mapping step collects reference poses, called *keyposes*. A keypose consists of a camera pose in relation to the model and a set of view-dependent and appearance-based lines. It covers possible camera poses for a relatively small area around the object and keeps the lines which are likely to be visible for these views. To eliminate drift each keypose is initialized with the lines of the coarse input model.

The mapping process extracts line segments from the current frame and extends and refines the set of lines for the closest keypose. In contrast,tracking uses the lines of the closest keypose to select good edges to track depending on a predicted view. Hence, a level of detail-based culling is performed automatically, as is hidden-line removal. Thus, the tracking always uses view-dependent and appearance-based lines that possess an appropriate level of detail and are not occluded. Keyposes are also useful for performing automatic re-initialization if tracking fails. Manual initialization procedures are no longer needed; the user only needs to go back to an already visited point of view.

Generally, newly added edges are refined during the tracking and are completely removed from the edge model if they are not constantly detected in the 2D image, or if their appearance varies often. As a result, non-relevant edges which could arise from dynamic objects or shadows, are removed from the model.

2.5 Method Overview

A concise overview of the method is shown in Figure 2.3. The system consists of three main components:

1. **Line Management** takes care of the line model. Depending on the pose it estimates the closest keypose of the model and decides which lines are used during the tracking step and in the mapping and refinement step.
2. **Tracking** uses the selected subset of line segments and follows a RAPiD-like scheme to estimate the pose of the camera. This is done by fitting 3D points of the line segments to strong image gradients.
3. **Mapping & Refinement** extracts new 3D line segments and adds new keyposes to the model, if the quality of the pose is good and it differs enough from existing keyposes. If the estimated pose is close to an existing one, the mapping and refinement component identifies existing 3D lines for the selected keypose and refines them.

In the following pages, Section 2.6 describes the line model that is administrated by the line management system. Sections 2.7 to 2.9 explain the three components and their functionality in detail.

2.6 Line Model

The line model is formed by the line segments of the input model and a set of keyposes that are generated during the tracking process. These keyposes approximate possible camera poses in the already visited environment around the known 3D object. A keypose keeps information about the 3D camera pose and a collection of 3D lines. These lines

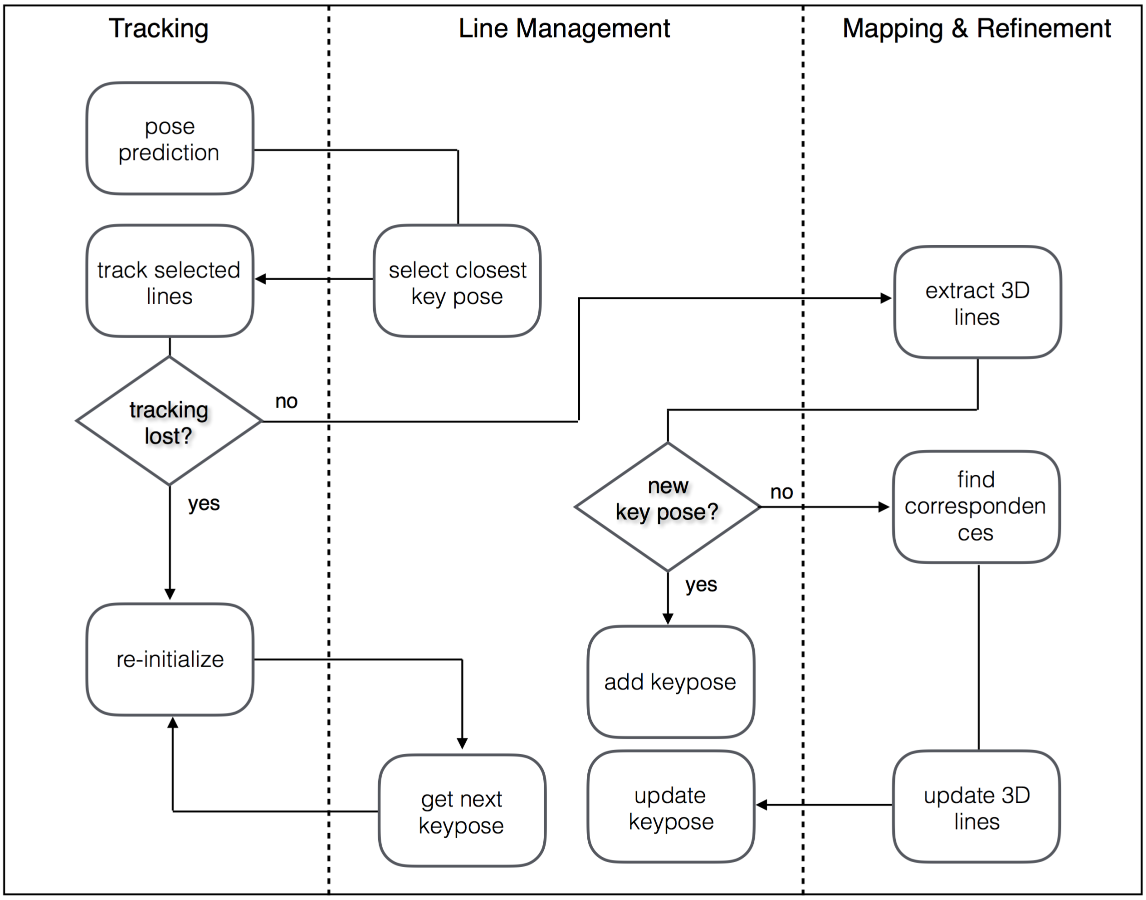


Figure 2.3: **Schematic overview of the proposed method.** The line management takes care of the administration of the line model and its keyposes. The model-based tracking system estimates a camera pose by fitting a selection of robust 3D lines to strong image gradients. If the quality of the pose is good and keypose is new, the mapping and refinement component adds new 3D lines to the model. Otherwise, if the pose is similar to a keypose, the 3D lines are refined for the current keypose.

are created and refined when the camera pose, estimated by the tracking component, is similar to a stored pose.

2.6.1 Keyposes

Depending on the input, the line model initially contains only edges that describe a relatively small part of the object or describe its contours. When the tracker is initialized the first keypose is added. If the current camera pose differs substantially from the initial pose or previous keyposes, a new keypose is added. Each generated keypose stores the

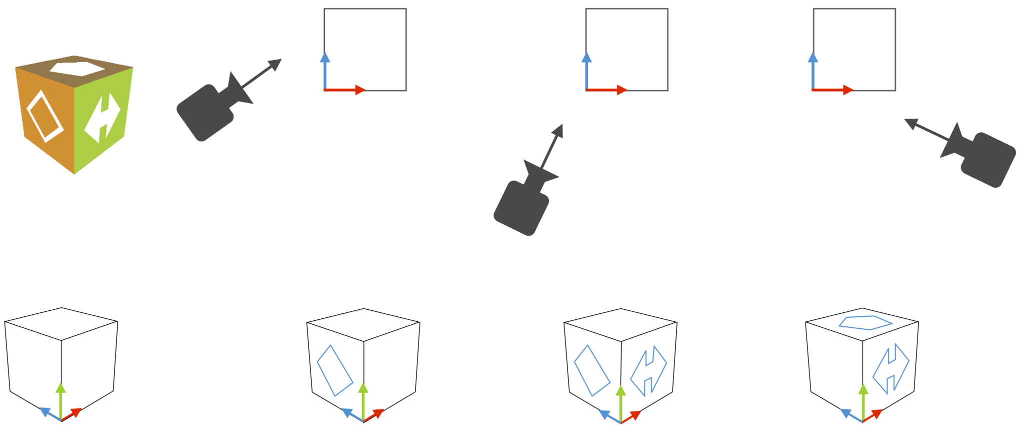


Figure 2.4: **Schematic visualization of different keyposes.** These keyposes are generated from three camera poses and their corresponding lines (blue). Basic lines (black) are defined by the coarse input model.

following information: a 3D camera rotation and translation with respect to the object, and a set of 3D lines extracted from the input images. The idea of keyposes is shown in Figure 2.4. At first, 2D line segments are extracted from the current frame to create a keypose. The 2D segments are then back-projected based on the surface model to obtain 3D edges. These observed lines are refined, and new 3D lines are mapped and added to the keypose if the estimated camera pose is close to a stored one.

Keyposes are added whenever the following conditions are met: the tracking quality is good; most of the object is visible; and the camera has a minimum distance from each existing keypose in the model. The calculation of tracking quality is based on the covariance matrix and the number of detected edges in the current frame. The distance between two camera poses is calculated by the translation distance and the angle difference between the viewing directions (currently set to 10°). The translation distance between two camera poses depends on the size of the model. An appropriate threshold is calculated by taking the size of the bounding box into account. The minimum translation distance is given as a percentage of the radius of the bounding sphere; experiments showed that 15% is an adequate value. In most cases the camera moves slowly, so it is sufficient to compare the distance between the current pose and the last active keypose. At worst, all keyposes are compared to check whether a new keypose is required.

2.6.2 Lines and Confidences

Starting from the input, the model keeps only the basic lines that are represented by their two endpoints. New lines are added to the model as the camera moves around the object or the first keypose is initialized, and it is very likely that some of these lines are incorrect because of the detection of non-physical edges in the image. As mentioned in Section 2.2.1, these edges are detected in the camera image, but are not associated with a physical object edge. Due to instability in their observation they are not suited for tracking. Moreover, some lines are incorrectly initialized in their 3D positions (e.g. wrong back-projected edges due to uncertainties).

To solve these problems, the first assumption is that a 3D line that occurs often in a 2D image is a good line to track. Therefore, new initialized lines are associated with a confidence factor. This factor increases through each successful observation and gives an indication of the robustness of a line in relation to the corresponding keypose. The second assumption is that a line that frequently changes its appearance or often corresponds to nearby edges in the image, such as parallel edges or background clutter, is an unsuitable line to track. These lines are unstable in their positions and thus the confidence is decreased. To deal with these lines, each new initialized edge is updated if the tracking quality is good. Furthermore, variance is calculated for the endpoints to obtain a measurement of the uncertainty of the 3D position. During tracking, the confidence factor in combination with the variance of the 3D position helps to identify good lines to track.

2.7 Line Management

The edge management component of the system takes care of the administration of the current model and the selection of lines for the tracking, mapping and refinement step. Different line states are used to determine which lines are good to track, which lines should be used only during the mapping and refinement process, and which lines should be eliminated.

2.7.1 Line states

Each line of the model is assigned to one of the following edge states: *truth*, *robust* or *under control*. All lines of the input model are marked as *truth*. These lines are correctly positioned, correspond to a real edge of the scene, and are therefore continuously used for tracking. Furthermore, these lines are useful for initializing a new keypose and thus for drift avoidance. In contrast, new initialized lines are not used for tracking immediately. They are marked as *under control* and are used only during the refinement step, which is processed in every frame. As described in Section 2.6.2, the endpoints of these lines are updated, and the confidence increases with every subsequent frame after initialization if the line occurs. If the number of detections reaches a threshold and the variance of the endpoints remains low, the line is assigned to *robust* and is used for the tracking procedure for that keypose. If the occurrence count does not increase within a specified number of frames or the variance becomes too high, the edge can be classified as a non-physical edge and is then eliminated from the associated keypose.

2.7.2 Re-initialization

In some cases, the tracking step fails. This can occur, because of fast camera movements, or because the object to be tracked is no longer visible. For this reason the tracking quality is observed in every frame. If the tracking quality is very low, the tracking continues for a few frames but the system does not allow any further steps, such as adding new keyposes or mapping and refinement. If the quality remains poor after the additional few frames, the tracking is considered as lost and a re-initialization process is begun. This re-initialization is realized based on the previously collected keypose information. Starting from the last pose, keyposes with corresponding robust edges are selected and the system tries to re-initialize the tracking process.

Ideally, the object should be visible and tracked for a while at the beginning of a tracking phase. In this way new keyposes are initialized and good lines to track can be classified.

2.8 Edge-based Tracking

In this research, a RAPiD-like tracker [WVS05] was used. Wuest *et al.* [WVS05] approach improves the standard RAPiD algorithm's robustness and accuracy. The focus is on reducing incorrect point correspondences. Wrong correspondences may arise from background clutter or other edges that lie near to the searched ones. Because there exists more than one potential candidate multiple hypothesis are considered to estimate the camera pose.

The tracking system uses as input the surface model and a subset of lines that are good to track. Once initialized, the camera pose is updated for each frame through the following steps:

A) **Create a set of 3D control points** via uniform sampling in image space. The number of sample points per line is determined by the length of the projected line.

B) **Perform a visibility test** to identify visible control points of the lines from the initial line model. This step is required because some lines might be occluded by the object itself. The search for correspondences to occluded control points can result in false matches, which leads to inaccurate pose estimation. To discard occluded control points, the surface model is rendered off-screen with regard to the camera pose into the z-buffer. Non-visible points are then automatically identified by rendering them into the same z-buffer and using occlusion queries; a depth test is automatically performed using the graphics processing unit (GPU) hardware.

Because the visibility of control points of the initial lines is constant, this step is performed only once if a new keypose is added.

C) **A gradient maxima search** projects each sample point onto the image plane and searches for significant edges in a direction perpendicular to the corresponding line. The method considers each significant gradient maximum as a possible correspondence.

D) **Pose estimate with a single hypothesis.** This step estimates a coarse pose by taking for each sample point the correspondences with the strongest gradient magnitude as the match. Based on these correspondences, the camera pose is estimated by non-linear minimization. In this way a pose close to the searched one is quickly estimated without interference from other hypotheses.

E) **Pose estimation with multiple hypotheses.** This step appends several of the most significant gradients and applies the minimization again based on the estimated pose guess from the previous step. This process greatly improves the results.

As described by Wuest *et al.* [WVS05] in detail, the initial pose is estimated by minimizing the distance between control points and their correspondences, for a single hypothesis, as follows:

$$err = \sum_{i=0}^{m} \rho_{Tuk}(\Delta(p_i, q_i)), \tag{2.1}$$

where m is the number of control points for all lines currently considered; p_{Tuk} describes the Tukey estimator function; and Δ is the distance function between two points in image space. The Tukey estimator function is defined by:

$$p_{Tuk}(x) = \begin{cases} \frac{c^2}{6}[1 - (1 - (\frac{x}{c})^2)^3] & \text{if } |x| \leq c \\ \frac{c^2}{6} & \text{otherwise} \end{cases} \tag{2.2}$$

The threshold is defined by $c = k \cdot \sigma$, where σ is the standard deviation of the estimation error and the constant k. The distance between a projected control point p_i and it's corresponding hypothesis q_i in the image can be obtained by:

$$\Delta(p_i, q_i) = |(q_i - p_i) \cdot n_i|, \tag{2.3}$$

where n_i indicates the normal of the projected line. The pose refinement via multiple hypotheses by minimizing the following error:

$$err = \sum_{i=0}^{n} \rho_{Tuk}(\underset{j}{min} \Delta(p_i, q_{i,j})), \tag{2.4}$$

This refinement ensures that the hypothesis with the closest distance is used. To maintain real-time performance, only a fixed number of hypothesis per control point are considered.

2.9 Line Mapping and Refinement

This section describes the process by which the model is updated if an accurate camera pose, estimated by the tracking component, is available. Existing lines are refined and new edges are added. This procedure is summarized by the following steps, shown in Figure 2.5. (Sections 2.9.1 to 2.9.3 explain steps A to C in greater detail.)

A) **Extracting lines** extracts straight line segments, represented by two 3D points in world coordinates out of the current camera image.

B) **Establishing correspondences** identifies existing lines and adds new 3D lines to the keypose closest to the estimated one.

C) **Adding new lines and updating confidences** updates all lines that were identified in the previous step and updates their confidence. New lines are added to the current keypose.

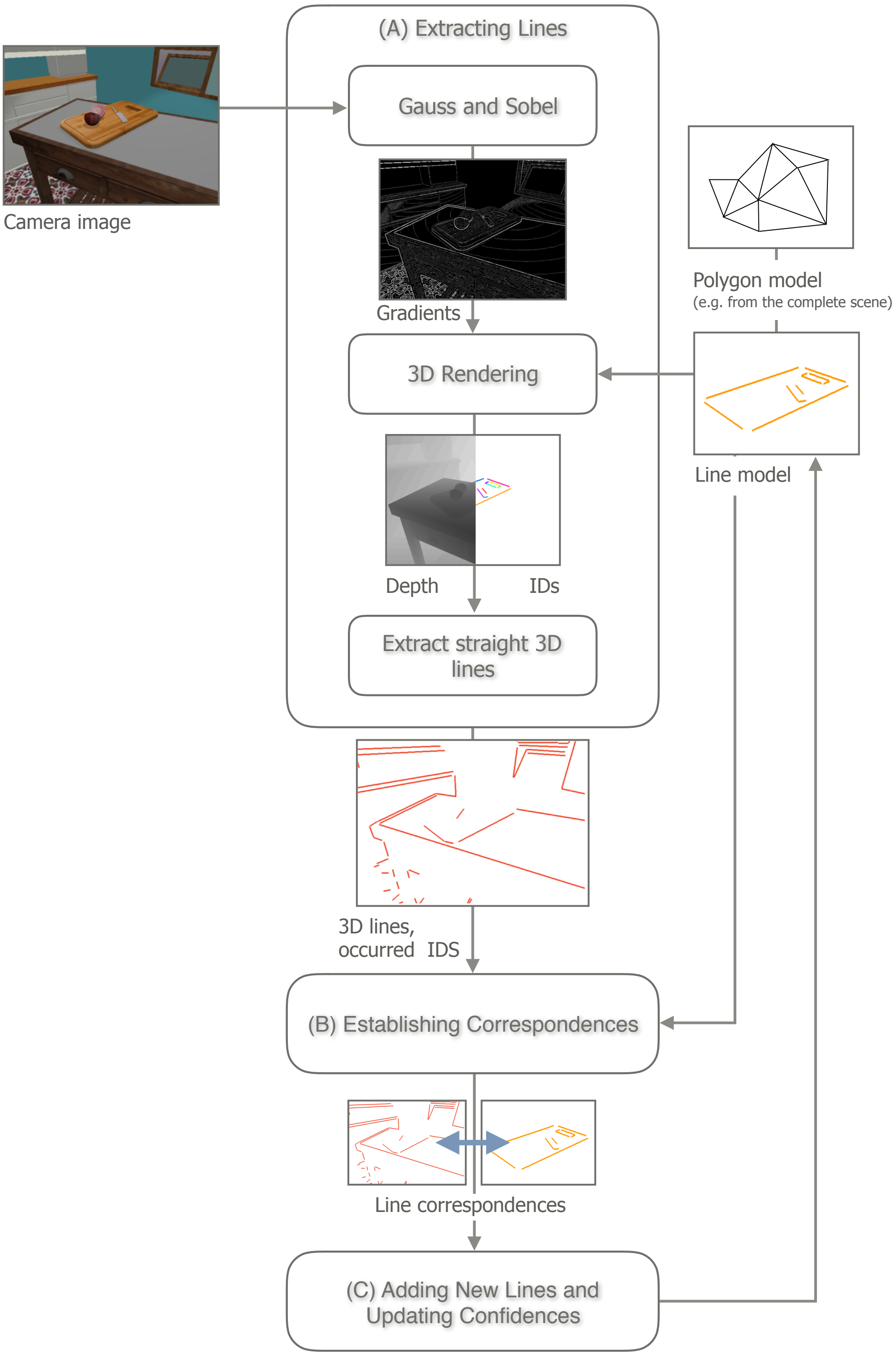


Figure 2.5: **Schematic pipeline of the line mapping and refinement.** The camera image is convolved with a Gauss and Sobel mask to create significant edge points and straight line segments (Step A). A depth map is used for a back-projection process and an ID map is used to establish line correspondence (Steps A and B). New lines are added to the current line model and existing lines are updated (Step C).

2.9.1 Extracting Lines

First, a gradient image is created by applying a simple Gauss and Sobel filter mask. The algorithm then walks along the gradient image, applies a non-maximum suppression and creates a structured representation of significant and connected 3D edge points.

To create structured 3D edge points efficiently, an offscreen rendering step is performed. This step uses the estimated camera pose and renders the surface model at the same image resolution as the input image. The resulting output image provides depth values (distances to the camera) per pixel. The depth values are used to perform a back-projection procedure of 2D gradient points with high values. At this step, all of the points are discarded which do not lie on the surface of the 3D model or have a large discrepancy in depth to previous connected points. Large differences in depth values are caused by uncertainties of the 2D edge detection, mostly in case of a silhouette or depth edges. Silhouette edges are located at the border of the object surface and the background, whereas depth edges are located within the object region (e.g. caused by self-occlusion). It may happen that the wrong depth value is selected.

The connected 3D edge points are then subdivided into straight line segments represented by two endpoints. The resulting 3D segments are described in the camera coordinate system. For further processing, line segments are transformed to the coordinate system of the model by applying inverse rotation and translation of the current camera pose.

2.9.2 Establishing Correspondences

In the system it is necessary to decide if a 3D line exists in the current edge model or must be added. According to the ID method described by Lima *et al.* [LSF$^+$10], a matching procedure between the existing and newly created 3D line segments is processed. The original method is often used to perform hidden-line removal but can be easily transferred to the matching problem if the camera rotation and translation are known.

Therefore, a unique identifier (ID) is assigned to each 3D line in a keypose. In the same manner as the depth map is rendered, each existing line with its ID encoded as color is rendered. This is done simply, in one rendering pass, using OpenGL frambuffers. To map the IDs to RGB and vice versa, a simple RGB coding scheme is used. With this scheme the maximum number of lines per keypose is $2^{32} - 1$. The line count per keypose depends on the complexity of the scene or object and on the threshold values for the difference between two keyposes. However, it will never reach this maximum.

To identify existing lines in the keypose, the line extraction algorithm (described in Section 2.9.1) is extended. Connected points are split to straight segments, and the ID image at the corresponding 2D point is evaluated and the occurrences per ID are counted. If an ID occurs more often than a threshold (currently set to half the length of the segment), it can be assumed that the ID already exists. The corresponding edge is then identified by decoding the ID and is updated. If many different IDs occur, it is quite likely that background clutter surrounds the line; such lines are ignored. Otherwise, a new 3D line is added to the current keypose. Lines are only updated or added when a certain number of edges are detected. Otherwise, it is assumed that the quality of the estimated pose is not accurate enough for the mapping and refinement step.

2.9.3 Adding New Lines and Updating Confidences

The confidence of a line per keypose is calculated by considering the occurrence and the appearance. If the number of detections in the 2D image n reaches a certain threshold t (with $t > 0$), the variance σ^2 for each endpoint is calculated by:

$$\sigma^2 = (\frac{1}{n}\sum_{i=0}^{n}(p_i - \mu)^2)/l, \tag{2.5}$$

where μ is the current position of the considered endpoint, $\{p_1, p_2, \ldots, p_n\}$ are positions already observed, and l is the length of the line in 3D space. The division by l

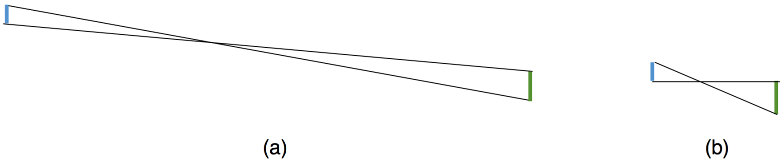


Figure 2.6: **Effectiveness of dividing by length.** Two observations of line segments with different lengths. The segments in (a) are more equal than in (b). Dividing of the variance by the length results in a higher value for segment (b) than segment (a).

assures equality of σ^2 for different edge lengths, as sketched in Figure 2.6. If the variance constantly grows, the line is classified as non-robust, as discussed in Section 2.7.

2.10 Experimental Results

In this section, the results of evaluating the method through several experiments are presented. The experiments were performed with different objects in a set of synthetic and real image sequences. The used models are differed in complexity and characteristics, as shown in the following sections.

2.10.1 Synthetic Image Sequences

A set of synthetic sequences was used to compare the proposed method with the standard edge tracker without any mapping or refinement step. These image sequences were generated by moving a virtual camera around a static object and storing ground truth pose data for each image. The model of the chopping board was placed in a kitchen model, as in Figure 2.7. This made the scene more realistic, for example with background clutter. The surface models of the chopping board and the kitchen are taken from 3D Warehouse [War]. The corresponding coarse edge models of the chopping board were generated manually using Blender [Ble].

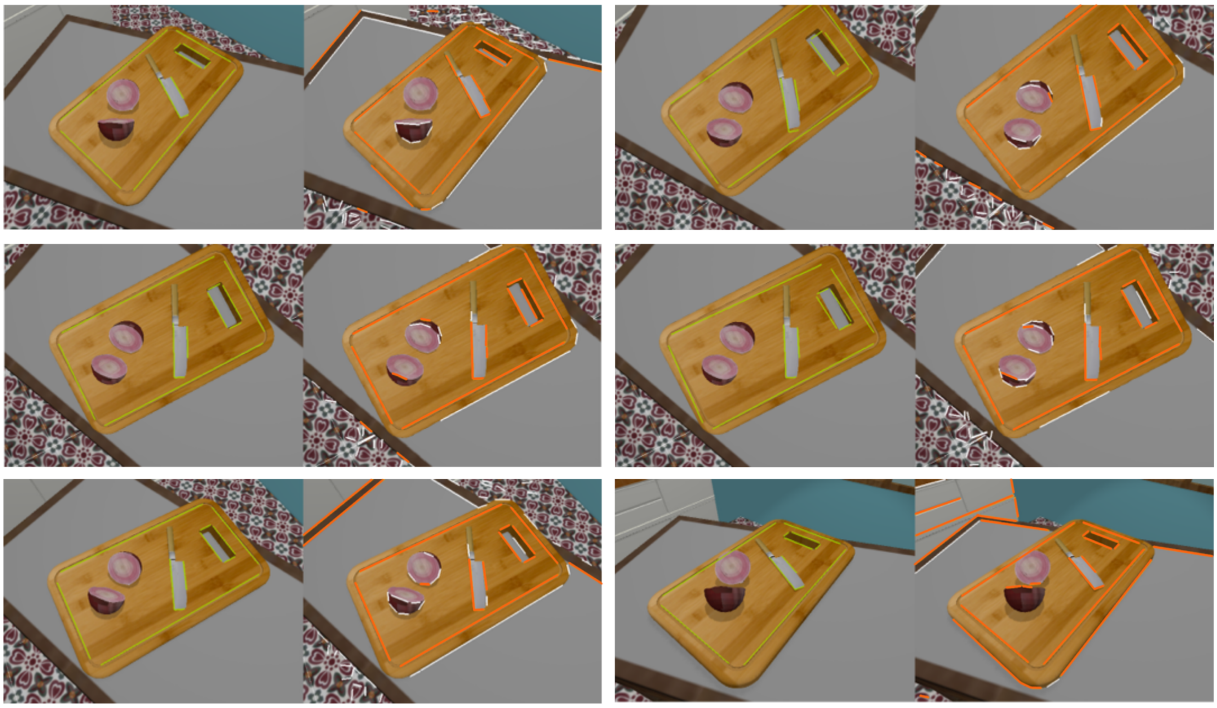


Figure 2.7: **Tracking results of a synthetic generated scene.** Results with the proposed mapping and refinement step (orange edges) and without this step (green edges) are shown in the synthetic sequence of the chopping board. The white edges were used only during the mapping and refinement step; they were not used for tracking.

Figure 2.7 shows the tracking results and Figure 2.8 shows the corresponding 6 degrees of freedom (DOF) camera pose plots for one image sequence. The tested image sequence started with a slow camera movement, which was rather easy to track. After some time, around frame 350, tracking became more complicated and the translation velocity increased. Based on the line plots, the proposed method consistently showed similar or better robustness and accuracy than the method without mapping and refinement step. Standard tracking was able to track only 56.4% of the images, whereas tracking with the mapping was able to track 92%. The method with mapping and refinement did not lose the camera pose easily and recovered faster if tracking was lost. Re-initialization was easily performed by finding the best matching keypose as described in Section 2.7.2. The camera simply needed to move close to any stored keypose. By contrast, tracking without the extension needed a camera pose close to the most recently tracked pose to reinitialize the tracking process.

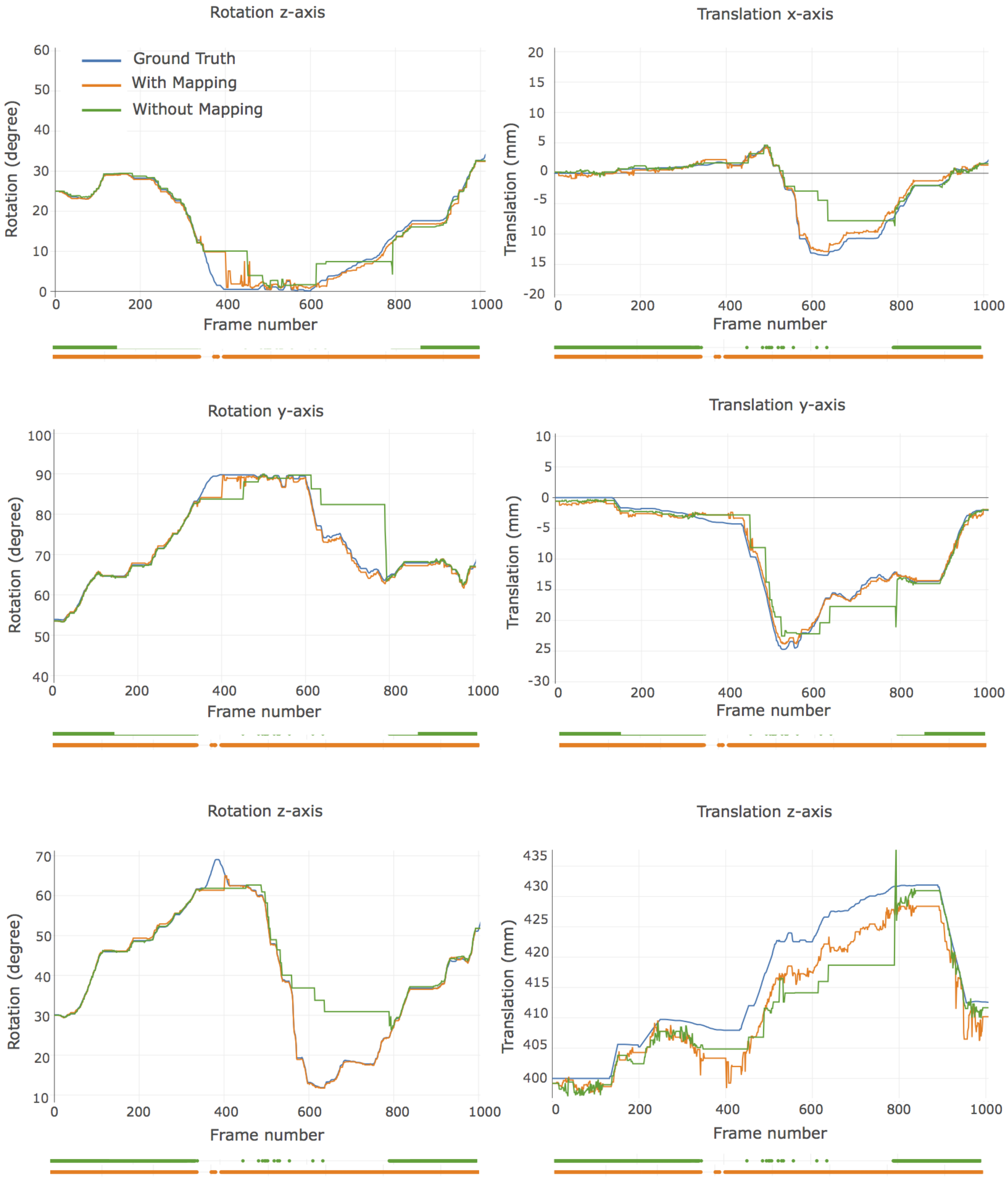


Figure 2.8: **6 DOF camera pose plots of the chopping board in a synthetic generated image sequence.** The line plots below the diagrams indicate successfully tracked frames. In general, the presented approach (orange) consistently achieved better results than tracking without mapping and refinement (green).

Moreover, tracking without a mapping and refinement step depends strongly on the quality of the input edges. There is no general statement about the selection of good edges to track. Increasing the number of input edges, for example, does not necessarily mean that the accuracy or robustness increases. By the way of demonstration, the root mean square (RMS) errors were calculated for the sample sequence with different numbers of input edges, as shown in Table 2.1. The generality of the presented system for different input edges can be seen. It was able to track the scene with different input edges with similar results in terms of accuracy and robustness.

Method	Edges (#)	Frames (%)	RMS (root mean square) error					
			R_x (degree)	R_y (degree)	R_z (degree)	t_x (mm)	t_y (mm)	t_z (mm)
With	5	90.9	0.59211	0.51526	0.59501	0.56182	1.19744	2.17623
Without		35.7	0.54594	0.43345	0.22434	0.59579	0.64225	3.20885
With	7	100	0.56358	1.30319	0.77255	1.20516	1.39944	5.42438
Without		100	1.10871	1.83799	1.03849	1.78927	1.23728	6.76075
With	8	100	0.94687	1.68201	0.93394	1.66943	1.31145	6.05157
Without		100	1.26483	2.06454	1.23755	2.12224	1.32286	7.64757
With	9	94.6	0.54248	0.84202	0.55344	0.90416	0.294224	5.09546
Without		73.1	1.22008	2.11999	1.03555	2.23612	0.563874	8.63815
With	10	92	0.31508	0.36303	0.45056	0.38775	0.86188	1.74504
Without		56.4	0.34329	0.32387	0.19249	0.24481	0.50929	2.01651

Table 2.1: **Root mean square (RMS) error in a synthetic generated sequences.** The upper rows show the results of the proposed method; the lower rows show the results of the method without any refinement or mapping procedure. The number of input edges is shown in the second column. The third column shows the percentage of tracked frames.

2.10.2 Real Image Sequences

The method was tested with real image sequences, which represented a much greater challenge than the synthetically generated scenes. Real image scenes contain noise, motion blur and different lighting conditions. A simple model of a house with many planar surfaces and textures was used. The surface model of this object and the coarse edge model were generated manually using a standard modeling tool. Only the surface model of the object to be tracked was available. Therefore, new edges were mapped to the edge model only if these lay on the polygon model. Figure 2.9 shows the tracking results with and without mapping and refinement. The tracking with mapping and refinement benefited from stored keyposes for re-initialization, as seen before in the synthetic scenes. Tracking with refinement and mapping achieved 98% tracked frames. Tracking without this procedure only achieved 83% tracked frames.

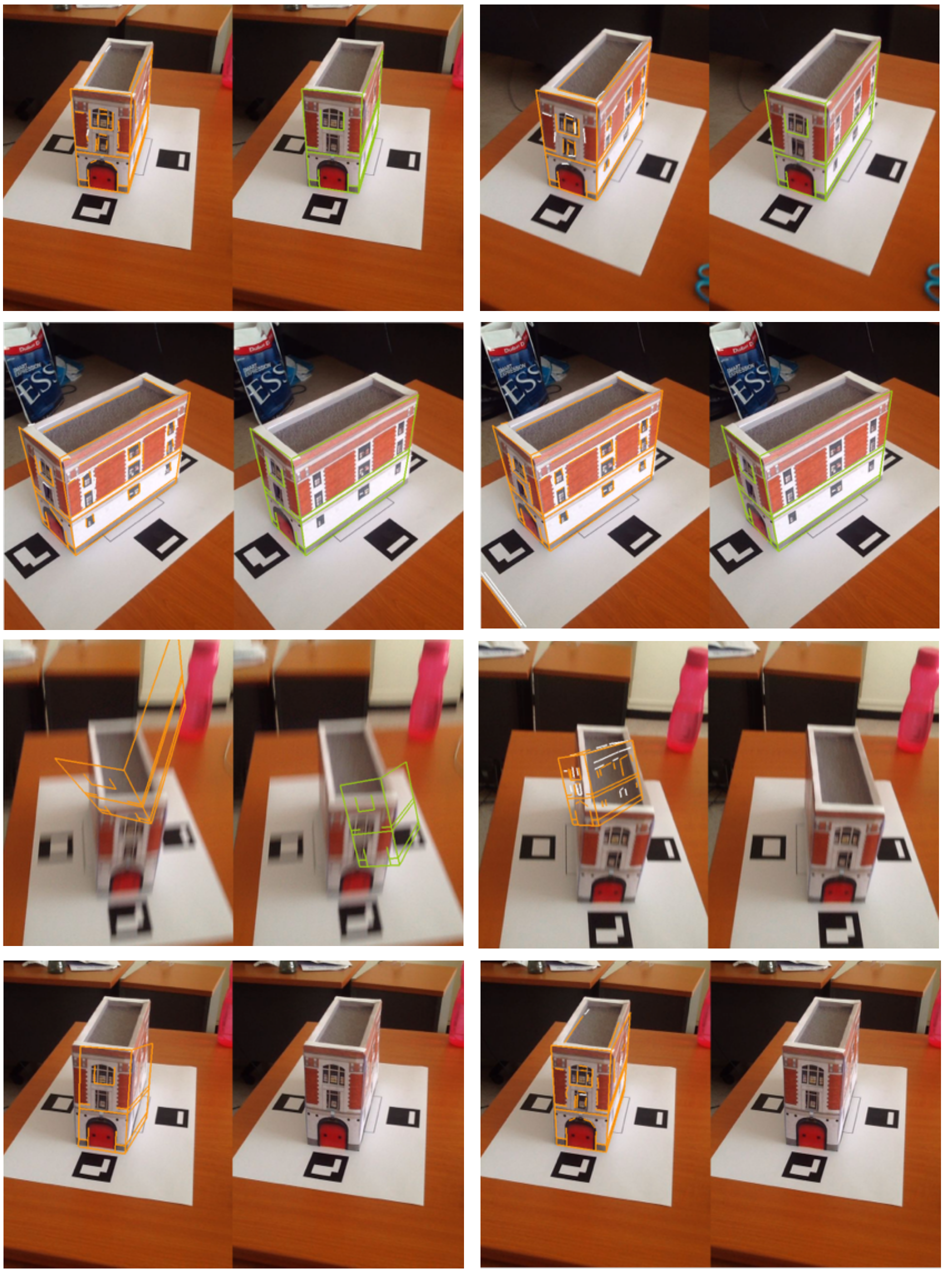


Figure 2.9: **Tracking results of a real scene.** The effectiveness of the keyposes for re-initialization is shown. Results with the proposed mapping and refinement step (orange edges) and without this step (green edges) are shown in the real sequence. The white edges were used only during the mapping and refinement step. They were not used for tracking. The markers were used only to initialize the tracking step.

2.11 Conclusion and Future Work

Model-based tracking using appearance-based line models presents a suitable approach to realize robust tracking for AR applications. This approach achieves sufficient runtime performance (≥ 30 frames per second) at a standard image resolution of 640×480 pixels. The main advantage of the algorithm is that it does not require extensive pre-processing steps, such as modeling a suitable line model. It also offers the advantages of model-based tracking in the context of AR: 1) no modification of the scene is required; 2) collision and occlusion handling can be realized by the surface model; 3) in contrast to well-known SfM approaches, virtual content can be registered in the coordinate system of the model; and 4) drift-free tracking.

Starting from a coarse line model, existing lines are refined and new lines are added as the user moves around in the scene or around the object to be tracked. New lines are detected and extracted from the incoming camera stream so that they are likely to appear in the next frames. So called keyposes enable the selection of view-dependent and appearance-based lines and make the tracking process more robust. The method uses lines that are likely to be visible and thereby good to track depending on the predicted viewpoint; in other words, it improves the accuracy of the correspondences. For example, the model is more detailed if the user is very close to it whereas the model becomes more coarser if the user moves away from it.

Generally, the approach can deal with several types of scenes or objects. It performs well in poorly textured scenes, but also performs well in textured scenes in which only sparse geometric structures are found. Moreover, in contrast to state-of-the-art approaches it adapts to changes in the environment or object to be tracked. The approach benefits from the use of keyposes that enable a semi-automatic re-initialization procedure after tracking failures. It outperforms the standard edge-based trackers which often require manual re-initialization. However, the user still has to go back to a previously visited view. The option to perform initialization and re-initialization steps automatically offers the use of tracking-by-detection approaches. Lines of the model could be enhanced by line descriptors that describe either a patch around their 2D projection in the camera stream (as in [HS12; LZL$^+$13; NSS$^+$16]) or their geometric structure in 3D, for example, if these represent orientation or texture edges.

Currently, the tracking process fails if the model to be tracked is non-visible. To overcome this limitation, future work should concentrate on the enhancement of the model beyond the known part. For example, special sensors like depths sensors or stereo cameras could be useful to obtain the 3D coordinates of line endpoints within areas of the camera image where the surface structure is unknown. Thus, tracking would no longer be limited to regions in which the model is visible, and the user would be able to move freely within the scene.

Chapter 3

Optimizing 3D Reconstructions: Propagating Surfaces, Filling Holes and De-noising

This chapter deals with the 3D reconstruction challenge in AR. An approach that extends, fills holes, and de-noises an ongoing surface reconstruction is presented. The presented algorithm is integrated in the well-known KinectFusion pipeline [NDI$^+$11].

This chapter is organized as follows. The challenge of reconstructing a suitable model of the environment for AR applications is discussed in Section 3.1, followed by a description of the KinectFusion pipeline and the technical background in Section 3.2. Section 3.3 provides an overview of the related work. Section 3.4 describes the idea of the algorithm and Section 3.5 outlines the pipeline of it. Sections 3.6 to 3.11 describe the algorithm in detail. Section 3.12 explains an easy way to realize occlusions and collisions on the implicit representation of the scene. The results are presented in Section 3.13 and the conclusion follows in Section 3.14. Parts of the work presented in this chapter were peer-reviewed and presented at a conference [HHM16].

3.1 3D Reconstruction Challenge

In addition to an accurate tracking system, an accurate and complete reconstruction of the environment is fundamental in AR. While it is true that virtual objects can only be positioned precisely by an accurate tracking approach, such objects should be able to interact with the real scene. For example, virtual objects in motion should not move through real objects but should collide with them.

Generally, an accurate 3D model of the real environment offers many possibilities for creating a believable and realistic AR application. Registration of the 3D model often called *phantom* so that it precisely matches its real counterpart, and hiding the phantom from the user, allows interactions between the real and virtual components to be realized. Beside collision detection and handling via standard computer graphics algorithms, the model can be used to realize occlusions in static scenes. The model can also be used to determine the spatial relationships between virtual and real objects, so that virtual objects can be occluded by real objects instead of constantly appearing in front of them, regardless of the relationship. By considering both real and virtual objects in the virtual scene, a realistic rendering with shadows, lightings and reflections can be realized [IKH+11]. For example, researchers [HDH03; KBS16] used a phantom to cast shadows from virtual objects onto real objects and vice versa (Figure 3.1). Virtual objects appear more realistic and give a better perception of depth; the immersion of the AR experience is improved.

In addition to offering several interaction opportunities, phantoms are useful for creating special effects in AR applications, such as reality diminishing [CGB+09], rearranging real world objects [KSW+11], x-ray visualizations [KSW+11], and re-texturing [BRF01; MNZ+15]. For example, Cosco *et al.* [CGB+09] used a phantom model to determine the region of the real scene (a haptic device) to be diminished.

Accurate tracking is a prerequisite for realizing types of interactions decribed above. The phantom model must be well-registered with the real one. Moreover, ideally the

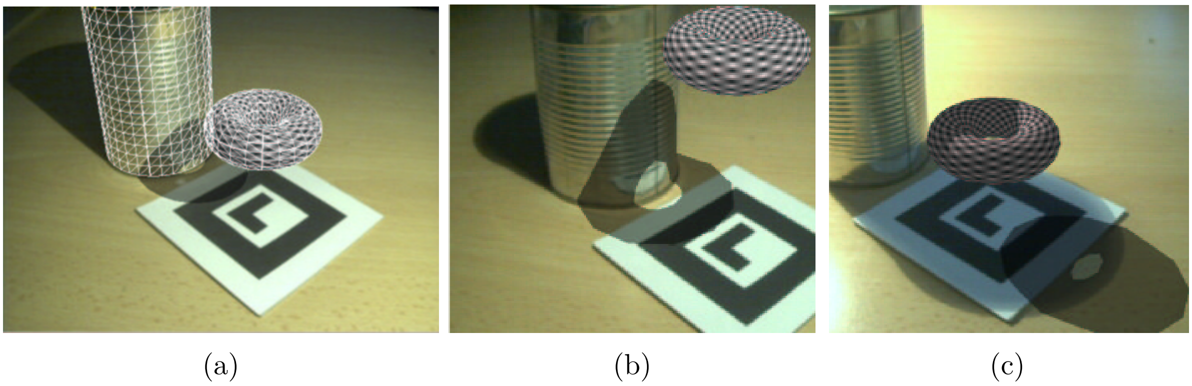
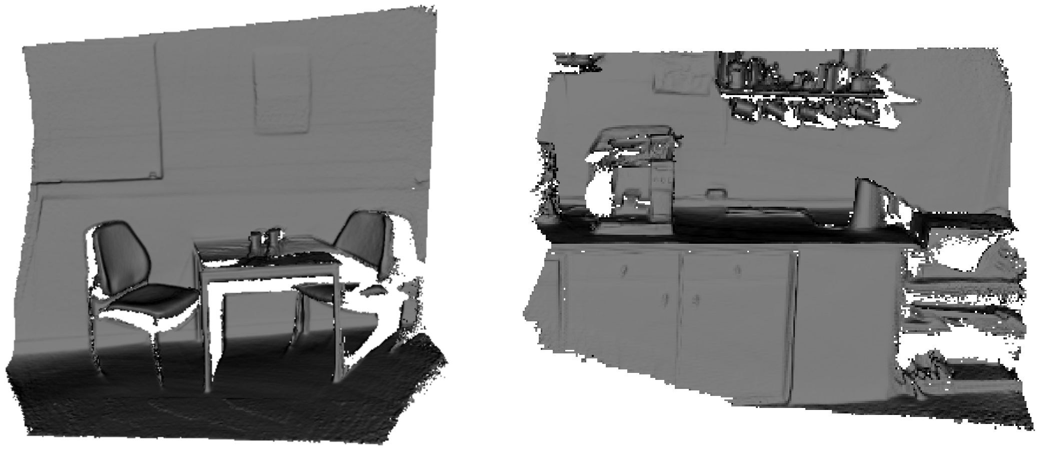

(a) (b) (c)

Figure 3.1: **An example of using a phantom model for shadowing.** (a) shows the model of the virtual object (torus) and the phantom model of the real object (can), (b) shows a shadow cast onto the real can and (c) shows a shadow cast from the can onto the torus. Image source: [HDH03]

model should be complete and accurate. Phantom models can be modeled manually using standard modeling software. However, reconstructing the scene before the AR application starts, or while it is running, takes far less work.

Reconstruction of the environment is a fundamental task in several computer vision areas, such as in path planning to navigate robotic devices on surfaces [BS12]. Hence, this become an important field of research. Currently algorithms can reconstruct a mostly dense and accurate surface model. Nevertheless, the results are often incomplete as parts of the scene are non-visible because of phenomena such as occlusion by other objects [SSG$^+$14], self-occlusion, or other issues in the field of reconstruction such as specular reflectance (as described in [WC07b]). Incomplete models of a scene, for example as visualized in Figure 3.2, are unsuitable for detecting and handling collision in a plausible manner. For instance, a rolling virtual sphere moves through the wall behind the chairs instead of colliding with it (see Figure 3.2a).

Currently, it is common to perform an offline capturing step to create a suitable reconstruction to be used as a phantom. The Vuforia AR SDK [SDKb] or the Microsoft HoloLens SDK [SDKa], for example, allow developers to create applications in which the user can scan the target objects or the environment before the main applications starts. During the capturing process gaps and holes can be filled if the user moves the sensor



(a) Coffee scene (b) Kitchen scene

Figure 3.2: **Examples of incomplete 3D surface reconstructions.** (a) and (b) show two different incomplete reconstructions obtained by the well-known KinectFusion algorithm (described in Section 3.2). The small holes are mainly caused by occlusions.

slightly so that the scene is recorded from different viewpoints. This method is time consuming and impractical, and a precise and complete model can be generated only if every object in the scene is obtained from different viewpoints using this procedure. A user who lacks expert knowledge will find it very complicated to fulfill this task. For example, due to the upright posture of the user, the area below the table is often not captured; which results in large holes and gaps in the surface.

That is why the reconstruction usually requires an optimization and a post-processing step. Occluded and unobserved regions are filled and this enables an almost realistic collision detection and handling in static environments. However, this technique cannot deal with dynamic environments. If new objects are placed in the observed scene, they are not part of the phantom model and virtual objects cannot collide with them. Moreover, the interaction between the real and virtual worlds is limited to the captured region of the user's surroundings. The user and the virtual objects cannot move freely.

Instead of acquiring a suitable model in an offline procedure the goal should be to generate the model during runtime of the AR application. Holes and gaps should be filled during the reconstruction procedure. In this way no preceding capturing step is required. Furthermore, the user can explore a dynamic environment without restriction. Reconstructing the environment during runtime of the AR applications leads to two other problems. The reconstruction changes with every new observation which results in wobbling surfaces and topology changes. Moreover, the reconstructions are noisy especially at the beginning [DSM+17]. Noise can impair the collision handling because of noisy surface normals, and rapid changes in geometry lead to problems during physical simulations. For example, if a planar surface changes with each observation, a static sphere will roll continuously.

3.2 Background to KinectFusion

KinectFusion [NDI+11] realizes a dense surface mapping and tracking algorithm for complex room-sized scenes, using a Kinect depth sensor. The pipeline reconstructs a single dense surface model by integrating incoming depth data from multiple viewpoints into the reconstruction, which is stored in a uniform voxel grid. The volume represents the surface by a truncated signed distance function (TSDF). The 6 DOF sensor pose is simultaneously tracked using the depth data. Therefore, the incoming depth data are aligned to the global model of the scene using a coarse-to-fine iterative closest point algorithm (ICP). In this way, frame-to-model tracking is realized that is more suitable for AR than mapping and tracking algorithms that perform frame-to-frame tracking. Frame-to-model approaches eliminate drift whereas frame-to-frame tracking approaches are sensitive to drift [NDI+11].

3.2.1 Truncated Signed Distance Function

The KinectFusion pipeline represents the surface by an implicit function, the truncated signed distance function (TSDF). Generally, signed distance functions (SDF) define the surface by the *zero-crossing*. The SDF values represent the distances to the nearest point on the surface (the closest zero-crossing). The sign indicates the location of the point in relation to the surface. As shown in Figure 3.3, negative values are inside the surface and positive values are outside the surface.

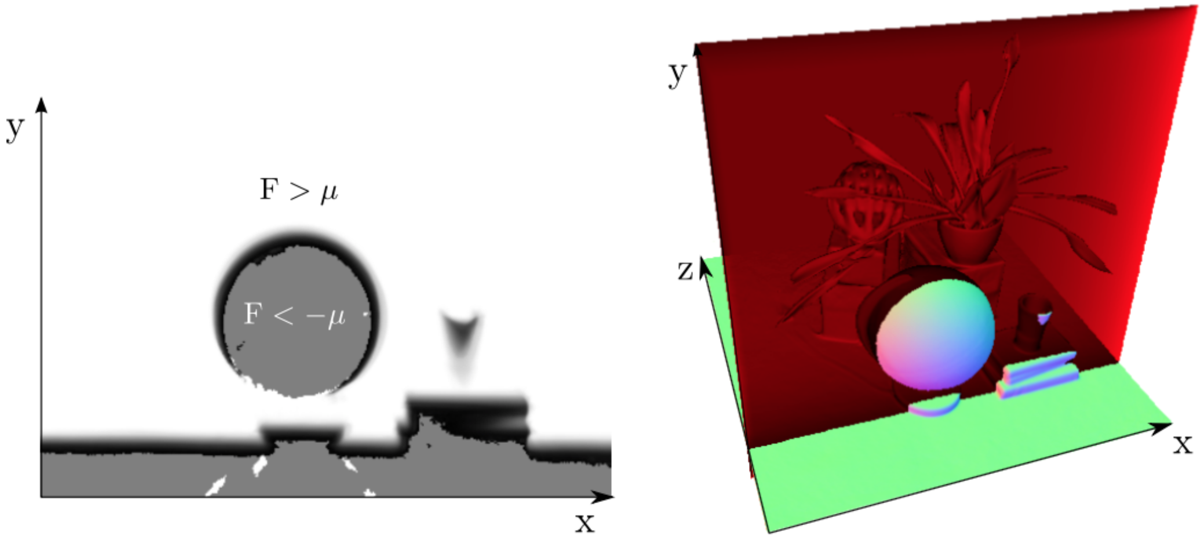


Figure 3.3: **Example of a TSDF volume.** The figure visualizes the TSDF values F of a slice (left) from a reconstruction volume (right). The slice shows the interpolated distances to the surface from positive values (white) $F > \mu$ over negative values (black) to unmeasured values $F < -\mu$ (gray). Image source: [NDI+11]

KinectFusion uses a discretization and clamped variant of the SDF. The environment is represented by a uniform and fixed-sized 3D voxel grid allocated on the GPU. Each voxel contains the clamped signed distance to the surface and a weight. Hence, for a global point $p \in \mathbb{R}^3$ within the 3D volume S_k, the current truncated signed distance value $F_k(p)$ and the weight $W_k(p)$ can be obtained by:

$$S_k(p) = [F_k(p), W_k(p)]. \tag{3.1}$$

As shown in Figure 3.4, the distance is truncated at $\pm\mu$ and scaled to $[-1, 1]$. In addition, values increase for visible parts of the captured scene from the surface to free-space, with $F_k(x) = 1$, and decrease for non-visible parts until $F_k(x) = -1$ is reached. The weight indicates the confidence of an observed TSDF value and is used to integrate incoming TSDF values by a weighted average. Voxels where the state is unknown are clearly defined by zero $\{p \in \mathbb{R}^3 | W_k(p) = 0\}$ and should be filled to generate a watertight surface.

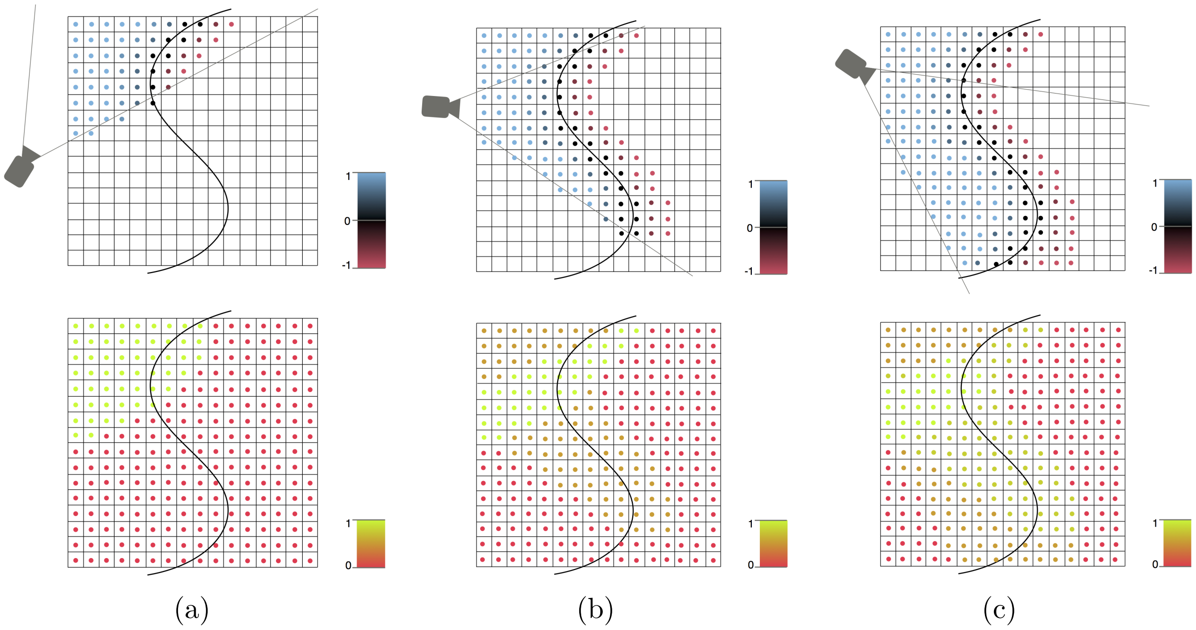


Figure 3.4: **Schematic visualization of the function S_k in a 2D voxel grid representation for three consecutive observations.** Panels (a) to (b) show different observations for consecutive frames k with $k \in \{0, 1, 2\}$. The upper voxel grids show the status of the TSDF value (F_k) schematically for each frame k. The surface to be observed is black and the distance is coded as color: a sign change is green; positive distances increase from blue to green and negative distances decreases from green to red. The lower voxel grids visualize the weights (W_k) for each frame k. The weights are normalized for current maximum weight in the whole volume.

Generally, signed distance fields are attractive for surface reconstruction in AR applications enhanced with virtual content. Collision detection algorithms for rigid bodies or deformable objects can be realized easily, as shown by Fisher and Lin [FL01] and Bridson *et al.* [BMF03]. Moreover, occlusions between the real and virtual objects can be achieved by ray casting. There is no need to transform the volume to an explicit surface representation. Section 3.12 demonstrates efficient collision detection as well as occlusion with the TSDF representation.

3.2.2 Overview of the Pipeline

The KinectFusion pipeline is first initialized by placing the volume and its origin of the coordinate system into the real scene. After initialization, the pipeline uses the depth data obtained from the Kinect device and processes the following four steps (as described in [NDI^{+}11; IKH^{+}11]):

A) **Surface measurement** applies a bilateral filter to the raw depth data to reduces the noise while preserving depth discontinues and calculates a dense vertex and normal map for a given frame k. The vertex map V_k is obtained through a back-projection process for the filtered depth frame. The calculation of the corresponding normal map N_k is based on the cross product of neighboring vertices.

B) **Pose estimation** performs a projective ICP to estimate the pose of the camera. This step aligns the surface measurements (V_k, N_k) of the current frame k against the surface prediction from the previous frame $(\hat{V}_{k-1}, \hat{N}_{k-1})$ to realize frame-to-model tracking. Generally, the ICP algorithm aligns 3D shapes by minimizing the distance between two point clouds. It iteratively adjusts the camera pose starting from an initial guess to minimize an error metric (usually the sum of squared differences between point correspondences). In the KinectFusion algorithm, the point correspondences are obtained through projection, so that the error can easily be calculated by a point-plane metric per pixel.

C) **Volume integration** performs a global scene fusion process. First, the surface measurements (V_k, N_k) are transformed from camera coordinates to the global coordinate system of the volume. Therefore, the camera pose determined by the previous step is used. Second, new data are integrated into the 3D voxel grid representation.

D) **Surface prediction** performs a ray casting of the TSDF to provide a dense surface prediction. The prediction consists of a vertex and normal map $(\hat{V}_k, \hat{N}_k)$ for the frame k. This prediction is used in the next frame $k + 1$ to estimate a frame-to-model pose. This step performs a per-pixel ray cast: it traverses all voxels along the ray (defined from the center of the sensor through the pixel) until a zero-crossing (a sign change of the TSDF values) is found. The intersection point is finally obtained by linear interpolation. The surface normal is estimated directly as the derivative of the TSDF.

The data association and pose estimation work with a coarse to fine strategy using a three level vertex and normal map pyramid. They start with the coarsest level and perform an ICP minimization to estimate the pose of the sensor, based on the pose obtained in the previous frame. Then this pose is used to perform an ICP minimization on the next level, and so on.

Each component of the pipeline has a parallelized structure, so that the pipeline is fully realized using the advantage of GPU hardware to achieve real-time capability. A detailed description appears in [NDI+11; IKH+11].

3.2.3 Limitations and Improvements

The KinectFusion system was one of the first approaches that was capable of reconstructing high-quality 3D models in small indoor scenarios [DSM+17]. Several improvements have focused on different aspects of the algorithm. Most of the extensions have aimed at extending the size of the scene the system can reconstruct or on improving the accuracy of the pose estimation.

The TSDF voxel representation enables accurate and precise reconstructions, but is very memory-intensive [KDS+15]. Hence, high-precision reconstructions are realizable only for scenes of limited size, such as single objects, small workspaces, or rooms.

KinectFusion's pose estimation relies on depth information; therefore, the accuracy of the estimation decreases if insufficient geometry features are available. Examples are when the camera is pointed at a flat wall or corridor that lacks significant 3D features. The ICP algorithm, moreover, assumes small motions from frame-to-frame and fails if the camera moves fast or abruptly [WJK+13].

Roth and Vona [RV12] presented *Moving Volume KinectFusion* which automatically moves the volume through space as the camera translates and rotates to provide both visual odometry and a dense spatial map of the local environment. Their system no longer store data of non-visible voxels. In contrast, Klingensmith *et al.* [KDS$^+$15] concentrated on reducing the amount of data to store. They presented an approach for mobile devices that uses a dynamic, spatially-hashed TSDF for mapping and localization. Unnecessary calculations and memory use are avoided by culling parts of the scene that do not contain surfaces. Whelan *et al.* [WJK$^+$13] integrated color information into the reconstruction and combined various odometry estimation techniques, which uses both dense geometric and photometric constraints to increase the tracking robustness across a various environments (e.g. desk size or corridors).

In AR scenarios, tracking is fundamental to register virtual objects and the reconstruction of large-scale environments offers new opportunities to move around attributively. However, quality is no less important. As mentioned before, reconstructions should ideally be complete, without holes or gaps, and should be geometrically consistent between consecutive frames. At present only a few approaches concentrate on optimizing reconstructions of the real world during the capturing process. For example, one strategy entails fitting planes to the TSDF to fill small holes or to reduce noise on flat surfaces. These methods are discussed in detail in Section 3.3.2.

3.3 Related Work on Optimizing 3D Surface Reconstructions

This section discusses relevant work in the field of plane segmentation and fitting. Planes are very useful in urban environments for the optimization of 3D surface reconstructions. Thereafter, current approaches for optimizing 3D surface reconstructions are presented.

3.3.1 Plane Segmentation and Fitting

Plane extraction and fitting techniques are useful for several applications in urban environments where surfaces are mainly planar. Planes can be used, among other things, to simplify the geometry [WGS03], to reduce noise [DSM+17], to obtain semantic information [HHR+12], or to perform a planar SLAM [LLL+12; TJR+13]. Holz *et al.* [HHR+12] used detected plane segments to identify obstacles and segment graspable objects on the floor in household environments or in table top scenarios. At present, most approaches work on 3D point clouds that are obtained from laser scanners, RGB-D or stereo cameras. These approaches can be divided into three categories [DG10; OSW+11], namely: region growing methods, Hough transform-based methods, and methods that follow the random sample consensus (RANSAC) procedure.

Region growing methods start with one or more seed regions and expand them iteratively. Neighbors are merged to the seed region if they satisfy certain conditions [DG10]. Most of these methods work on organized point clouds, such as those obtained from a depth sensor, where the neighbors are easy to identify.

Poppinga *et al.* [PVB+08] used an organized point cloud as input and followed a point-based procedure and grew planar regions by adding neighboring points. The algorithm starts with random points to define a seed plane. A neighboring point extends the plane if it is likely to be a part of the plane – that is, if the point-plane distance and mean square error are below thresholds. The region grows until no more suitable points can be found. The centroid and covariance matrix for estimating the plane parameters are updated incrementally. Thus, the results are mostly accurate, but the method is somewhat inefficient because complex computations, such as the calculation of the covariance matrix, are performed very often.

Holz and Behnke [HB13] presented an approach for reconstruction of the surface as mesh and segmentation of the range image by growing regions using the local mesh neighborhoods. They processed four main steps: 1) approximating a mesh from an image neighborhood; 2) approximating local surface normals and estimating curvature by using this mesh neighborhood; 3) using bilateral filtering to smooth points and normals; and 4) segmenting points to a given model, such as a plane, by region growing.

For plane segmentation, seed planes are defined by a centroid and a normal. A neighbor is merged with the plane if the angle between its normal and the normal of the plane, as well as the point-plane distance, both lie below a threshold. To avoid complex calculations the algorithm updates only the centroid and normal instead of incrementally updating a covariance matrix to derive a plane normal.

These methods use point clouds that are organized in an image-point structure to compute the nearest neighbors. They cannot be directly applied to (unorganized) point clouds that are obtained from multiple views. They are only able to consider the data that are visible from the current view.

Deschaud and Goulette [DG10] used a voxel representation instead and presented an algorithm to detect planes in unorganized and noisy point clouds, using filtered normals and voxel growing. First, their system uses an octree to split the point cloud into voxels and estimates normals. Then the system estimates a score of local planarity for each point and its neighborhood. The point with the best score is then used as seed to start the region growing by voxels. The algorithm detects large and small planes in massive data sets and extracts connected components. However, the voxel growing method has a complexity of O(N) and needs a processing time up to a minute or more (e.g. 452 seconds for 398k points, ~ 1 millisecond per point).

It can be concluded that voxel growing approaches are often not efficient enough to optimize planar regions during the reconstruction process.

Hough transform [Hou62] is a fundamental algorithm to find multiple instances of objects with specific shapes that are parametrizable. This is achieved through transformation to a parameter space and a voting procedure. Typically, the method is used to find regular curves such as lines or circles in 2D images. Calculations with 3D data sets have not received much attention to date [BEL$^+$11] .

Boormann *et al.* [BEL$^+$11] presented variants of the Hough transform to detect planes in a 3D point cloud. In their system, each plane in $\mathbb{R}^3$ is defined by the hesse normal form, a normal vector perpendicular to the plane and a distance to the origin. The 3D Hough space (θ, ϕ, d) is defined by the normal in polar coordinates θ, ϕ and the distance to the origin d. To find planes, each point of the point cloud is transformed into this parameter space and votes for all planes on which the point may lie. Clusters with many votes in the Hough space then represent the detected planes. The 3D Hough transform approaches tend to be slow in practice and the results are imprecise, as these methods require discrete data [WGS03].

Holz *et al.* [HHR$^+$12] used two parameter spaces and an iterative procedure to realize a fast plane segmentation in point clouds obtained via a RGB-D camera. First, the approach estimates local surface normals and clusters the point cloud in the normal space to obtain plane candidates. Second, it clusters each candidate group by its distance from the origin. Discretization effects are compensated using a post-processing step, which merges similar neighboring segments. However, unconnected planar segments might become merged into the same cluster.

RANSAC, presented by Fischler and Bolles [FB81], is an iterative algorithm to find a model for a dataset that may contains outlier. RANSAC-based plane fitting methods, such as [DG10], usually fit planes to a model by 1) iteratively selecting three random points to describe a plane mathematically; 2) calculating the inliers (e.g. all points with a distance to the plane below a threshold); and 3) repeating these two steps until a good plane is found or a certain number of iterations is reached.

Several approaches [TRC12; TJR$^+$13] process the RANSAC procedure several times to obtain multiple planes. The approaches first detect the globally significant plane in the whole point cloud, remove all inliers from the point cloud, and perform these two steps to find the next significant plane. These steps are repeated until a specific number of planes is found, or until no more significant planes (with a certain number of inliers) can be found.

In contrast to region growing approaches, approaches based on Hough and RANSAC are processed globally. Plane segments are merged if they share a similar orientation and distance from the origin. Hence, points of one plane segment are not necessarily connected with each other [HB13].

To overcome this problem and to increase the speed several approaches apply the RANSAC procedure only to local regions, instead of the entire point cloud. Afterwards, planes are merged to obtain global planes.

The approach, presented by **Weingarten *et al.*** [WGS03], splits the set of points into equally sized 3D cubes and fits a single plane to each cube using RANSAC. Plane segments are afterwards merged through a region growing approach. For every cube, the neighbors are merged if two constraints are satisfied: the best fitting planes of the cubes are similar in their orientation and their centers of gravity lie close together.

Lee *et al.* [LLL+12] presented a rapid approach to extract planes in RGB-D sensor data by processing a segmentation to obtain plane candidates first. The approach estimates normals and edges from the incoming depth map and uses them to split the image into segments of planar structures. Segments having a certain number of pixels are potential candidates for planes. RANSAC is then applied to these candidates and segments having sufficient inliers are refined.

The method, proposed by **Hulik *et al.*** [HBS+12], segments a depth map into planar regions by splitting the image into tiles. The algorithm computes the normals and processes RANSAC for each tile. If a plane is found, a seed-fill procedure is performed to obtain planes with connected components.

Other methods also exist that cannot be categorized into the three groups. For example, **Feng *et al.*** [FTK14] divided an organized point cloud into uniform and non-overlapping groups in image space. Their approach constructs a graph that represented the groups and their neighborhood. Then an agglomerative hierarchical clustering on this graph systematically merges graph nodes belonging to the same plane. This is repeated until the mean squared error of plane fitting exceeds a threshold. Lastly, the algorithm uses a pixel-wise region growing to refine the plane.

Generally, it can be concluded that fast algorithms mostly work on organized point clouds, such as those obtained by an RGB-D device, and do not consider previously obtained surface information. Approaches that work on point clouds obtained from multiple views of the surface are much more accurate. However, they are often too slow to achieve real-time performance.

3.3.2 Hole Filling, De-noising and Propagating Surfaces

Filling holes and gaps in a surface reconstruction is an important issue and many approaches have already been proposed. Many of these use an implicit representation.

One method is to fit functions to the distance field first and to extract the iso-surface from these functions afterwards. Carr *et al.* [CBC+01] used radial basis functions to describe a smooth and closed surface. Marschner *et al.* [DMG+02] presented another method that extends the signed distance values from the boundaries of the observed parts to the whole volume. This process is called *diffusion* and is done by alternating steps: blurring and compositing. This method is explained in detail in Section 3.10.

Most of these approaches are performed as a post-processing step after the reconstruction and thus are not designed to achieve real-time performance. Moreover, solving problems like wobbling surfaces, hole filling, and noise reduction in real-time during the acquisition phase has not yet received much attention [DSM+17].

Silberman *et al.* [SSG$^+$14] focused on extending planar surfaces to fill occluded and non-visible parts of the scene. Their system extends the KinectFusion pipeline by accessing the TSDF volume and filling the unknown parts. The system first extracts planar surfaces and classifies each with a label (e.g. floor, wall) or as a part of an internal object. The system uses a Hough transform to select a finite set of planes. It then sorts the candidates in descending order, depending on the votes received during Hough transform. After that, the algorithm iteratively associates points to each plane that are likely to correspond to it. Planes are then extended by solving a labeling problem. However, this pipeline is currently implemented only in Matlab [Mat] and does not achieve real-time rates; up to a minute of processing time is required.

Dzitsiuk *et al.* [DSM$^+$17] de-noised, stabilized and filled holes of a SDF representation during a reconstruction of a scene in real-time by fitting planes into the reconstruction. Their approach splits the volume into equally sized regions and fits local planes to the SDF. Then it merges these local plane candidates to find globally consistent planes. Dzitsiuk *et al.* presented different approaches to perform both steps, namely local plane fitting and merging.

To estimate local planes, the researchers presented a RANSAC and a least squares approach. To perform a traditional 3-point RANSAC procedure (see Section 3.3.1), the system generates 3D points by creating a mesh for each local region of the volume. Hence, each local region represents the dominant plane. The least squares approach does not extract an explicit representation. It fits planes directly to the local SDF volumes by minimizing the difference between the distance from the plane to the voxel center and the SDF. Because a local region of the volume can contain more than one plane, and to provide robustness against outliers, a weighted least squares method is used.

Larger planes, such as walls, are reconstructed by merging them at a global level and propagating them back to small volumes. This is done by a 1-point RANSAC algorithm or a greedy region growing. The RANSAC method selects a plane candidate randomly, identifies inlier planes and refines the plane having the largest set of inliers. Region growing, in contrast, selects one candidate, performs a breadth-first traversal of its neighbors, and merges similar planes. The main drawback of the region growing method is to find a good candidate to start with.

Dzitsiuk*et al.* [DSM$^+$17] implemented these different methods on the central processing unit (CPU) of a mobile device. The researchers concluded that the least squares fitting to find plane candidates, followed by a plane merging by RANSAC, was the fastest combination. Using this combination, they were able to obtain interactive frame rates.

To reduce noise, the SDF values are adapted to the global planes. To stabilize the estimated planes which continued to change with each new observation the system updates only those planes that differs by more than a degree. The hole filling is processed by extending planes to unfilled voxels. Therefore, the refined planes are propagated to intersected volumes in a specific range.

3.4 Propagating Surfaces, Hole Filling and De-noising during the 3D Reconstruction

To date, few approaches have concentrated on optimizing ongoing surface reconstructions (described in Section 3.3.2). Existing approaches have followed the same procedure: they fit global planes to the reconstruction and use them to de-noise and fill holes in planar regions. This method clearly works relatively well in man-made environments in which many flat structures occur be found (e.g. table tops, walls or floors). But, taking a closer look at objects in man-made environments, it can be noted that also a lot of non-flat structures can be found (e.g. cups, bottles, sinks or balls).

One of the main contributions of this thesis is an algorithm that can reduce noise, fill holes, and extend unknown parts of an ongoing reconstruction according to the geometric structure of the scene. In contrast to state-of-the-art methods, the proposed method generates watertight and visually appealing geometric reconstructions on several types of surfaces. The algorithm is integrated in the widely used KinectFusion reconstruction pipeline [NDI+11]. The KinectFusion system uses a voxel grid representation of the scene and reconstructs a dense surface model by integrating the depth data from the sensor, as described in Section 3.2. The extension de-noises the partial dense reconstruction of the observed surface and extends it to fill unknown voxels so that holes and gaps where depth data is not captured are filled in. As the camera moves, new incoming structure data are continuously integrated so that hypothetical filled in voxels are refined. Therefore, the algorithm continuously propagates the actual observed TSDF values towards unknown parts of the volume. First, it estimates local planes within subregions of the volume and uses them to extend and to de-noise the current observed planar structures of the scene. Note: local planes are signified *local plane priors* in further descriptions of the algorithm, similar as in [DSM+17]. If this process is finished, a diffusion algorithm continuously fills in small holes and extends curved structures of the scene to generate a watertight and complete reconstruction. In contrast to current methods that estimate global planes, the presented approach uses local planes, fits TSDF values to locally merged planes and provides smooth transitions between subregions. Thus, it does not require a step to merge planes globally. Furthermore, existing methods are implemented on the CPU whereas the presented approach is completely integrated into the KinectFusion pipeline and uses the capabilities of the GPU hardware. Thus, there is no need to transfer the reconstruction data to the CPU.

3.5 Method Overview

The pipeline takes the depth data from a sensor and applies the following six main stages to de-noise the current reconstruction and continuously fill unknown parts of the volume. The stages are shown and labeled in Figure 3.5.

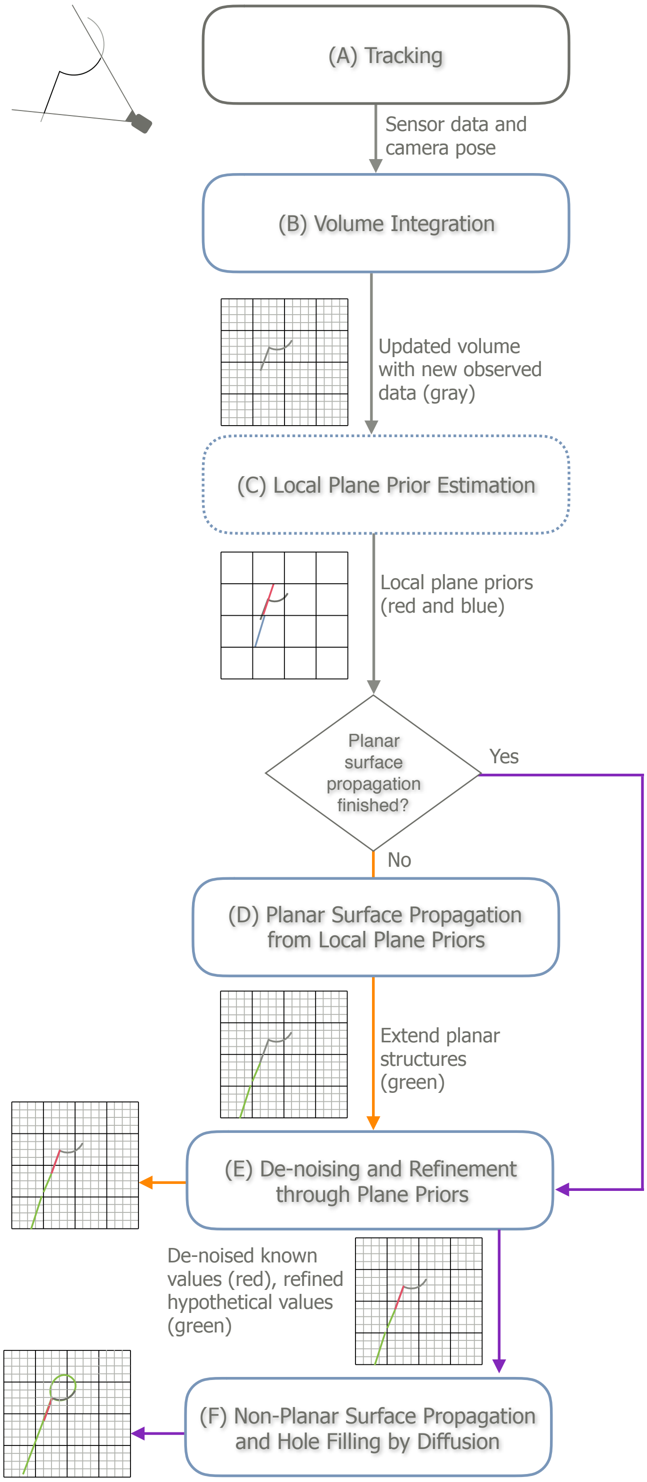

(A) Tracking
Sensor data and camera pose
(B) Volume Integration
Updated volume with new observed data (gray)
(C) Local Plane Prior Estimation
Local plane priors (red and blue)
Planar surface propagation finished?
Yes
No
(D) Planar Surface Propagation from Local Plane Priors
Extend planar structures (green)
(E) De-noising and Refinement through Plane Priors
De-noised known values (red), refined hypothetical values (green)
(F) Non-Planar Surface Propagation and Hole Filling by Diffusion

Figure 3.5: **Overview of propagation, hole filling and de-noising pipeline.** Visualization of main stages of the algorithm to propagate, fill and de-noise a reconstruction. The algorithm uses the depth data of the scene from a sensor and estimates the camera pose, integrates new incoming depth data, and optimizes the ongoing surface reconstruction. A short description of each stage appears in Section 3.5.

A) **Tracking** uses the depth data obtained by the sensor to perform projective ICP tracking to estimate the camera pose (described in Section 3.2.2).

B) **Volume integration** integrates new incoming depth data to the current reconstruction. New observed data are fused to the current surface data. Hypothetical surface data (which are created in Step D and Step F) are adapted accordingly.

C) **Local plane prior estimation** splits the volume into equally sized subregions, called to hereafter *subvolumes*. The step estimates the dominant local plane of the actual observed surface for each region.

D) **Planar surface propagation from local plane priors** continuously extends planar structures (like walls or floors) towards unknown regions. As a first step, it propagates local planes to empty subvolumes by examining the neighboring subvolumes and determining whether an empty subvolume is affected by a local plane. Thereafter, for each unknown voxel the superordinate subvolume is checked to see if it contains a local plane. If so, the voxel is filled using the distance from the voxel center to the local plane.

   By applying this process multiple times iteratively, local planes are quickly extended and known planar structures of the scene grow towards unknown regions.

E) **De-noising and refinement through local plane priors** reduces noise of known flat structures of the scene by adapting the TSDF values to local planes, on the one hand, and refines propagated TSDF values (created in the Step D), on the other hand. It estimates a partially consistent plane in the neighborhood of the considered voxel and optimizes its TSDF value. Partially consistent planes prevent discretization artifacts like unwanted steps between adjacent subvolumes.

   If the propagation of planar structures is finished, only observed values are denoised. This is necessary because new data are continuously integrated and must also updated.

F) **Non-planar surface propagation and hole filling by diffusion** starts when the propagation of planar structures is finished and extends the actual TSDF values (observed and hypothetical) to fill small holes. This stage is especially necessary for non-planar surfaces. It alternates two steps, blurring and compositing, to diffuse the known TSDF values. If this step is repeated many times, similar to Step D, the known non-planar structures of the scene grow towards unknown regions.

The local plane estimation (Step C) is not processed for each frame. The reason is threefold: 1) it reduces the effect of wobbling surfaces; 2) it keeps the real-time capability of this approach; and 3) surface data do not change significantly within several frames, so it is unnecessary to estimate new local planes for each frame. Instead, it is sufficient to perform this step only for keyframes.

To keep the algorithm real-time capable, only one iteration of steps D and F which continuously extend the observed reconstruction is processed per frame. The volume is not filled immediately, but the algorithm converges relatively fast. It usually finishes after a few iterations. As discussed in Section 3.13.2, a common scene from an man-made environment is filled rapidly after the user starts the reconstruction procedure. For example, the volume of the kitchen front is filled completely within 70 frames, which corresponds to 2.5 seconds with a frame rate of 30 frames per second (fps).

The notation used to describe the reconstruction volume is explained in Section 3.6, and Sections 3.7 to 3.10 describe steps C to F in detail. Lastly, Section 3.11 describes the volume integration (Step B).

3.6 TSDF Volume

The reconstruction volume that will be extended and optimized is defined analogously, as illustrated in Section 3.2.1. For a voxel v within volume S_k which keeps observed data from 0..k, the TSDF value $T_k(v)$ and the weight $W_k(v)$ can be obtained by:

$$S_k(v) = [T_k(v), W_k(v)]. \tag{3.2}$$

Unobserved TSDF values of all voxel v within the volume are identifiable by $W_k(v) = 0$. These values should be filled by the propagation and hole filling procedures and adapted by the volume integration, if new surface data become available.

3.7 Local Plane Prior Estimation

To estimate local plane priors the algorithm splits the volume in subvolumes and finds the most significant plane inside the local region of each subvolume by following a 1-point RANSAC procedure. To do so, a sampling step creates a set of sample points on the surface of the current reconstruction. Each sample is defined by its 3D point and its normal. Afterwards, the algorithm iteratively selects a random sample to define a plane candidate and calculates its inliers. This step is repeated to find the candidate with the largest number of inliers and stops if a certain number of iterations is reached.

In contrast to previous work by Dzitsiuk *et al.* [DSM+17], the time-consuming steps to create a mesh (to process a 3-point RANSAC) and to estimate the normals by calculating the covariance matrix are not required. Instead, the presented algorithm is inspired by the 1-point RANSAC plane merging strategy of Dzitsiuk *et al.*.

The first step of the algorithm splits the volume into equal subvolumes. A size of 0.2 m^3 for each subvolume is a good choice, in terms of quality and runtime performance. For example, when using a volume with 256^3 voxels which is a commonly used configuration to reconstruct a 3 m^3 environment, it is then grouped into 17^3 subvolumes. To perform the 1-point RANSAC for a given subvolume the first step generates a set of plane candidates. The algorithm identifies all voxels of the subvolume that lie at a zero-crossing and calculates their surface normal and corresponding point on the surface. For each voxel center v_i in $V = \{v_0, \ldots, v_m\} \subset \mathbb{R}^3$ at a zero-crossing the normalized normal vector $\vec{n_i}$ is estimated by $|\nabla T_k(v_i)|$. The point on the surface x is calculated by $x = v_i - T_k(v_i)\vec{n_i}$. These points are then used to build a set of possible plane candidates $P = \{p_0, \ldots, p_m | p_i = (\vec{n_i}, x_i)\}$.

The algorithm then iteratively selects a random plane candidate p_k (with $0 \leq k \leq m$) and calculates the number of inliers. Inliers are all other plane candidates of the subvolume $I = \{p_i \in P | k \neq i\}$ that fulfill the following two conditions, similar to those described in [DSM+17]):

1) the point-to-plane distance between a hypothetical inlier p_i and the plane candidate p_k is below a certain threshold ϵ

$$|n_k \circ (x_i - x_k)| < \epsilon, \tag{3.3}$$

2) the normals of both are approximately similar; for example the angle between the hypothetical inlier p_i and the plane candidate p_k is below a certain threshold δ

$$|n_k \circ n_i| < \cos(\delta). \tag{3.4}$$

In most cases, $\epsilon = 0.01$ m and $\delta \approx 25°$ work well. After a certain number of iterations have been performed, the plane candidate with the highest number of inliers is accepted as the most significant plane. This, however, only applies if $\frac{|I|}{m}$ is above a threshold (where 0.5 is a good choice of threshold). Approximately 100 iterations are absolutely sufficient to obtain good results. Finally, the local plane is estimated by the average of all inliers.

3.8 Planar Surface Propagation from Local Plane Priors

Before planar surfaces can be propagated throughout the whole volume it is necessary to propagate local plane priors towards empty subvolumes. To do so, the algorithm follows a region growing procedure and processes two steps for each iteration: 1) it grows local planes to adjacent and intersected subvolumes; and 2) it fills unknown voxels with their distance to the superordinated local plane. If these two steps are iterated on planar structures will quickly grow towards unknown regions.

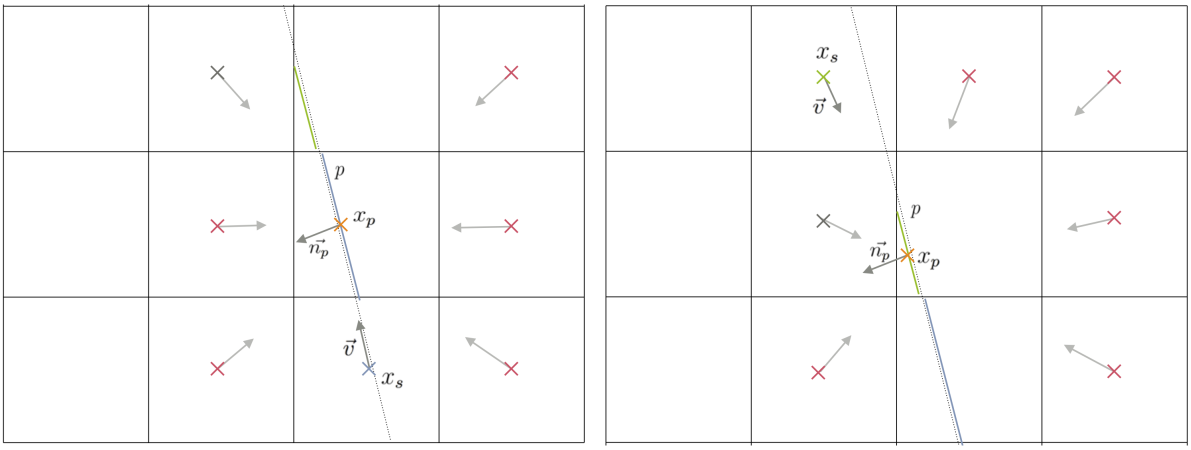
To grow local plane priors, every iteration of the algorithm examines the neighboring subvolumes of each empty subvolume. Hence, an empty subvolume is filled by the average of all planes of the neighboring subvolumes that intersect the empty subvolume significantly.

To determine if a local plane p of a subvolume affects an adjacent and empty subvolume the normal $\vec{n_p}$, the center x_p of the plane and the center of the empty subvolume x_s in global coordinates are considered:

$$|\vec{n_p} \circ \frac{(x_p - x_s)}{\|(x_p - x_s)\|}| < \lambda, \tag{3.5}$$

where λ is a certain threshold. If the dot product is approximately zero the plane intersects the subvolume clearly. Thus, λ regularizes whether a plane is considered or not depending on the size of the intersection ($\lambda = 0.5$ works well in many scenarios). Figure 3.6 shows two examples of how local plane priors are propagated. The number of required iterations depends on the size of the subvolumes and the number of subvolumes that are already filled with data obtained by the sensor.

The propagation of local plane priors ceases if dominant planes intersect – for example in the corner of a room where walls intersect. It is likely that the local planes intersect and from a corner if their normal vectors are differ significantly from each other. Accordingly,



(a) (b)

Figure 3.6: **Two examples of local plane propagation**. (a) Shows a plane p (blue) which should be propagated to the whole volume (black dots). The plane is defined by its center x_p (orange) and normal $\vec{n_p}$ (gray). Considering the adjacent subvolumes the plane p will be propagated only to the subvolume marked with a blue cross (where x_s is the center of this subvolume and $\vec{v}$ the normalized vector between x_p and x_s). Subvolumes which are not affected by the plane are marked with red and gray crosses at the center. (b) Shows to which subvolumes the green plane will be propagated accordingly.

the variance of the unsigned dot product between the planes is calculated and the propagation procedure stops if it is below a threshold (0.9 provides good results). The variance σ^2 is calculated by:

$$\sigma^2 = \frac{1}{m} \sum_{i=0}^{m} (n_i \circ \mu)^2, \tag{3.6}$$

where m is the number of considered planes, n is the normal vector of a plane, and μ is the mean of all normal vectors. The second step of every iteration checks for each unknown voxel of the volume if there exists a local plane in the superordinate subvolume. If there exists such a plane, the distance between the voxel center and the plane is calculated (as in Equation 3.3) and saved in the current voxel.

After the propagation process is stopped, the surface still contains holes and gaps in regions where no significant planar structure could be found. These small holes are closed by the diffusion algorithm (see Section 3.10).

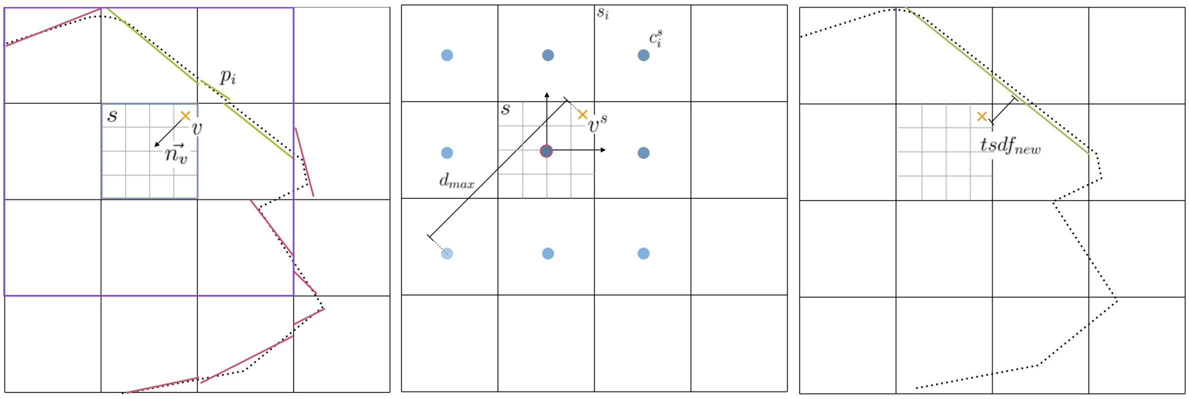
3.9 De-noising and Refinement through Local Plane Priors

To obtain a smooth and appealing surface reconstruction of flat structures observed TSDF values are de-noised and hypothetical TSDF values are refined. To eliminate unwanted step artifacts that occur because of the partitioning into subvolumes, TSDF values are refined by adapting them to partially consistent planes. On the one hand this guarantees smooth transitions between planes of adjacent subvolumes; on the other hand it prevents small holes (which occur if nearby planes cover only a small region of the subvolume and are not designated as dominant).

To de-noise or refine a TSDF value of a voxel with center $v \in \mathbb{R}^3$, the normal $\vec{n_v} \in \mathbb{R}^3$ is calculated by $|\nabla T_k(v)|$ in the first step. After that, the plane of the superordinate subvolume s and all planes in its $3 \times 3 \times 3$ neighborhood are potentially candidates to build a partially consistent plane. Only planes to which the voxel v potentially corresponds to should be considered. Accordingly, all plane candidates that are too far away from the voxel or do not have a similar normal are removed. A plane is considered as a candidate if the point-to-plane distance and angular difference using the Equations 3.3 and 3.4 are small. The remaining planes are then merged to a single plane by calculating the weighted average. An example is illustrated in Figure 3.7. The weight w_i of a subvolume plane p_i depends on the distance of the subvolume s_i to the current voxel v and is calculated by :

$$w_i = (1 - \frac{\|v^s - c_i^s\|}{d_{max}})^3 , \tag{3.7}$$

where $v^s \in \mathbb{R}^3$ describes the voxel center and $c_i^s \in \mathbb{R}^3$ the center of the subvolume s_i. Both values are given in the coordinate system of the superordinate subvolume s. The term $d_{max} \in \mathbb{R}$ describes the maximal distances between the v^s, c^s within the considered neighborhood.



(a) (b) (c)

Figure 3.7: **Estimation of partially consistent plane by merging local planes.** The volume with its voxels (gray) is split into subvolumes (black). (a) Shows the surface of the real environment (black dots), the current viewed voxel v (yellow), and its normal $\vec{n_v}$ (black). The superordinate subvolume s is outlined in blue and the considered neighborhood is outlined in purple. Panel (a) also shows the local planes for all subvolumes (red and green) and the plane candidates that are finally merged to a single plane (green). (b) Shows the weights of each local plane (blue points) that are used to build the weighted average. (c) Shows the merged plane (green) and the new calculated TSDF value $tsdf_{new}$.

The next step calculates the new TSDF value $tsdf_{new}$. This is the distance from the voxel center v to the merged plane. After a normalization and clamping procedure, the TSDF value replaces the current one if $tsdf_{new}$ does not deviate significantly from the previous value. This is true if the difference between the values is below a certain threshold (2 cm provided good results).

3.10 Non-Planar Surface Propagation and Hole Filling by Diffusion

Similar to the planar surface propagation, the surface diffusion algorithm continuously extends the actual known TSDF values to unknown parts of the volume. The method is used to fill small holes and to extend curved structures of the scene. Each diffusion iteration consists of two steps: blurring and compositing (similar as in Marschner *et al.* [DMG$^+$02]). The idea is to blur the whole volume and to write back only blurred values where the TSDF is unknown. If this step is processed repeatedly, the TSDF values diffuse to the whole volume. If the whole volume is blurred and unobserved parts of the volume are replaced naively, unsightly stair-step artifacts of two consecutive diffusion steps arise. The compositing step ensures smooth transitions by interpolating between consecutive iteration steps.

To alternate between blurring and compositing the algorithm uses a second volume, diffusion volume D_k, in addition to the reconstruction volume S_k. The reconstruction volume is used to realize the reconstruction and interactions (e.g. occlusion or collision handling). In contrast, the diffusion volume is used to diffuse observed and hypothetical values towards to unknown regions. Values that are observed by the sensor should not be modified by the diffusion algorithm.

The reconstruction volume S_k is defined as described in Section 3.6 and the diffusion volume D_k is adequately defined by $D_k(v) = [T_k^d(v), W_k^d(v)]$. Whereas the weight of the reconstruction volume describes the confidence of an observed value, the weight of the diffusion volume is used to realize the compositing between two iteration steps.

The pseudocode Algorithm 1 illustrates the main steps of the algorithm for a single iteration. To process the blurring step the known TSDF values are convolved with a kernel h (line 6). The kernel h is defined by the weights of the diffusion volume D_k

Algorithm 1 Diffusion algorithm for a single iteration at frame k.

```
 1: for Each voxel $v$ in the diffusion volume $D_k$ do
 2:     $w^k \leftarrow W_k(v)$ current confidence in volume $S_k$
 3:     if $w^k == 0$ then                                   ▷ TSDF value is unobserved
 4:         $w^k_{max} \leftarrow$ actual maximum weight of the source volume $S_k$
 5:         $h \leftarrow$ prepare kernel h with weights and normalize it
 6:         $T_k^d(v) \leftarrow h * T_k(v)$    ▷ convolve values from the last iteration with kernel h
 7:         $T_k^d(v) \leftarrow W_k^d(v)T_k(v) + (1 - W_k^d(v))T_k^d(v)$ ▷ interpolate between two iterations
 8:         $W_k^d(v) \leftarrow min(W_k^d(v) + 1, w^k_{max})$              ▷ update the weights
 9:     end if
10: end for
11: copy all TSDF values with $W_k^d > 0$ from volume $D_k$ to the source volume $S_k$
```

in the n^3 neighborhood (line 5). Hence, TSDF values having a large weight that are observed more often have a potentially higher accuracy and exert more influence during the first convolution. Values that are unknown are ignored automatically. The weights of the TSDF values, which were estimated by the previous propagation procedure of planar structures, are initialized with 1. This generates smooth transitions between planar and curved structures.

The compositing step processes a linear interpolation between the previous TSDF values to smooth between two blurring steps (line 7), and it updates the weight (line 8). Increasing the weight for each iteration ensures that the linear interpolation (line 7) is reduced to $T_k(v)$ so the blurring process stops after a certain number of iterations. As an additional benefit the interpolation of values near to an accurate observed surface stops earlier. The filter size n is not critical because of the repeated blurring step [DMG+02]. In terms of blurring and performance results, $n = 3$ offers a good compromise.

3.11 Volume Integration

New incoming depth data are continuously integrated into the volume as the sensor is moved around. New observed data are fused to the previous observed TSDF values and estimated TSDF values are adapted. Thus, the complete propagation process does not have to start anew. The pseudocode Algorithm 2 illustrates the volume integration.

First, the new TSDF value of a voxel is calculated (lines 2-4). To calculate the incoming TSDF observed by the sensor (line 2), each voxel center of the volume is are converted into a global position using the resolution of the volume and its dimension in the real world. To obtain their distances to the camera center, these global voxel points are transformed into camera coordinates. After their correspondences are calculated by a back projection in image space and looking up measured depth values, the TSDF can be calculated. The TSDF for a voxel is calculated by the difference between the measured and calculated

Algorithm 2 Integration step at frame k.

```
 1: for Each voxel $v$ in the volume $S_k$ do
 2:     $tsdf_{k+1} \leftarrow$ incoming tsdf value
 3:     $w_{k+1} \leftarrow min(W_k(v) + 1,\ w_{max})$
 4:     $tsdf_{k+1} \leftarrow \frac{T_k(v)W_k(v)+tsdf_{k+1}w_{k+1}}{W_k(v)+w_{k+1}}$        ▷ weighted average
 5:     if $W_k^d(v) > 0$ and $W_k(v) == 0$ then        ▷ diffused tsdf value
 6:         if $tsdf_{k+1} < 0$ and $tsdf_{k+1} < T_k(v)$ then
 7:             $tsdf_{k+1} \leftarrow T_k(v)$        ▷ take diffused tsdf value
 8:             $w_{k+1} \leftarrow W_k(v)$
 9:         end if
10:     end if
11:     $T_k(v) \leftarrow tsdf_{k+1}$
12:     $W_k(v) \leftarrow w_{k+1}$
13: end for
```

distance to the camera. The TSDF is then truncated and ranged to $[-1, 1]$. A detailed description about TSDF calculation for a given depth map appears in [NDI+11] and [IKH+11]. Then the weight is increased (line 3) and the TSDF values are fused (line 4) by taking the weighted average of the newly calculated and the previously stored value, in which S_k is the TSDF integrated up to frame k.

Second, new incoming TSDF values are merged with the diffused values (lines 5-10). If the new TSDF value ($tsdf_{k+1}$) lies in front of the surface ($tsdf_{k+1} > 0$), the diffused values are replaced. Alternatively, if the new TSDF lies behind the surface, a minimum operation is used to form the union. In this way, surfaces without self-intersections are created. Lastly, new values are stored at the current voxel v (lines 11 and 12).

3.12 Collisions and Occlusions with the TSDF

As mentioned before, collision and occlusion handling between the real and the virtual world is a well-known task and is essential to create believable AR applications. This section shows how a TSDF representation can be used to detect and handle both collisions and occlusions. Generally, there is no need to create an explicit representation, since both techniques work directly on the implicit surface data. Permanent and time-consuming remeshing of the TSDF (e.g. via Marching Cubes [LC87]) if new data are integrated into the reconstruction can be avoided.

3.12.1 Collision Detection and Response

Typically, virtual objects in an AR scene are represented by polygonal meshes, whose faces consist of triangles or squares. Hence, a simple algorithm that handles collisions between polygonal surfaces of rigid bodies and the TSDF reconstruction is used. It focuses on an efficient GPU implementation by avoiding unnecessary data transfer between main and GPU memory, as the volume resides on the graphics card by design. The algorithm performs well enough to allow the collision of real and virtual objects, even while the surface reconstruction and optimization are in progress.

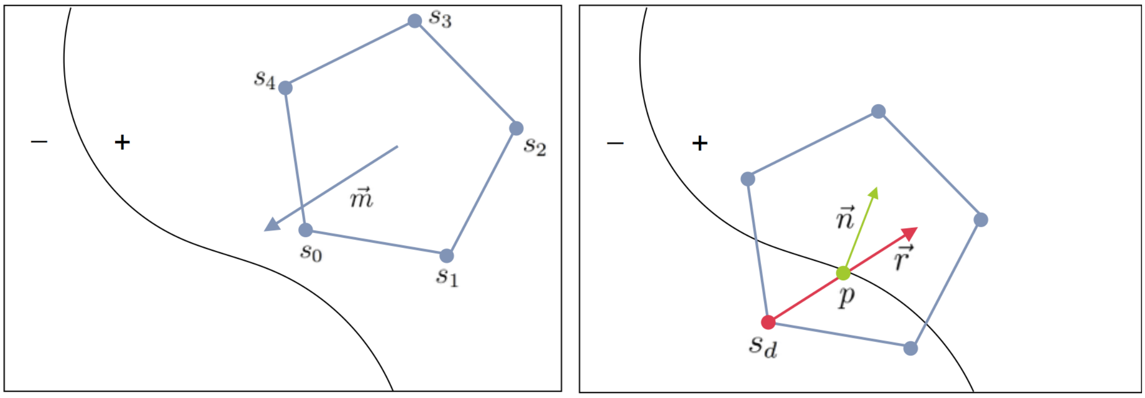
Collision handling can be divided into two main stages: collision detection and collision response. To detect collisions between virtual objects and the real world, the reconstruction is always aligned with its real counterpart, and predictions are made about where the virtual objects will move to. All virtual objects are temporarily moved to their new positions and the algorithm checks whether they collide with objects in the scene. Each virtual object is represented by a set of sample points on the surface (the vertices of the polygonal mesh). By simply testing all sample points against the TSDF, a collision between the object and the real world can be detected (a similar approach was used in [GBF03]).

Let t_0 be a time step in which no interpenetration between a virtual object and real world occurs. By considering the time interval $[t_0, t_0 + \triangle t]$ between two consecutive updates, the position and orientation of the moving virtual object is updated to time $t_0 + \triangle t$. A collision occurs during this time interval if one or more sample points s of an object are inside the surface of the real world (as shown in Figure 3.8):

$$c_d(s) = \begin{cases} 1, & \text{if } T_k^d(s) < 0 \text{ and } W_k^d(s) > 0 \\ 0, & \text{otherwise,} \end{cases} \tag{3.8}$$

where T_k^d represents the distance function and W_k^d the weight function (which indicates if a point is already known) of the optimized reconstruction. Depending on the time of the interval $[t_0, t_0 + \triangle t]$, one or more sample points might already be inside the real world surface.

For the collision response, the collision point of the virtual object – the first sample point at which the virtual object contacts the real world – must be found. This point s_d can be approximated by the deepest point of interpenetration (which corresponds to the lowest negative TSDF value). The contact point p on the real world surface is then approximated by tracing a ray $\vec{r}$ in the inverse movement direction $\vec{m}$. The surface normal $\vec{n}$ is defined by evaluating the gradient $\nabla T_k^d(p)$ at the contact point p on the surface. Based on the contact point and the normal a collision response can be performed by following standard procedure (as described in [MW88]).



(a) (b)

Figure 3.8: **An example collision detection.** (a) Shows the surface of the real world represented by the TSDF (black) and a virtual object represented by its vertices s_0, s_1, s_2, s_3, s_4 (blue) at time step t_0. The virtual object moves in direction $\vec{m}$. (b) Shows a collision at time step $t_0 + \triangle t$ where s_d is the deepest point of interpenetration, $\vec{r}$ is the ray used to find the contact point p, and $\vec{n}$ is the corresponding normal.

The quality of collision handling depends on the number of sample points used to approximate the surface. Using the vertices works well for sharp objects (e.g. cubes) but it can be critical for curved objects (e.g. spheres) that have low resolutions. It might be worthwhile to use a dense point sampling (as in [Erl04]) instead of simply using the vertices. To simulate collisions between deformable virtual objects (e.g. clothes) and the TSDF reconstruction, particle based approaches, such as [FSG03; TKH$^+$05], can be useful.

3.12.2 Occlusion Handling

To realize occlusions, two render passes must be used. The first pass performs a ray casting of the reconstruction volume from the camera's point of view in which the depth data of all surfaces are saved to a separate frame buffer. When rendering virtual objects in a second render pass, it can be decided whether the stored depth values are in front of the rendered objects. If that is the case the alpha value of the corresponding fragment is set to zero, which results in an occlusion. More details about occlusion handling and its challenges, using a reconstruction of the environment, are given in Chapter 4.

3.13 Experimental Results

This section shows the applicability of the algorithm and the effectiveness of local planes priors on different indoor scenes which retain both planar and curved structures. First, the diffusion algorithm was evaluated. Second, changes in the results due to considering local plane priors were noted. Finally, the runtime performance of the algorithms was measured to show that the overall system performance was good enough to obtain real-time rates.

Unless stated otherwise, all scenes were reconstructed and extended with standard parameters, as described in detail in the corresponding sections, and recorded with a Kinect (v1 or v2).

3.13.1 Surface Propagation without using Plane Priors

Figure 3.9 illustrates the progression of the diffusion over a period of iteration steps and shows that the diffusion starts relatively fast, rapidly covering large parts of an unfilled area. This area is refined as the diffusion progresses. Due to the progression being fast during the first iterations, it is less important that the reconstruction is not completed in one frame.

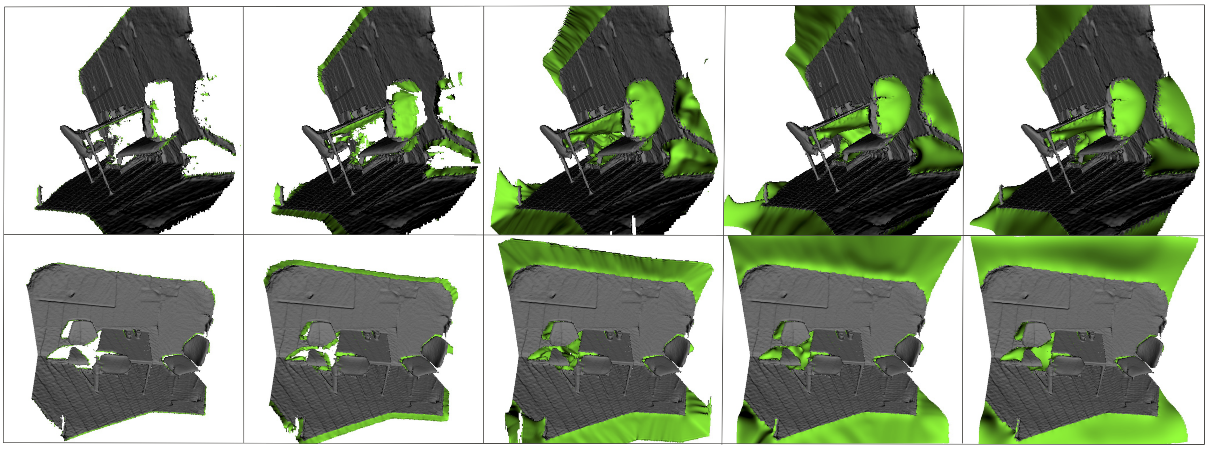
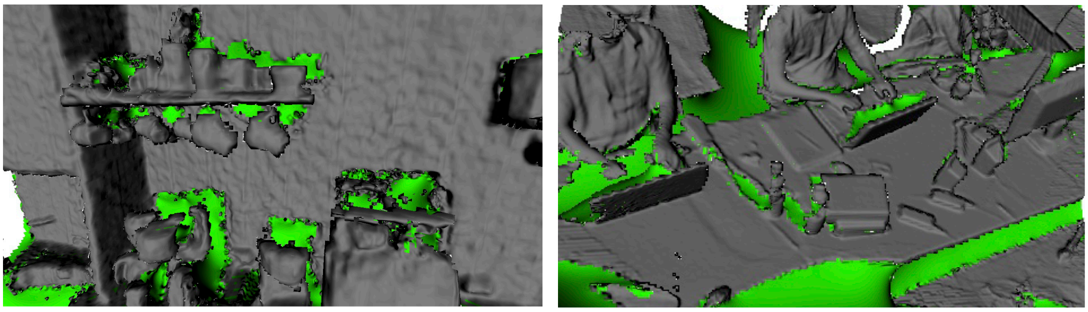
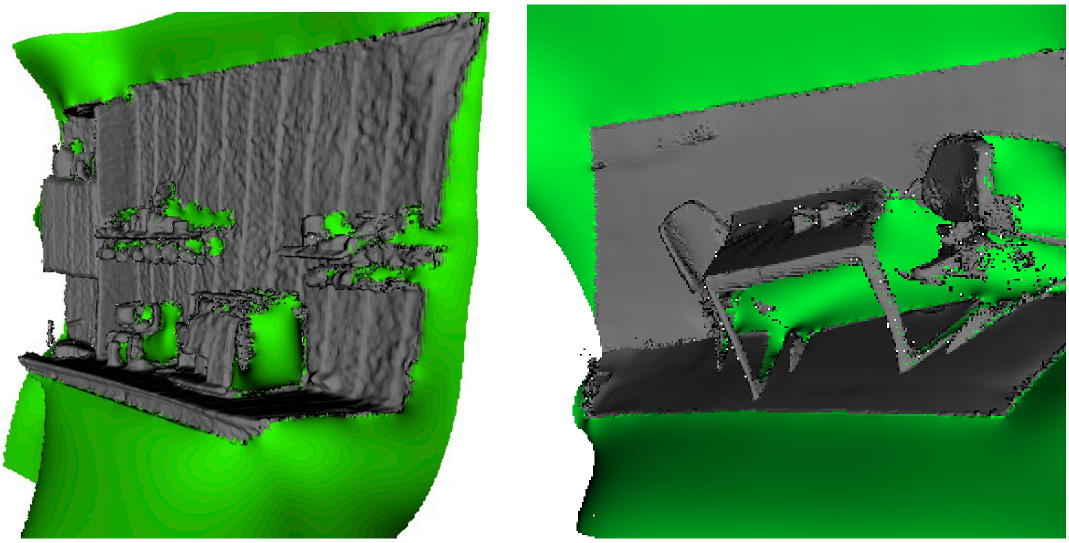


Figure 3.9: **Surface propagation without plane priors.** Starting with an observed (gray) surface reconstruction, the approach fills holes and extends the surface (green). The two rows show the same scene from different viewpoints. The iteration interval of the diffusion algorithm was as follows: 1, 4, 16, 64, 256.

Figure 3.10 shows different scenes that contain several types of surfaces which are common in man-made environments. The propagation algorithm filled small holes in a scene having scattered and planar surfaces. The small holes behind the cups and kitchen equipment (Figure 3.10a) and small holes on the office desk (Figure 3.10b) were filled in a plausible way. In addition, the coffee table (Figure 3.10d) was approximated reasonably, as the diffusion followed the edge of the table and left an empty space underneath. Figures 3.10c and 3.10d show that the algorithm was also able to diffuse known parts of the scene towards unknown ones. For example, the front side of the kitchen was fulfilled through a vertical progression (Figure 3.10c). However, the diffusion tended to curved surfaces even if the distance increased.



(a) Kitchen front with cupboard (b) Office scene



(c) Kitchen scene (d) Coffee scene

Figure 3.10: **Propagation results without plane priors.** The unsuccessful reconstruction data (gray) and the diffusion without using plane priors (green) on different scenes with scattered and planar surfaces. Panels (a) and (b) show scenes with small holes on scattered surfaces. Panels (c) and (d) show scenes with unsuccessfully reconstructed planar surfaces.

Generally speaking, small holes were repaired effectively as the diffusion converged relatively fast. In contrast, on parts of the scene, where the known surface should be extended in a planar manner, the diffusion converged more slowly and tended to curved shapes.

3.13.2 Surface Propagation using Plane Priors

In man-made scenarios with large flat surfaces, propagation with plane priors produced more reasonable results than a simple propagation procedure through diffusion. Figure 3.11 compares the results with and without plane priors on different types of surfaces.

The algorithm with plane priors was able to deal with small holes and large unknown parts. It diffused planar surfaces towards unknown parts of the scene without curved forms and filled holes without planar structures.

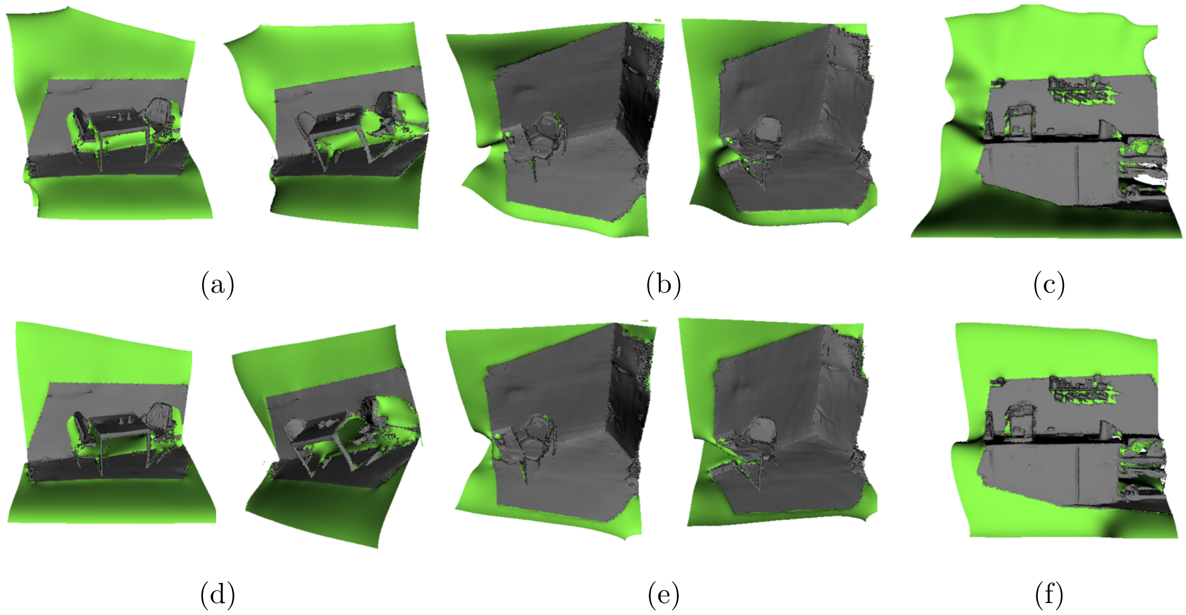

(a) (b) (c)
(d) (e) (f)

Figure 3.11: **Comparison between results created with and without plane priors.** Panels (a) to (c) results without local plane priors, and (d) to (f) show the results with local planes.

Plane priors ensured that the algorithm converged even more quickly and effectively. Figure 3.12 illustrates the surface propagation using plane priors over several steps of iteration. The extension of planes was notably rapid. Unfilled planar parts of the scene were filled in after only a few iterations (e.g. 40 iterations). Then the diffusion without plane priors (which started between frame 40 and 60) filled small holes and eliminated small artifacts by smoothing regions in which no significant planes were found.

Figure 3.13 shows the effect of refinement and de-noising through partially consistent planes. Notably, unpleasant steps between adjacent subvolumes were eliminated. Noise from the observed reconstruction data in planar regions was successfully reduced. However, it can happen that filigree structures may be lost. For example, shown in Figure 3.13f, the structure of the drawer was no longer visible. This can be controlled by the parameter δ, which decides based on the angular difference if a local plane should be

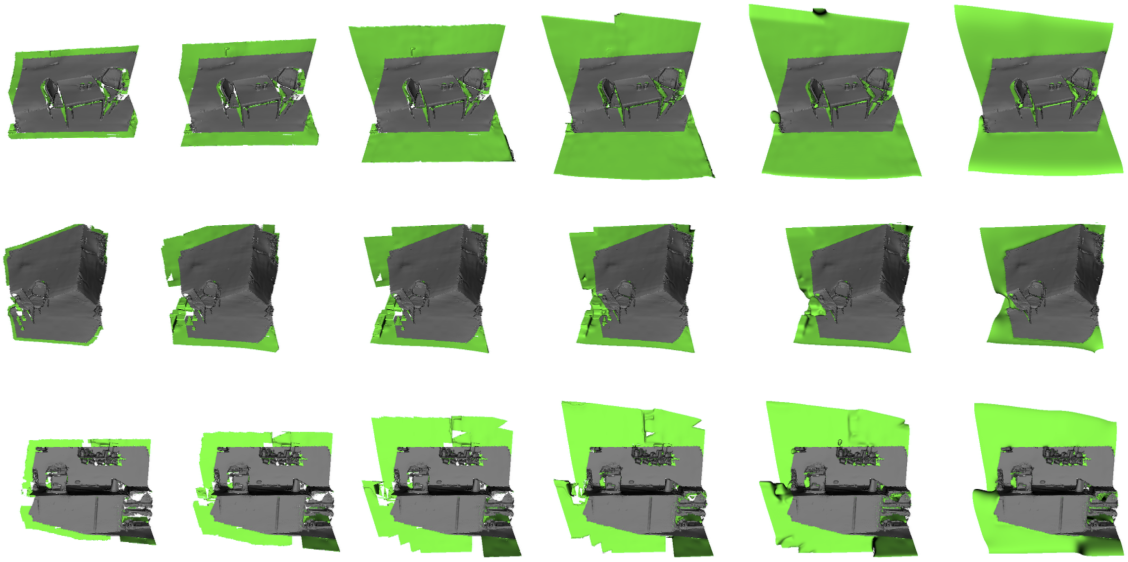
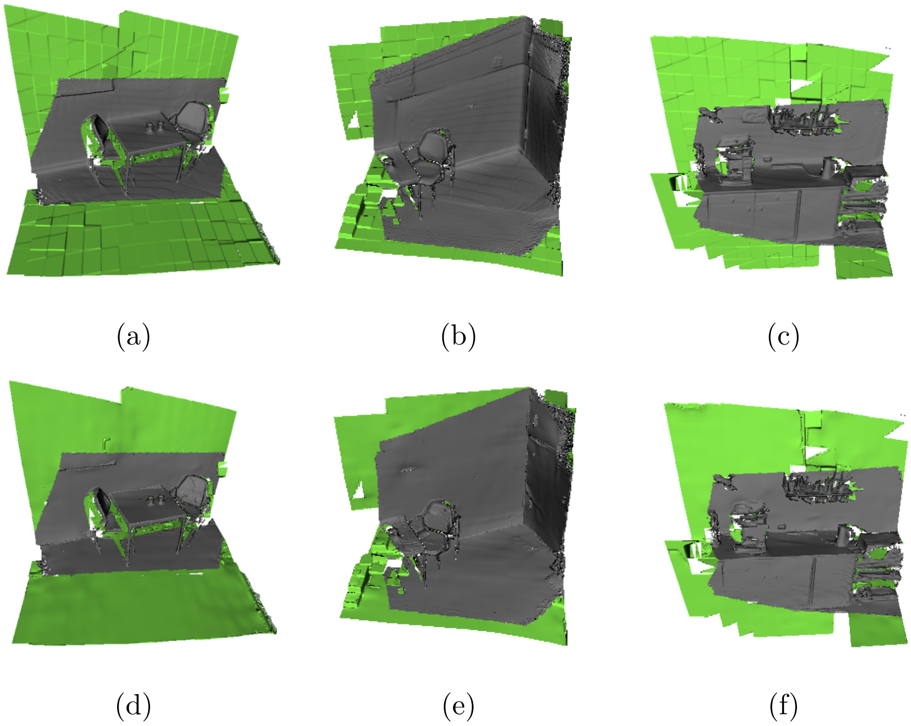


Figure 3.12: **Surface Propagation using Plane Priors for three scenes.** Starting from an observed (gray) surface reconstruction, the approach filled holes and extended the surface (green) in a plausible way. The iteration interval of the algorithm was as follows: 10, 20, 40, 60, 80, 380.



(a) (b) (c)

(d) (e) (f)

Figure 3.13: **Effect of refinement and de-noising through partially consistent planes.** Panels (a) to (c) show results of the propagation of planar structures without refinement and de-noising. Panels (d) to (f) show the results with refinement and de-noising, respectively. The diffusion step, which eliminates small artifacts and closes small holes, were not performed on these examples.

considered during the merging step (as described in Section 3.9). A large δ leads to the elimination of filigree structures, whereas a small δ leads to ignoring many local planes so that only a small part of the scene is refined.

3.13.3 Implementation Details and Runtime Performance

The presented approach used the capabilities of the GPU hardware using CUDA [CUD] technologies. In addition, all calculations were performed on the GPU to overcome limitations in transferring reconstruction data between GPU and CPU memory. The runtime performances were evaluated for different scenes. All benchmarks were performed on a PC with a 4 GHz i7 processor and a Geforce GTX Titan X graphics card. An overview of runtime performances at each stage of the algorithm, for the different test scene, appears in Table 3.1. All measurements given in this section were obtained by the NVIDIA Visual Profiler [Pro] and are shown as an average of 1000 invocations.

Pipeline stage	Coffee scene	Kitchen front	Corner scene
Local Plane Estimation	113.093	126.230	202.151
Local Plane Propagation	0.810	0.124	0.759
Total (ms)	113.903	126.354	202.910
Surface Propagation Plane Priors	3.821	3.771	3.542
Refinement & De-noising	3.690	3.729	5.092
Total per frame (ms)	7.511	7.500	8.634
Surface Propagation Diffusion	6.759	6.682	6.560
De-noising	2.031	2.105	4.107
Total per frame (ms)	8.790	8.787	10.667

Table 3.1: **Runtime performance measurements for different test scenes.** All these attributes define times in milliseconds (ms). Each scene was reconstructed using a volume with 256^3 voxels grouped into 17^3 subvolumes. The whole volume was thus $3\,m^3$ and each subvolume was $0,2\,m^3$. The local plane estimation was performed using 100 RANSAC iterations. Local plane estimation and propagation was not performed for each frame.

Local plane estimation. The estimation of local planes consumed most of the runtime of the algorithm. Furthermore, the execution time increased with the size of a known scene. Currently, the realization of this step does not focus on an efficient implementation and could be optimized using common GPU acceleration methods. As described in Section 3.5, the local plane estimation was only performed for keyframes; moreover, it could be performed asynchronously. Optimization of this step nevertheless makes sense to reduce the GPU utilization and memory capacity of the algorithm, as an asynchronous execution would need a second volume. The execution time of the current implementation depended strongly on the number of subvolumes, as Table 3.2 shows. The RANSAC procedure was performed by evaluating each plane candidate (counting its number of inliers) in parallel. Thus plane estimation was performed more rapidly for larger subvolumes, which had more plane candidates, because more instructions were processed in parallel. However, the algorithm tended to provide better results when small subvolumes were used.

Number of subvolumes (size)	Coffee scene	Kitchen front	Corner scene
17^3 (0.2 m)	113.093	126.230	202.151
25^3 (0.3 m)	63.936	63.548	119.881
34^3 (0.4 m)	33.485	31.475	53.945
42^3 (0.5 m)	24.189	25.664	40.321

Table 3.2: **Runtime performance measurements for the local plane estimation at various subvolumes.** All these attributes define times with milliseconds. Each scene is given by a $3\,m^3$ volume with 256^3 voxels.

Surface propagation and hole filling per frame. The runtime needed to propagate planar surfaces was perfectly sufficient to fulfill real-time requirements. Moreover, it was largely independent from the size of the known surface. The diffusion algorithm, which extends curved structures and closes small holes, was also independent from the known size of the surface and achieved real-time rates. However, it was computationally more intensive and almost twice as times slower than the propagation of planar surfaces because a convolution of the volume is needed. Thus it is focused on an efficient implementation in this regard. In the last years much research has been done on how convolution algorithms can be efficiently implemented on GPU. The performance of 2D convolutions is significantly improved by utilizing parallelization and overlapping kernel windows via shared memory [Pod07]. As a 3D convolution is required to process volume data, a 2D convolution for every volume slice is performed, followed by an accumulation of the result inside the kernel. This resulted in three slices that were accumulated per voxel for a 3^3 kernel. A similar approach is described in [ED14] for non-separable 3D convolutions.

Refinement and de-noising per frame. The refinement of propagated values and the de-noising of observed values were processed per voxel in parallel. As shown in Table 3.1, the processing time of this step varied only slightly according to the size of the scene, which kept the complete pipeline real-time capable.

For the sake of completeness, the collision handling (described in Section 3.12.1) between a mesh with 14837 vertices and an optimized volume took less than a millisecond to compute.

3.14 Conclusion and Future Work

A strategy to optimize a surface reconstruction while the reconstruction is in progress has been presented. The approach was able to de-noise observed surfaces and extended them towards unknown parts of the scene so that unobserved parts were filled in and holes were closed in a believable way. Generally, this marked an immense step forward to realize more realistic AR applications in which virtual content interacts with the real world. The user was not tasked with obtaining a complete reconstruction of his or her surroundings before the main AR application started. Moreover, interactions between the real and virtual worlds, such as collisions, were no longer limited to observed regions.

The contribution of this chapter is twofold. First, it details an approach to integrate a suitable filling and de-noising algorithm into the KinectFusion reconstruction pipeline, which can fill in unknown parts in a plausible way and de-noise known parts. The approach handled different types of structures, such as curved or planar structures, and overcame limitations of state-of-the-art approaches whose filling procedures are restricted to planar structures. Therefore, the proposed approach fitted local planes to the reconstruction and propagated planar structures, and diffused the surfaces to extend curved structures and to close small gaps. Hypothetical surfaces were smooth, without self-intersections, and were updated if the sensor captured new surface data. Second, this chapter demonstrates how to use the extended implicit surface reconstruction to realize collision and occlusion, without a transfer to an explicit representation. All algorithms used the capabilities of the GPU. By design they did not require high transfer rates and thus achieved real-time performance.

A limitation of the implementation was that the whole volume was not filled at once, as the processing time per propagation step (via plane priors or diffusion) was too lengthy. However, for a rough approximation of the surface just a few iterations are required (see Section 3.13.2), which enables real-time capability. Additionally, the processing speed of the local plane estimation stage may be enhanced by adapting the implementation to the use of small subvolumes that achieve the best overall appearance in different scenes.

Chapter 4

Occlusion Matting: Realistic Occlusion Handling

This chapter deals with the occlusion problem in AR applications and presents a method of handling occlusions realistically through alpha matting. This new approach is called *occlusion matting* and was proposed for the first time in the context of this thesis.

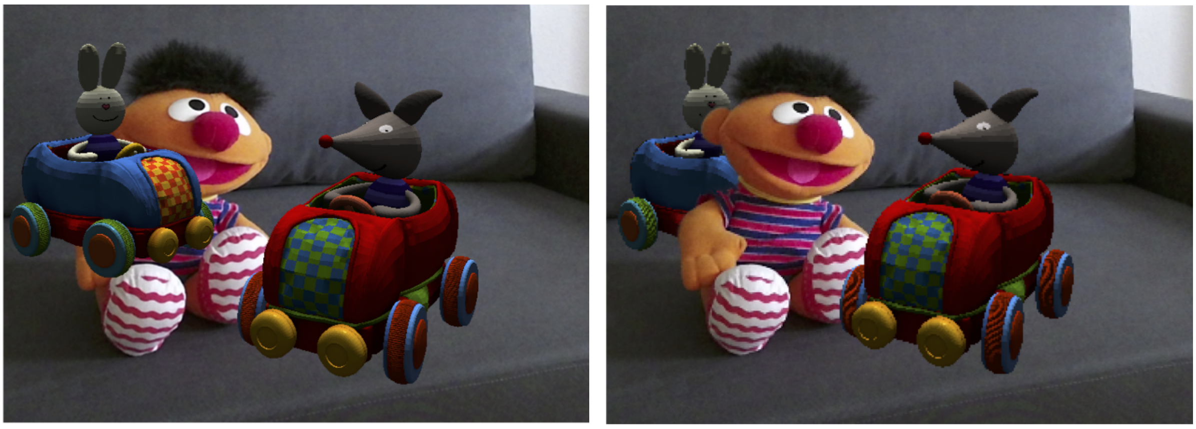
The occlusion handling challenge is discussed in Section 4.1, followed by Section 4.2 which gives an overview of related work. Section 4.3 explains the idea of occlusion matting and re-formulates the occlusion problem as an alpha matting problem. Afterwards, Section 4.4 discusses relevant approaches in the field of alpha matting and focuses on the applicability of those approaches to occlusion matting. An overview of the occlusion matting approach is given in Section 4.5, and its realization is detailed in Sections 4.6 to 4.8. Section 4.9 illustrates the runtime performance and applicability of the algorithm across several cases. The results are compared with those obtained by state-of-the-art methods. Lastly, a summary and discussion of future work is given in Section 4.12. Most of the work presented in this chapter was peer-reviewed and presented at a conference [HHM17].

4.1 Occlusion Handling Challenge

The rendering of virtual objects in AR must be as realistic as possible nowadays. In addition to the illumination of virtual objects, occlusions between the real and virtual worlds are important to achieve an outstanding AR experience.

By default, virtual objects always appear in the front of the user's view; they overlay the video image. However, virtual objects are sometimes softly, partially or totally occluded by real objects. Hidden parts of the objects should not be rendered or blended together to create realistic occlusions. Accordingly, techniques are needed to determine which parts of virtual objects are visible, transparent or non-visible.

Occlusion errors are not only unsightly; they lead to incorrect depth perception because the spatial relationship between the real and virtual worlds is unrecognizable. This results in misunderstanding and confusion in users – for example if they try to grab virtual objects. The effect of occlusion handling in AR is shown in Figure 4.1. Moreover, the probability of eyestrain and motion sickness is increased [FHF$^+$99].



(a) Without occlusion handling (b) With occlusion handling

Figure 4.1: **Occlusion handling in AR.** (a) Shows the virtual content is well registered but occlusion by the real object is not handled. (b) Shows the same scene with occlusion handling, which creates better depth perception as the relationship between real and virtual scene is clear.

4.2 Related Work on Occlusion Handling

Approaches to occlusion handling in AR can be divided into three categories: 1) object-based, 2) model-based and 3) depth-based. The following sections describe and discuss in detail relevant methods in the each category.

4.2.1 Object-based Occlusion Handling

Object-based approaches usually estimate a 2D region representing the objects that lie in front of the virtual object. By tracking or detection of this region, an occlusion mask can be calculated and used to handle occlusions.

Berger [Ber97] used a contour-based approach to solve the occlusion problem. In their system, each contour point is labeled with *behind* or *in front of*. The occlusion mask is then generated by tracking the contour from frame-to-frame. This algorithm is easy to implement and does not need a 3D model of the real scene. However, the user must specify the spatial relationship between real and virtual objects, which cannot change during the runtime of the application. Moreover, the approach suffers from missing contours. If a contour cannot be detected it has a strong influence on the final occlusion mask. Even one missing contour can lead to noticeable errors when using this algorithm.

Lepetit and Berger [LB00] extended the contour-based method to resolve occlusion semi-automatically in an offline scenario. The user specifies the outline of occluding objects in key-views, thus, occlusions in intermediate views can be generated automatically.

Fischer *et al.* [FRB03] proposed an alternative method, based on the background subtraction principle, which can run in real-time. They assumed a static background, namely a plane like a marker or a template image, with the virtual scene lying in front of it. The idea is that an object that lies in front of the static background also lies in front of the virtual scene. The algorithm detects a known background (e.g. a marker board) in the current camera image and creates an occlusion mask by performing an image comparison. This concept only works for AR scenarios in which the occluding objects are closer to the camera than the rendered objects (such as hands or pointing devices). In addition, the occluding objects must be in front of markers or templates. Occlusions are only handled in front of the known background plane; occlusions with real objects that lie in front of the virtual scene but do not cover the plane are not handled.

Feng *et al.* [FDG+06] presented an approach for realizing multilayer occlusions. They focused on indoor scenarios in which occluded objects were several meters away from the user. The approach splits the real indoor space into several parts, from the front to the back, out of the users view. To realize occlusions, the virtual scene is divided into these parts accordingly. If real objects are moving, occlusions can be handled depending on their location in the room. In contrast to most object-based approaches, the approach of Feng *et al.* can deal with large distances and multiple objects. However, the occlusion results are very coarse and depended on the level of subdivision of the room.

Tian *et al.* [TGW10b] proposed an approach that can deal with similar background colors and large viewpoint changes. As in other object-based approaches, the occlusion handling is divided into three steps: selection, tracking, and masking. First, the user selects the occluding object using an interactive segmentation method. Then the contour of the object region is tracked through optical flow and the correct occlusion is obtained. Lastly, the boundary between real and virtual objects is smoothed. This procedure does not work if the user or the camera moves too fast and the tracking fails; in that case, the user must re-initialize the occluding object first. To summarize, these methods do not need a 3D model of the considered scene or special sensors to realize occlusions. However, they must be initialized by the user or the background must be known.

Tian *et al.* [TGW10a] presented an approach that obtains the contour of the occluding real object automatically, using a disparity map generated from a stereo image pair. The precise contour is estimated by a mean-shift algorithm in an offline step. As in other methods, the contour is then tracked online to handle occlusions. However, object-based approaches depend on a fixed relationship between real and virtual objects, which is not realistic in many AR cases.

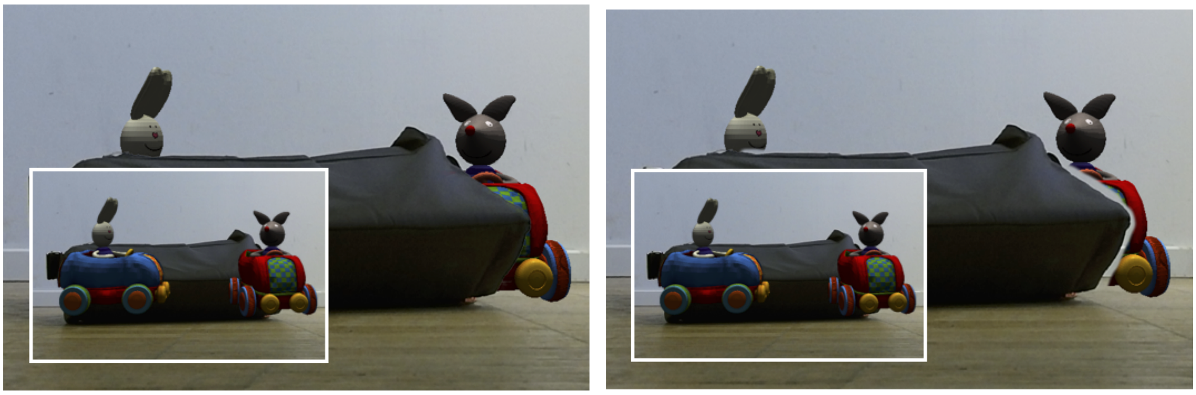
4.2.2 Model-based Occlusion Handling

Model-based approaches require a 3D model of either the real scene or the occluding object. Then occlusion handling can be performed by aligning the virtual and real objects through tracking, followed by an off-screen rendering step.

This strategy was introduced by **Breen *et al.*** [BWR$^+$96] to handle static occlusions. The main idea is to define a representation of real objects in the user's surroundings, which is often called a phantom (see Section 3.1). After a registration step, the virtual representations should be aligned precisely with their real counterparts. Phantoms are then rendered only into the z-buffer and not to the screen. With this approach occlusions are automatically handled by the rendering hardware. A pixel of the virtual object will only be drawn if it is in front of the real scene, as indicated by a smaller depth value.

Fuhrmann *et al.* [FHF$^+$99] extended this method to handle dynamic occlusion caused by people moving in a collaborative environment. Human motion capturing enables the registration of the phantom. The phantom represents the user and is modeled as kinematic chains of articulated solids.

Model-based approaches suffer from (among others) registration errors, jitter, and incorrect projection parameters [KD04]. Even small errors, such as those caused by inaccurate calibration or tracking, result in noticeable occlusion errors. In addition to accurate registration, an accurate and complete 3D model of the real environment is needed. The phantom model must match exactly its counterpart in the real scene, otherwise the occlusion does not look appealing. An example of an inaccurate registration is shown in Figure 4.2.



(a) Accurate registration (b) Inaccurate registration

Figure 4.2: **Example of model-based occlusion handling and the registration problem**. (a) Shows a scene with occlusion handling, with a precise phantom of the real object (backpack) that is well registered. (b) Shows the phantom not well registered, so occlusions do not look appealing. Small images (outlined in white) show the scene without occlusion handling.

Klein and Drummond [KD04] proposed a method in which tracking is followed by occlusion refinement. After a tracking step, the algorithm identifies and refines the boundaries where real objects occluded virtual objects. The approach calculates a 2D clipping polygon that is in front of the virtual object. Each clipping edge of the polygon is then refined using an edge-based tracking approach. This procedure can be summarized in three steps: 1) sampling of the clipping edge; 2) perpendicular search for strong image gradients in the camera image for each sample point; and 3) optimization to fit a clipping edge to its counterpart in the camera image. The refined polygon is used to clip the virtual object, and each clipping edge is thus blended into the camera image to generate seamless occlusion boundaries.

DiVerdi and Höllerer [DH06] proposed a similar technique. Instead of refining each edge of the clipping polygon as in Klein and Drummond's method, they proposed an image-based post-processing approach. Using the GPU the algorithm estimates a per-pixel offset that is used to render clipped polygons of the virtual objects.

Both approaches reduce the visual registration errors. The results are convincing for simple scenarios and jitter effects are mostly eliminated. But in more complex situations, these approaches suffer from low contrast and incorrect correspondences. Occluding edges cannot be found or are too close to other texture edges or shadows [KD04]. Using the pixel-wise approach, DiVerdi and Höllerer were able to deal with coarsely approximated geometries, where errors caused by inaccurate phantoms were reduced.

Zheng *et al.* [ZSW14] corrected registration errors, both in world space through camera pose refinement and in local image space through pixel-wise refinement. Better registration in world space was obtained by refining the camera poses through selective weighting of important regions of the image. The image space correction computed the optical flow between the camera image and a rendering of the real model, using the refined pose in a post-processing step, to deal with various non-rigid errors. However, visual differences between the rendered model image and the camera image can be problematic. This method cannot deal with strong lighting changes, shadows, motion blur, or large occlusions.

In addition to registration and modeling errors, another drawback of model-based methods is the need for a 3D model. Currently, the required model is obtained directly from existing resources or is modeled using a modeling software or is obtained by a 3D scanning process. Thus, any representation – whether mesh, implicit surface, or point cloud – can be used to create a depth map.

Tian *et al.* [TLX+15] proposed a method based on a 3D reconstruction. The method uses a low-cost depth sensor and consisted of two stages: 1) an offline stage in which a 3D point cloud is generated; and 2) an online stage in which occlusions are handled. The occlusion results depend strongly on the quality of the reconstruction and the method is not suitable for dynamic scenes where the real environment changes. Occlusions with real objects that are placed into the scene or moved out of the scene cannot be handled. In contrast, a real-time reconstruction procedure during the online stage to update the model would be able to handle this.

Generally, current reconstruction algorithms [NLD11; WLS+15] are able to reconstruct a semi-dense and accurate model in real-time. The boundaries, however, often remain noisy. A detailed discussion about this challenge is presented in Section 3.1. Overall, it can be concluded that the accuracy of the model or reconstruction and the tracking have strongly influence the occlusion results. Consistent occlusions can be achieved only if a complete and precise 3D model is available and accurate tracking without jitter is used. In contrast to the object-based methods, these approaches estimate spatial relationships and handle occlusions between real and virtual objects without user interactivity. However, model-based methods are generally unable to deal with dynamic occlusions or deformable objects. To deal with these aspects each dynamic object must be known and tracked.

4.2.3 Depth-based Occlusion Handling

Due to the spread and advancement of low-cost sensor technologies, such as depth sensors or stereo cameras, current algorithms use incoming depth maps from sensors to handle the occlusion problem [FHS07; DFK13]. As model-based approaches do, they compare the depth values to decide which parts of the rendered scene are visible or non-visible. These methods can realize occlusions for unknown real objects without any user input. No prior knowledge of the shape, size or position of the objects is required.

Despite strong improvements of low-cost depth sensors, the following challenges remain: 1) several types of noise occur, especially at the boundaries between foreground and background; 2) incomplete data may occur, for example because of shadow effects or limited sensor ranges; and 3) inaccurate mapping occurs between color and depth data. Thus, occlusion handling based on raw depth maps suffers from missing data, coarse edges, and inconsistency between color and depth boundaries. The result is visual artifacts, as shown in Figure 4.3. These artifacts are not only unsightly but also detract from an immersive AR experience considerably [KSF10]. Hence, common depth-based methods improve the incoming data by a pre-processing step. Because high-quality depth maps are essential to many tasks in computer vision, many algorithms to improve and enhance depth maps have been proposed already. Filtering approaches [TM98; $\text{KCL}^{+}07$; HST13]

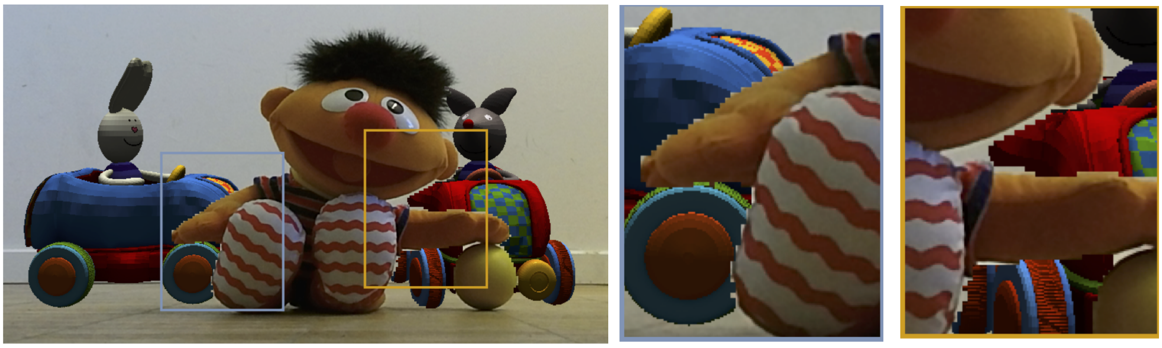

(a) (b) (c)

Figure 4.3: **Occlusion handling with raw depth data.** (a) Results of occlusion handling using raw depth data, which yields visual artifacts. (b) Noisy depth data, especially at the boundaries between foreground and background, result in unsightly zigzag patterns. (c) Inaccurate mapping between color and depth data results in inconsistency between edges, and false occlusions.

can reduce noise and fill holes but they are only partially suitable for solving the AR occlusion problem. The reasons are twofold: firstly, the computational cost of these algorithms is high; secondly, the boundaries often remain coarse and unaligned with the boundaries of the color image [DCY+16]. Popular edge-aware filters, such as a guided filter [HST13] or bilateral filters [TM98; KCL+07] which smooth images but preserve edges, tend to generate interpolation artifacts around boundaries [DCY+16]. Such artifacts can falsify the occlusion result.

Leal-Meléndrez *et al.* [LAG13] filled holes using an in-painting algorithm. The boundaries generated using this method are still imprecise and noisy, and noisy boundaries result in zigzag patterns between the foreground and the virtual object.

Schmidt *et al.* [SNV02] presented a method to obtain a high-quality, dense disparity map from stereo images, with a focus on sharp edges. However, these edges are not always aligned with the edges of the color image. Misalignments between depth and color edges lead to false occlusions. Background pixels at the boundaries are considered as foreground – or vice versa.

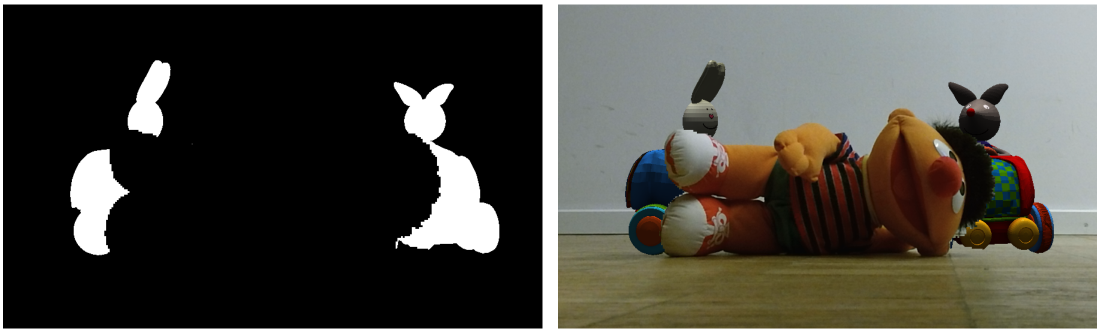
Du *et al.* [DCY+16] also focused on improving depth maps with sharp edges that were aligned to the edges in the color image. The algorithm snaps depth edges to color edges to improve the consistency between the two. This procedure is very time-consuming for higher resolutions but produces good results if the occluding object can be separated from the background precisely. This method suffers from background clutter and weak gradients between foreground and background. Moreover, it cannot handle smooth occlusions that arise through motion blur or fuzzy objects (as shown in Section 4.9).

4.3 Occlusion Matting

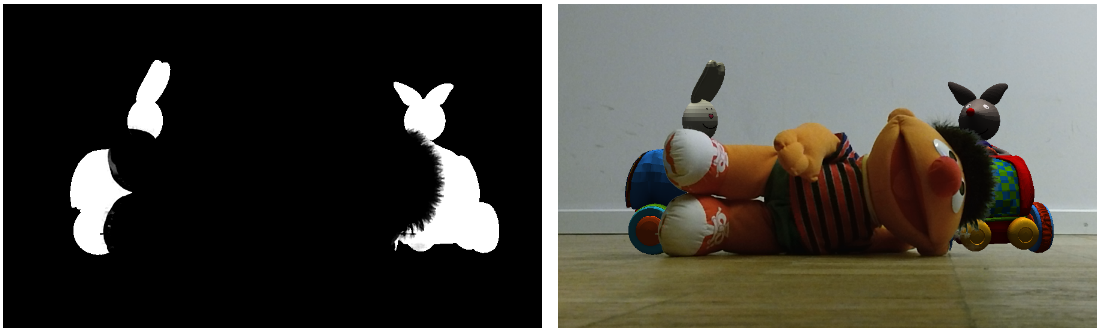
This section motivates the handling of the occlusion problem as an alpha matting problem, to overcome the limitations in state-of-art methods.

4.3.1 Idea and Motivation

As shown in the previous section (Section 4.2), several solutions to handling occlusions in AR exist. Depth-based methods are becoming more popular and their accuracy has increased. In comparison with other methods, their distinctive advantage is that they can deal with occlusions of unknown and dynamic objects without any user input. The results of these methods often remain inaccurate or are accurate only in regions in which the foreground and background are clearly separable. However, sometimes pixels or regions close to the boundary of an object are not strictly foreground or background – for example, they might be caused by fuzzy objects such as hair, or motion blur. Accordingly, occlusion handling should involve determining both the full and partial pixel transparency of virtual objects. Whereas state-of-the-art methods try to identify which pixel of the rendered virtual object is visible or non-visible, the goal should be to determine a blending coefficient for each pixel. These blending coefficients are called the *alpha matte*. The approach presented in this thesis formulates the occlusion problem as an alpha matting problem. Instead of calculating the visibility for each pixel of the virtual objects, a blending coefficient is estimated. This enables seamless integration of virtual objects into the real world, even for fuzzy foreground objects like hair. A comparison between occlusion matting and state-of-the-art methods is given in Figure 4.4. The approach uses raw depth information from the real scene (e.g. obtained by a low-cost depth sensor) to realize rough foreground and background segmentation. The blending coefficient between transitions in which depth values are typically noisy is estimated from the color image.



(a) Binary mask (b) Result using (a)



(c) Alpha matte (d) Result using (c)

Figure 4.4: **State-of-the-art approaches vs. occlusion matting.** (a) Binary mask from state-of-art methods; they judge only whether a pixel of the rendered scene is visible or non-visible. (b) Results of the occlusion handling using the binary mask (a). (c) Alpha matte of the occlusion matting approach, in which the opacity of each pixel is precisely estimated. (d) Results of the occlusion handling using the alpha matte (c).

4.3.2 Occlusion as an Alpha Matting Problem

The motivation for the proposed method is to handle the occlusion problem as an alpha matting problem. Generally, alpha matting addresses the problem of extracting foreground objects from static images or video sequences. Precisely separating a foreground object from the background requires estimating the opacity for each pixel, also known as

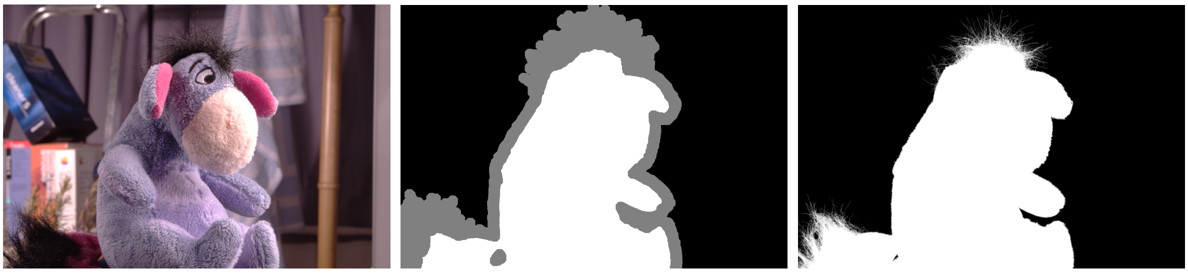

(a) (b) (c)

Figure 4.5: **A natural image matting example.** (a) Shows the input image. (b) Shows a user-specified trimap. The trimap splits the input image in three regions: definitive foreground (white), definitive background (black) and unknown (gray). (c) Shows the alpha matte. Image source: Benchmark test data set with ground truth data [RRW$^+$09].

pulling a matte. An observed image I can be described mathematically by a compositing equation (see [PD84]). I is a convex combination of the foreground image F and background image B:

$$I = \alpha F + (1 - \alpha)B \tag{4.1}$$

where α defines the alpha matte, with $\alpha \in [0, 1]$. Matting describes the inverse process of compositing. To handle occlusions in AR, alpha matting can be used to determine a precise alpha matte of the virtual scene. The task is the same: to precisely separate the foreground (parts of the real scene which occlude virtual objects) from the background. Thus, foreground and background are separated by the alpha matte in relation to the virtual objects. Then the composition of the augmented scene can be obtained by redefining Equation 4.1.

Matting is an under-constrained problem, since in a three-channel image seven unknowns need to be solved from three inputs. The seven unknowns include the foreground F color, the background B color, and alpha α; the three inputs represent the color of the input image I. Thus, there exists no unique solution and constraints are necessary to ensure a good alpha estimation. Hence, almost all common alpha matting methods use a user-specified three-level map, called a *trimap*, as the starting point. A trimap segments the input image into three non-overlapping regions: definitive foreground, definitive background and unknown. Figure 4.5 shows an example of a trimap.

It can be concluded that two main challenges exist. The first is to automatically generate a trimap based on the relation between the real and virtual objects. The trimap should be as precise as possible and unknown regions should be as small as possible. This enables a high-quality alpha matte estimation and avoids unnecessary computations. The second challenge is to estimate the alpha matte by solving the optimization problem for all unknown regions in real-time.

4.4 Related Work on Alpha Matting

This section gives a short overview of relevant work on alpha matting. Because alpha matting is a well-known problem in many image and video editing applications, it has been extensively studied. Methods that solve the matting problem for scenes in which the background is arbitrary and unknown are important to the current work. This problem is known as *natural image matting.* Methods related to automatic trimap generation are also of interest.

4.4.1 Natural Image Matting

Natural image matting methods can be classified into four categories: 1) sampling-based; 2) propagation-based; 3) a combination of sampling and propagation; and 4) learning-based approaches [ZSL$^+$15].

Sampling-based methods, [BDV00a; BDV00b; RT00], assume that the true foreground and background colors of an unknown pixel can be estimated from known foreground and background samples. Unknown alpha values are then calculated by solving the inverse compositing equation. The results depend on the selected samples and the quality of the trimap [WC07a]. Thus, the methods work well on distinct foreground and background color distributions, where samples are easy to collect.

Propagation-based methods, [LLW08; SJT$^+$], do not estimate explicit foreground and background colors. Instead they assume that foreground and background colors are locally smooth and treat the problem as propagating known alpha values towards unknown regions. Compared with sampling-based approaches, these methods produce local smooth results. However, the quality rapidly degrades when foreground and background patterns become complex, because colors are wrongly propagated [WC07b].

Combination of sampling and propagation. Several methods fuse these two approaches to achieve high-quality alpha mattes. These techniques collect sample sets (pairs of foreground of background samples) and estimate the alpha matte in a global optimization process, by selecting a single or several best sample pairs [GO10; WC07b]. Two main challenges occur with these algorithms: 1) how to collect suitable sample sets, and 2) how to define an objective function to select good sample pairs. For example, samples are generated from spatial nearest boundary pixels [WC07b] or from the closest superpixel[1] [JVC$^+$16]. Once sample pairs have been selected, the next step is to choose a good pair using an objective function and calculating an alpha value from this pair [GO10; WC07b]. These methods only perform well if the true foreground and background colors are represented in the sample set. The true colors are not always covered because these methods collect samples only near each unknown pixel, and the number of samples is rather limited. Several improvements have been devised to overcome this problem. He *et al.* [HRR$^+$11] presented a global sampling method that uses all available samples in the image to avoid missing good samples. The approach, proposed by Johnson *et al.* [JVC$^+$16], generates samples from the spatially closest superpixels and treats the matting problem as a sparse coding problem; alpha is estimated from the complete set of selected samples.

Learning-based approaches treat the matting problem as a supervised or semi-supervised learning task. These methods usually perform a training procedure and estimate the matting parameters in a learning-based framework [ZSL$^+$15]. Xu *et al.* [XPC$^+$17] pro-

---

[1] Superpixels are segments of a so called superpixel segmentation, which over-segments the image by grouping pixels that share similar properties [LMC17].

posed a novel deep learning based algorithm to overcome problems like similar foreground and background colors or complicated textures. The method uses a deep convolutional encoder-decoder network that takes an image and the corresponding trimap as inputs to predict the alpha matte. Thereafter, a small convolutional network refines the alpha matte to obtain accurate alpha values and sharper edges.

4.4.2 Real-time Natural Image Matting and Video Matting

Algorithms that achieve a good trade-off between almost real-time performance and accuracy are of interest to this work. Most image matting algorithms presented in Section 4.4.1 are unable to run in real-time and thus are not suited for processing live videos. First, the matting problem requires global optimization. Second, these approaches are mostly used for static images in editing applications, where almost real-time capability is sufficient. Video matting extends the matting problem to video sequences and is often used in television and film production. It is not possible to use user-specified constraints – such as trimaps or strokes – for every video frame. Moreover, in online video matting, the alpha matte must be estimated in real-time. Recent algorithms have generally processed two steps: 1) generate trimaps by segmentation methods and 2) pull a matte with image matting techniques [ZSL$^+$15]. Real-time video matting approaches, such as chroma key matting [YZ14; XDW$^+$14], often simplify the problem by using a known green or blue background. Hence these methods can be used only in controlled environments (e.g. movie studios). To handle the occlusion problem, the challenge remains computing the alpha matte from video frames in natural, unknown, uncontrolled environments. Moreover, the relationship between foreground and background can change at any time if the real and virtual objects move.

Joshi *et al.* [JMA06] presented a rapid automatic system for high-quality video matting in a natural environment using a camera array. Based on the relative parallax in the array images, the system constructs a trimap and estimates alpha at near real-time rates. The matting problem is constrained by assuming a sufficiently textured background. Therefore, the algorithm projects colors from each camera to the depth of the foreground object. The trimap and alpha values are then estimated using mean and variance statistics from these colors. The approach achieves good results in scenarios where the background is well-textured. However, it fails in scenarios where background colors are constant, such as a white wall. With regard to this limitation and the need for several cameras, the method is currently not transferable to the occlusion problem.

Gastal and Oliveira [GO10] presented an approach that combines sampling and propagation. The algorithm traces rays from the unknown pixels to select potential sample pairs. An objective function is then used to select the best sample pair; the objective function considered spatial, photometric, and probabilistic information. This method was realized in OpenGL shading language (GLSL) and could perform in real-time, depending on the number of selected samples. However, the researchers did not have to deal with the automatic trimap generation problem.

Wang *et al.* [WGZ+12] used information from a depth sensor to generate a trimap automatically through a segmentation procedure. In their system, the alpha matte is estimated by a propagation-based method called *multichannel Poisson matting*. Real-time rates were achieved by optimizing the algorithm for parallel processing on the GPU. In addition, problems of propagation-based methods (see Section 4.4.1), the method yields unpleasant visual artifacts that result from temporal inconsistency. The reason for this temporal inconsistency is that the segmentation is solved for each frame independently.

4.4.3 Depth-based Trimap Generation and Matting

Automatic high-quality trimap generation is a fundamental problem in alpha matting. Many proposed approaches use depth information from the scene [LL12; WFQ+07] and almost all of them follow the same procedure. They create a binary map by calculating a segmentation of foreground and background, as a first step. For example, a separation of foreground and background can be generated using a segmentation algorithm, such as k-means [ZLY+09]. In the second step, unknown regions are determined by applying an erosion and dilation technique to the foreground.

These approaches use a fixed structuring element for the morphological operations. This results in inaccurate alpha estimation, because a wide unknown region is better suited to fuzzy areas whereas a narrow region is more suitable for objects with sharp edges [CYA+11]. Ideally, fuzzy regions (e.g. foreground object with fur) should be covered by larger unknown regions than regions having sharp edges (e.g. foreground object with significant boundaries).

Wang *et al.* [WFQ+07] proposed an adaptive dilation according to the fuzziness of the foreground object. Like other systems, their system performs two main steps: segmentation, followed by morphological operations. The segmentation between foreground and background is obtained through a user-defined plane. The amount of erosion and dilation are also specified by the user. Thus, the user decides which parts of the object needed a larger dilation.

Cho *et al.* [CYA+11] proposed an alternative approach. The approach generates trimaps in which the size of unknown regions differs automatically according to fuzziness. The algorithm traces the exterior boundaries of foreground objects and calculates the size of structuring elements based on the fuzziness. The fuzziness is estimated from the variance between the previously calculated alpha values of the neighboring pixels.

In addition to being used for automatic trimap generation, depth information can be used to reduce artifacts that arise from ambiguities between foreground and background colors [LL12; WFQ+07].

4.5 Method Overview

The occlusion matting pipeline takes a color image and a raw, registered depth map from a sensor, and applies the following five main stages. The process is shown in Figure 4.6).

A) **3D Rendering** renders the color and depth values of the virtual scene into textures.

B) **Adaptive trimap generation** specifies invalid, foreground, background and unknown regions of the image. It uses the depth information acquired by the sensor and rendered scene. It also takes the color image of the sensor into account to provide a precise assumption of which pixels certainly belong to the foreground or the background in relation to the virtual scene. Furthermore, a dilation is applied to unknown regions based on the boundaries and fuzziness in the color image.

C) **Foreground and background propagation** propagates known foreground and background regions of the color image towards unknown regions.

D) **Alpha estimation** identifies, for each pixel in the unknown regions, the best pairs of foreground and background samples using a simple objective function. Based on the obtained sample sets, the final alpha value is estimated.

E) **Compositing** combines the color images of the rendered and real scene into a single image, called *composite*.

The following sections (Sections 4.6 to 4.8) describe steps B to D in detail.

4.6 Adaptive Trimap Generation

The image is divided into two parts: segments that are in the background of the virtual objects and segments in the foreground of these objects. This division allows for easy

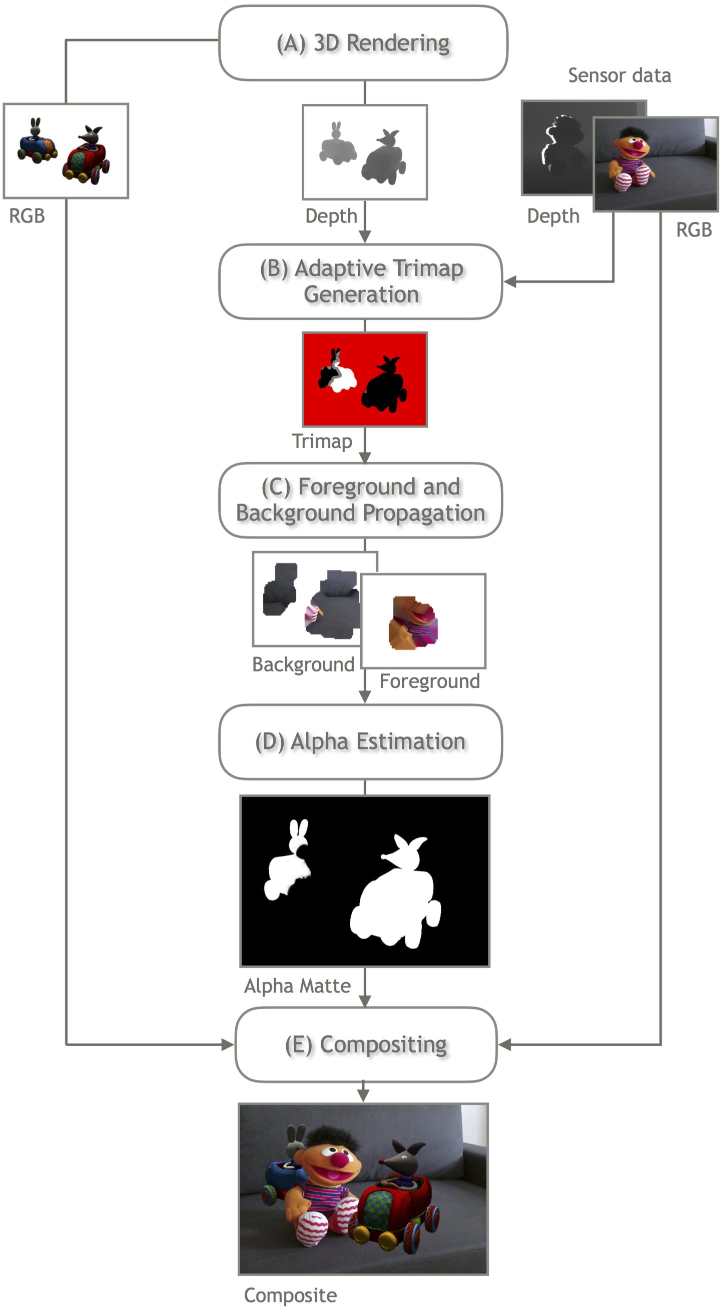


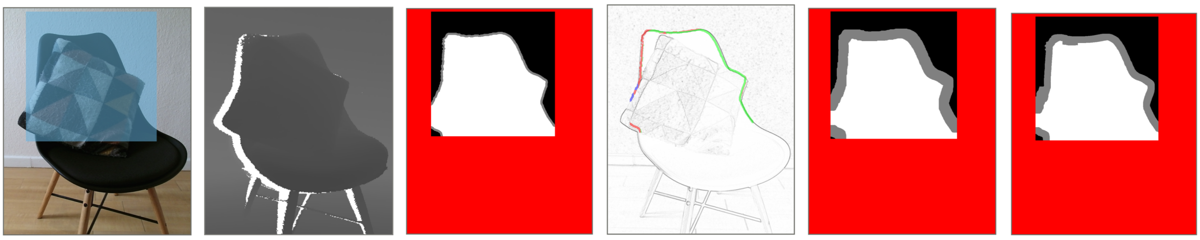
Figure 4.6: **Overview of occlusion matting pipeline.** Visualization of main stages, from a raw depth map to the rendered view of the virtual scene, with occlusion handling through alpha matting. A short description of each stage appears in Section 4.5.

segmentation of the foreground and background by performing a depth test. Unknown regions can then be obtained by identifying the transition between the foreground and background, followed by a dilation procedure (see Section 4.4.1). However, this would result in an inaccurate trimap. Unknown areas lie around the boundaries of the depth image but not necessarily around the boundaries of the color image. This scenario strongly influences the alpha estimation, because some background regions are wrongly taken as foreground and vice versa. However, a large dilation means large unknown regions and color boundaries may be covered; it also means that known foreground regions are likely to shrink. Foreground regions of small objects, such as fingers of a hand, can be fully eliminated and a precise alpha estimation is no longer possible. To overcome these problems, unknown regions are enlarged towards the boundaries of the color image. In contrast, regions without significant edges (e.g. hair) are enlarged in all directions.

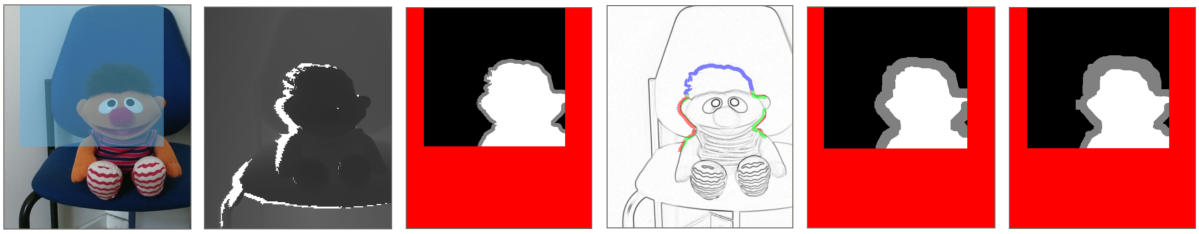
The adaptive trimap generation is processed in three steps: 1) coarse segmentation of foreground, background, and unknown 2) labeling of unknown regions, which helps to decide how to dilate them; and 3) the adaptive dilation.

4.6.1 Coarse Segmentation

The segmentation takes the raw depth map as input and generates an initial trimap. Typically, a trimap defines foreground, background and unknown regions using a three color image (e.g. white, black and gray). In this algorithm a fourth color (red) is needed to represent invalid values, because information is valid only in regions onto which virtual objects are projected. Known foreground pixels (F) and known background pixels (B) are identified by applying a simple depth test between virtual objects and valid pixels of the scene. Invalid depth values occur mostly because of shadowing effects so that these are defined as background. Unknown regions (U) are represented by the boundaries of the foreground object (transitions between F and B). To obtain these regions, the valid pixels are convolved with a 3×3 Sobel kernel. Because raw depth edges are typically noisy as evident in Figures 4.7b and 4.7h, low-pass filtering is used to smooth the valid pixels first. Invalid values (red) are ignored, so that pixels with large gradients within a window are the unknown regions. The results of coarse segmentation are shown in Figures 4.7c and 4.7i.



(a) Scene
(b) Sensor depth
(c) Coarse trimap
(d) Labeling $w = 25$
(e) Dilation $s = 32$
(f) Dilation $s = \{32, 40\}$



(g) Scene
(h) Sensor depth
(i) Coarse trimap
(j) Labeling $w = 25$
(k) Dilation $s = 15$
(l) Dilation $s = \{15, 25\}$

Figure 4.7: **Adaptive trimap generation.** Panels (a) and (g) show a virtual plane that should lie partially behind a real object; (b) and (h) are raw sensor depth data; and (c) and (i) represent the smoothed segmentation between foreground (white), background (black), unknown (gray), and invalid (red) pixels. Panels (d) and (j) show the edges of the color image overlaid with the labeled unknown regions, where w is the size of the $w \times w$ search window. Color edges that lie in the front-half-space of the unknown pixel (red), in the back-half-space (green), or where no significant edge exists nearby the unknown pixel (blue). Panels (e) and (k) show the extended unknown regions with a fixed dilation amount s. Panels (f) and (l) show the adaptive dilation with two sizes s. Large structuring elements are used to visualize the differences, although, in several cases a smaller size is sufficient.

4.6.2 Labeling of Unknown Regions

The purpose of the labeling step is to categorize all initial unknown pixels into three classes: *front half-space*, *back half-space* and *no edge*. An unknown pixel i (with $i \in U$) is labeled as *front half-space* if i lies in the front half-space of a significant edge in the color image (Figure 4.8b). If pixel i lies in the back half-space instead (Figure 4.8c), it is labeled as *back half-space*. If there is no edge nearby, it is marked as *no edge*. To realize this distinction, significant edges in the color image are detected by applying a 3×3 Sobel filter. Significant edge points are then extracted by applying a threshold to the gradient magnitudes. For each unknown pixel i, the method then collects edge points

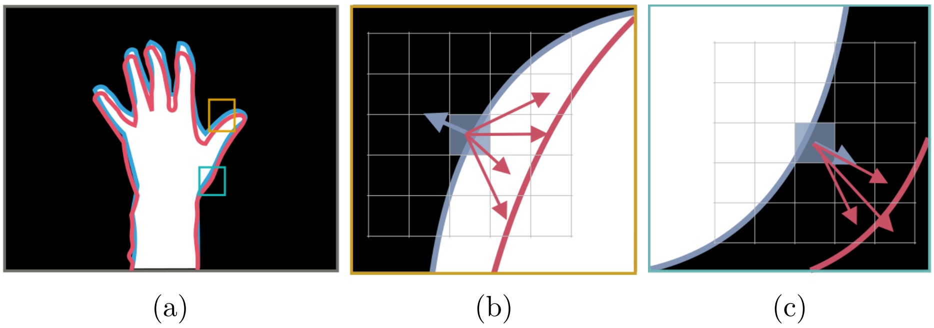

(a) (b) (c)

Figure 4.8: **Schematic illustration of the labeling procedure.** (a) coarse segmentation between foreground, background and unknown regions (blue) overlaid with boundaries in the color image (red), (b) the unknown pixel lies in the front half-space of a significant edge in the color image, (c) the unknown pixel lies in the back half-space of a significant edge in the color image.

of the color boundaries within a specific search window. Based on the smoothed gradient of the unknown pixel, the algorithm decides whether the pixel lies in the front or back half-space of the selected edge points. This can be decided efficiently by calculating the scalar product, as visualized in Figure 4.8. If only a few edge points are selected, the unknown pixel is labeled as *no edge*. The results of the labeling step are shown in Figures 4.7d and 4.7j.

4.6.3 Adaptive Dilation

As mentioned before, the idea behind adaptive dilation is to extend unknown regions in a such way that both depth and color boundaries are covered while keeping the unknown regions as small as possible. The dilation process consists of checking for each pixel $k \in F, B$ if there exists a pixel $i \in U$ within a window around k. Pixel k is also labeled as unknown if there exists such a pixel i which is either labeled as *front half-space* and k lies in back half-space of i or labeled as *back half-space* and k lies in front half-space of i. Figure 4.7e shows a simple dilation. The corresponding trimap generated by the adaptive dilation is shown in Figure 4.7f. Different unknown regions are obtained, and the adaptive dilation extends unknown regions towards the boundaries of the color image.

However, large fuzzy areas are not always covered by the unknown regions. Generally, regions with sharp edges only need small extensions, whereas fuzzy regions or those with corners need a wider extension (see Section 4.4.3). To solve this problem, the amount of dilation depends on the number of *no edge* labeled points e. If e is above a threshold, the dilation amount is increased. The threshold should depend on the size of the structuring element s (where $\frac{s}{3}$ is a good choice). This results in a larger dilation for fuzzy foreground objects (Figure 4.7l). Figure 4.7k shows a trimap based on a structuring element, in which the size is fixed.

4.7 Foreground and Background Propagation

For robust alpha estimation, it is necessary to choose good samples of the known foreground and background colors. Therefore, the known background and foreground colors are propagated towards the unknown regions, while maintaining a distance metric to calculate the uncertainty of the propagation. This pre-processing step allows for the use of a universal search window for the alpha estimation, which reduces the cost of selecting samples.

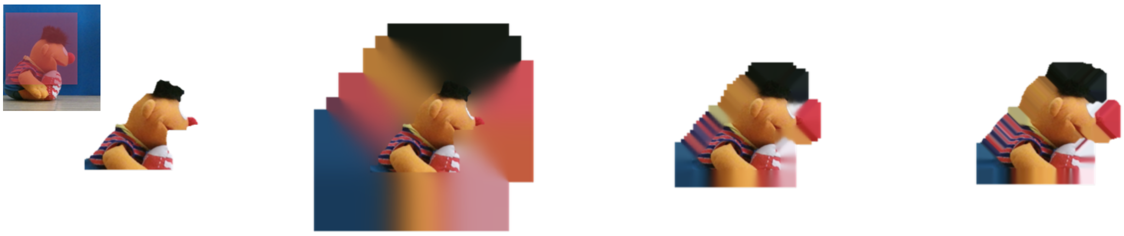
The main idea of the propagation method is to blur the whole image, foreground or background, and to write back only blurred colors for unknown regions. If this step is processed repeatedly, the known colors diffuse to the unknown regions. The idea of this algorithm is similar to the hole-filling technique for 3D volumes [DMG+02]. The realization of a fast blurring is inspired by pyramid-based filter algorithms [Bur81; OAB+85]. Such approaches create an image pyramid by downscaling the input many times, up to a specific level. Later, the lower resolution images are used to blur the source image.

The process is explained through following the steps of the propagation of foreground pixels; the background propagation works identically. Algorithm 3 illustrates the main steps of the diffusion for a single iteration.

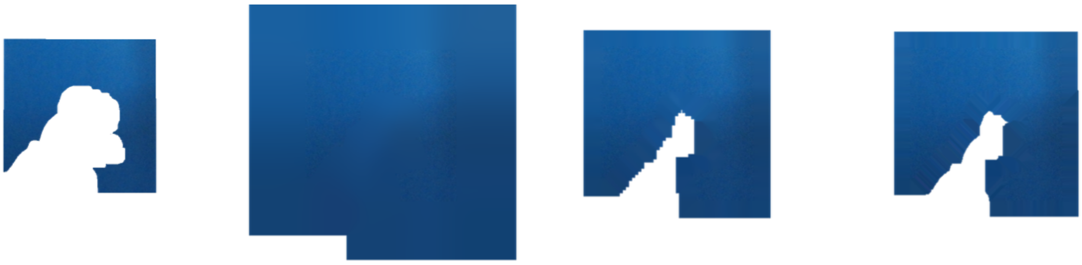
Algorithm 3 Diffusion algorithm for a single iteration i.

```
 1: copy input image $F$ with 4 channels to image $S$
 2: for each pixel $p^s$ of the image $S$ do
 3:     if $p^s_a > 0$ then                                  ▷ color value is unknown
 4:         $p^s_a \leftarrow 1$
 5:     else
 6:         $p^s_a \leftarrow 0$
 7:     end if
 8: end for
 9: build image pyramid of $S$ with $l$ levels
10: go top-down the pyramid to smooth $S$
11: for each pixel $p^f$ of the input image $F$ do
12:     if $p^s_a > 0$ then                    ▷ corresponding smoothed color is known
13:         $w^{i-1} \leftarrow \frac{p^f_a}{n}$                       ▷ calculate normalized weight
14:         $p^f_{rgb} \leftarrow w_{i-1}p^f_{rgb} + (1 - w_{i-1})p^s_{rgb}$
15:         $p^f_a \leftarrow min(p^f_a + 1, n)$                  ▷ increase and clamp alpha
16:     end if
17: end for
```

First, the foreground image F is copied (line 1) and an image pyramid with l levels of the copy S is created (lines 2-9). The finest level is initialized with all known foreground colors and their α channels are set to 1 (line 4). Unknown pixels are initialized by $\alpha = 0$ (line 6). The coarser levels are created by reducing the resolution of the previous finer level using a filtering operation (e.g. Gaussian filter) on the collapsing known foreground colors (line 9). The following blurring steps work top-down (line 10). New pixel colors of finer levels are calculated from the coarser levels by a quadratic B-Spline interpolation (as in [CC78]). The interpolation factors are weighted by the α values. Thus, unknown regions are not involved during this filtering process. The next steps (lines 11-18) write the smoothed foreground colors back into the input image F, where the colors are unknown. To achieve a smooth transition between two blurring steps, a linear interpolation between the previous colors and the new one (lines 13-15) is processed, where value n regularizes how often a color value is interpolated. To maintain the original colors the α values of F (with $\alpha > 0$) are initialized in the first iteration ($i = 0$) with n. If these steps are processed repeatedly, the known values diffuse towards unknown regions. Additionally, for each filled pixel, the corresponding diffusion iteration is saved as a distance metric, as mentioned. The amount of diffusion depends on the number of pyramid levels l and the number of diffusion steps i. A small number of levels leads to clearly diffused unknown colors and fills unknown regions by processing the diffusion several times. Conversely, a high number of levels results in smooth propagated values and fills unknown regions by processing the diffusion only a few times. Figure 4.9 shows the results of the propagation process and the interaction between the number of pyramid levels and diffusion steps. The values $l = 4$ and $i = 5$ are good compromises in terms of blurring, filling, and performance in several of the presented test cases. To profit from hardware optimization, the implementation utilizes mipmapping[2] to increase texture access times. The whole pyramid is represented as a texture with mipmap levels that are bound to framebuffer objects, which allows a manual mipmapping process.



(a) Foreground (b) $l = 6$, $i = 4$ (c) $l = 4$, $i = 5$ (d) $l = 2$, $i = 20$



(e) Background (f) $l = 6$, $i = 4$ (g) $l = 4$, $i = 5$ (h) $l = 2$, $i = 20$

Figure 4.9: **Propagation of foreground and background.** Different propagation results of an example scene shown in (a). (a) Shows definitive foreground colors and (b) shows definitive background colors. Panels (b) to (d) show propagated foreground colors with different numbers of pyramid levels l and diffusions i. Panels (f) to (h) show the propagated background colors.

---

[2] Mipmapping is a method of precomputing textures of reduced resolution [Bus03].

4.8 Alpha Estimation

To estimate the alpha matte on the basis of the propagated foreground and background colors a sample-based method is realized. Local samples of the foreground and the background are collected, pairs are formed and the best pair by an optimization of an objective function is selected. The objective function takes color information and the propagation distance metric into account. For a given sample pair F_i, B_j (with indexes i and j), alpha can be estimated by the following equation:

$$\hat{\alpha}_p(F_i, B_j, I_p) = \frac{(I_p - B_j)(F_i - B_j)}{\|F_i - B_j\|^2}, \tag{4.2}$$

where I_p is the color of pixel p.

Color Cost A good sample pair should be able to represent the pixel I_p of the color image as a linear combination of themselves. Thus, for a good pair of foreground and background colors F_i and B_i the color cost C_{col} should be small:

$$C_{col}(F_i, B_j, I_p) = \|I_p - (\hat{\alpha} F_i + (1 - \hat{\alpha}) B_j)\|, \tag{4.3}$$

$\hat{\alpha}$ is the estimated alpha value for I_p based on the sample pair F_i, B_j (using Equation 4.2). This type of function is used in other sample-based alpha matting methods [GO10; HRR$^+$11; RRG08].

Propagation Cost In addition to the color information, information about the confidence of the propagated foreground and background colors is used. This reduces the effect of incorrect propagated colors and helps to achieve quality results. Each foreground and background pixel of the propagated maps retains information about which diffusion step the pixel was created in (as described in Section 4.7). Colors that are created during the first diffusion lie close to the known foreground and background colors, and have a high confidence. They should result in a low cost. To achieve this, the normalized propagation cost C_{pro} for a sample pair (F_i, B_j) is defined by:

$$C_{pro}(F_i, B_j) = \frac{d(F_i) + d(B_j)}{2d_m}, \tag{4.4}$$

where the function d returns the number of iterations d_i (with $d_i \in [0, d_m - 1]$) when the color (F_i or B_j) is created. The term d_m represents the number of total diffusions.

Objective Function To find the best pairs, first the possible sample candidates in a $n \times n$ neighborhood around the unknown pixel p are selected from the propagated foreground and background colors $(\{F_i, \ldots, F_{nn}\}, \{B_j, \ldots, B_{nn}\})$. Then an objective function that combines the color and propagation cost for all possible pairs (F_i, B_j) is minimized:

$$(\hat{F}_i, \hat{B}_j) = \arg\min_{F,B} \; w \; C_{col}(F_i, B_j, I_p) + C_{pro}(F_i, B_j) \tag{4.5}$$

where w is a weight that defines the influence of the color cost. The final alpha value $\hat{\alpha}_p$ of pixel p is then estimated based on the best sample pair $(\hat{F}_i, \hat{B}_j)$. For completeness, the alpha values of known foreground pixels are set to 1 and alpha values of known background pixels are set to 0. The alpha values of the virtual object are then calculated by the inversion $(1 - \hat{\alpha}_p)$.

4.9 Experimental Results

To prove the viability of the presented algorithm in real scenarios, different scenes that were representative for real world scenarios were tested. In addition, runtime performance of the algorithm was measured because real-time capability is a crucial factor in an AR experience.

4.9.1 Implementation Details and Runtime Performance

The algorithm is implemented in C++ and GLSL and is portable to mobile devices using OpenGL for embedded systems (OpenGL ES). The algorithm performs all operations on the graphics card as each step can be computed independently per pixel. This includes the image-processing tasks and the optimization process to select good sample pairs. Thus, the algorithm do not depend on data transfer between the CPU and GPU, apart from the color and depth data at the start to eliminate potential bottlenecks.

The time measurements were obtained for an image resolution of 1920×1080 (color and depth image), whereas the propagation of known values operated on a 1024×1024 window around the area of interest. The texture upload time ($\sim 7.5\,\text{ms}$) was excluded in all test runs as it was a constant time factor, independent of the contribution. All benchmarks were executed on a test system with an Intel i7 4770K processor and an NVIDIA GeForce GTX 1080 graphics card. Both devices were used with fixed clock frequencies and disabled turbo settings to reduce variations during the measurements.

As shown in Table 4.1, the number of unknown pixels had no significant influence on the overall performance of the algorithm. Interestingly, the alpha estimation was the only part that depended on unknown pixels. However, the algorithm had a nearly constant runtime for the generated trimaps in full high definition resolution. The trimap in Figure 4.7f, for example, had 42748 unknown pixels and produced good results. Although the elapsed time increased for ~ 100000 unknown pixels, it remained insignificant compared to the total runtime. The computationally most expensive step was the propagation of known

No. unknown pixels	16703	46129	99653
Trimap (ms)	0.252862	0.263313	0.28445
Propagation (ms)	1.147508	1.145953	1.118089
Alpha (ms)	0.04997	0.04639	0.13538
Total (ms)	1.45034	1.455656	1.537919

Table 4.1: **Performance measurements** for different numbers of unknown pixels for $l = 4$ mipmap levels and $i = 5$ diffusion steps.

values into unknown regions. There was a linear correlation between the runtime of the propagation and the number of diffusion steps and mipmap levels (see Table 4.2). The algorithm required ~ 1.5 milliseconds to generate visually pleasing results and therefore surpassed real-time requirements for modern hardware.

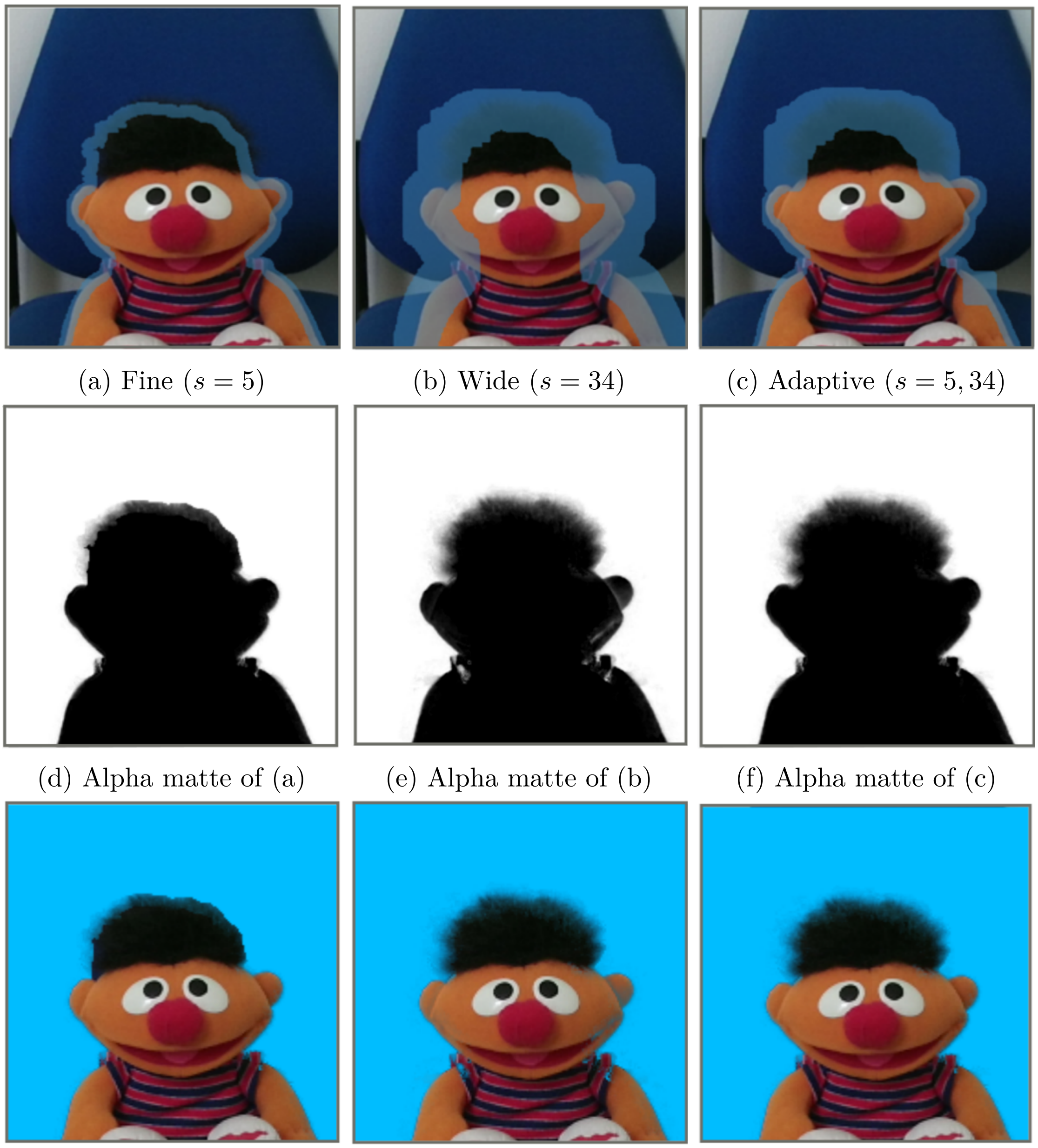
Diffusion steps i	mipmap levels l	Time in ms
10	2	0.740895
10	5	2.049906
10	7	3.030899
1	2	0.112646
20	2	1.54018
40	2	2.950295

Table 4.2: **Performance measurements** for the propagation of known values in milliseconds (ms).

4.9.2 Evaluation of the Algorithm

This section describes how the adaptive trimap generation affected the alpha matte estimation. An alpha matte obtained with different extended trimaps is shown in Figure 4.10. Panel 4.10d shows that fine unknown regions, obtained by a simple dilation, resulted in an alpha matte of poor quality. Foreground and background values were wrongly propagated because the unknown regions did not include fuzzy areas of the color image. Wide unknown regions included fine details but led to a loss of color information (Figure 4.10e). A good trade-off was obtained by the adaptive generated trimap (Figure 4.10f).

The effects of the objective function are also evaluated in this section. Figure 4.11 shows that the effect of wrongly propagated colors was reduced by using the objective function. Moreover, the use of the objective function yielded better results if the separation between foreground and background colors was ambiguous. This effect is shown in Figure 4.11f, where the pillow in the foreground has partially similar colors to those of the wall in the background. Generally, the algorithm achieved good results in quality and runtime by selecting the sample pairs in a 9×9 neighborhood.


(a) Fine ($s = 5$) (b) Wide ($s = 34$) (c) Adaptive ($s = 5, 34$)

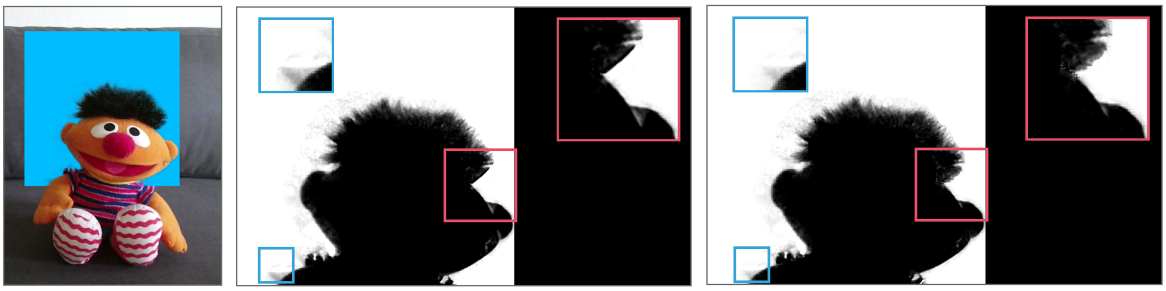
(d) Alpha matte of (a) (e) Alpha matte of (b) (f) Alpha matte of (c)

(g) Composite of (d) (h) Composite of (e) (i) Composite of (f)

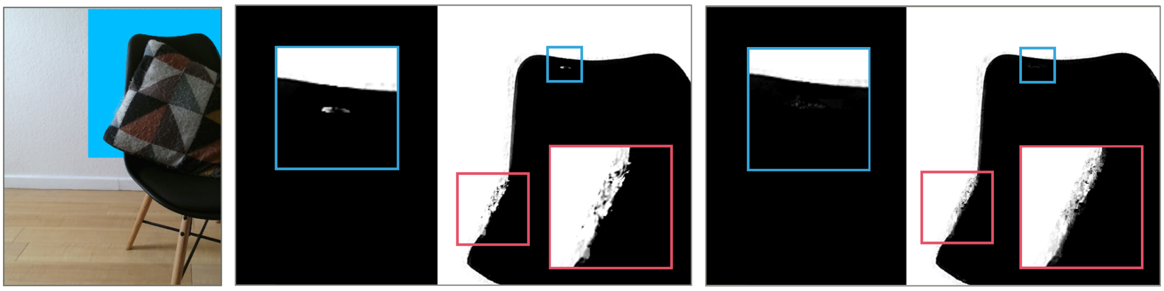
Figure 4.10: **Comparison of trimaps and their effects on the alpha matte.** Panels (a) to (c) show unknown regions (colored blue) obtained by different dilation procedures, where s is the size of the squared structuring element. Panels (d) to (f) represent the alpha mattes. Panels (g) to (i) are the final occlusion results.

4.9.3 Comparison with Edge-aware Filtering Methods

Different scenarios were tested to demonstrate the robustness and transferability of the algorithm and to compare the results with edge-aware filtering methods, including the guided image filtering [HST13] and adaptive manifolds [GO12]. In a simple scene with



(a) Composite (b) Alpha matte (without) (c) Alpha matte (with)

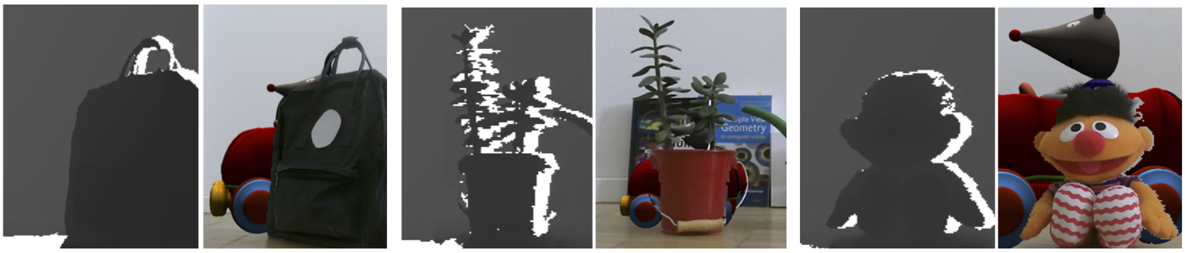


(d) Composite (e) Alpha matte (without) (f) Alpha matte (with)

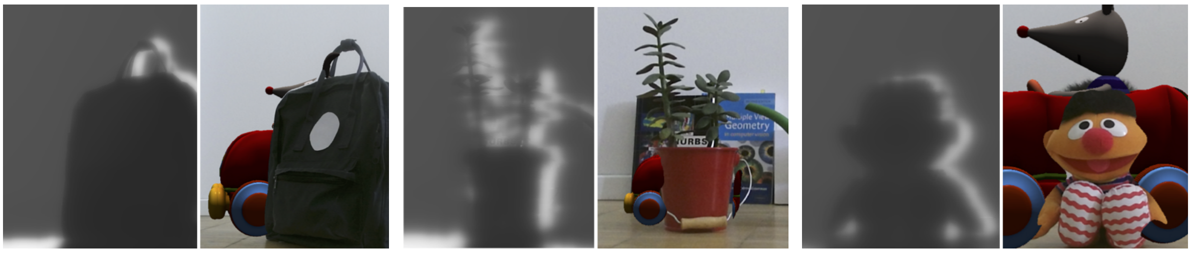
Figure 4.11: **Effect of the objective function: two examples.** Panels (a) and (d) show the composites using alpha matte (c) and (f) respectively. Panels (b) and (e) show the alpha mattes calculated directly from the propagated foreground and background colors. Panels (c) and (f) present the alpha mattes obtained by the sample pair that best fits the objective function (within a 9 × 9 neighborhood). Interesting regions are highlighted with rectangles.

clear and hard borders the resulting depth of both edge-aware filters was smoother than the raw depth data, which led to acceptable occlusions (Figure 4.12a, 4.12d and 4.12g). The zigzag pattern of the raw depth was completely eliminated. However, the manifold filter produced wrong occlusions in some regions, whereas the guided filtering provided consistently good results. Occlusion matting generated a nearly perfect alpha matte and occlusion.

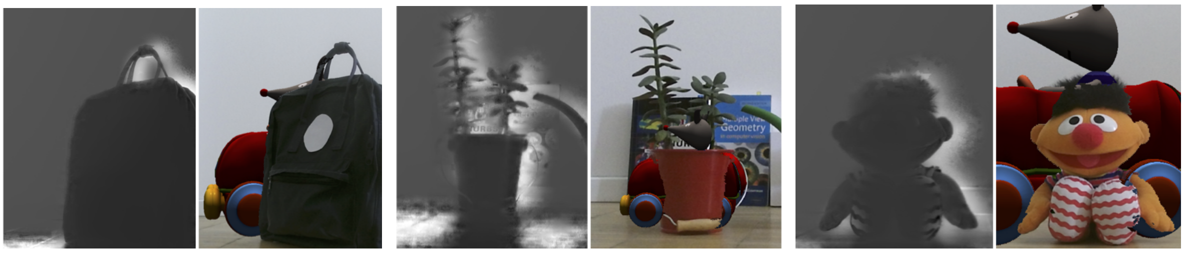
Filigree objects were heavily blurred by the manifold filter, which resulted in wrong depth values and an occluded head of the virtual object (Figure 4.12e). The guided filter, although not as blurry, suffered from the same problem (Figure 4.12h). Additionally, the background clutter influenced the filtering of depth values within the plant. The presented approach generated a visually pleasing occlusion, although the alpha matte had small artifacts (Figure 4.12n). This was due to the soft transition in areas having



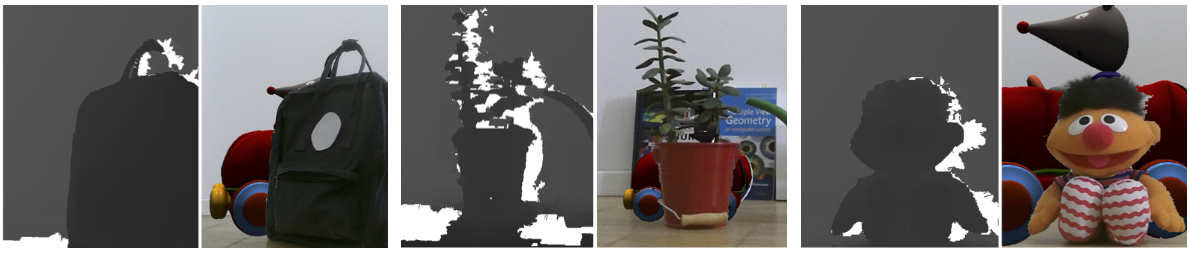
(a) Raw (b) Raw (c) Raw



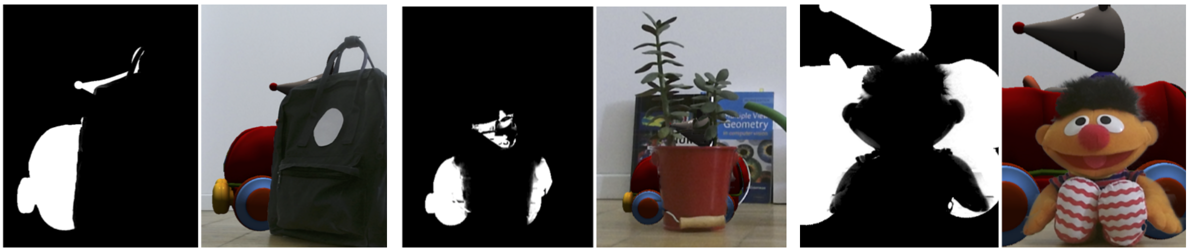
(d) Manifold Filtering (e) Manifold Filtering (f) Manifold Filtering



(g) Guided Filtering (h) Guided Filtering (i) Guided Filtering



(j) Edge Snapping-Based (k) Edge Snapping-Based (l) Edge Snapping-Based



(m) Occlusion Matting (n) Occlusion Matting (o) Occlusion Matting

Figure 4.12: **Overall comparison with state-of-the-art methods.** The algorithm was compared with raw depth data, manifold filtering, guided filtering, and an edge snapping-based approach in three AR scenarios. See Section 4.9.3 and Section 4.9.4 for detailed explanation and discussion.

high uncertainty. Fuzzy objects were also challenging. In comparison to occlusion matting the edge-aware filters were unable to achieve a smooth transition between fuzzy regions and the virtual object (Figure 4.12f, Figure 4.12i and Figure 4.12o).

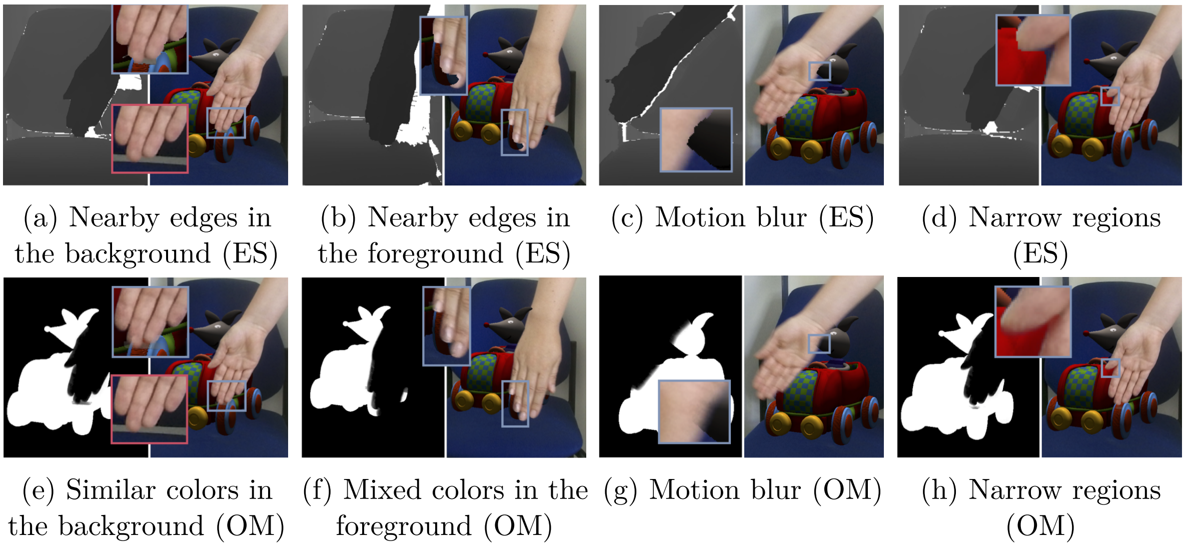

(a) Nearby edges in the background (ES)
(b) Nearby edges in the foreground (ES)
(c) Motion blur (ES)
(d) Narrow regions (ES)
(e) Similar colors in the background (OM)
(f) Mixed colors in the foreground (OM)
(g) Motion blur (OM)
(h) Narrow regions (OM)

Figure 4.13: **Comparison with edge snapping-based method.** Results for various hand scenes with common difficulties. Panels (a) to (d) show results of the edge snapping-based approach (ES). Panels (e) to (h) show results of the occlusion matting approach (OM).

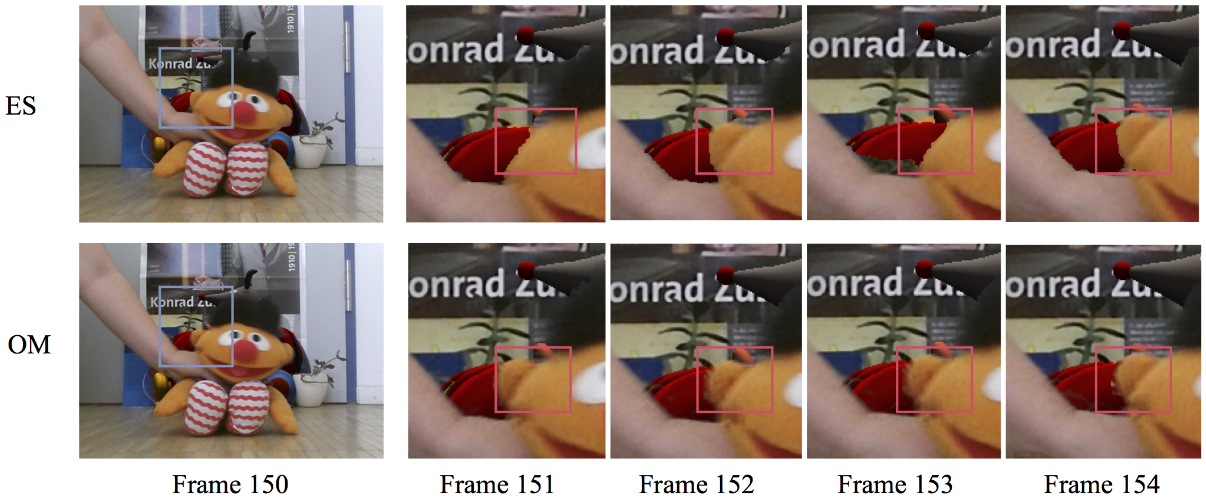
4.9.4 Comparison with Edge Snapping-Based Method

This section provides an extensive comparison between the presented method and the edge snapping-based depth enhancement by Du *et al.* [DCY+16]. The method of Du *et al.* is the latest state-of-the-art approach that deals with dynamic occlusion in AR scenarios.

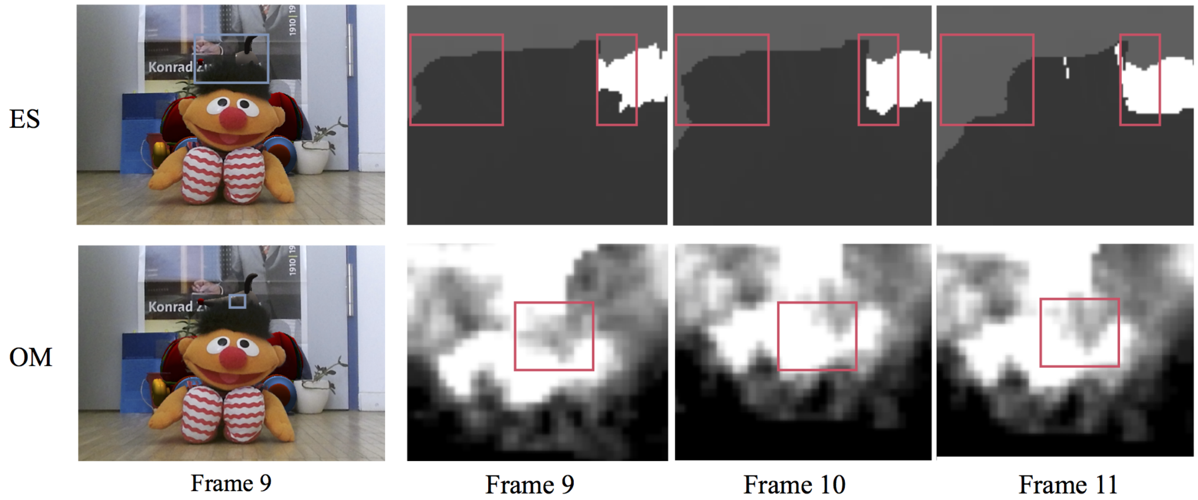
Du *et al.* focused on refining boundaries of skin colored objects like hands by snapping depth edges to color edges. Figure 4.13 shows such a case. It illustrates significant frames with challenging situations, such as ambiguous colors or edges in the background. The edge-snapping-based algorithm produced more visual artifacts (Figure 4.13a) than the occlusion matting approach (Figure 4.13e). The edge snapping-based method suffered from nearby edges and followed a seemingly stronger but incorrect edge (blue chair border), resulting in false occlusions. The presented method had difficulties in distinguish foreground versus background, because of similar colors. This uncertainty was reflected in the alpha values. The final result, however, was much more appealing than that of the edge snapping-based method, as the transparent transition was less obvious than the hard border in Figure 4.13a. Figure 4.13b shows a similar effect for the method of Du *et al.*; the algorithm chose undesired gradients (the edge near the fingernail in the foreground), which resulted in a wrong boundary. The presented approach (Figure 4.13f), given the easily distinguishable colors, produced a correct, smooth transition. In Figure 4.13c, the border had shifted inside the foreground object due to motion blur. Blurred edges tend to have weak gradients on the outside, although a visible boundary remains. Edge based-snapping, however, prefers to snap to strong gradients inside the object. Here the occlusion matting procedure profited from the smooth alpha transition and provided a more realistic object boundary.

Figure 4.13d and Figure 4.13h depict the differences in finding borders of regions that were very narrow compared to the whole object. The edge snapping-based approach generated a round border, based on its optimization process, whereas occlusion matting resulted in a clear boundary. Figure 4.14 gives two examples of how both algorithms behaved temporally. In contrast to the presented approach, the edge-based-snapping tended to switch its boundary from frame-to-frame if ambiguous edges were present (Figure 4.14a and Figure 4.14b). This resulted in considerable temporal discontinuities. The presented algorithm tended to generate small flickering effects because of noisy depth values (Figure 4.14b), which becomes noticeable in static scenes.

Generally, the edge snapping-based approach achieved appealing results in regions having clear foreground boundaries (Figure 4.13 and Figure 4.12i). However, the method could not deal with ambiguous edges in the background (Figure 4.12k) and foreground (see the ear of the cuddly toy in Figure 4.12l). Moreover, it was unsuitable for generating smooth transitions on fuzzy intersections (Figure 4.12l). In such scenes, it was hard to choose good parameters for the neighbor weights in such a way that the edges followed the zigzag



(a) Dynamic object



(b) Static object

Figure 4.14: **Temporal comparison.** Results of edge snapping (ES) and occlusion matting (OM) in significant frames of a scene with a dynamic object and static object. (a) Shows the final occlusion results; (b) shows the enhanced depth map and the alpha matte. Regions of interest are highlighted with red rectangles.

pattern while avoiding incorrect correspondences. The presented method combined both characteristics and was able to generate hard borders, as well as soft transitions where needed (Figure 4.12o).

4.9.5 Comparison of Runtime Performance

The performance of all algorithms on different image resolutions (see Table 4.3) was compared using the same system as in Section 4.9.1. According to Du *et al.* [DCY+16] edge based-snapping maintained acceptable frame rates on low resolution images (22 ms to 40 ms for 640 × 480 images). In the current work, an own implementation of their approach was used that relied on the CPU alone, and was therefore slower. Nonetheless, the frame time increased drastically for higher resolutions. The same applied to the manifold and guided filter, as their runtimes more than doubled from 720p to 1080p (Table 4.3). As shown in Section 4.9.1, presented algorithm's runtime mainly depended on the propagation of known values, but achieved real-time performance. Table 4.3 shows that the performance did not change significantly for higher resolutions.

Resolution	1920 × 1080	1280 × 720
Edge snapping (ms)	251.284	118.593
Guided filter (ms)	95.583	41.078
Manifold filter (ms)	170.502	70.531
Occlusion matting (ms)	1.481 (9.025)	1.372 (4.257)

Table 4.3: **Comparison of runtime performance for several state-of-the-art algorithms.** The time is the average of 500 executions. Brackets show the total calculation time with texture upload.

Furthermore, the table illustrates the increased runtime of the other algorithms, depending on image resolution. Although an increase in calculation time by ~0.1 ms from 720p to 1080p is evident, the main difference is caused by the upload time of the necessary texture data.

4.10 Limitations

The evaluation showed that the algorithm can produce visually pleasing results, even in areas in which previous techniques failed. However, it is often hard to find good sample pairs in regions where foreground and background colors are difficult to distinguish. This is a general problem in alpha matting and can be improved by a better split into known and unknown regions using structural information from the color image. Furthermore, it might be beneficial to use a more advanced objective function. Gastal and Oliveira [GO10], for example, found a good sample pair by minimizing the chromatic cost within a specific neighborhood of the input pixel. Furthermore, noisy depth data, especially at the boundaries, results in flickering that is visible throughout multiple frames. The raw depth data could be filtered to reduce potential noise, or temporal information could be used saving the trimaps of previous frames. Another limitation was that large holes in foreground objects ere not handled, as invalid depth values marked as definitive background. Small holes were usually not a problem if the size of the unknown regions were large enough to cover the hole. Hole-filling techniques [YKP12; XZW12; LCK16] could be used to overcome this limitation. However, such algorithms are often able to deal only with small holes and are often very time-consuming. The next section (Section 4.11) presents a simple and efficient solution.

4.11 Optimizing Sensor Data with 3D Reconstruction

As mentioned in Section 4.2.3, depth maps obtained from today's low-cost sensors have several limitations that depth-based occlusion handling methods must deal with: 1) noise, especially at the boundaries; 2) inaccurate mapping between color and depth data; and 3) incomplete data. Occlusion matting overcomes the limitations regarding noise and inaccurate mapping, but has problems dealing with large holes in the foreground for which no depth data can be obtained.

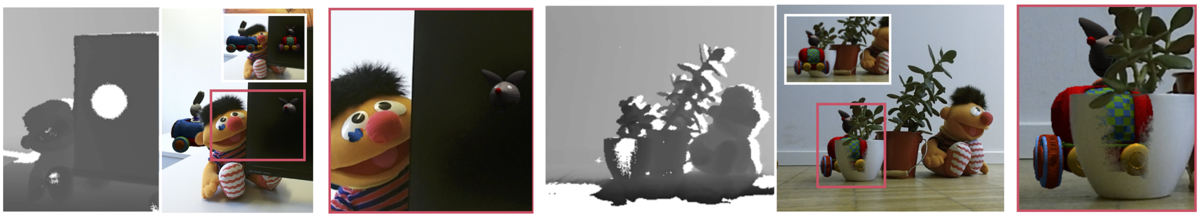
Depth maps of the scene can be also obtained from an existing model of the scene, as is the case with model-based methods (Section 4.2.2). Generally, model-based methods need an accurate and dense model or reconstruction of the environment. If a complete model of the scene, which exactly matches its counterpart in the real scene is available, model-based methods benefit from complete depth maps. However, these methods suffer from registration errors and jitter, which results in imprecise and unpleasant occlusions. Because the alpha estimation of the occlusion matting relies exclusively on the RGB data in the regions that separate the foreground from the background, small registration errors or jitter effects can almost be eliminated. However, limitations such as incomplete models or dynamic objects mean that model-based methods still tend to be impractical in many AR scenarios.

To benefit from both, depth-based and model-based methods, the depth map obtained by a low-cost sensor is optimized with data from the reconstruction (as presented in Chapter 3). Large holes in the sensor map mainly arise because of shadowing effects, reflections (Figure 4.15a) or limited sensor ranges (Figure 4.15b); they can be filled in with reconstruction data (Figures 4.15a and 4.15b). It is quite likely that a user observes a scene from several viewpoints and thereby reconstructs critical parts of the scene so that missing parts can be filled in. Sensor data instead are primarily useful for dealing with dynamic objects or incomplete or imprecise reconstruction data (as visualized in Figures 4.15c and 4.15d).

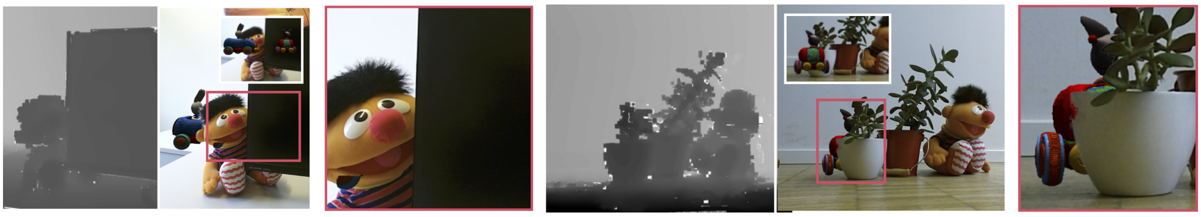
To optimize the sensor depth map, the reconstruction is rendered into a depth map from the current view. Depth values are then easily combined by preferring the sensor depth values and filling invalid values with the rendered depth values. As shown in Figure 4.15, large holes in the sensor data were correctly filled and clearly improved the occlusion.

4.12 Conclusion and Future Work

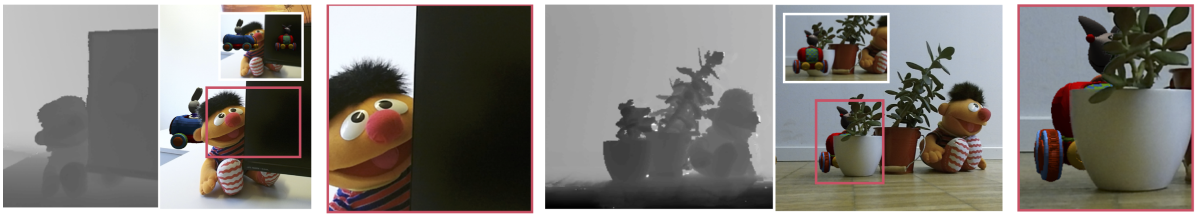
Occlusion matting combines and extends ideas from different research areas. A novel real-time approach was created to solve the occlusion problem in several AR scenarios.



(a) Sensor data (b) Sensor data



(c) Reconstruction data (d) Reconstruction data



(e) Optimized sensor data (f) Optimized sensor data

Figure 4.15: **Comparison of occlusion matting results for different depth maps: two examples.** Panels (a) and (b) show occlusion matting results based on raw depth maps obtained by a Kinect depth sensor; (c) and (d) show results based on depth maps created with reconstruction data; and (e) and(f) show results based on optimized depth maps that combine sensor and reconstruction data. Small images outlined in white show the scene with occlusion handling, whereas images outlined in red highlight critical parts of the scene. For a detailed discussion, see Section 4.11.

Reinterpreting the known challenge as an alpha matting problem enables not only to separate visible and non-visible regions of the virtual object, but also to realize smooth transitions by estimating an alpha value as a blending coefficient.

To handle a broad range of occlusion types (clear and soft object boundaries), an adaptive trimap generation was introduced that extended the regions of unknown pixels. The extension was influenced by the boundary and fuzziness in the color image. Based on the trimap the colors of the known foreground and background were propagated towards unknown regions. This allowed the use of a fixed search window while looking for suitable sample pairs, and led to the real-time capability of the presented algorithm. Section 4.9 verifies that occlusion matting compares well with current state-of-the-art techniques, while maintaining real-time capability and offering wider versatility due to smoother transitions. Occlusion matting suffers from the same problems as alpha matting approaches in general. Small artifacts occur because of similar foreground and background colors or from flickering throughout multiple frames, flickering is especially visible in static scenes. Section 4.10 presents more details on these issues and offers suggestions for future research. Generally, future work should concentrate on solving the occlusion problem as an alpha matting problem.

Chapter 5

Conclusion

This chapter summarizes the presented approaches to solve the identified problems, highlights the main achievements, discusses suggestions for future work.

5.1 Summary and Achievements

This thesis has presented various approaches to overcoming well-known problems in AR, and offers the realization of more believable, realistic AR applications. The problems of accurate and jitter-free registration and tracking, complete and de-noised 3D reconstructions and realistic occlusion handling have been researched and discussed.

Registration and Tracking. The presented tracking system can track an object based on a line model, and overcomes known limitations of standard edge-based trackers using an appearance-based line model. While state-of-the-art approaches still require a complete and accurate line model as input, a coarse line model is sufficient for the presented method. It is no longer necessary to expend great effort to manual create a suitable model. The presented approach performs a mapping and refinement step to generate a suitable model. As the user moves around new lines are added to the model and existing ones are refined.

Lines are added to the model if they appear and are detected by standard edge-detection algorithms in the camera stream. It is likely that lines which occur often will be detected again in subsequent frames.

In this context, the idea of so-called keyposes was presented. Each keypose retains a camera pose and a set of lines that are likely to be visible for similar camera poses. This data enables a selection of good edges to track; it also enables detail-based culling, automatic hidden-line removal, and an enhanced re-initialization procedure.

3D Surface Reconstruction. The proposed algorithm to optimize 3D surface reconstructions offers new possibilities for realistic AR applications. Realistic interactions between the real and virtual worlds, such as collisions, are realizable. Whereas previous work mainly focused on improving surface reconstructions offline after a capturing step, the presented method improves and extends the reconstruction during the capturing process; it also fills holes. User's are not required to capture the environment first but can explore it without any preparation. Moreover, interactions between the real and virtual world are not limited to captured regions or static parts of the scene. The presented approach extends the well-known KinectFusion pipeline and estimates local planes in subregions of the reconstruction volume. Local planes are used to de-noise and propagate planar structures of the current observed surface towards unknown regions. The subsequent surface diffusion can to fill small gaps and holes, especially on curved structures that lack dominant planes.

Generally, the method de-noises and extends, and fills holes in planar structures such as table tops, walls, or floors which are common in man-made environments. In contrast to existing work, the presented approach can deal with several types of structures other than flat ones. It also extends and fill holes in curved structures like bottles or balls, in a plausible manner.

The thesis has described how to use an implicit representation, represented by a volume allocated on the GPU memory, to realize collisions and occlusions without transferring the reconstruction data to the CPU memory or extracting an explicit representation, such as a triangle mesh. Thereby, the proposed method is real-time capable.

Occlusion Handling. The novel approach presented here, for handling occlusions by reinterpreting the occlusion challenge as an alpha matting problem offers new perspectives to solve the occlusion problem more realistically. Whereas state-of-the-art methods calculate whether pixels of the rendered scene are visible or non-visible, the presented approach estimates a blending coefficient per pixel. This enables more realistic occlusions in regions where the foreground and background are not strictly separable (e.g. caused by fuzzy objects like fur or hair, or by motion blur). The presented algorithm estimates an adaptive trimap to enable realistic occlusions, both in regions where foreground and background are clearly separable and in regions where they are not strictly separable. If definitive foreground, background, and unknown regions are defined, the algorithm quickly propagates known foreground and background color of the boundaries towards unknown regions. It then estimates an alpha matte based on an objective function, which takes color information and a propagation distance metric into account.

The results showed that the approach generated a high-quality alpha matte without any user input; it also overcame the limitations of low-cost sensor data and outperformed previous work in terms of quality, realism, and practical applicability. Furthermore, the presented algorithm solved the alpha matting within real-time rates by using the capabilities of the GPU hardware. This feature not only provides real-time occlusion matting but also offers opportunities in other fields of research, such as video matting.

5.2 Future Work

This thesis has reviewed and explained different approaches to realize more realistic and believable AR applications. However, several problems remain that need to be solved to meet the goal of merging the real and virtual worlds into a single unit.

Combination with other tracking approaches. This thesis has focused on an edge-based tracking approach using a coarse line model. The presented algorithm improves the robustness of the pose estimation and eliminates time-consuming pre-processing steps. However, it would be beneficial to combine the approach with other methods to become even more robust and practical for AR. Currently, the user has to move the camera towards a specific camera pose before the camera is registered to the model and the tracker is initialized. A tracking-by-detection approach could allow the tracking to be initialized or re-initialized without requiring user interaction. Currently, it is common to analyze the model to learn features, such as point-features, during an offline stage, and then during the online stage, to use those features to initialize and re-initialize the tracking. Research to date almost invariably used point-feature descriptors [LVT$^+$03; BPS05; RPS$^+$18] to realize the initialization. However, other approaches have described a line feature in an image [WLW09; WWH09; HS12], which are currently not used in a model-based approach. Future research could analyze whether these methods are suitable. Moreover, if structural information about the scene is available (e.g. obtained from a RGB-D sensor), it seems preferable to use a feature descriptor that considers the geometric structure of a feature, instead of only its appearance in the RGB image. The approach would then remain suitable for poorly textured scenes; only a polygonal model without texture would be required. Another suggestion for future work is to use the presented approach in a SLAM context. First, it can be used to offer an absolute reference to register virtual objects in the world; second, it can be used to reduce drift. The idea is that one or more models of static objects in the environment serve as anchors. Instead of registering virtual objects to a single coordinate system, they are registered in the coordinate system of the anchor next to them. The tracking and mapping starts and is adapted (e.g. to the current scale) whenever an anchor is detected.

Considering dynamic and deformable objects. While the presented approach to handling occlusions can deal with both dynamic and static objects, the presented approach to surface reconstruction assumes a static scene. If dynamic or deformable objects – such as a moving person – appear in the scene, the algorithm tries to integrate these objects into the current reconstruction. This results in incorrect surfaces, which do not look appealing and can result in incorrect collision or occlusion handling. One approach could be to ignore dynamic and deformable objects so they are not even integrated. An algorithm would be needed that detects which parts of the incoming depth image represent such an object. This is a challenging task in computer vision, particularly if the camera is also moving [WHN17]. If it is known which pixels of the incoming depth image correspond to moving or changeable objects, a depth-based collision handling method (e.g. [BRW95; SYT+12]) would be the next step. Another solution might be to reconstruct dynamic and deformable objects separately. Reconstructing deformable objects is a well-known problem and is useful in several other fields of research, such as body motion capturing. For example, a proposed approach called *DynamicFusion* [NFS15] addressed this issue and offers a good starting point. The researchers in that work adapted the idea of KinectFusion to reconstruct and track a non-rigidly deforming shape in real-time.

Semantic understanding of the real world. The presented approach to reconstruct the real world focuses on a complete geometric representation and enables geometry-aware AR interactions between real and virtual objects. In addition to a geometric reconstruction, semantic segmentation of the real world enables several possibilities in AR. It can be used to realize physical interactions even more realistically. Deep understanding of the semantics of the scene can help to simulate the behavior of virtual objects. For example, if information about the material properties are available – like segmentation into two classes, *hard* and *soft* – collisions between objects of both worlds can be handled more believably. A virtual drinking glass will be broken if it is dropped onto the stone floor and will remain intact if dropped onto a sofa. Information about material properties is also useful for occlusion handling. Occlusions can be handled more realistically if it is known whether an occluding object is transparent or opaque. Semantic understanding can therefore support the realization of AR applications. Dzitsiuk *et al.* [DSM+17] showed that detected planes could be used to obtain segmentation in classes such as *floor*, *wall*, *ceiling*, and *other*. They proposed using such segmentation to remove objects from the reconstruction to obtain a 3D floor plan of a building, for example. However, segmentation can be also used to place or replace objects in the scene. Furniture in the real scene can be replaced by a new piece of furniture to evaluate how it fits with the existing interior decoration. To date, relatively little attention has been given to semantic understanding in the context of AR [CFT17; DSM^{+}17]. Progress in meaningful semantic scene understanding and segmentation has been made mainly in other fields of research, including virtual reality and robotics.

This thesis has focused on various computer vision problems. It has shown that tracking, reconstruction, and occlusion handling can be more successfully addressed to achieve greater practical value in AR. AR researchers need to investigate these types of topics more thoroughly to move step closer to the goal of perfectly co-existing worlds.

List of Tables

2.1 Root mean square (RMS) error in a synthetic generated sequences. 38

3.1 Runtime performance measurements for different test scenes. 80

3.2 Runtime performance measurements for the local plane estimation at various subvolumes. 81

4.1 Performance measurements for different numbers of unknown pixels. 112

4.2 Performance measurements for the propagation of known values. 113

4.3 Comparison of runtime performance for several state-of-the-art algorithms. 120

List of Figures

2.1 Classes of edges. 12

2.2 Problem of silhouette edges. 16

2.3 Schematic overview of the proposed method. 25

2.4 Schematic visualization of different keyposes. 26

2.5 Schematic pipeline of line mapping and refinement. 32

2.6 Effectiveness of dividing by length. 35

2.7 Tracking results of a synthetic generated scene. 36

2.8 6 DOF camera pose plots of the chopping board in a synthetic generated image sequence. 37

2.9 Tracking results of a real scene. 39

3.1 An example of using a phantom model for shadowing. 45

3.2 Examples of incomplete 3D surface reconstructions. 46

3.3 Example of a TSDF volume. 48

3.4 Schematic visualization of the S_k function in a 2D voxel grid representation for three consecutive observations. 49

3.5 Overview of propagation, hole filling and de-noising pipeline. 61

3.6 Two examples of local plane propagation 67

3.7 Estimation of partially consistent plane by merging local planes. 69

3.8 An example for collision detection. 74

3.9 Surface propagation without plane priors. 76

3.10 Propagation results without plane priors. 77

3.11 Comparison between results created with and without plane priors. 78

3.12 Surface Propagation using Plane Priors for three scenes. 79

3.13 Effect of refinement and de-noising through partially consistent planes. . . 79

4.1 Occlusion handling in AR. 86

4.2 Example of model-based occlusion handling and the registration problem. . 90

4.3 Occlusion handling with raw depth data. 93

4.4 State-of-the-art approaches vs. occlusion matting. 95

4.5 A natural image matting example. 96

4.6 Overview of occlusion matting pipeline. 103

4.7 Adaptive trimap generation. 105

4.8 Schematic illustration of the labeling procedure. 106

4.9 Propagation of foreground and background. 109

4.10 Comparison of trimaps and their effects on the alpha matte. 114

4.11 Effect of the objective function: two examples. 115

4.12 Overall comparison with state-of-the-art methods. 116

4.13 Comparison with edge snapping-based method. 117

4.14 Temporal comparison. 119

4.15 Comparison of occlusion matting results for different depth maps: two examples. 123

Own Publications

[HDD$^+$15] A. K. Hebborn, M. Dilberovic, A. Derstroff, A. Franke, N. Höhner, P. Krechel, L. Prinz, A. Szirmai, F. Weigend, and S. Müller. OscARsWelt: A Collaborative Augmented Reality Game. In *Proceedings of the Second International Conference on Augmented and Virtual Reality (AVR)*, pages 135–150, Lecce, Italy, August 2015.

[HEM15] A. K. Hebborn, M. Erdt, and S. Müller. Robust Model Based Tracking Using Edge Mapping and Refinement. In *Proceedings of the Second International Conference on Augmented and Virtual Reality (AVR)*, pages 109–124, Lecce, Italy, August 2015.

[HHM14] J. Hunz, A. K. Hebborn, and S. Müller. A GPU based Real-Time Line Detector using a Cascaded 2D Line Space. In *Proceedings of the International Conference Computer Graphics, Multimedia and Image Processing (CGMIP)*, pages 56–63, Kuala Lumpur, Malaysia, November 2014.

[HHM16] A. K. Hebborn, N. Höhner, and S. Müller. Augmenting Surface Reconstructions. In *Proceedings of the 2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR Adjunct)*, pages 38–42, Merida, Yucatan, Mexico, September 2016.

[HHM17] A. K. Hebborn, N. Höhner, and S. Müller. Occlusion Matting: Realistic Occlusion Handling for Augmented Reality Applications. In *Proceedings of the*

2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pages 62–71, Nantes, France, October 2017.

Bibliography

[Azu97] R. T. Azuma. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[BCL15] M. Billinghurst, A. Clark, and G. Lee. A Survey of Augmented Reality. *Foundations and Trends in Human-Computer Interaction*, 8(2-3):73–272, March 2015.

[BDV00a] A. Berman, A. Dadourian, and P. Vlahos. Comprehensive method for removing from an image the background surrounding a selected subject. *U.S. Patent 6,134,345*, 2000.

[BDV00b] A. Berman, A. Dadourian, and P. Vlahos. Method for removing from an image the background surrounding a selected object. *U.S. Patent 6,134,346*, 2000.

[BEL$^+$11] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter. The 3D Hough Transform for Plane Detection in Point Clouds: A Review and a New Accumulator Design. *3D Research*, 2(2):32:1–32:13, 2011.

[Ber97] M. O. Berger. Resolving Occlusion in Augmented Reality: A Contour Based Approach without 3D Reconstruction. In *Proceedings of 1997 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 91–96, San Juan, Puerto Rico, USA, June 1997.

[Ble] Blender. URL `http://www.blender.org/`. Accessed: October 8th, 2014.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of Clothing with Folds and Wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pages 28–36, San Diego, CA, USA, July 2003.

[BPS05] G. Bleser, Y. Pastarmov, and D. Stricker. Real-time 3D Camera Tracking for Industrial Augmented Reality Applications. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 47–54, Plzen, Czech Republic, January 2005.

[BR16] L. Bose and A. Richards. Fast Depth Edge Detection and Edge Based RGB-D SLAM. In *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1323–1330, Stockholm, Sweden, May 2016.

[BRF01] D. Bandyopadhyay, R. Raskar, and H. Fuchs. Dynamic Shader Lamps: Painting on Movable Objects. In *Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR)*, pages 207–216, New York, NY, USA, October 2001.

[BRW95] D. Breen, E. Rose, and R. T. Whitaker. Interactive Occlusion and Collision of Real and Virtual Objects in Augmented Reality. Technical Report ECRC-95-02, ECRC, Munich, Germany, 1995.

[BS12] A. Breitenmoser and R. Siegwart. Surface Reconstruction and Path Planning for Industrial Inspection with a Climbing Robot. In *Proceedings of the 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, pages 22–27, Zurich, Switzerland, September 2012.

[Bur81] P. J. Burt. Fast Filter Transform for Image Processing. *Computer Graphics and Image Processing*, 16(1):20–51, May 1981.

[Bus03] S. R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, New York, NY, USA, 2003. ISBN: 0521821037.

[BWR+96] D. E. Breen, R. T. Whitaker, E. Rose, and M. Tuceryan. Interactive Occlusion and Automatic Object Placement for Augmented Reality. *Computer Graphics Forum*, 15(3):11–22, August 1996.

[BWS06] G. Bleser, H. Wuest, and D. Stricker. Online camera pose estimation in partially known and dynamic scenes. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 56–65, Santa Barbara, CA, USA, October 2006.

[CBC+01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 67–76, Los Angeles, California, USA, August 2001.

[CC10] C. Choi and H. I. Christensen. Real-time 3D Model-based Tracking Using Edge and Keypoint Features for Robotic Manipulation. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4048–4055, Anchorage, AK, USA, May 2010.

[CC78] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[CFT17] L. Chen, K. Francis, and W. Tang. Semantic Augmented Reality Environment with Material-Aware Physical Interactions. In *Proceedings of the 16th IEEE International Symposium on Mixed and Augmented Reality (ISMAR Adjunct)*, pages 135–136, Nantes, France, October 2017.

[CGB+09] F. I. Cosco, C. Garre, F. Bruno, M. Muzzupappa, and M. A. Otaduy. Augmented Touch without Visual Obtrusion. In *Proceedings of the 8th IEEE In-*

ternational Symposium on Mixed and Augmented Reality (ISMAR), pages 99–102, Orlando, FL, USA, October 2009.

[CUD] CUDA. URL `https://www.geforce.com/hardware/technology/cuda`. Accessed: February 2th, 2016.

[CYA+11] J. H. Cho, T. Yamasaki, K. Aizawa, and K. H. Lee. Depth video camera based temporal alpha matting for natural 3D scene generation. In *Proceedings of the 3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, Antalya, Turkey, May 2011.

[DC02] T. Drummond and R. Cipolla. Real-time Visual Tracking of Complex Structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, July 2002.

[DCY+16] C. Du, Y. Chen, M. Ye, and L. Ren. Edge Snapping-Based Depth Enhancement for Dynamic Occlusion Handling in Augmented Reality. In *Proceedings of the 15th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 54–62, Merida, Yucatan, Mexico, September 2016.

[DFK13] S. Dong, C. Feng, and V. R. Kamat. Real-Time Occlusion Handling for Dynamic Augmented Reality Using Geometric Sensing and Graphical Shading. *Journal of Computing in Civil Engineering*, 27(6):607–621, November 2013.

[DG10] J.-E. Deschaud and F. Goulette. A Fast and Accurate Plane Detection Algorithm for Large Noisy Point Clouds Using Filtered Normals and Voxel Growing. In *Proceedings of the 5th International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, Paris, France, May 2010.

[DH06] S. DiVerdi and T. Höllerer. Image-space Correction of AR Registration Errors Using Graphics Hardware. In *Proceedings of the IEEE Virtual Reality Conference (VR)*, pages 241–244, Alexandria, VA, USA, March 2006.

[DMG+02] J. Davis, S. Marschner, M. Garr, and M. Levoy. Filing Holes in Complex Surfaces Using Volumetric Diffusion. In *Proceedings of the first International Symposium on 3D Data Processing Visualization and Transmission*, pages 428–441, Padova, Italy, June 2002.

[DMM03] A. J. Davison, W. W. Mayol, and D. W. Murray. Real-Time Workspace Localisation and Mapping for Wearable Robot. In *Proceedings of the 2th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 315–316, Tokyo, Japan, October 2003.

[DSM+17] M. Dzitsiuk, J. Sturm, R. Maier, L. Ma, and D. Cremers. De-noising, Stabilizing and Completing 3D Reconstructions On-the-go using Plane Priors. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3976–3983, Singapore, 2017.

[ED06] E. Eade and T. Drummond. Edge Landmarks in Monocular SLAM. In *Proceedings of the British Machine Vision Conference 2006 (BMVC)*, pages 7–16, Edinburgh, UK, September 2006.

[ED14] A. Eklund and P. Dufort. Non-separable 2D, 3D, and 4D Filtering with CUDA. In *GPU Pro 5*, pages 469–492. CRC Press, 2014.

[Erl04] K. Erleben. *Stable, Robust, and Versatile Multibody Dynamics Animation.* PhD thesis, University of Copenhagen, Deptartment of Computer Science, 2004.

[FB81] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[FDF+90] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practic (2Nd Ed.)* Addison-Wesley Professional, 1990.

[FDG$^+$06] Y. Feng, W. Du, X. Guan, F. Gao, and Y. Chen. Realization of Multilayer Occlusion between Real and Virtual Scenes in Augmented Reality. In *Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 1–5, Nanjing, China, May 2006.

[FHF$^+$99] A. Fuhrmann, G. Hesina, F. Faure, and M. Gervautz. Occlusion in collaborative augmented environments. *Computers & Graphics*, 23(6):809–819, 1999.

[FHS07] J. Fischer, B. Huhle, and A. Schilling. Using Time-of-flight Range Data for Occlusion Handling in Augmented Reality. In *Proceedings of the 13th Eurographics Conference on Virtual Environments (EGVE)*, pages 109–116, Weimar, Germany, July 2007.

[FL01] S. Fisher and M. C. Lin. Deformed Distance Fields for Simulation of Non-penetrating Flexible Bodies. In *Proceedings of the Eurographic Workshop on Computer Animation and Simulation*, pages 99–111, Manchester, UK, September 2001.

[FRB03] J. Fischer, H. Regenbrecht, and G. Baratoff. Detecting Dynamic Occlusion in Front of Static Backgrounds for AR Scenes. In *Proceedings of the Workshop on Virtual Environments 2003 (EGVE)*, pages 153–161, Zurich, Switzerland, May 2003.

[FSG03] A. Fuhrmann, G. Sobottka, and C. Groß. Distance Fields for Rapid Collision Detection in Physically Based Modeling. In *Proceedings of the International Conference on Computer Graphics and Vision (Graphicon)*, pages 58–65, Moscow, Russia, September 2003.

[FTK14] C. Feng, Y. Taguchi, and V. R. Kamat. Fast Plane Extraction in Organized Point Clouds Using Agglomerative Hierarchical Clustering. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6218–6225, Hong Kong, China, May 2014.

[GBF03] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex Rigid Bodies with Stacking. In *Proceedings of ACM SIGGRAPH 2003 Papers (SIGGRAPH)*, pages 871–878, San Diego, CA, USA, July 2003.

[GHH01] S. Gibson, T. Howard, and R. Hubbold. Flexible Image-Based Photometric Reconstruction using Virtual Light Sources. *Computer Graphics Forum*, 20(3):203–214, September 2001.

[GJM$^+$10] R. Grompone von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall. LSD: A Fast Line Segment Detector with a False Detection Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010.

[GM06] A. P. Gee and W. Mayol-Cuevas. Real-Time Model-Based SLAM Using Line Segments. In *Proceedings of the International Symposium on Visual Computing (ISVC)*, pages 354–363, Lake Tahoe, NV, USA, November 2006.

[GMS$^+$17] R. Gomez-Ojeda, F.-A. Moreno, D. Scaramuzza, and J. Gonzalez-Jimenez. Pl-slam: a stereo slam system through the combination of points and line segments. *arXiv preprint arXiv:1705.09479*, 2017.

[GO10] E. S. L. Gastal and M.l M. Oliveira. Shared Sampling for Real-Time Alpha Matting. *Computer Graphics Forum*, 29(2):575–584, May 2010.

[GO12] E. S. L. Gastal and M. M. Oliveira. Adaptive Manifolds for Real-Time High-Dimensional Filtering. *ACM Transactions on Graphics (TOG), Proceedings of ACM SIGGRAPH 2012*, 31(4):33:1–33:13, 2012.

[Har93] C. Harris. Tracking with rigid models. In *Active Vision*, pages 59–73. MIT Press, 1993.

[HB13] D. Holz and S. Behnke. Fast Range Image Segmentation and Smoothing Using Approximate Surface Reconstruction and Region Growing. In *Intelligent Autonomous Systems 12: Volume 2 Proceedings of the 12th International Conference IAS-12*, pages 61–73, Jeju Island, Korea, June 2013.

[HBS$^+$12] R. Hulik, V. Beran, M. Spanel, P. Krsek, and P. Smrz. Fast and Accurate Plane Segmentation in Depth Maps for Indoor Scenes. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1665–1670, Vilamoura, Portugal, October 2012.

[HDH03] M. Haller, S. Drab, and W. Hartmann. A Real-time Shadow Approach for an Augmented Reality Application Using Shadow Volumes. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 56–65, Osaka, Japan, October 2003.

[HHR$^+$12] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke. Real-Time Plane Segmentation Using RGB-D Cameras. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317, 2012.

[Hou62] P. V. C. Hough. Method and means for recognizing complex patterns. *U.S. Patent 3,069,654*, 1962.

[HRR$^+$11] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun. A Global Sampling Method for Alpha Matting. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2049–2056, Colorado Springs, CO, USA, June 2011.

[HS12] K. Hirose and H. Saito. Fast Line Description for Line-based SLAM. In *Proceedings of the British Machine Vision Conference 2012 (BMVC)*, pages 83.1–83.11, Surrey, UK, September 2012.

[HS90] C. Harris and C. Stennett. RAPID - A Video Rate Object Tracker. In *Proceedings of the British Machine Vision Conference 1990 (BMVC)*, pages 15.1–15.6, Oxford, UK, September 1990.

[HST13] K. He, J. Sun, and X. Tang. Guided Image Filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, June 2013.

[IKH+11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 559–568, Santa Barbara, California, USA, October 2011.

[JMA06] N. Joshi, W. Matusik, and S. Avidan. Natural Video Matting Using Camera Arrays. In *Proceedings of ACM SIGGRAPH 2006 Papers (SIGGRAPH)*, pages 779–786, Boston, Massachusetts, August 2006.

[JVC+16] J. Johnson, E. S. Varnousfaderani, H. Cholakkal, and D. Rajan. Sparse Coding for Alpha Matting. *IEEE Transactions on Image Processing*, 25(7):3032–3043, 2016.

[KA08] A. Koschan and M. A. Abidi. *Digital Color Image Processing*. Wiley-Interscience, 2008.

[KBS16] H. Kolivand, M. Billinghurst, and M. S. Sunar. Livephantom: retrieving virtual world light data to real environments. *PLOS ONE*, 11(12):1–19, December 2016.

[KCL+07] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele. Joint Bilateral Upsampling. In *Proceedings of ACM SIGGRAPH 2007 Papers (SIGGRAPH)*, San Diego, California, USA, August 2007.

[KD04] G. Klein and T. Drummond. Sensor Fusion and Occlusion Refinement for Tablet-Based AR. In *Proceedings of the 3th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 38–47, Arlington, VA, USA, November 2004.

[KDN93] D. Koller, K. Daniilidis, and H.-H. Nagel. Model-based Object Tracking in Monocular Image Sequences of Road Traffic Scenes. *International Journal of Computer Vision*, 10(3):257–281, June 1993.

[KDS$^+$15] M. Klingensmith, I. Dryanovski, S. S. Srinivasa, and J. Xiao. Chisel: real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015.

[Kli00] G. Klinker. Augmented reality: a problem in need of many computer vision-based solutions. In *Confluence of Computer Vision and Computer Graphics*, pages 267–284. Springer Netherlands, 2000.

[KM07] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234, Nara, Japan, November 2007.

[KM08] G. Klein and D. Murray. Improving the Agility of Keyframe-Based SLAM. In *Proceedings of the 10th European Conference on Computer Vision (ECCV)*, pages 802–815, Marseille, France, October 2008.

[KMB$^+$14] A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, and C. Rother. 6-dof model based tracking via object coordinate regression. In *Proceedings of the 12th Asian Conference on Computer Vision (ACCV)*, Singapore, November 2014.

[KSF10] E. Kruijff, J. E. Swan, and S. Feiner. Perceptual Issues in Augmented Reality Revisited. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 3–12, Seoul, South Korea, October 2010.

[KSW$^+$11] D. Kalkofen, C. Sandor, S. White, and D. Schmalstieg. *Visualization Techniques for Augmented Reality.* In *Handbook of Augmented Reality.* Springer New York, 2011, pages 65–98.

[LAG13] J. A. Leal-Meléndrez, L. Altamirano-Robles, and J. A. Gonzalez. Occlusion Handling in Video-Based Augmented Reality Using the Kinect Sensor for

Indoor Registration. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (CIARP)*, pages 447–454, Havana, Cuba, November 2013.

[LB00] V. Lepetit and M.-O. Berger. A Semi-Automatic Method for Resolving Occlusion in Augmented Reality. In *Proceedings of 2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 225–230, Hilton Head Island, SC, USA, June 2000.

[LC87] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 163–169, Anaheim, California, USA, August 1987.

[LCK16] S. Liu, C. Chen, and N. Kehtarnavaz. A Computationally Efficient Denoising and Hole-Filling Method for Depth Image Enhancement. *Real-Time Image and Video Processing 2016*, 9897, 2016.

[LF05] V. Lepetit and P. Fua. Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, January 2005.

[LL12] T. Lu and S. Li. Image Matting with Color and Depth Information. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, pages 3787–3790, Tsukuba, Japan, November 2012.

[LLL+12] T.-k. Lee, S. Lim, S. Lee, S. An, and S.-y. Oh. Indoor Mapping Using Planes Extracted from Noisy RGB-D Sensors. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1727–1733, Vilamoura, Portugal, October 2012.

[LLW08] A. Levin, D. Lischinski, and Y. Weiss. A Closed-Form Solution to Natural Image Matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242, February 2008.

[LMC17] O. Lézoray, C. Meurie, and E. Celebi. Superpixels for Image Processing and Computer Vision. *Journal of Electronic Imaging, Society of Photo-optical Instrumentation Engineers*, October 2017.

[Low92] D. G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, 8(2):113–122, August 1992.

[LSF$^+$10] J. P. Lima, F. Simões, L. Figueiredo, and J. Kelner. Model Based Markerless 3D Tracking applied to Augmented Reality. *SBC Journal on 3D Interactive Systems*, 1:2–15, 2010.

[LVT$^+$03] V. Lepetit, L. Vacchetti, D. Thalmann, and P. Fua. Fully Automated and Stable Registration for Augmented Reality Applications. In *Proceedings of the 2th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 93–102, Tokyo, Japan, October 2003.

[LZL$^+$13] J. H. Lee, G. Zhang, J. Lim, and I. H. Suh. Place Recognition using Straight Lines for Vision-based SLAM. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3799–3806, Karlsruhe, Germany, May 2013.

[Mat] Matlab. URL `https://de.mathworks.com/`. Accessed: March 10th, 2018.

[MKS$^+$11] T. Mörwald, M. Kopicki, R. Stolkin, J. Wyatt, S. Zurek, M. Zillich, and M. Vincze. Predicting the Unobservable Visual 3D Tracking with a Probabilistic Motion Model. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1849–1855, Shanghai, China, May 2011.

[MNZ$^+$15] S. Magnenat, D. T. Ngo, F. Zund, M. Ryffel, G. Noris, G. Rothlin, A. Marra, M. Nitti, P. Fua, M. Gross, and R. W. Sumner. Live Texturing of Augmented Reality Characters from Colored Drawings. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1201–1210, November 2015.

[MW88] M. Moore and J. Wilhelms. Collision Detection and Response for Computer Animation. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 289–298, Atlanta, Georgia, USA, August 1988.

[NDI+11] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. KinectFusion: Real-time Dense Surface Mapping and Tracking. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, Basel, Switzerland, October 2011.

[NFS15] R. A. Newcombe, D. Fox, and S. M. Seitz. DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, Boston, MA, USA, June 2015.

[NLD11] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV)*, pages 2320–2327, Barcelona, Spain, November 2011.

[NPD07] J. Neubert, J. Pretlove, and T. Drummond. Semi-Autonomous Generation of Appearance-based Edge Models from Image Sequences. In *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 79–89, Nara, Japan, November 2007.

[NS09] Z. Noh and M. S. Sunar. A Review of Shadow Techniques in Augmented Reality. In *Proceedings of the Second International Conference on Machine Vision (ICMV)*, pages 320–324, Dubai, United Arab Emirates, December 2009.

[NSS+16] Y. Nakayama, H. Saito, M. Shimizu, and N. Yamaguchi. Marker-Less Augmented Reality Framework Using On-Site 3D Line-Segment-based Model

Generation. *Journal of Imaging Science and Technology*, 60(2):20401-1–20401-24, March 2016.

[OAB+85] J. M. Ogden, E. H. Adelson, J R. Bergen, and P. J. Burt. Pyramid-Based computer graphics. *RCA Engineer*, 30(5):4–15, 1985.

[OSW+11] B. Oehler, J. Stueckler, J. Welle, D. Schulz, and S. Behnke. Efficient Multi-resolution Plane Segmentation of 3D Point Clouds. In *Proceedings of the 4th International Conference on Intelligent Robotics and Applications (ICIRA)*, pages 145–156, Aachen, Germany, December 2011.

[Pan11] G. Panin. *Model-based Visual Tracking: The OpenTL Framework*. Wiley, 2011. ISBN: 9781118002131.

[PD84] T. Porter and T. Duff. Compositing digital images. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 253–259, Minneapolis, Minnesota, USA, July 1984.

[PL07] J. Platonov and M. Langer. Automatic Contour Model Creation out of Polygonal CAD Models for Markerless Augmented Reality. In *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 75–78, Nara, Japan, November 2007.

[Pod07] V. Podlozhnyuk. Image Convolution with CUDA. *NVIDIA Corporation white paper*:4–21, June 2007.

[Pro] NVIDIA Visual Profiler. URL `https://developer.nvidia.com/nvidia-visual-profiler`. Accessed: February 2th, 2016.

[PVB+08] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak. Fast Plane Detection and Polygonalization in Noisy 3D Range Images. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3378–3383, Nice, France, September 2008.

[RD06] G. Reitmayr and T. W. Drummond. Going out: Robust Model-based Tracking for Outdoor Augmented Reality. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 109–118, Santa Barbara, CA, USA, October 2006.

[RN17] John C. Russ and F. Brent Neal. *The Image Processing Handbook, Seventh Edition*. CRC Press, Inc., 7th edition, 2017.

[RPS+18] J. Rambach, A. Pagani, M. Schneider, O. Artemenko, and D. Stricker. 6DoF Object Tracking based on 3D Scans for Augmented Reality Remote Live Support. *Computers*, 7(1), 2018.

[RRG08] C. Rhemann, C. Rother, and M. Gelautz. Improving Color Modeling for Alpha Matting. In *Proceedings of the British Machine Vision Conference 2008 (BMVC)*, Leeds, UK, September 2008.

[RRW+09] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A Perceptually Motivated Online Benchmark for Image Matting. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1826–1833, Miami, FL, USA, June 2009.

[RT00] M. A. Ruzon and C. Tomasi. Alpha Estimation in Natural Images. In *Proceedings of 2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18–25, Hilton Head Island, SC, USA, June 2000.

[RV12] H. Roth and M. Vona. Moving Volume KinectFusion. In *Proceedings of the British Machine Vision Conference 2012 (BMVC)*, pages 1–11, Surrey, UK, 2012.

[SDKa] Microsoft HoloLens SDK. Spatial mapping. URL `https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping`. Accessed: February 19th, 2018.

[SDKb] Vuforia AR SDK. Smart Terrain. URL `https://library.vuforia.com/articles/Training/Getting-Started-with-Smart-Terrain`. Accessed: February 19th, 2018.

[SH15] D. Schmalstieg and T. Höllerer. *Augmented Reality: Principles and Practice*. Addison-Wesley usability and HCI series. Addison-Wesley, 2015. ISBN: 9780321883575.

[SJT$^+$] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson Matting. In *Proceedings of ACM SIGGRAPH 2004 Papers (SIGGRAPH)*.

[SMG05] R. Subbarao, P. Meer, and Y. Gene. A Balanced Approach to 3D Tracking from Image Streams. In *Proceedings of the 4th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 70–78, Vienna, Austria, October 2005.

[SNV02] J. Schmidt, H. Niemann, and S. Vogt. Dense Disparity Maps in Real-Time with an Application to Augmented Reality. In *Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV)*, pages 225–230, Orlando, FL, USA, December 2002.

[SSG$^+$14] N. Silberman, L. Shapira, R. Gal, and P. Kohli. A Contour Completion Model for Augmenting Surface Reconstructions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 488–503, Zurich, Switzerland, September 2014.

[SSI03] I. Sato, Y. Sato, and K. Ikeuchi. Illumination from Shadows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(3):290–300, March 2003.

[SYT$^+$12] Y. Souma, H. Yamachi, Y. Tsujimura, and Y. Kambayashi. Interaction in Augmented Reality by Means of Z-buffer Based Collision Detection. In *Proceedings of the Fifth International Conference on Advances in Computer-Human Interactions (ACHI)*, pages 315–318, Valencia, Spain, January 2012.

[TGW10a] Y. Tian, T. Guan, and C. Wang. An automatic occlusion handling method in augmented reality. *Sensor Review*, 30(3):210–218, 2010.

[TGW10b] Y. Tian, T. Guan, and C. Wang. Real-Time Occlusion Handling in Augmented Reality Based on an Object Tracking Approach. *Sensors*, 10(4):2885–2900, 2010.

[TJR$^+$13] Y. Taguchi, Y. D. Jian, S. Ramalingam, and C. Feng. Point-Plane SLAM for Hand-Held 3D Sensors. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5182–5189, Karlsruhe, Germany, May 2013.

[TKH$^+$05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision Detection for Deformable Objects. *Computer Graphics Forum*, 24(1):61–81, March 2005.

[TLX$^+$15] Y. Tian, Y. Long, D. Xia, H. Yao, and J. Zhang. Handling Occlusions in Augmented Reality Based on 3D Reconstruction Method. *Neurocomputing*, 156(C):96–104, May 2015.

[TM98] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *Proceedings of the Sixth International Conference on Computer Vision (ICCV)*, pages 839–846, Bombay, India, January 1998.

[TRC12] A. J. B. Trevor, J. G. Rogers, and H. I. Christensen. Planar Surface SLAM with 3D and 2D Sensors. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3041–3048, Saint Paul, MN, USA, May 2012.

[War] 3D Warehouse. URL `https://3dwarehouse.sketchup.com/`. Accessed: October 8th, 2014.

[WC07a] J. Wang and M. F. Cohen. Image and Video Matting: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 3(2):97–175, January 2007.

[WC07b] J. Wang and M. F. Cohen. Optimized Color Sampling for Robust Matting. In *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Minneapolis, MN, USA, June 2007.

[WFQ$^+$07] O. Wang, J. Finger, Y. Qingxiong, J. Davis, and Y. Ruigang. Automatic Natural Video Matting with Depth. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG)*, pages 469–472, Maui, HI, USA, November 2007.

[WGS03] J. Weingarten, G. Gruener, and R. Siegwart. A Fast and Robust 3D Feature Extraction Algorithm for Structured Environment Reconstruction. In *Proceedings of the 11th International Conference on Advanced Robotics*, Coimbra, Portugal, June 2003.

[WGZ$^+$12] L. Wang, M. Gong, C. Zhang, R. Yang, C. Zhang, and Y.-H.Yang. Automatic Real-Time Video Matting Using Time-of-Flight Camera and Multichannel Poisson Equations. *International Journal of Computer Vision*, 97(1):104–121, March 2012.

[WHN17] Y. Wu, X. He, and T. Q. Nguyen. Moving Object Detection With a Freely Moving Camera via Background Motion Subtraction. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(2):236–248, February 2017.

[WJK$^+$13] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust Real-Time Visual Odometry for Dense RGB-D Mapping. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5724–5731, Karlsruhe, Germany, May 2013.

[WLS$^+$15] T. Whelan, S. Leutenegger, R. F. Salas-moreno, B. Glocker, and A. Davison. ElasticFusion : Dense SLAM Without A Pose Graph. In *Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015.

[WLW09] Z. Wang, H. Liu, and F. Wu. HLD: A robust descriptor for line matching. In *Proceedings of the11th IEEE International Conference on Computer-Aided*

Design and Computer Graphics (CAD/Graphics), pages 128–133, Huangshan, China, August 2009.

[WS07] H. Wuest and D. Stricker. Tracking of industrial objects by using CAD models. *Journal of Virtual Reality and Broadcasting*, 4(1):155–164, 2007.

[WVS05] H. Wuest, F. Vial, and D. Stricker. Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality. In *Proceedings of the 4th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 62–69, Vienna, Austria, October 2005.

[WWH09] Z. Wang, F. Wu, and Z. Hu. Msld: a robust descriptor for line matching. *Pattern Recognition*, 42(5):941–953, May 2009.

[XDW+14] J. Bing Xiahou, X. Na Deng, Q. Qian Wei, and X. Wei Liu. Real-Time Video Matting Algorithm Based on Chroma Key. *Advanced Materials Research*, 926-930:3161–3164, May 2014.

[XPC+17] N. Xu, B. Price, S. Cohen, and T. Huang. Deep image matting. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 311–320, Honolulu, HI, USA, July 2017.

[XZW12] K. Xu, J. Zhou, and Z. Wang. A Method of Hole-filling for the Depth Map Generated by Kinect with Moving Objects Detection. In *Proceedings of the IEEE international Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–5, Seoul, South Korea, June 2012.

[YKP12] N.-E. Yang, Y.-G. Kim, and R.-H. Park. Depth Hole Filling Using the Depth Distribution of Neighboring Regions of Depth Holes in the Kinect Sensor. In *Proceedings of the 2012 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC)*, pages 658–661, Hong Kong, China, August 2012.

[YZ14] L. Yin and J. Zhao. Real-Time Automatic Chroma-Key Matting Using Perceptual Analysis and Prediction. In *Proceedings of the IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4, Toronto, ON, Canada, May 2014.

[ZLY$^+$09] J. Zhu, M. Liao, R. Yang, and Z. Pan. Joint Depth and Alpha Matte Optimization via Fusion of Stereo and Time-of-Flight Sensor. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 453–460, Miami, FL, USA, June 2009.

[ZSL$^+$15] Q. Zhu, L. Shao, X. Li, and L. Wang. Targeting Accurate Object Extraction From an Image: A Comprehensive Study of Natural Image Matting. *IEEE Transactions on Neural Networks and Learning Systems*, 26(2):185–207, February 2015.

[ZSW14] F. Zheng, D. Schmalstieg, and G. Welch. Pixel-wise Closed-Loop Registration in Video-Based Augmented Reality. In *Proceedings of the 13th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 135–143, Munich, Germany, September 2014.

Curriculum Vitae

Name	**Anna Katharina Hebborn**
Date of birth	June 28, 1985
Place of birth	Bergisch Gladbach
Nationality	German

2005	**General qualification for university entrance** Gymnasium Herkenrath, Bergisch Gladbach
2005–2008	**Professional training as Media Designer** Book and offset printing Häuser KG, Cologne Digital and print media with focus on media technology
2008–2011	**Bachelor of Science, Computational Visualistics** University of Koblenz-Landau, Koblenz Bachelor thesis at research group Computer Graphics: *Acceptance of Tangible User Interfaces in AR Gaming*
2011–2013	**Master of Science, Computational Visualistics** University of Koblenz-Landau, Koblenz Master thesis at metaio GmbH, Munich: *Automatic Generation of 3D Edge Models from Polygon Models for Model-based Tracking*
since 2013	**Ph.D. Student and Research Associate** University of Koblenz-Landau, Koblenz Institute for Computational Visualistics Research group Computer Graphics