Real-time Simulation of Time-of-Flight Sensors and Accumulation of Range Camera Data

Echtzeit Simulation von Time-of-Flight Sensoren und Akkumulation von Tiefendaten

bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen

zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Maik Keller

17. Juni 2015

1. Gutachter:	Prof. Dr. Andreas Kolb, Universität Siegen
2. Gutachter:	Prof. Dr. Reinhard Koch, Christian-Albrechts-Universität zu Kiel
Mündliche Prüfung:	13. November 2015

Printed on non-aging wood- and acid-free paper.
Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Acknowledgments I would like to thank all those who contributed to the success of this work. First and foremost, I would like to thank my supervisor Prof. Dr. Andreas Kolb for his great support and the opportunity he gave me to work in his research group. He gave me the freedom to explore my own ideas and he was always open for discussion and supervision of my work. I thank Prof. Dr. Reinhard Koch for agreeing to act as the second referee.

I gratefully acknowledge the financial support of the Federal Ministry of Education and Research within the Lynkeus project.

Sincere thanks are given to all my colleagues from the Computer Graphics Group in Siegen, I enjoyed working with them very much. Special thanks go to Prof. Dr. Christof Rezk-Salama who introduced me to the world of Computer Graphics. I feel deeply grateful for co-working with my friend Dr. Jens Orthmann. He is a great person.

I thank all co-authors of my publications. Especially, I would like to thank Dr. Shahram Izadi from Microsoft Research in Cambridge, and Prof. Dr. Tim Weyrich from University College London. I also would like to thank Dr. Martin Lambers, Dr. Nicolas Cuntz, and Dr. Marvin Lindner for valuable discussions and their scientific contributions to my papers. Particularly, I would like to mention Damien Lefloch and Thomas Hoegg for their inspiring discussions and their work in the field of Time-of-Flight simulation and 3D reconstruction.

I thank Andrea Anders and Willi Gräfrath from the Computer Graphics Group for the handling of all administrative matters.

I am very much obliged to my supervisors and colleagues from pmd technologies: Dr. Bernd Buxbaum, Dr. Robert Lange, and Jochen Penne for their support and for giving me the freedom to concentrate on finishing my research. Furthermore, I would like to thank Dr. Lutz Heyne, Stefan Hoberg, Martin Profittlich, Alexander Strauch, Michelle Schneider, and Daniel Queisser for valuable discussions and their help in the context of this thesis.

I would also like to thank Maria Knaier for proof-reading this thesis.

Finally, and above all, I would like to express my deep gratitude to my family: my parents Renate and Jürgen Keller, my sisters Tina and Daniela, and my parents-in-law Brigitte and Reiner Ohrndorf. They always believed in me, which I appreciate very much.

Last, but not least, I would like to thank my wife Kathrin, the love of my life, and our sweet children Momo and Kasimir. My wife sacrificed a lot in order to simplify things for me. Her patience and her support enabled me to finish this work, for which I am very grateful and which is beyond any words.

– Maik Keller
June 2015

Abstract

The present thesis addresses the topic of 3D range imaging in a twofold way: new methods in the field of range data simulation as well as the accumulation of range images into a consistent data representation, namely 3D environment models, for high quality 3D object reconstruction are proposed.

Within the last years inexpensive Time-of-Flight (ToF) range imaging devices have become an alternative to traditional depth measuring approaches. ToF cameras measure full-range distance information by actively illuminating a scene and measuring the time until back-scattered light is detected. The final distance information is computed from multiple raw images. This thesis proposes a method for simulating the ToF principle in real-time along with the major sensor characteristics. The approach is motivated by physically-based illumination models and applied to the simulation of Photonic Mixing Devices, a specific type of ToF sensors.

Furthermore, this thesis presents new methods of range data accumulation in real-time. While the hierarchical volumetric approach supports merging and subtraction of sub-volumes with arbitrary resolutions, the point-based fusion method accounts for spatial limitations of previous approaches and addresses high quality 3D reconstructions at extended scales. Additionally, dynamically changing scenes are supported which results in advanced camera pose estimation as well as reduced drift errors. The algorithms are evaluated using simulated data as well as real camera data from structured light and ToF devices.

The algorithms presented in this thesis feature an extensive data-parallel implementation on current graphics processing units in order to ensure the online capability of the methods but without restricting the algorithms to hardware-specific features.

Zusammenfassung

Die vorliegende Arbeit befasst sich im Bereich der 3D Tiefen-Bildverarbeitung mit zwei unterschiedlichen Themengebieten: zum einen geht es um neue Ansätze zur Simulation von Tiefenbilddaten, zum anderen geht es um die Akkumulierung von Entfernungsbildern in eine konsistente Datenbasis - auch 3D Umgebungsmodelle genannt - für qualitativ hochwertige 3D Objektrekonstruktionen.

Im Laufe der letzten Jahre haben sich kostengünstige 3D Laufzeitkamerasysteme (Time-of-Flight, ToF) zu einer ernstzunehmenden Alternative zu traditionellen Abstandsmessverfahren etabliert. ToF-Kameras erfassen Entfernungsbilder, indem die Szene aktiv beleuchtet und die Laufzeit des reflektierten Lichts bestimmt wird. Dabei werden die Tiefendaten aus mehreren Rohbildern rekonstruiert. Diese Arbeit stellt einen Ansatz zur Echtzeit-Simulation des ToF-Verfahrens vor, welcher auch die Simulation von relevanten Sensor-Charakteristiken umfasst. Der Ansatz ist physikalisch motiviert und findet seine Anwendung in der Simulation von Photomischdetektoren (Photonic Mixing Devices), welche auf dem Lichtlaufzeitverfahren basieren.

Darüber hinaus werden in der vorliegenden Arbeit neue Verfahren für die Akkumulierung von Entfernungsbildern präsentiert. Der hierarchisch-volumetrische Ansatz unterstützt das Hinzufügen und Entfernen von Teil-Volumen mit unterschiedlicher Auflösung. Im Gegensatz zu bisherigen Ansätzen stellt die punkt-basierte Methode qualitativ hochwertige Rekonstruktion von großen Szenen sicher. Des Weiteren werden dynamische Szenen unterstützt. Dadurch verbessern sich die Abschätzung der Kamerapose sowie das Abdriften während der Datenfusion. Die Anwendung der Algorithmen wird unter Nutzung von Simulationsdaten und realen Kameradaten (basierend sowohl auf dem strukturierten Licht-Ansatz als auch auf dem ToF Verfahren) demonstriert.

Alle präsentierten Algorithmen zeichnen sich durch eine umfangreiche daten-parallele Implementierung auf derzeit verfügbaren Grafikkarten aus, um die Echtzeitverarbeitung sicherzustellen. Die Algorithmen werden dabei nicht auf hardware-spezifische Aspekte eingeschränkt.

Contents

Chapter 1

Introduction

Nowadays, a wide range of applications rely not only on two dimensional image data but utilize additional sources such as depth information in order to increase accuracy and to solve ill-posed problems. Many applications in the field of gaming, robotics, automotive, and augmented reality rely on range images which are used for capturing the real world environment in 3D. The ability of rapidly responding to environmental changes requires real-time camera systems for range data acquisition as well as online 3D reconstruction methods.

The emergence of inexpensive consumer depth cameras received much attention by developers in the 3D reconstruction community since classical range imaging devices are rather expensive and inconvenient to use. Microsoft's Kinect camera is a famous example of a widely distributed range imaging system and its successful usage in many application areas. The Kinect has originally been provided along with Microsoft's Xbox for steering games with gestures in order to allow for a novel and immersive gaming experience. Its technology is based on the structured light principle. However, the mass distribution of the Kinect helps also other range imaging systems to be accepted by developers and by the market. Within the last years Time-of-Flight (ToF) range imaging has become an alternative to traditional depth measuring approaches. These cameras have the potential of efficiently generating depth data at the desired quality for a wide range of applications. ToF cameras measure full-range distance information by estimating the elapsed time between the emission and the reception of active light in real-time. Such sensors are inexpensive, compact, and they have a high performance. In contrast to stereoscopic approaches, ToF sensors compute range data by simple calculations which require only a small amount of computational resources. However, current ToF sensors are affected by systematic error sources as well as motion artifacts. Furthermore, their spatial resolution is low compared to other depth sensing technologies.

In this dissertation a new physically-based ToF sensor simulation is proposed which is capable of simulating the major sensor artifacts in real-time. Furthermore, new 3D reconstruction algorithms of environmental modeling and high quality surface rendering are introduced which utilize latest consumer depth cameras. The problems of real-time processing of noisy range images are addressed and solutions are presented for capturing large scale environments.

1.1 Context

Parts of this thesis have been developed within the Lykeus 3D project, an interdisciplinary collaboration of various universities and companies. The project has been funded by the Federal Ministry of Education and Research (BMBF) and its main goal is the creation of 3D real-time camera systems for intelligent environment capturing using Photo-Mixing-Device (PMD) technology which is based on the ToF principle (Lynkeus promotional reference: 16SV2296). In particular, the project focused on the development of hardware standards, interfaces, and modular software components in the field of 3D vision. The project covered the design of a newly developed 3D camera chip, the creation of novel image processing algorithms as well as the depth sensor's application in real world scenarios. In detail, the 3D camera system has been demonstrated in a robot's bin picking application, a driverless transport system, a human-robot interaction application, and finally in the area of terahertz imaging.

In the context of Lynkeus 3D, the Computer Graphics and Multimedia Systems Group at the University of Siegen has been responsible for two main components of the camera system. On the one hand, the project demanded for an application which simulates ToF sensors in real-time. The physically-based simulation of such sensors is an essential building block for hardware design and application development. Therefore, the simulation data must capture the major sensor characteristics. On the other hand, the creation of a 3D model of the environment in real-time has been required. The major purpose of environment modeling is the temporal accumulation of sensor data into a consistent data basis which then is available for subsequent algorithms of the various demonstrators.

After successfully finalizing the work in the Lynkeus project the research has been extended in the area of environment modeling. Ongoing collaborations with the *German Aerospace Center* in Munich, *pmdtechnologies* in Siegen, *Microsoft Research* in Cambridge, and the *University College London* led to various publications as well as fruitful co-operations during the time this thesis has been prepared.

1.2 Overall Goals

In this dissertation the topic of 3D range imaging is investigated in a twofold way: range data simulation as well as the accumulation of range images into a consistent data basis, namely 3D environment models, is discussed. However, from a general point of view similar goals can be pointed out:

High quality results The results of the various approaches are of high quality.

Regarding the ToF sensor simulation the computed output data is as close as possible to the data of real world devices reflecting also technology-based artifacts. In the field of range data accumulation the reconstructed 3D models represent the environment in original scale including small details and an overall minimal drift error.

Real-time capability The real-time application of all methods is important. The ToF sensor simulation is seen as a substitute for a real world device and thus adequate frame-rates need to be maintained. Since the data accumulation methods reflect changes in the environment immediately the processing of the devices' full frame is targeted.

Applicability Parts of this thesis rely on the processing power of GPUs in order to ensure the real-time capability. However, the algorithms are designed and implemented regardless of latest specific GPU features in order to allow for a wider acceptance by the community. Additionally, all approaches are described as complete as possible not assuming any prerequisites which further increases the usability of the methods.

1.3 Contributions

This thesis contributes to two major parts in the field of 3D range imaging: firstly, the real-time simulation of ToF range data is addressed. Secondly, new methods of the accumulation of range images for high quality 3D object reconstruction are proposed. Most parts of this work have been published in several scientific articles [KOKP07, KK09, KCK09, OKK09, FHK*10, KLL*13]. In particular, the contributions of the presented work are:

Real-time simulation of Time-of-Flight sensors This approach proposes a novel method for simulating the ToF principle along with its major artifacts. The approach is physically-based and applied to the simulation of PMD sensors, a specific type of ToF sensors. The simulation incorporates various new elements:

- Development of a theoretical physically-based sensor model for the ToF principle.
- Simulation of the major temporal and spatial ToF artifacts as well as the systematic deviation error.
- Implementation of data-parallel algorithms for real-time simulation.
- Development of a GPU-based generic sensor simulation framework. The framework allows for the integration and adaption of new sensor concepts

as well as for dynamic scene configuration and simulation sequence export.

A standalone version of the simulation framework is available to project partners and therefore has been actively used for the development of subsequent processing algorithms as well as for the generation of ground-truth data. Besides the presented work, the proposed simulation approach contributed to the following publications (mostly for comparison as well as for evaluation purpose): [HLK13a, LHK13, KLL*13, MNK13]. Recently, the presented simulation approach has been taken one step further by Lambers et al. [LHK15] by using physical units throughout the parametrization and simulation process.

Methods of range data accumulation Various methods for the generation of environmental models are proposed. Therefore, the accumulation of single range images is processed in real-time in order to reflect changes in the environment immediately. In this context new strategies are developed:

- Development of a novel hierarchical volumetric GPU-based data structure. The merging and subtraction of (sub-)volumes with nearly arbitrary resolution is supported.
- Development of a new point-based fusion approach for depth data accumulation. No data structure, nor topological assumptions of the points are required for smooth high quality surface reconstructions.
- A fully point-based range image accumulation-pipeline is presented, without converting between multiple representations. Thus, data is merged on point-level for memory efficiency to support scalable dense reconstructions of large scenes.
- The application of a new sensor uncertainty model leads to better denoised reconstruction results.
- Improvement of camera pose estimation by segmenting dynamic objects in the scene, even when the camera is moving, which results in reduced drift errors.
- Proposal of a combined occupancy grid and surface reconstruction method in order to allow for autonomous exploration in unknown areas.

Main objective of all methods has been their real-time application. The algorithms are optimized for the GPU's parallel processing architecture in order to ensure the online capability of range data accumulation and 3D reconstruction: the point-based fusion approach defines new state-of-the-art capabilities for high quality reconstruction of point clouds in real-time. Recently, the basic point-based fusion technique has been extended by other researchers: Whelan et al. [WLSM*15] expand the basic point-based fusion technique by online loop closure whereas Lefloch et al. [LWK15] further improve the reconstruction quality by accumulating anisotropic point-representations. The volumetric data structure has been integrated into a robot's bin picking showcase application in collaboration with the German Aerospace Center (DLR) [FHK*10] which led to best paper nomination in the category *best application paper award* at IROS 2010 conference.

1.4 Outline

The main parts of the research proposed in this dissertation have been published in various publications during the time at the Computer Graphics and Multimedia Systems Group at the University of Siegen. Therefore, the key chapters of this thesis reflect the contributions of the publications. The work is structured in five main chapters:

Chapter 2 gives a general introduction to current range imaging techniques and outlines the ToF principle and its technology-based error effects. Furthermore, an overview about environmental modeling is given. The chapter closes with an overview about current graphics hardware and its relation to real-time processing.

Chapter 3 describes the "Real-time Simulation of Time-Of-Flight Sensors" [KK09] which has been published in the journal Simulation Practice and Theory (Simpat 2009). The theoretical sensor model's integration into "A Simulation Framework for Time-Of-Flight Sensors" [KOKP07], published on the IEEE Symposium on Signals, Circuits and Systems (ISSCS 2007), is also described.

Chapter 4 presents the "Interactive Dynamic Volume Trees on the GPU" [KCK09], published in the Proc. of the Vision, Modeling, and Visualization workshop (VMV 2009) which proposes a temporal accumulation of sensor data into a consistent data basis. The approach has been integrated into "Cooperative Bin Picking with ToF-Camera and Impedance Controlled DLR Light-Weight Robot III" [FHK*10], as presented in the Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2010) in collaboration with the Institute of Robotics and Mechatronics of the German Aerospace Center in Munich.

Chapter 5 outlines the "Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion" [KLL*13], presented in the Proceedings of the Joint 3DIM/3DPVT Conference (3DV 2013). This work has been published in collaboration with Microsoft Research in Cambridge and the University College London. Furthermore, an alternative fusion technique based on moving least squares is described as well as the proposal of an occupancy grid integration. The work utilizes a GPU framework described in "Integrating GPGPU Functionality into Scene Graphs" [OKK09], which is published in the Proc. of the Vision, Modeling, and Visualization workshop (VMV 2009).

Finally, Chapter 6 gives a summary and discusses the contributions of this thesis along with providing an outlook of future research.

Chapter 2

Fundamentals

The following sections give a general introduction to current range imaging techniques. The Time-of-Flight principle is outlined and compared to other depth sensing approaches. Furthermore, the artifacts of the Time-of-Flight data are categorized and described. With regard to the processing of depth sensor generated data this chapter presents an overview of the modeling of environments, i.e. processing approaches for 3D data accumulation. Finally, with respect to real-time data processing, an overview about current graphics hardware is given to complete the fundamental background for this thesis.

2.1 Current Range Imaging Techniques

This work addresses two aspects in the field of range imaging: on the one hand, the topic of real-time simulation of Time-of-Flight (ToF) depth sensors is discussed. On the other hand, approaches of depth data processing are described. This section outlines today's depth imaging techniques with respect to the topic of this thesis to allow an insight into the details of the sensor simulation and the subsequent processing of depth data.

In general, a range image is a 2D matrix consisting of depth information in each entry. Each value of this matrix corresponds to a specific distance to the scene imaged by an optical system. This image is called depth map and it contains the distances to all objects in the scene within the field of view of the acquisition system. The sensor device that is capable of producing such depth maps is often referred to as a range camera. Range cameras operate with a number of different depth measurement techniques which can be classified into triangulation approaches (including passive stereoscopic as well as active approaches such as structured light), and active depth sensing approaches such as ToF concept.

Passive Stereoscopic Approaches Classical triangulation approaches (also known as stereoscopic approaches) compute the depth information of a single pixel by analyzing the projections of two optical systems. Typically, a stereo setup consists of two cameras which are mounted next to each other with a certain distance observing

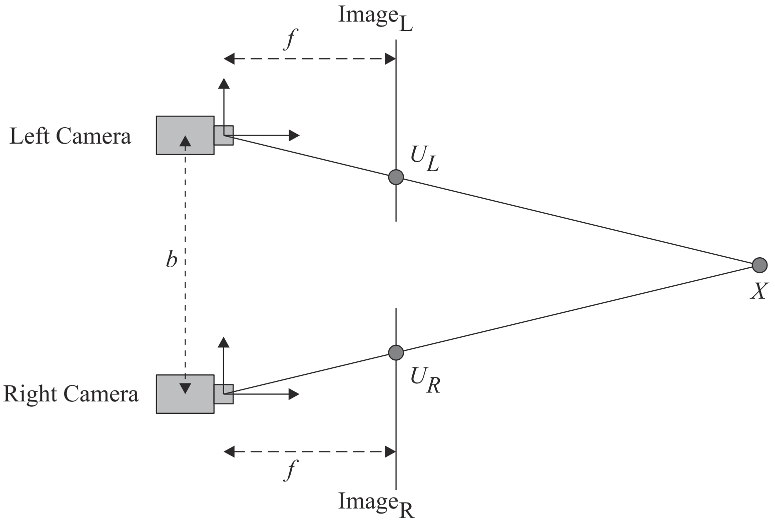


Figure 2.1: Triangulation in passive stereoscopy. The working principle of passive stereoscopy is illustrated by a simplified two-dimensional sketch.

the same scene. This distance defines the baseline b of the system. Knowing the internal and external camera parameters the depth information can be estimated. A triangle is defined by the two lines of sight observing a particular surface point as well as the baseline connecting both lines in a common image plane. The depth information is then computed by taking into account the baseline's length as well as the two angles of the lines of sight. Fig. 2.1 displays a simple example where the depth d of point X can be reconstructed by detecting this point in both images Image_L and Image_R and calculating its disparity $x_p = U_L - U_R$. d is then computed by $d = \frac{f*b}{x_p}$. If the epipolar geometry is known the problem of finding the best correspondence in Image_L and Image_R can be limited to search for a best match along a scan line [BF82].

In general, passive stereoscopic approaches require the identification of correspondences along the acquired images. Finding such matches in real-time demands for powerful computational hardware [SS02]. However, the main challenge is to find correspondences anyway since in sparsely textured areas or homogeneous regions ambiguities cannot be avoided and unique matches cannot be identified. The accuracy of the data strongly depends on the properties of the acquired scene (e.g. contrast and quality of light) as well as on the computational resources. Several approaches have been published which try to overcome these limitations by utilizing various techniques: such as dynamic programming [OK83], relaxation techniques [MP79, PMF85], and prediction methods [AF87]. All in all, stereo vision is a very flexible method to generate depth maps. It can be applied in various environments and since it is a passive system it is not affected by the illumination of direct sunlight.

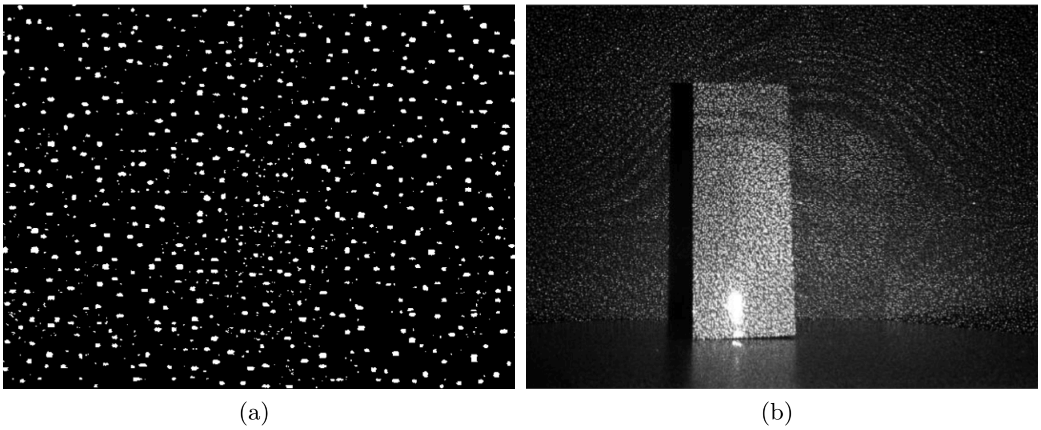


Figure 2.2: Structured light projection pattern. The speckle pattern exemplarily described in the original Primesense patent is shown in 2.2(a). 2.2(b) displays the infrared image of Microsoft Kinect's speckle pattern projected onto a sample scene. Fig. courtesy of [SZ08, KE12].

Structured Light The method of structured light belongs to the group of active stereoscopic approaches. The most challenging task in stereo vision is the identification of correspondences along acquired images and the principle of structured light overcomes this problem by using a projection unit. A pattern is generated by this unit on the imaged surface (see Fig. 2.2). In practice the projection of a single pattern is not enough for a unique detection of correspondences. Thus, time- or color-multiplexed illumination is used. In comparison to passive stereoscopy one of the cameras is substituted by an infrared projection unit.

A famous example for a depth measurement system based on structured light technology is Microsoft's Kinect [Kin] which is shipped for a very low consumer price with the Xbox 360 gaming console. Microsoft licensed this approach of Primesense. Since details about the specific algorithms are not publicly available their description is mostly speculative and based on Primsense's patents. The patents describe the use of a classic computer vision technique called depth-from-focus which is applied besides the stereo vision approach. Depth-from-focus is based on the principle that projections which are more blurry are also further away. The Kinect consists of a special lens with different focal lengths in the lateral domain. Thus, a projected circle then becomes an ellipse whose orientation depends on depth. Finally, depth maps are produced with 640 × 480 pixels at 30 frames per second (fps). The depth resolution decreases with increasing distance, with 1 cm depth resolution at 2 m and 7 cm resolution at 5 m [KE12].

Although Microsoft's Kinect is a successful example of a widely distributed consumer depth camera triangulation techniques in general suffer from a non-neglectable drawback: a large baseline may be required depending on the scene depth and the required accuracy. Thus, stereoscopic approaches may not be usable for the acquisition of large distances. Furthermore, occlusion and shadow effects still remain as significant problems based on different viewing and illumination positions. These effects may result in incomplete depth maps.

Time-of-Flight Within the last years ToF range imaging has become an alternative to traditional depth measuring approaches. The range information is estimated by measuring the time between the emission of a signal and the detection of its backscattering from an object. The distance to an object is then computed as $d = \frac{v*t}{2}$, v being the signal's velocity and t the time measured for the signal to return to the sensor. The signal can be an optical, electromagnetic or acoustic signal. Historically, radar and sonar are two well known ToF systems. The latter uses slow sound waves in contrast to an electromagnetic signal as the electronics responsible for the time measurement does not need to be of high accuracy. However, it is very difficult to achieve a high lateral resolution for depth maps since narrow beams could not be maintained successfully. Therefore, illumination based on visible or infrared light is desirable which is significantly easier to focus than sound waves. But this requires a much higher precision in the timing electronics for distance measurement. ToF systems are commonly categorized into direct-, shuttered-pulse-based-, and continuous-wave ToF systems.

Direct ToF systems typically use a diode or a laser for emitting a light pulse and then measure the time until the light pulse returns. These systems achieve a high accuracy and are available for acquiring a single point, a scan-line, or also a fully covered field of view. The main drawback of these relatively expensive systems is their mechanical vulnerability since laser scanners are mechanically complex devices with moving components.

The **shuttered pulse-based technique** emits discrete light pulses and requires an image sensor which integrates the incident light (from the backscattered pulse) over a given exposure time. A very fast shutter is needed in order to estimate the pulsed-light delay from the total amount of light observed by the sensor. For instance, this technique is used by the ZCam device developed by 3DV Systems [JVC, GY06].

Continuous-wave (CW) ToF systems do not require high speed shutter electronics in contrast to pulse-based techniques. Using an amplitude modulated light source the depth is determined by measuring the phase shift between the emitted and the received optical signal. In contrast to stereoscopic approaches, devices which utilize the CW-ToF approach as well as the pulse-based ToF principle compute depth maps by simple calculations requiring only a small amount of computational resources. Both ToF systems suffer also less from shadowing effects. However, due to the periodicity of the emitted optical signal the CW-ToF systems' non-ambiguity range is limited. The CW-ToF approach is utilized by a number of ToF depth camera manufacturers: e.g. Canesta/Microsoft, MESA Imaging, SoftKinetic, and PMD [GYB04, MES, KVN07, RH07].

2.2 Photo Mixing Device Technology

In this section the principles of ToF systems as well as their error characteristics are briefly discussed for Photo Mixing Devices (PMD) developed by the company *pmdtechnologies*. PMD sensors gained a lot of attention in literature because of their open design as well as for the known processing pipeline [Lan00]. The general working principle should be also applicable to other ToF device manufacturers who use very similar techniques [OLK*04, GYB04]. Thus, the overview which is given in this section should be valid in a rather general sense for ToF systems.

2.2.1 Signal Theory

PMD sensors consist of two key components: a light source which illuminates the scene with modulated, incoherent near infrared (NIR) light as well as a CMOS ToF sensor which measures the returning light (see Fig. 2.3). So-called smart pixel sensors [XSH*98, Lan00] gather the reflected light. Usually a complete sensor array and proper imaging optics are used to provide a full field of view. However, in comparison to multi mega-pixel RGB cameras ToF cameras have a rather low lateral resolution, e.g. current models are available from 160×120, and 200×200 pixels [PMD] up to 512×424 pixels (Microsoft ToF Kinect for Windows, [Kin]).

A single pixel samples and correlates the incoming optical signal with the internal reference signal of the modulated illumination which results in an image of a per-pixel sampling of the correlation function with an additional internal phase delay. This image is also called phase image. A PMD sensor takes typically four of these phase images to determine the distance related phase shift and thus the distance to the respective object region can be calculated.

This process can be expressed by the following equations: given a reference signal $g(t)$ and the optical signal $s(t)$ incident to a PMD pixel, the pixel samples the correlation function $c(\tau)$ for a given internal phase delay τ:

$$c(\tau) = s \otimes g = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} s(t) \cdot g(t+\tau)\,\mathrm{d}t. \tag{2.1}$$

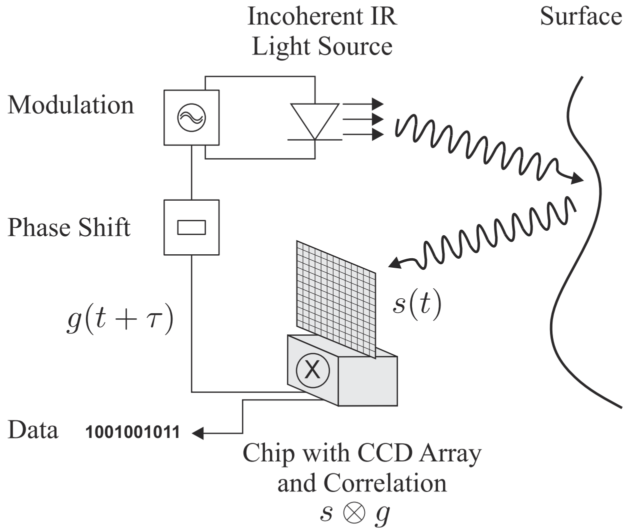


Figure 2.3: ToF principle. The principle of PMD's ToF technology is illustrated.

Assuming the reference signal to be sinusoidal, i.e. $g(t) = \cos(\omega t)$ and no additional non-linear signal deformation, the optical response signal is given by $s(t) = b + a\cos(\omega t + \varphi)$, with $\omega = 2\pi f$ being the angular modulation frequency, a the amplitude of the incident optical signal, b the correlation function's bias (e.g. background light), and φ the phase shift relating to the object distance. It should be noted that the modulation frequency defines the distance unambiguity for the distance sensing, e.g. a modulation frequency of 20 MHz results in an unambiguous range of about 7.5 m. Basic trigonometric calculations yield:

$$c_\tau(\varphi) = b + \tfrac{a}{2}\cos(\tau + \varphi). \tag{2.2}$$

To be more precise, the demodulation of the correlation function is achieved by using several samples of $c_\tau(\varphi)$ which are obtained by four sequential PMD phase images $A_i = c_{\tau_i}(\varphi)$ which use internal phase delays $\tau_i = i \cdot \frac{\pi}{2}$, $i \in \{0, 1, 2, 3\}$. The final distance d to the object's surface (see Eq. (2.4)) can be calculated based on the four phase images:

$$\varphi = \arctan\left(\frac{A_3 - A_1}{A_0 - A_2}\right), \tag{2.3}$$

$$d = \frac{c_{\text{light}}}{2\omega}\varphi, \tag{2.4}$$

with $c_{\text{light}} \sim 3 \cdot 10^8 \frac{m}{s}$ being the speed of light. The factor 1/2 represents the fact that the light travels the distance twice between the light source and the PMD sensor.

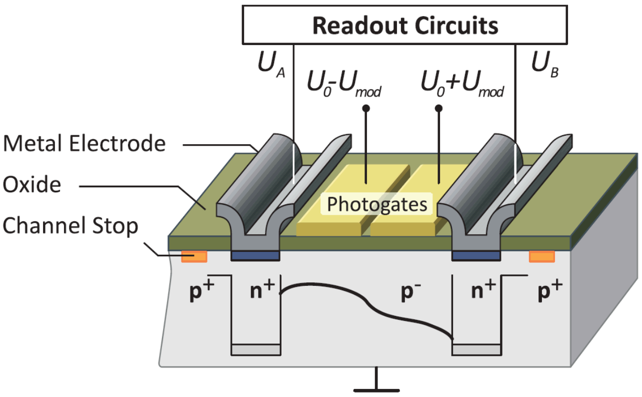


Figure 2.4: PMD pixel design. Simplified pixel design of a single PMD pixel. Fig. courtesy of [MKF*05, Lin10].

Additionally to the distance the intensity values b as well as the signal's correlation amplitude a (see Eq. (2.5)) can be calculated:

$$b = \frac{A_0 + A_1 + A_2 + A_3}{4} \quad \text{and} \quad a = \frac{\sqrt{(A_3 - A_1)^2 + (A_0 - A_2)^2}}{2}\,. \tag{2.5}$$

While the intensity values b are comparable to a grey scale image the signal's amplitude values can be interpreted as a confidence value for the distance measurement.

2.2.2 Pixel Design

The pixel's hardware design is based on the CCD principle as Complementary Metal Oxide Semiconductor (CMOS) sensors which provides a low cost production process, small-sized pixels as well as low power requirements. In contrast to standard CMOS pixels which typically contain a single photodiode and a charge accumulating capacitor, each PMD pixel consists of several photogates above a light sensitive area and two corresponding capacitors. This principle is called 2-tap approach since multiple measurements (i.e. two) of the correlation function can be performed in parallel [Lan00]. A variable electrical field is generated across the pixel which shifts the electrons generated by the incident photons to one or the other photodiode. This principle accumulates electrons either to the left or the right integration node (see Fig. 2.4) and mixes the incoming photo-electrons with the phase shifted electronic signal by synchronizing the electrical field with the modulated light source. The mixed signals of the two integration nodes are then integrated over a certain exposure time in order to accumulate enough electrons for a valid signal level.

Both of the integration nodes (often referred to as channel A and channel B) are sensitive to ambient background light. Thus, both channels will contain an equal offset of electrons. However, this offset does not influence the overall measurement since after each integration period the two channels A and B are subtracted. This result is then used for distance calculation.

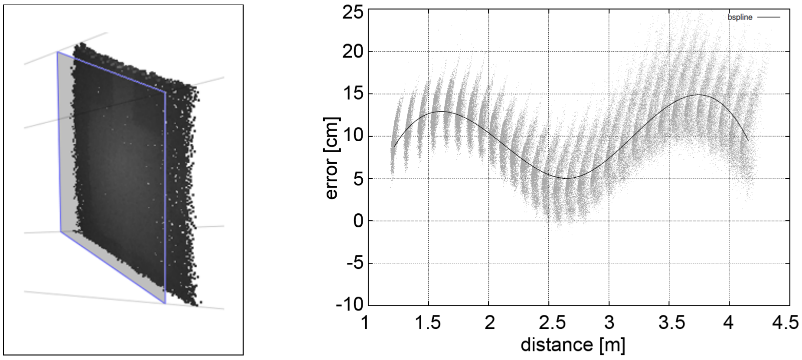
Especially outdoor applications suffer from limited sensor dynamics since a certain amount of the capacitor is already occupied by uncorrelated sunlight carrying no distance information. Thus, the final signal amplitude is comparatively small and the accuracy of the distance measurement is reduced significantly. The use of longer exposure times in order to achieve a better resolution of the signal may be limited by the pixel's early saturation. To reduce the influence of excessive background light a special suppression of background illumination (SBI) circuit has been developed. The SBI circuit dynamically adjusts the charge level for both photodiodes. This effectively improves the dynamic range of a PMD pixel for correlated signals. For detailed information about the technological background of PMD sensors see Lange [Lan00].

2.2.3 Sensor Effects

The data of a ToF sensor, here illustrated by the example of PMD, has a couple of artifacts coming along with the implementation of this kind of technology. An understanding of these artifacts is required in order to get an idea of the data quality.

Wiggling Algorithms which determine the phase shift and compute the distance measurement assume a certain waveform of the correlation function (see Eq. (2.1)). The wiggling error is the error which arises from the higher harmonics in the optical and the reference signal, i.e. the signal is not perfectly sinusoidal. This error is also known as the systematic deviation error. Fig. 2.5(a) displays the sensor data which is measured by capturing a planar surface. However, the actual distance of the surface is marked by the position of the reference plane. In Fig. 2.5(b) the wiggling error is plotted over the interval 1-4 m. Rapp et al. [RFHJ07] investigated the systematic errors and statistical uncertainties of PMD sensors. The distance based errors of the PMD sensors are also examined by Lindner et al. [LK06, LK07] who presented a lateral and depth calibration approach of such sensors in order to account for wiggling effects.

Noise The signal quality and thus the resulting distance measurement is affected by various categories of noise resulting from the pixel's underlying CMOS design: time-dependent noise, time-independent noise, and signal noise influence the accuracy of the measured distance.



(a)

(b)

Figure 2.5: Systematic deviation error. In 2.5(a) the wiggling error is exposed by the reference plane which marks the actual distance from the surface. In 2.5(b) the error is displayed in [cm] over the interval 1-4 m with a b-spline fitting (solid line), Fig. courtesy of [LKR08].

Thermal noise, dark current noise and reset noise [Kle08] are *time-dependent* and significantly increase with rising temperature since electronic components of a camera react sensitively to a variation in temperature. This kind of noise can be reduced or eliminated by a proper cooling of the system as well as linear correction functions [Tem96].

So-called fixed pattern noise introduces an individual offset to each pixel due to statistical process variations and different signal propagation delays in conventional CMOS imagers. This noise artifact can be compensated by determining the offset values which are then finally subtracted from the actual sensor output.

The last noise-category emerges from the photon shot noise: the *signal noise*. It cannot be compensated and therefore has a significant impact on the effective signal-to-noise ratio. In Fig. 2.5(a) the noise becomes visible in the cloud of points while capturing a planer surface.

Motion Blur The total capturing time for a single depth image is composed of the acquisition times of several (typically four) phase images and the respective readout times between the phase image measurements. This temporal integration leads to a motion blurring effect in dynamic scene setups (i.e. moving objects or moving camera itself) since the phase images A_i lead to varying object points observed by the respective pixels (see Fig. 2.6). This results in unmatching phase values during the demodulation of the correlation function. The level of motion blur increases with the speed of the moving objects as well as the duration of the integration time. A compensation model is described by Lindner and Kolb [LK09].

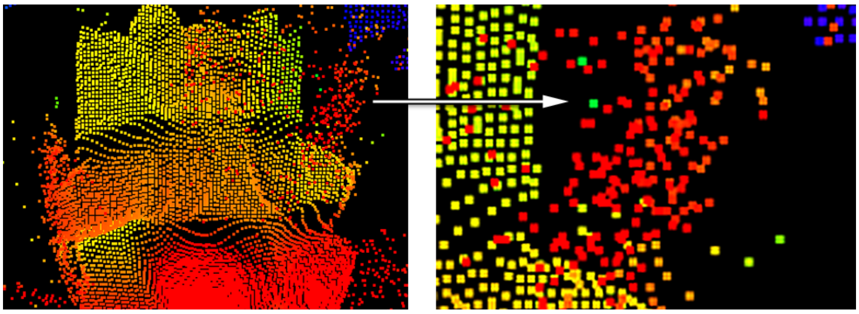


Figure 2.6: Motion artifacts. The object is moving from the right to the left side. This results in motion artifacts which are especially visible at the boundary of the object.

Mixed Phases The artifacts which are covered by mixed phases can have a variety of reasons. In all cases the integrated light for a PMD pixel does not originate from a single, well defined distance of a single object. In these situations different phases from the various distances interfere and are mixed in a single pixel. Flying pixels as well as multipath-interference are typically caused by mixed phases.

If a reconstructed depth value provides inaccurate distance information and its solid angle covers depth inhomogeneities, this pixel is called *flying pixel*. The typical artifacts where the object's boundaries tend to drift either towards the background or the camera sensor are illustrated in Fig. 2.7(a). A common approach is the classification of flying pixels by the number of valid neighbor pixels which then can be discarded if the number of nearby pixels is below a certain threshold [LLK08].

If light from the active light source not only illuminates the object directly (normally referred to as the primary return of the light) but also partly by a reflection in the scene the measured distance to the object is corrupted. This effect is called *multipath-interference*. While this type of artifact is hard to quantify and depends on the actual scene it is clearly visible in situations where two planar walls enclose a certain angle (see Fig. 2.7(b)). The acquisition of the scene with multiple frequencies and its subsequent data evaluation help to detect and to eliminate multipath effects [Fuc10, DGC*11, FSK*14].

2.3 Environment Modeling

As the second focus of this thesis is the processing of depth sensor generated data the following sections present an overview of the basic processing techniques with regard to the modeling of environments. The major purpose of environment modeling is the temporal accumulation of sensor data, namely 3D depth maps, into a consistent

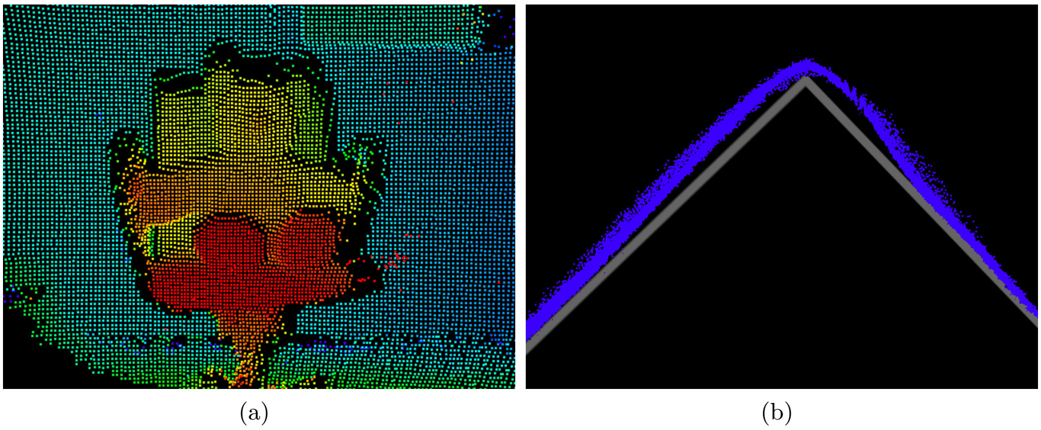
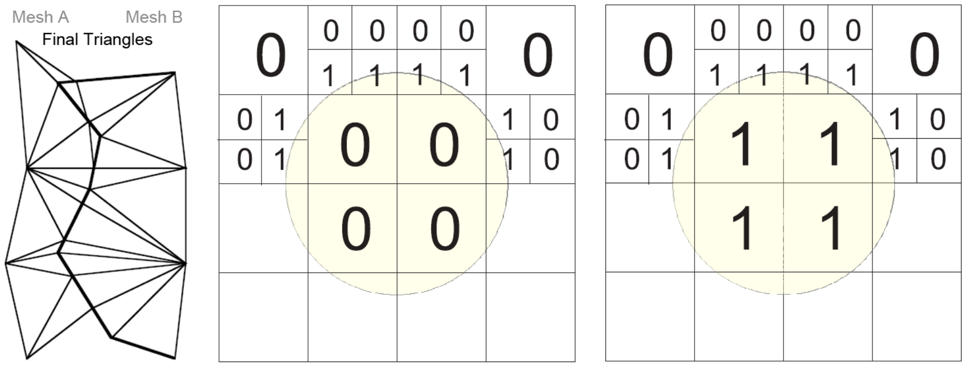


Figure 2.7: Artifacts caused by mixed phases. In 2.7(a) the flying pixels of the objects' boundaries are displayed which mainly drift towards the background. Top view of a corner scene is shown in 2.7(b). The planar reference walls are depicted in grey whereas the actual multipath-interference measurement is colored in blue.

3D model. However, the pure accumulation of several depth maps into a common coordinate space is often not enough. The result would be an unstructured 3D point cloud containing multiple 3D points for the same observed scene point due to the noisy character of the sensor data. Thus, proper environment modeling requires additional computation steps.

In literature numerous approaches describe the acquisition of 3D objects. Many of the publications in the field of environment modeling are often referred to as *(online) 3D reconstruction* approaches for reconstructing complex geometrical objects [CM91, BDL95, CL96, Neu97, RA99, RHHL02]. The generation and application of environment models traditionally resides in the area of mobile robotics and is also used in the fields of 3D outdoor scenery acquisition, navigation and self-localization. Some of the applications utilize 2D laser scanners for data acquisition [HBT03, LEC*01, FZ01]. Other approaches use 3D laser scanners for acquiring the scene [ASG*01, HDH*01, NSH03]. More recent examples for the use of environment models are in-door navigation systems [Pro] as well as hand held scanning applications [IKH*11, NZIS13].

The first step for 3D scene reconstruction is the data *acquisition* step which is performed by utilizing mainly one of the range imaging techniques described in Sec. 2.1, i.e. 3D laser devices, structured-light devices, or ToF devices. Next, the data *pre-processing* stage includes the filtering of raw depth maps as well as determining outliers for better reconstruction results (see also [NIH*11] for details about individual pre-processing steps).



(a) (b) (c)

Figure 2.8: Various types of environment models are shown: polygonal model (2.8(a), Fig. courtesy of [TL94]), surface voxel model (2.8(b)), and full voxel model (2.8(c)).

The remaining steps of 3D reconstruction involve the *registration* of multiple datasets, followed by *merging* the depth data into a common model which usually includes also a data *fusion* step in order to reduce the overall data load. The following sections summarize the various types of environment models and further describe the basic steps of environment modeling.

2.3.1 Types of Representation

In general, environment models can be represented by various geometric representations. The most important types are polygon models, surface and full voxel models (see Fig. 2.8) as well as point-based representations (see Fig. 2.11(a)). Polygonal models consist of several individual polygon meshes which are merged into a single polygonal model as proposed by Turk and Levoy [TL94]. In addition to the surface voxel model which contains only voxels close to the object's surface, the full voxel model also includes the voxels which are located in the inner region of the object. Moravec and Elfes [ME85] introduced occupancy grid mapping in the field of robotics dividing the full model's space into unknown, occupied, and free areas. Point-based representations are more amenable to the input data acquired from depth sensors in contrast to the afore-mentioned methods. A dense surface is represented by the accumulation of point data suitable for high quality scanning of small objects [WWLVG09] as well as large scale reconstructions [HKH*12].

Visual Hull The visual hull of an object is defined as the full voxel model comprising the intersections of all 2D silhouettes showing the object from arbitrary views. Therefore, the foreground object is segmented from the background for each of the

2D source images. The computation of the visual hull is based on *volume carving* starting from an initially occupied volume of voxels. The intermediate model representation during the carving process may be a binary occupancy grid. Kuzu and Sinram [KS04] refine the volume carving algorithm by checking the voxels of the visual hull for color correspondence. Tosovic et al. [TSK02] describe a combination of structured light- and visual hull-algorithms using a hierarchical data structure. The structured-light approach allows for the reconstruction of concavities on an object's surface. However, Matusik et al. [MBR*00] describe an image-based method for the computation of visual hulls which does not rely on a volumetric representation, but computes a viewpoint dependent visual hull in image space.

Grid types Grids are used for storing the spatial data information of the specific type of environment model. Grids are also utilized as acceleration structures for fast access of the data within the environment model. The most important data structures are *regular grids* as well as *hierarchical grids*.

Regular grids divide the space into axes-parallel, rectangular volume elements (voxels) having all edges of a single axis of the same size. The simplest volumetric case is the so-called cartesian grid composed of cubical cells. A regular grid ensures fast access to its elements, e.g. neighborhood searches. Its main disadvantage is the large memory footprint needed for the entire allocation of the volume - memory is also allocated for space which will remain unoccupied. Large sized volumes as well as volumes with a very fine grained resolution, i.e. small size of individual cells, are challenging and cannot be handled efficiently.

The solution is the usage of hierarchical grids which allocate the memory on demand only. The advantage of hierarchical data structures is their high local adaptivity for representing spatial details. The adaptive size of the grid cells can be determined by taking the lateral resolution of the depth camera into account as well as its depth accuracy. Prominent types of those grids are octrees and kd-trees.

Octree The octree is a recursive data structure in which a single octree node has either eight or zero child-nodes. Each octree node represents a position in 3D space and occupies its respective volume. In general, every node consists of an element which carries the node's data. However, usually it is sufficient that only the leafs of the octree structure contain the application relevant data elements. The root node corresponds to the entire 3D space being occupied by the octree. In [Mea82, LC94] first octrees have been used with regard to the representation of depth maps. Schiller and Koch [SK09] explore octrees for capturing dynamic scenes. Octomap presented by Hornung et al. [HWB*13] is a modern example of environment modeling using octress for representing occupancy grids.

Kd-tree The kd-tree is a binary tree consisting of nodes which represent a k-dimensional point, with $k = 3$ in 3D space. A kd-tree recursively subdivides the space into two subnodes by generating a splitting hyperplane. Every node is associated with one of the k-dimensions with the hyperplane perpendicular to that dimension's axis [Ben75, ZHWG08].

2.3.2 Registration

The acquisition of a single depth map is only an intermediate step with regard to the entire acquisition process. Multiple depth maps acquired from various sensor positions require knowledge about the position and the orientation of the sensor in order to transform the data, namely the respective 3D points, into a common coordinate system. The localization of the sensor could be done either based on an initial calibration and measurement process determining the exact sensor position, e.g. by taking a robot's pose into account. Or a so-called *registration* step could be performed on the depth map data directly.

Iterative Closest Point At the beginning of the registration usually depth maps are transformed into a good initial starting position, sometimes this is done manually. In a second step a variant of the well known iterative closest point (ICP) algorithm [BM92, CM91] is performed for fine tuning the initial registration guess. This method has become the prevalent algorithm for aligning 3D datasets into a common model purely based on their geometry.

As a prerequisite this algorithm requires that the two depth maps which are transformed to each other are overlapping to a sufficient amount. Pairs of points are identified as point-correspondences based on appropriate criteria, e.g. points' distance, surface orientation, and color. The error, i.e. the point pairs' criterion, is minimized using the least squares approach as applied by Rusinkiewicz et al. [RHHL02]. Often the procedure of finding point correspondences and optimizing the transformation result is performed multiple times until the overall error is small enough, i.e. the transformation of the depth map into a common coordinate space is found (see Fig. 2.9). This type of registration is called pairwise registration. This means that a sequence of consecutive depth maps are sequentially registered to each other. The various variants of the ICP algorithm mostly differ regarding the choice of the correspondence criterion.

Simultaneous Registration Better than only registering consecutive frames to each other is the approach of registering all views simultaneously [Pul99]. This minimizes the global error and avoids inconsistent scene reconstructions. The simultaneous approach uses the result of the pairwise registration while diffusing the pairwise

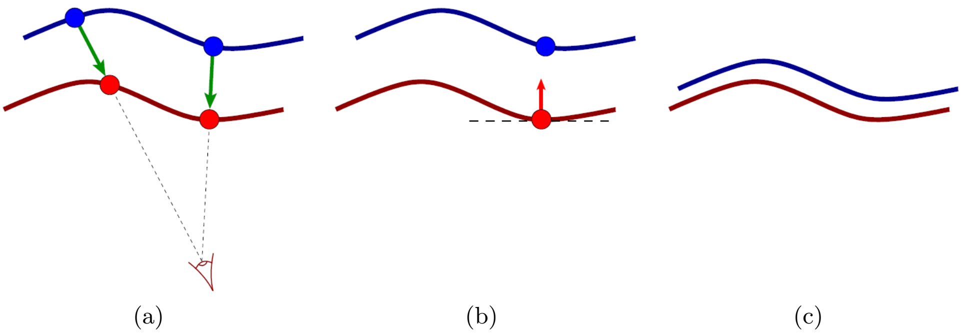


Figure 2.9: Iterative closest point algorithm. Minimizing the error metric moves the blue dataset closer to the red dataset. Fig. courtesy of [RHHL02].

registration errors by directly registering a view's neighboring depth maps. Further approaches of simultaneous registration are described in [BDL95, Neu97, BS97]. Bernardini et al. [BR02] use also optical properties of the scene in comparison to traditional registration algorithms. The color is used to significantly improve the registration result for objects with less geometric features.

2.3.3 Merging

A consistent data representation, preferably without holes in the geometry, is created in the merging stage as soon as the registration of the individual depth maps is finished. Merging algorithms need to be efficient concerning computation time and resources. Literature differentiates between the reconstruction of 3D point clouds [BMR*99, GKS00] and 3D reconstruction based on depth maps [TL94, CL96].

Mesh-based Approaches Zippering [TL94] is a popular approach for transforming multiple depth maps into a consistent triangulated mesh. Depth maps are connected to each other on the basis of polygonal meshes. For that to happen, depth maps are tessellated and their vertices are weighted. Redundant triangles in overlapping regions are discarded. The remaining meshes are zippered together. Once all meshes have been combined, the individual depth maps contribute to the resulting surface by finding the consensus geometry. The final position of a vertex is calculated by taking the nearby positions from each of the original depth maps. Surface refinement is postponed until the overall geometry has been determined. Thus, discontinuities are eliminated which may have been introduced in the zippered regions.

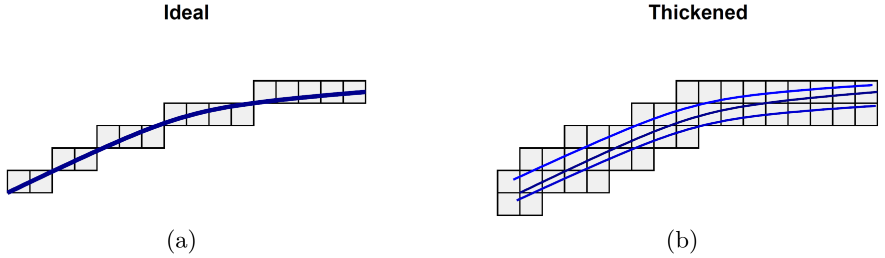
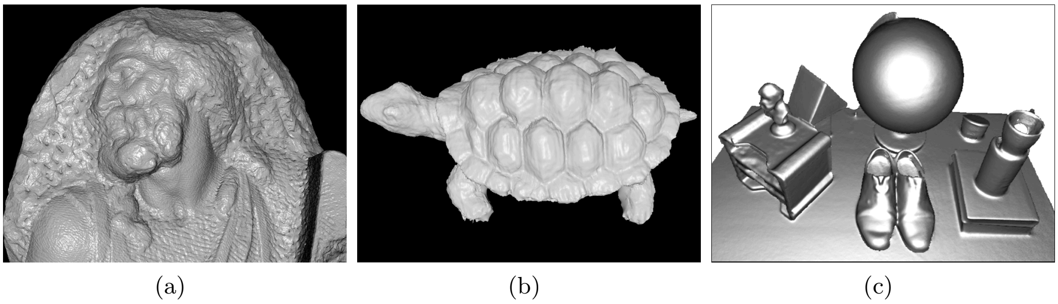


Figure 2.10: Volumetric merging. 2.10(a) shows an ideal merging result for no noisy data. In practice, the noise in the data leads to a wider layer of voxels as illustrated in 2.10(b). Fig. courtesy of [RHHL02].

Implicit Approaches: Function- and Volume-based The approach of combining depth maps into a volumetric model avoids problems which occur in the previously described approach of combining polygon meshes. Curless and Levoy [CL96] use weighted signed distance functions. Therefore, each depth map is converted to a signed distance function and finally added to the already existing model data. The resulting surface is extracted via isosurface rendering based on marching cubes as proposed by Lorensen et al. [LC87].

Rusinkiewicz et al. [RHHL02] present a first system for real-time acquisition of 3D models using a voxel-based merging approach. All points of the depth map are added to the voxel grid by quantizing their positions and determining their orientation, i.e. normal calculation. While adding new depth maps to the voxel model also the normal per voxel is accumulated. Sensor noise and registration inaccuracies are only partly taken into account: depth maps are temporarily triangulated before being added to the volume structure. Thus, tiny triangles are discarded to account for measurement inaccuracies. A sequence of ideal or highly accurate depth data would result in a narrow band of surface voxels. However, noisy measurements involve much more surface voxels (see Fig. 2.10). Rusinkiewicz et al. [RHHL02] use this online accumulation only for interactive user feedback. The final model is constructed by utilizing an offline registration approach based on [Pul99] in order to achieve high quality results. The actual merging step is then based on weighted signed distance functions. Finally, the surface is converted to a mesh using the marching cubes algorithm.

With the emergence of inexpensive consumer depth cameras, e.g. Microsoft's Kinect, real-time variations of Curless and Levoy's volumetric fusion approach [CL96] have been proposed [IKH*11, NIH*11]. *Moving volume* variants [RV12, WKF*12] try to overcome memory limitations and allow for the reconstruction of larger volumes. Nießner et al. [NZIS13] use a hashing data structure in order to reconstruct scenes at larger scales.



(a) (b) (c)

Figure 2.11: Rendering techniques. The image in 2.11(a) is rendered using QSplats - a splatting approach for rendering large scale models. In 2.11(b), marching cubes is applied for extracting a polygonal mesh. 2.11(c) employs a ray casting approach for high quality surface reconstruction. Fig. courtesy of [RL00, RHHL02, NIH*11].

2.3.4 Rendering

The environment model is visualized by rendering its data representation. Various techniques are capable of generating synthetic views of the accumulated data (see Fig. 2.11).

Splatting The splatting approach [Wes90] projects the 3D points' footprints directly onto the image plane. Additional properties such as orientation of the surface and the distance to the sensor's projection center are taken into account [RL00, RPZ02]. The approach by Stolte [Sto11] is based on hierarchical octree structures. Splatting offers a way for rendering also large point-based models in real-time.

Polygon Surface The advantage of extracting polygonal surfaces from volumetric grid cells is that standard methods of computer graphics and image synthesis can be applied to polygonal meshes for further processing. The marching cubes algorithm [LC87] is directly employed on regular volumetric grid structures and creates a mesh. Alternative approaches work also on octree structures. The generation of polygonal surfaces may be challenging for large models in real-time.

Ray Casting One of the most popular image-order methods for rendering volumetric data is the ray casting algorithm (see discussion in Levoy [Lev88]). For each pixel in the image domain a single ray is cast into through the volumetric grid. The data is then resampled at discrete positions along the ray. As the graphics processing units evolve and become even more powerful ray casting approaches can handle also large datasets at reasonable frame-rates [RGW*03, KW03].

2.3.5 Discussion

The choice of algorithm strongly depends on the project's requirements, i.e. the features which are supported by the environment model. The decision whether to use the full voxel model or the surface model depends on the support of volume carving, the model spatial's extension as well as the availability of processing power. Furthermore, the decision for the accuracy of the registration strongly influences the requirements for the computational resources.

Full Voxel Model vs. Surface Model Approaches such as volume carving algorithms and the generation of occupancy grids are based on full voxel models [KS04, HWB*13]. This type of representation is computationally expensive since not only the surface voxels need to be determined but also the inner as well as outer voxels need to be classified and processed. This means, if the processing of the environment model is critical in terms of available processing power then the full voxel model will be a rather inappropriate choice. However, e.g. in the area of robotics this type of environment model is often mandatory. Autonomous guidance as well as the ability of rapidly responding to environmental changes ideally require full voxel models.

If a full voxel model is not mandatory then a surface voxel model only or a model which completely works without an underlying data structure would be the better option. If requirements demand for large sized environment models with regard to spatial extension or memory resources then adaptive grids provide a high spatial resolution locally, e.g. octrees or kd-trees. Selecting the proper merging strategy may also depend on the choice of the environment model's type of representation. Intermediate data representations may introduce discontinuities, limit the scalability and automatically add computational complexity to the approach [SB12, RHHL02]. Thus, intermediate forms of representation are preferably avoided in real-time applications.

The volumetric tree approach described in Chap. 4 uses a voxel-grid structure which supports volume carving on the environment model. A redesign of the original algorithm leads to an approach which does not employ a spatial data structure at all for high quality surface reconstruction (see the point-based fusion approach proposed in Chap. 5). However, optionally the point-based approach can be extended by a probabilistic occupancy grid for the use in robotic scenarios (see Sec. 5.5).

Registration Accuracy vs. Processing Speed The choice of the specific registration algorithm mainly depends on the overall setup. Additional registration steps might be required even if the exact sensor position and orientation is provided along with the depth maps, e.g. by taking a robot pose into account of a depth sensor which is mounted onto the robot's end effector. The accuracy of the orientation of the robot's joints still introduces small errors in terms of computing the six degree-of-freedom (6 DoF) robot pose [FHK*10]. The pairwise ICP registration approach may be used in order to improve inaccuracies of given registration poses. Alternatively, ICP can be applied for calculating the transformation information of two subsequent depth maps from scratch starting with an initial guess. It should be noted that the major drawback of the pairwise ICP registration approach is the accumulation of small errors during the registration process of two consecutive frames which leads to significant errors after registering multiple depth maps.

On the other hand, global registration approaches such as simultaneous registration [Pul99] are more accurate compared to simple pairwise registration. However, simultaneous registration algorithms suffer from their computational complexity. The temporal resolution of depth maps is considerably high and thus requires a lot of computation power. Additionally, the original depth maps are stored for the subsequent registration steps which demands computational resources as well as sufficient memory. Usually, global registration approaches are performed as an offline post-processing step [RHHL02].

2.4 Graphics Hardware

The high temporal resolution of sensor data and the constraint of real-time data processing require the utilization of multi-core processing units such as graphics processing units (GPUs) instead of using a single central processing unit (CPU) only. The nature of GPUs is the ability of handling massive amounts of data by its parallel processing architecture. This means that programming of GPUs has to be taken into account which requires an adaption of the algorithms to the GPU's parallel processing layout.

This section gives a brief overview about graphics hardware and GPU programming to allow for a better understanding of the design and implementation of the processing algorithms presented in this thesis. The proposed simulation of ToF data as well as the methods for the processing of millions of 3D points into an environment model are computationally intensive approaches. These algorithms run at interactive frame-rates by using the parallel execution power of GPUs which cannot be addressed by traditional CPUs. While CPUs sequentially apply a single instruction to a single data element at a time modern GPUs are highly optimized data-parallel streaming processors following the SIMD architecture principle (single instruction, multiple data) [KH10]. In recent years, developments regarding a more generalized architecture and programming model provide computing capabilities for a variety of application areas such as computational fluid dynamics, medical imaging, and

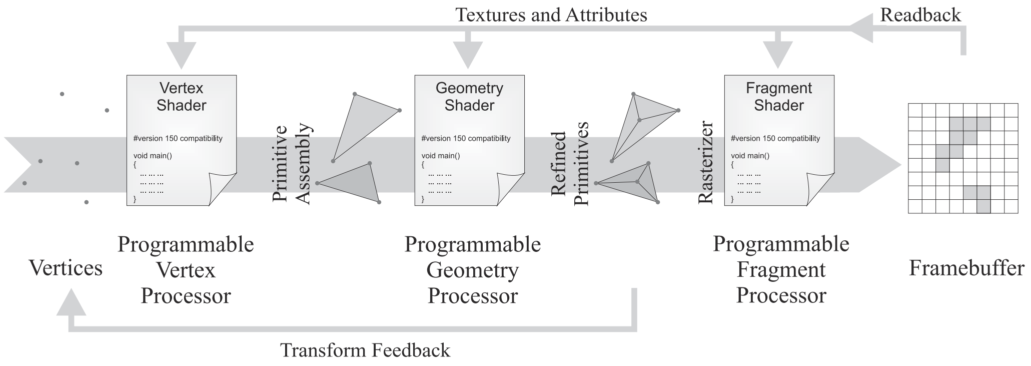


Figure 2.12: Programmable graphics pipeline. A single render pass, i.e. the process from vertices to screen pixels, comprises the three programmable processing stages.

environmental science [SK10].

2.4.1 Graphics Pipeline

The GPU is designed for generating raster images extremely fast and efficiently. Therefore, a virtual scene must be decomposed into planar polygons in the tessellation step. In the display traversal step all polygonal primitives are then converted into a raster image. However, all 3D graphics processors carry out the display traversal as a pipeline of processing stages. While first graphics processors implemented the so-called fixed function pipeline, modern GPUs are rather flexible and programmable [HKRs*06, PF05, FK03]. Earlier programmable hardware architectures used dedicated hardware called *shader units* which where optimized for a special task. With the introduction of Nvidia's GeForce 8 and ATI's Radeon HD 2000 series the unified shader architecture has been introduced. This hardware consists of many computing (shader) units, each capable of processing any task within the stages of the graphics pipeline. This leads to a better load balancing and thus increases computation throughput of the GPU significantly. Fig. 2.12 shows the three processing stages which set up a single render-pass.

Vertex Processing In the vertex stage a *vertex program* processes incoming vertices, i.e. geometry, and computes linear transformations such as rotation, translation and scaling of the vertices in the 3D domain. Modeling transformation (transform local coordinates into world space), viewing transformation (convert into camera space) and finally the projection matrix (transform into screen space) are also applied in this stage. Before projecting the vertices a simple per-vertex light model can be applied [Bli77]. Geometric primitives are then assembled and handed over to the fragment stage.

Geometry Processing With the specification of shader model 4.0 the vertex processing stage is extended by the so-called geometry processing stage which is applied after assembling the primitives. The primitives can be modified by *geometry programs*. This means, additional primitives can be created as well as the entire geometry can be generated procedurally by emitting or discarding arbitrary vertices.

Fragment Processing In the rasterization stage each geometric primitive is decomposed into a set of fragments. Every fragment corresponds to a specific pixel in screen space. A *fragment program* can perform several texture fetches as well as filtering operations per fragment and finally computes the color of each pixel.

By using *uniform* variables all stages can be supplied with additional input data. Processing results can be stored and supplied to subsequent render passes as input data, i.e. geometry shader output is fed via the *transform feedback* feature back into the pipeline and processed fragments can be read back as textures by being written to a separate render target instead of the frame buffer.

The application developer gets access to the graphics pipeline by utilizing the open graphics library (OpenGL)[1] or DirectX[2], the two main application programming interfaces (APIs) for graphics. The various shader stages are programmed by using C for Graphics (CG)[3], OpenGL shading language(GLSL)[4], or high level shading language (HLSL) - all are inspired by C-like syntax and allow e.g. control flow statements such as loops and conditional branches.

2.4.2 General Purpose Computation

General purpose computing on GPU (GPGPU) performs computations in application areas which are traditionally handled by the CPU. These programs gain a massive performance boost based on the highly parallel streaming processors of the GPU. The invention of the programmable shader pipeline as well as floating point support on the GPU ($\sim$ 2001) made general purpose programming popular. Nowadays, the GPU can be programmed in two different ways with regard to general purpose computation: using the graphics API (shader-based) or using CUDA/OpenCL. Both of the methods are used for implementing the algorithms presented in this thesis.

Shader-based GPGPU In the early days, programming GPGPU required reformulating algorithmic problems to the graphics pipeline specification. Since GPUs

[1] OpenGL, `https://www.opengl.org`.

[2] DirectX and HLSL, `https://msdn.microsoft.com/`.

[3] CG, `https://developer.nvidia.com/cg-toolkit`.

[4] GLSL, `https://www.opengl.org/documentation/glsl/`.

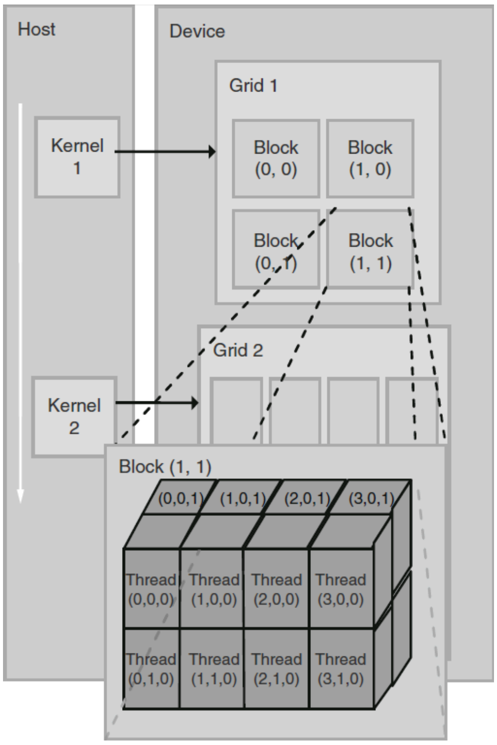


Figure 2.13: Cuda architecture. Cuda threads are organized in a multi-dimensional grid and block structure. Fig. courtesy of [KH10].

only process vertices and fragments the programmer needs to learn graphics terminology (such as textures, vertices, and frame buffers) for non-graphics problems. Owens et al. [OLG*07, OHL*08] describe many of the commonly used techniques which map complex applications to the GPU. Not only that researchers need to learn OpenGL or DirectX as well as the specific shading language to get access to the GPU's computation power - the main drawback of using the graphics API is the cumbersome translation of problems to map them onto the programmable graphics pipeline.

CUDA and OpenCL: non graphics interfaces In 2006, Nvidia introduced the compute unified device architecture (CUDA) which includes several new components specifically designed for GPU computing. CUDA aims to get rid of the limitations which prevented previous graphics processors from being used by more researchers in even more application areas for GPGPU. It is not necessary anymore to access the GPU by programming the shader units only. The new architecture allowed all algorithmic logic units (ALUs) to be marshaled by programs performing general purpose computations and thus the ALUs comply with IEEE requirements for single-precision and floating-point arithmetic [WFF11]. CUDA C became the first language to facilitate GPGPU without having knowledge about the graphics pipeline which is only accessed though OpenGL and DirectX. Fig. 2.13 illustrates the basic working principle of CUDA's thread invocation: when a kernel is launched, it is executed as a grid of parallel threads. Each CUDA thread grid consists of many (e.g. millions) of lightweight GPU threads per kernel invocation. All threads within the same block can synchronize their execution and efficiently share data through a low-latency memory.

In 2009, Apple, ATI/AMD, Intel and Nvidia developed the Open Computing Language[1] (OpenCL) as a standardized programming model. OpenCL defines language extensions and runtime APIs which allow developers to program parallelized code on massively parallel processors such as GPUs. Applications which have been developed in OpenCL can run without being modified on all processors that support also OpenCL in contrast to CUDA which is only supported by Nvidia GPUs.

2.4.3 Discussion

The choice of shader-based versus non-graphics pipeline-based GPGPU strongly depends on the application and on the formulation of the algorithmic problem. Furthermore, the developer needs to determine which GPU vendors are supported and which GPU features are required for the algorithm in order to achieve the maximum performance speed-up. The algorithms presented in this thesis are implemented using both: the graphics pipeline based GPGPU approach as well as using Nvidia's CUDA. The ToF sensor simulation approach (see Chap. 3) is implemented using pure shader-based GPGPU techniques since the synthesis of the depth camera's image-output perfectly fits to the shader-based graphics pipeline. This way also older GPU models are supported which results in lower hardware requirements and better acceptance by the users.

The volume tree approach (see Chap. 4) for the accumulation of depth maps is also programmed with shader-based GPGPU techniques. It exhaustively uses the transform feedback feature of the graphics pipeline. However, the complexity of the shaders is noticeable and the overall render-pass structure is rather sophisticated which is mandatory for performing the data structure's expansion and reduction algorithms correctly. Anyhow, this way also older graphics card models are supported which achieve already a substantial performance boost for real-time data processing.

In Chap. 5 the point-based fusion method is presented for depth map accumulation and high quality online 3D scene reconstruction. This approach is implemented using a mixture of the shader-based pipeline and Nvidia's CUDA. Technically, central parts of the algorithms could also be thought of being converted to the standard graphics pipeline although they are implemented in CUDA which has been the decision for

[1] Khronos Group - OpenCL, `https://www.khronos.org/opencl`.

developing the algorithms faster as well as having better debugging possibilities. The reason for taking CUDA over OpenCL is that OpenCL programming constructs are at a lower level and thus more tedious to use compared to CUDA which follows a rather high-level programming interface.

Chapter 3

Time-of-Flight Sensor Simulation

In the previous chapter current range imaging techniques have been briefly described and the PMD-based ToF principle has been outlined (see Sec. 2.1 and 2.2). The current chapter focuses on real-time sensor simulation for ToF sensors. Dynamic motion blurring and resolution artifacts such as flying pixels as well as the typical deviation error are prominent effects of real world systems and therefore the modeling of these artifacts (see Sec. 2.2.3) is essential for an authentic simulation approach.

After starting with a general section about motivation and research objectives, the structure of this chapter is as follows: in Sec. 3.2, an overview of the related work on ToF sensor simulations is given. Sec. 3.3 proposes the physical sensor model which is the basis for the simulation approach. While in Sec. 3.4 the generic simulation framework for the integration of sensor concepts is presented, a realization of the theoretical sensor model is described in Sec. 3.5. The results of the sensor simulation are presented in Sec. 3.6 along with the evaluation of simulation data. Finally, Sec. 3.7 discusses the presented simulation approach.

Publications The sensor simulation framework which provides the technical basis for the simulation approach has been published in [KOKP07]. The theoretical approach of simulating ToF sensors in real-time and a comparison of simulation results and real sensor data has been presented in [KK09].

3.1 Motivation

The real-time simulation approach for ToF sensors has been developed in the context of the Lynkeus 3D project (cmp. Chap. 1). The project demanded for a physically-based simulation of such sensors as an essential building block for hardware design and application development.

A simulation of ToF sensors is helpful in numerous use cases. On the one hand, the sensor parameters such as resolution, focal-length and frequency can easily be modified and distortion effects can be evaluated in detail. On the other hand, experiments can be carried out under reproducible conditions, especially with regard to dynamic scene setups. The simulation approach should not be based on an ideal type of sensor, of course, rather the simulation result must reflect the major sensor characteristics in order to produce results comparable to real sensor data. The development of subsequent algorithms is significantly effected by the simulation data, e.g. in the fields of object recognition and image analysis as well as correction of calibration errors. A real-time simulation is favored since the algorithms themselves handle the sensor data also in real-time within the data processing pipelines. This means that the sensor simulation is preferably as fast as real world sensors.

Challenges Since subsequent algorithms may use the simulation as a substitute for a real world device the simulation approach focuses on real-time simulation. This means, not only the generation of synthetic sensor data and its high precision is challenging but also the manipulation of single camera parameters is required. High performance can be achieved by GPU programming. However, using standard rasterization techniques limits the data processing to per-pixel data only which requires an adaption of the simulation algorithms to the GPUs parallel SIMD architecture (cmp. Sec. 2.4).

Objectives The goal of this chapter is the proposal of a physically-based simulation approach for ToF sensors. The simulation covers major artifacts such as flying pixels, motion blur, as well as the systematic deviation error which are typical effects of ToF cameras. The capability of real-time simulation is ensured by developing a parallelized simulation approach using the programmability of GPUs. This way, the simulation allows for interactive feedback and real-time processing of subsequent processing algorithms.

3.2 Related Work

Researchers often apply their algorithms on both real sensor data and simulated sensor data. Peters et al. [PHKL06] use synthetic test data for the localization of mobile robots. Their ToF simulator application is Matlab-based and not suitable for real-time simulation. The analysis of the synthetic scene geometry with regard to its visibility to the simulated PMD pixels is part of their simulation approach. In contrast to Peters, the approach proposed in this thesis uses the rendering pipeline of the graphics card which automatically provides the relevant scene geometry with respect to the current camera pose instead of calculating it laboriously manually. This enables the simulation approach being suitable for real-time applications. In [PLHK07] they extend their application in order to model the bistatic effects which are caused by the spatially separated illuminator and receiver positions in real 3D cameras.

Streckel et al. [SBKK07] use synthetic ideal depth data to test their structure from motion algorithms in combination with ToF sensors. Their simulation of depth noise is based on results presented by Kuhnert and Stommel [KS06] who assume a quadric relation between the camera-object distance and the standard deviation of the PMD sensor's depth data. The latter approaches neglect sensor characteristics like flying pixels and distance deviation (see Fig. 3.9). These artifacts are covered by the approach presented in this chapter.

Meister et al. [MNK13] extend the simplified local illumination approach presented in Sec. 3.3.2 by taking into account realistic light propagation using bidirectional path-tracing. Their simulation model is not real-time capable. However, their technique successfully simulates multipath-interference effects. In contrast to the previously presented simulation models Schmidt et al. [SJ09] focus on the simulation of sensor hardware and thus do not handle the light propagation and illumination in their approach, instead they use pre-computed light maps as input.

3.3 Physically-based Sensor Model

In this section a physically motivated sensor model is presented. The model is the basis for the simulation approach and it comprises the sensor characteristics described in Sec. 2.2.3 in order to be comparable to real sensor data. In particular, the approach aims at the simulation of the following ToF specific artifacts:

Flying pixels Pixels which cover depth inhomogeneities of the scene.

Motion blur The temporal integration during the acquisition of the individual phase images leads to blurring artifacts in the resulting depth map.

Wiggling This systematic deviation error occurs from the higher harmonics in the optical- and the reference signal.

Noise Various noise sources lead to inaccurate measurements.

3.3.1 Illumination and Radiosity

Radiosity is defined as the density of the total emitted and reflected radiation leaving an infinite small surface element. The radiosity B is calculated for a point P of a lambertian surface which is illuminated by an area light source L as follows:

$$B(P) = \int_{Q \in L} f(-\hat{\omega}_Q) \frac{1}{{d_{P,Q}}^2} (\hat{\omega}_Q \cdot \hat{n}_P) \cdot k_P \, \mathrm{d}Q \ , \tag{3.1}$$

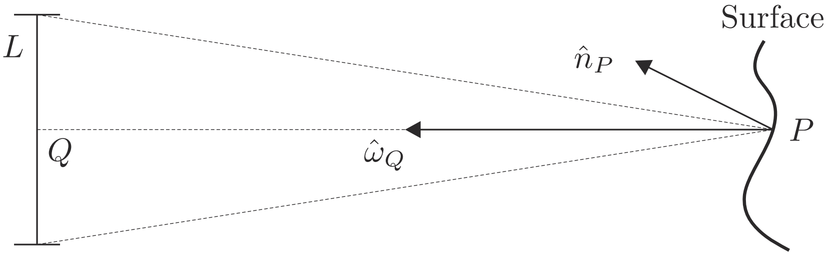


Figure 3.1: Radiosity. The surface is illuminated by an area light source L. The brightness of the light which is emitted in P depends on $\hat{\omega}_Q$ and $\hat{n}_P$. These vectors form the light's angle of incidence.

where f is assumed to be the emission distribution function of L. Then $B(P)$ is the outgoing intensity at point P (see Fig. 3.1). The term $d_{P,Q}$ stands for the distance from P to a point Q of L. The brightness depends on the light's angle of incidence which is calculated by the dot product of the light direction $\hat{\omega}_Q$ and the surface normal $\hat{n}_P$ of P. k_P is a constant which expresses the surface's diffuse reflectivity at P.

3.3.2 Simple Model with Point Light Illumination

The demodulation of the correlation function c has been presented in Sec. 2.2 for ideal sinusoidal signals (see Eq. (2.1)). The attenuation of the signal in proportion to a point's distance d_P to the sensor is already included in the optical function s, with $s \propto \frac{1}{{d_P}^2}$. Furthermore, s includes the radiosity in point P of the object's surface which means $s \propto B(P)$. If it is assumed that s equals the reference function g with an additional phase shift φ_P, then this can be expressed by:

$$s(t) = g(t + \varphi_P) \cdot \frac{B(P)}{{d_P}^2} \,. \tag{3.2}$$

In this first approach the illumination of the scene is restricted to a point light source. This simplification limits the light source L in Eq. (3.1) so that it consists of one sample Q which means it is not integrated over the area of the light source. Furthermore, if it is assumed that a PMD pixel is affected by just a single point P of the surface, this results in the equation:

$$A_i = \lim_{T \to \infty} \frac{1}{T} \int_{-T/2}^{T/2} g(t + \varphi_P) \cdot \frac{B(P)}{{d_P}^2} \cdot g(t + \tau) \, \mathrm{d}t \,. \tag{3.3}$$

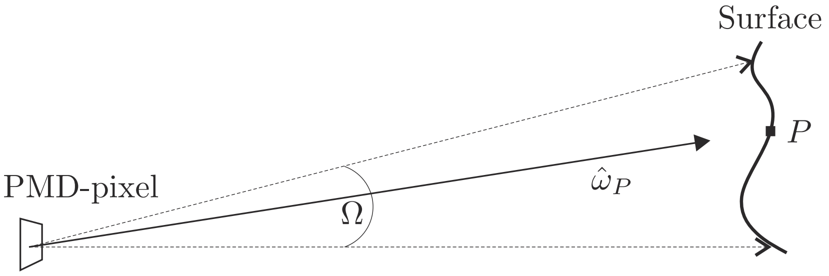


Figure 3.2: Solid angle. A single PMD pixel covers not just the distance to one point of the surface, it rather represents also depth inhomogeneities which are covered by its solid angle.

A closer look at the physical setup of the ToF sensors shows that a modification of Eq. (3.3) is still required. A single PMD pixel covers not only the distance to one point of an object's surface, it rather receives the reflected optical signal within its solid angle Ω which is illustrated in Fig. 3.2. Therefore, it is necessary to integrate over Ω:

$$\begin{aligned} A_i &= \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} \left[\int_{\hat{\omega}_P \in \Omega} g(t+\varphi_P) \cdot \frac{B(P)}{{d_P}^2} \, \mathrm{d}\hat{\omega}_P \right] \cdot g(t+\tau) \, \mathrm{d}t \\ &= \int_{\hat{\omega}_P \in \Omega} \frac{B(P)}{{d_P}^2} \left[\lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} g(t+\varphi_P) \cdot g(t+\tau) \, \mathrm{d}t \right] \mathrm{d}\hat{\omega}_P \, . \end{aligned} \tag{3.4}$$

Now $\bar{c}$ is introduced as the *normalized* correlation function which is - in contrast to c in Eq. (2.1) - independent of the illumination and the attenuation of the signal, thus Eq. (3.4) can be written as

$$\begin{aligned} A_i &= \int_{\hat{\omega}_P \in \Omega} \frac{B(P)}{{d_P}^2} \cdot \bar{c}_\tau(\varphi_P) \, \mathrm{d}\hat{\omega}_P \\ &= \int_{\hat{\omega}_P \in \Omega} f(-\hat{\omega}_Q) \frac{1}{{d_{P,Q}}^2} (\hat{\omega}_Q \cdot \hat{n}_P) \cdot k_P \cdot \frac{1}{{d_P}^2} \cdot \bar{c}_\tau(\varphi_P) \, \mathrm{d}\hat{\omega}_P \, . \end{aligned} \tag{3.5}$$

If the position of the point light source is set exactly to the position of the sensor, which means that $d_{P,Q} = d_P$, a discretization of Eq. (3.5) yields:

$$A_i = \sum_{\hat{\omega}_P \in \Omega} f(-\hat{\omega}_Q) \frac{\hat{\omega}_Q \cdot \hat{n}_P}{{d_P}^4} \cdot k_P \cdot \bar{c}_\tau(\varphi_P) \, . \tag{3.6}$$

3.3.3 Enhanced Model including Area Light Illumination

The approach of a point light illumination which is described in the previous section can be extended to an illumination with area lights which represents the realistic light model of a real world device. Assuming an area light source L, then all quantities which are related to the light position have to be varied. Then Eq. (3.5) leads to:

$$A_i = \int_{\hat{\omega}_P \in \Omega} \frac{k_P}{{d_P}^2} \left[\int_{Q \in L} f(-\hat{\omega}_Q) \frac{1}{{d_{P,Q}}^2} (\hat{\omega}_Q \cdot \hat{n}_P) \cdot \bar{c}_\tau(\varphi_{P,Q}) \, \mathrm{d}Q \right] \mathrm{d}\hat{\omega}_P \, . \tag{3.7}$$

It should be noted that the phase shift φ depends on the distance $d_{P,Q}$ from the point Q of the light source to the point of the surface P as well as on the distance d_P from the surface to the sensor, which means that $\varphi_{P,Q} \propto \|P - Q\| + d_P$. The discretized version of Eq. (3.7) reads:

$$A_i = \sum_{\hat{\omega}_P \in \Omega} \frac{k_P}{{d_P}^2} \sum_{Q \in L} f(-\hat{\omega}_Q) \frac{1}{{d_{P,Q}}^2} (\hat{\omega}_Q \cdot \hat{n}_P) \cdot \bar{c}_\tau(\varphi_P) \, . \tag{3.8}$$

3.4 Simulation Framework

The simulation approach particularly aims at the generation of synthetic sensor data and the manipulation of camera parameters in real-time, while incorporating the most relevant artifacts of PMD-based ToF cameras. The idea of facilitating the generation of such synthetic data led to the development of a framework which allows for the simulation of various full-range camera-like ToF sensors [KOKP07]. Its interfaces regarding data-manipulation and sensor-simulation are generic, thus new sensor concepts can be incorporated into the framework.

The high performance of the framework is achieved by utilizing programmable GPUs which allow for interactive simulator feedback. All relevant scene-, object- and camera-parameters are editable and accessible to the frameworks's simulation core. The framework provides the possibility to add user-defined parameters which are essential for integrating additional sensor characteristics. The basic architecture of the simulation framework is illustrated in Fig. 3.3.

Configuration The scene consisting of various geometric objects and state definitions is loaded into the application using a VRML compatible data format [Web97, Aut06]. Optical camera parameters like exposure time, focal length and image resolution as well as sensor specific parameters such as modulation frequency are accessible

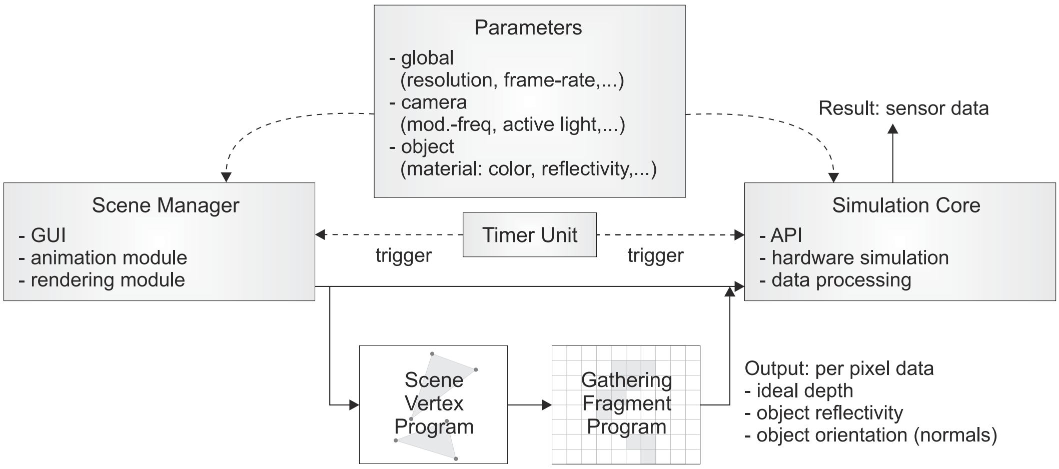
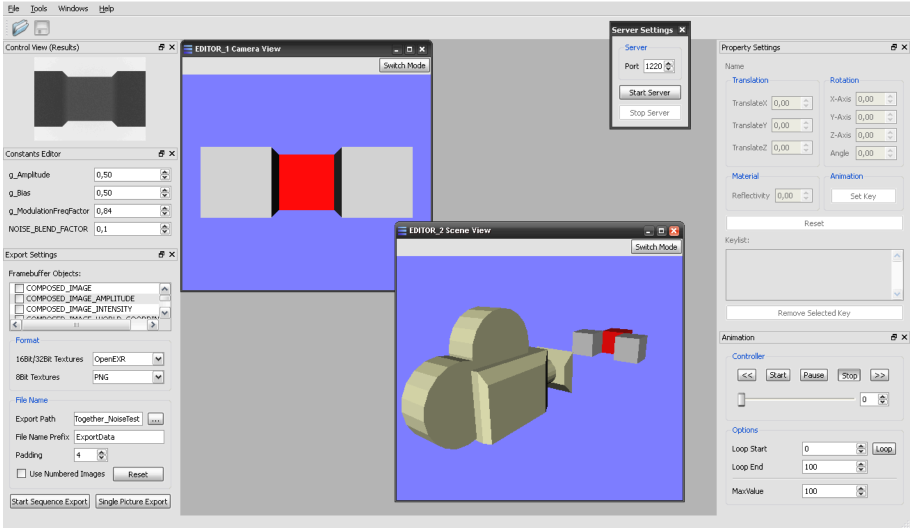


Figure 3.3: Framework architecture. All parameters are freely configurable and accessible by the scene manager and the simulation core. A timer triggers the various rendering passes. The simulation core has full access to the output of the gathering fragment program which calculates several data per pixel.

and editable via the user interface. Additionally, the configuration of object and camera motion during a simulation sequence can be specified. The simulation itself is then based on all specified parameters. During a simulation sequence each frame can be selected and viewed separately due to the real-time capability of the system. Also all parameters are reconfigurable during the simulation which simplifies the evaluation process.

Programmable Simulation Core The framework's simulation core resembles the characteristics of the sensor's hardware and offers the ability to simulate the sensor-internal data generation process. The emulation of ToF sensors is achieved by implementing the simulation core's freely configurable vertex and fragment programs on the GPU according to the sensor model (see Fig. 3.3). The image data is directly stored in the GPU's memory, thus providing full access to the GPU's flexibility and efficiency [SDK05, Owe05]. Its parallel stream processor concept offers a way to outsource a considerable part of the sensor simulation to the graphics card and thus a sensor-pixel is processed very fast.

Limitations and Advanced Features The approach of using standard rasterization techniques of the graphics card is limited to the simulation of *one ray* per pixel at *one moment* in time. For more detailed information about the standard rendering pipeline in computer graphics see [AMH02]. In order to compensate for this disadvantage the framework supplies several features: the spatial super-sampling feature



g_Amplitude 0,50
g_Bias 0,50
g_ModulationFreqFactor 0,84
NOISE_BLEND_FACTOR 0,1

□ COMPOSED_IMAGE
□ COMPOSED_IMAGE_AMPLITUDE
□ COMPOSED_IMAGE_INTENSITY

16Bit/32Bit Textures OpenEXR
8Bit Textures PNG

Export Path Together_NoiseTest
File Name Prefix ExportData
Padding 4
□ Use Numbered Images

Port 1220

TranslateX 0,00
TranslateY 0,00
TranslateZ 0,00
X-Axis 0,00
Y-Axis 0,00
Z-Axis 0,00
Angle 0,00
Reflectivity 0,00

0

Loop Start 0
Loop End 100
MaxValue 100

Figure 3.4: Simulation application. The user interface of the simulation framework including manipulation and simulation sequence modules.

allows the sampling of multiple rays per pixel at one moment in time. This way, the information from neighboring pixels is gathered and evaluated. Furthermore, a super-sampling on the time-axis is provided by the framework, which is useful for the implementation of motion blur artifacts by integrating the data on the temporal basis. The framework also supports the execution of multiple render passes which is necessary in the case of more sophisticated and complex networks of vertex and fragment programs. In Fig. 3.4 a screenshot of the framework's user interface is displayed. A detailed description of the implementation aspects with regard to the simulation framework can be found in [KOKP07].

3.5 Implementation

In this section the theoretical sensor model of Sec. 3.3.2 is implemented which incorporates a point light illumination. The goal is the simulation of the generation process of the four phase images A_i. These phase images also form the basis for further data output of the PMD sensor such as phase shift, depth, intensity and amplitude data.

3.5.1 Phase Image Calculation

As standard graphics rasterization techniques are used it is possible to take advantage of the scene's depth information which is automatically provided by the graphics pipeline. The depth data is available within the range of $[0, 1]$ and includes a projective depth deformation, i.e. nonlinear. The ideal distance information d_{ideal} is then calculated by a perspective correction of the depth data. d_{ideal} also contains the transformation from cartesian coordinates into radial coordinates in order to fit the ToF approach where light source and sensor are located nearly in the same position.

First, the scene is rendered four times according to the phase delay τ_i in order to generate the four phase images A_i. The resulting distance d_{ideal} is then used for the calculation of the ideal phase shift, cmp. Eq. (2.4):

$$\varphi_{\text{ideal}} = \frac{2\omega}{c_{\text{light}}} d_{\text{ideal}}. \tag{3.9}$$

Second, the phase images A_i are calculated by sampling the normalized correlation function $\bar{c}$ and incorporating the object's IR reflectivity and its orientation as well as the intensity attenuation. Thus, the adaption of Eq. (3.6) results in

$$A_i = f(-\hat{\omega}_Q) \frac{\hat{v} \cdot \hat{n}}{{d_{\text{ideal}}}^2} \cdot k \cdot \bar{c}_{\tau_i}(\varphi_{\text{ideal}}),\ \tau_i = i \cdot \tfrac{\pi}{2},\ i \in \{0, 1, 2, 3\}\ , \quad ^{(*)} \tag{3.10}$$

with the intensity of the lambertian light source being $f(-\hat{\omega}_Q) = 1$. k is the object's reflectivity constant, $\hat{v}$ is the vector from the object point towards the camera position and its illumination, and $\hat{n}$ is the surface normal. The brightness is determined by the dot product of $\hat{v}$ and $\hat{n}$ (see Eq. (3.1)).

3.5.2 Flying Pixels

In Eq. (3.6), a PMD pixel is influenced by the information which is covered by its solid angle Ω. Since only per-pixel data is available at the GPU a spatial super-sampling algorithm is used which is applied to a higher resolution rendering of the scene. Thus, a simulated pixel will also contain the information which relates to several directional samples within its solid angle (see Fig. 3.2). In Fig. 3.5, a simple super-sampling algorithm is illustrated which simulates the coverage of a pixel's solid angle and thus leads to a satisfactory simulation of a real world sensor's flying pixel

[(*)] It should be noted that the total amount of light received by a PMD pixel does not depend on its distance since the surface area covered by the pixel's solid angle scales up with the pixel's distance. The original publication presented in [KK09] stated wrongly a $1/d^4_{\text{ideal}}$ relation.

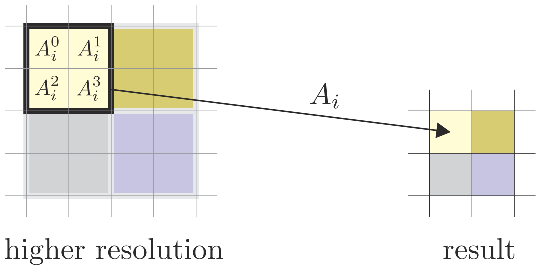


Figure 3.5: Spatial super sampling. The higher resolution image is sampled several times (here: four times). The value for the resulting pixel is based on the mean value. Thus, an approximation of the flying pixel effect of a PMD device is implemented.

effect. The algorithm splits a pixel into several sub-pixels, and a sample is taken for each of the sub-pixels. The resulting pixel's phase value A for a certain phase image i is then defined as the mean value of all m sub-pixels j:

$$A_i = \frac{1}{m} \sum_{j=0}^{m-1} A_i^j \ . \tag{3.11}$$

It should be noted that in contrast to Peters et al. [PLHK07] the sampling of the normalized correlation function $\bar{c}$ is applied to each sub-pixel (see Eq. (3.10)), whereas Peters first computes the phase φ_{ideal} by superpositioning individual point responses and applies the correlation function afterwards. Peters et al. [PLHK07] do not give results regarding the flying pixel effect.

3.5.3 Motion Blur

The motion blurring effect of a ToF device is implemented by rendering the scene several times according to the quantity of phase images A_i. Here the original scene is rendered four times subsequently at four different points of time. This means that during a simulation sequence the geometric setup of the scene, e.g. camera and object position as well as orientation, will differ between the phase images. The left part of Fig. 3.6 shows the various images which are taken during a camera movement. The four phase images A_i which are calculated on the basis of the rendered images are depicted in the middle. Since phase images are used to calculate the resulting data such as the depth information, each of the phase images contributes to the simulation result (figure on the right). Thus, the implementation of motion blur is achieved by temporal super-sampling.

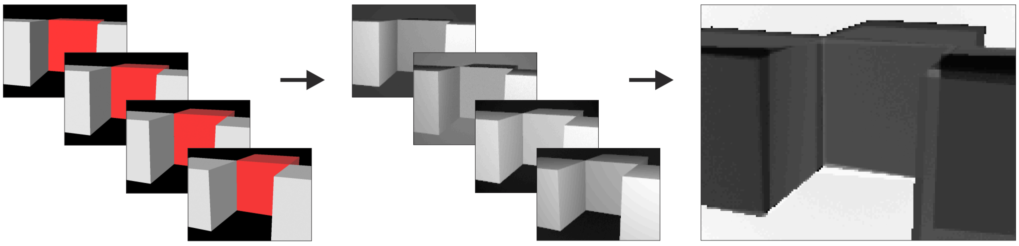


Figure 3.6: Temporal super-sampling in a dynamic scene. The figure on the left shows the rendered images of a dynamic simulation scenario which contain color and depth information. The computation of the phase images (middle) is based on the rendered images. The temporal super-sampling leads to artifacts in the depth values (figure on the right) which are calculated on the basis of the simulated phase images.

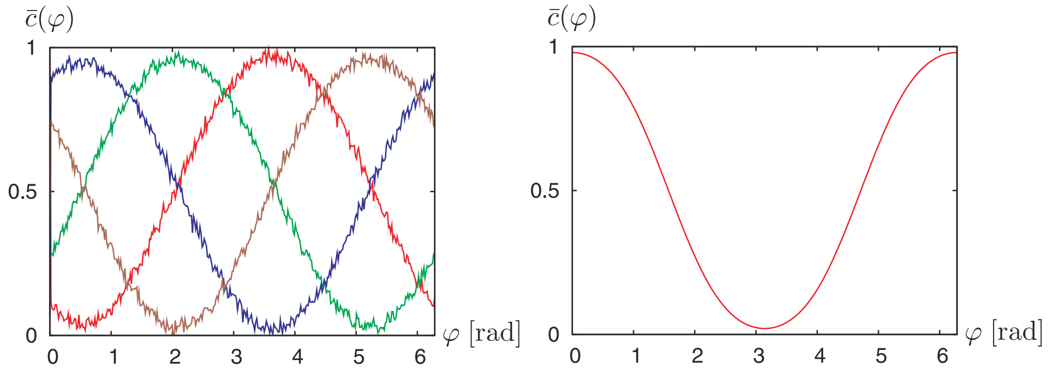
3.5.4 Correlation Function

The actual function cannot be measured exactly since the precise form of the correlation function c of a PMD device is influenced by numerous physical and electronic effects. Although a measurement of the signal response of a real PMD pixel can be achieved by directly illuminating it while applying all values of the internal phase delays τ_i, the simulation result differs from the real sensor behavior, i.e. the systematic deviation error is not reproducible (see Sec. 2.2.3). The simulation framework is flexible and configurable and thus offers a comfortable way of testing the influence of the correlation function's various implementations on the results. The values of the measured correlation function, which are displayed in Fig. 3.7(a), are written to a texture which afterwards is available for a texture look-up on the GPU. Thus, the correlation function's values are integrated into the calculation of the phase images A_i. However, the measured correlation function suffers from the static amount of noise which is present in the data as well as from the lack of the wiggling effect.

More realistic simulation results of the systematic deviation error are achieved by an approximation of the sinusoidal signal by various Fourier-modes:

$$c\left(\tau_i, \varphi_{\text{ideal}}\right) = \frac{1}{2} \cdot \left(1 + \cos(\varphi_{\text{ideal}} + \tau_i) - 0.041 \cdot \cos\left(3 \cdot \left(\varphi_{\text{ideal}} + \tau_i\right)\right)\right), \tag{3.12}$$

with $\tau_i = i \cdot \frac{\pi}{2}$, $i \in \{0, 1, 2, 3\}$. This signal form results from a phenomenological approach, the plot of Eq. (3.12) is shown in Fig. 3.7(b). The approach refers to the publications by Rapp et al. [RFHJ07, Rap07] who evaluated the higher harmonics in the signal.



(a) (b)

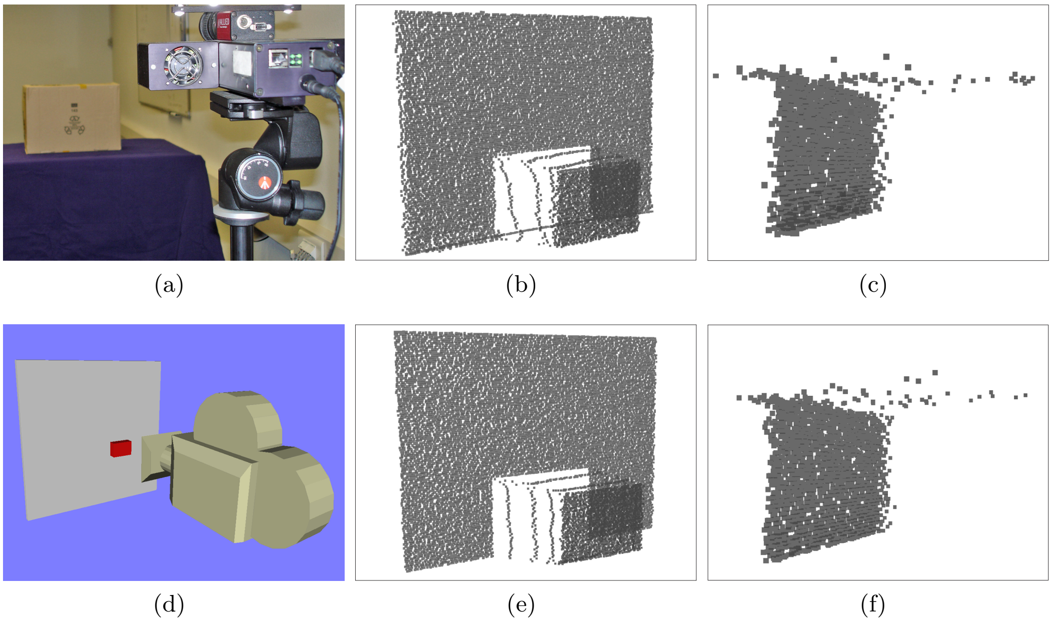
Figure 3.7: Correlation functions. The measured correlation function for all values of the phase shift φ and the internal phase delay τ_i is displayed in 3.7(a). The correlation function of Eq. (3.12) is shown in 3.7(b).

3.5.5 Integration of Sensor Noise

A simple noise model is used in the sensor simulation which results in the plausible behavior of the simulated sensor. The idea is to include the noise even on the level of phase images. To be more precise, a certain amount of randomly chosen noise values is added to the values of the phase image A_i. Due to the fact that the noise is a per-pixel noise it is added to the resulting phase images after super-sampling the higher resolution image. This means, A_i of Eq. (3.11) is modified in the following way:

$$A_{\text{noisy},i} = A_i + \alpha \cdot n_{\text{rand}} + A_i \cdot \beta \cdot n_{\text{rand}} \ , \tag{3.13}$$

with $\alpha, \beta \geq 0$ being the noise coefficients, $n_{\text{rand}} \in [-1, 1]$ is a random Gaussian number, and $A_{\text{noisy},i}$ is the resulting pixel's phase value A for a certain phase image i. The signal-to-noise ratio is described by $\alpha \cdot n_{\text{rand}}$ which results in a constant pixel noise. The term $A_i \cdot \beta \cdot n_{\text{rand}}$ models intensity related noise effects. Since it is tricky to generate random numbers on graphics hardware, a texture is sampled which contains random numbers. The influence of the noise is then controlled by adjusting α and β. Noisy phase images then result in depth images which contain also noisy measurements.


(a) (b) (c)

(d) (e) (f)

Figure 3.8: Comparison of flying pixel behavior of real PMD data and simulation results. The top row shows the data of the real PMD sensor while the bottom row displays the simulation data. 3.8(a) and 3.8(d) show the scene setup. The object's width is 0.34 m and its height is 0.24 m. All depth data is transformed into 3D world coordinates for visualization purpose. The flying pixels' behavior of the object with a short distance to the background is shown in 3.8(b) and 3.8(e). 3.8(c) and 3.8(f) display the effect of a large distance between object and background.

3.6 Results and Analysis

The simulation of PMD's phase images allows for the reconstruction of depth, intensity and amplitude data by using known demodulation formulas (see Sec. 2.2). In this section, the proposed ToF sensor simulation approach is evaluated and compared to real sensor data. Furthermore the simulation is applied to a series of experimental scenes.

3.6.1 Evaluation: Comparison to Real Sensor Data

The side-by-side comparison of the real data and the simulation data (at modulation frequency of 20 MHz) shows correspondence to a great extent regarding the sensor behavior. In the first scenario, the object is positioned in a short distance from the wall in order to demonstrate the flying pixels which tend to drift from the object towards the wall. The distance between object and camera sensor is 1.67 m. The

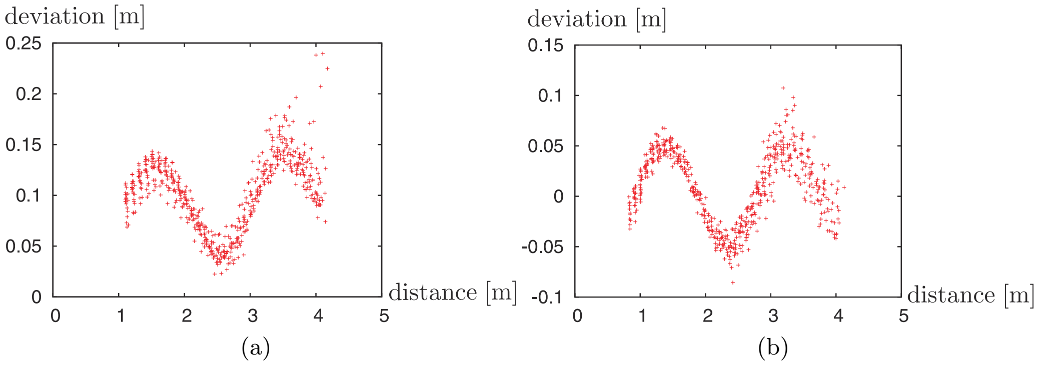


Figure 3.9: Deviation error. The graph 3.9(a) of a real PMD device with a resolution of 160 × 120 pixels is compared to the simulation data in 3.9(b). On the x-axis the measured distances are shown and the deviation error is marked on the y-axis (both in meters).

object itself is placed in front of the wall at a distance of 0.36 m (see Fig. 3.8(b) and 3.8(e)). In the second scenario, a large distance of 4.13 m between the object and the wall results in flying pixels which are located in front of the object and drift in the direction of the camera sensor (see Fig. 3.8(c) and 3.8(f)).

3.6.2 Wiggling Evaluation

The validation of the simulation data is quite elaborate. The wiggling errors of a real PMD device and the sensor simulation are displayed in Fig. 3.9. This simulation uses the Fourier-mode correlation function which is described in Eq. (3.12), since the correlation function which is based on a measured signal response (see Fig. 3.7(a)) results in slightly noisy data due to the specific measuring technique. The graph is plotted as a function of the measured distance for the image depth information of distances from 0.9 m to 4.0 m. Fig. 3.9(a) displays the real world sensor results and Fig. 3.9(b) shows this function measured by the sensor simulation. The simulation uses a spatial super-sampling on a higher resolution rendering with 640 × 480 pixels and with a down-sampling of 4 × 4 sub-pixels. The noise model uses $\alpha = 0.000164$ and $\beta = 0.047$. As the simulation results in relative intensity values, the data is linearly transformed on the y-axis which depends on the internal camera design with its latency time. However, the simulated data clearly indicate the typical deviation error of the real world sensor.

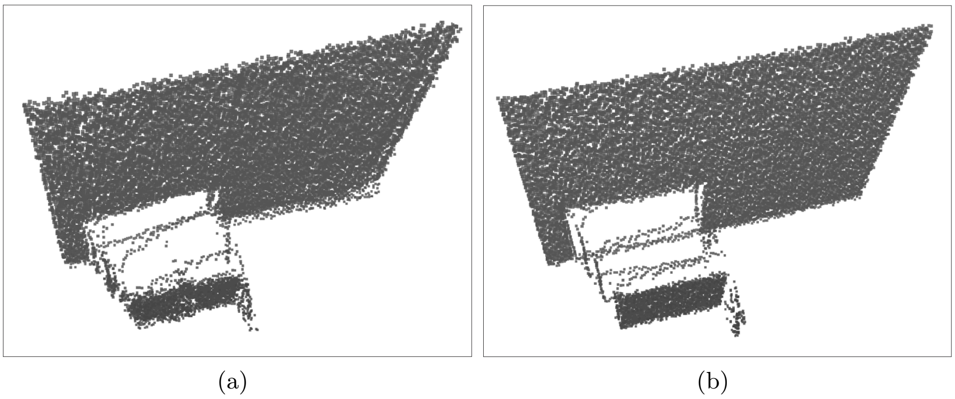


Figure 3.10: Comparison of depth data of a dynamic scene. 3.10(a) shows the scene in 3D world coordinate view acquired by a real PMD device. The scene displays a horizontal movement of a box object (which is shown in Fig. 3.8) from the left to the right. The simulated data during the same object movement is displayed in 3.10(b).

3.6.3 Motion Evaluation

Compared to images of static scenes which are relatively easy to verify, the validation of dynamic scenes is quite difficult because of the spatial and temporal artifacts which occur at the same time. The comparison of real dynamic PMD sensor data with simulated data on a quantitative basis is difficult. Therefore, only a qualitative comparison is presented in Fig. 3.10. This shows a moving box in front of a wall with a lateral velocity of approximately 2 m/s. The typical temporal artifacts visible in the simulated data on the left and the right end of the object are highly comparable to real sensor data as well as spatial artifacts.

3.6.4 Illumination: Point Light vs. Area Light

In Sec. 3.5 the theoretical sensor model of Sec. 3.3.2 has been implemented which supports a point light source at camera position. The advantage of this approach is that the graphics hardware needs less computation time for the illumination of the scene and thus the simulation feedback is achieved in real-time at interactive frame-rates. With regard to the model of the graphics card, the frame-rates are between 15 and 30 fps. The approach of a scene illumination with area lights as proposed in Sec. 3.3.3 has been also implemented. According to Eq. (3.8), an area light illumination is simulated which is approximated by up to 112 single lights which are located next to the sensor and are regularly distributed onto an area of 19×4.5 cm. This corresponds to the specification of the 19 k model by PMD. The distance from the camera and its illumination to the center of the scene is 2 m. The difference

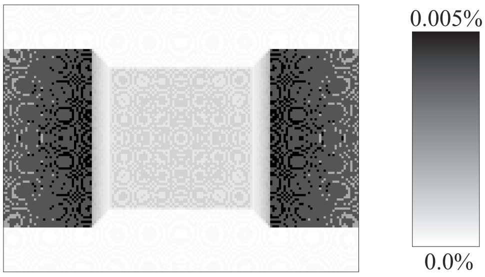


Figure 3.11: Area light vs. point light approach. Difference between both of the phase images A_0 generated by the area light approach and point light approach.

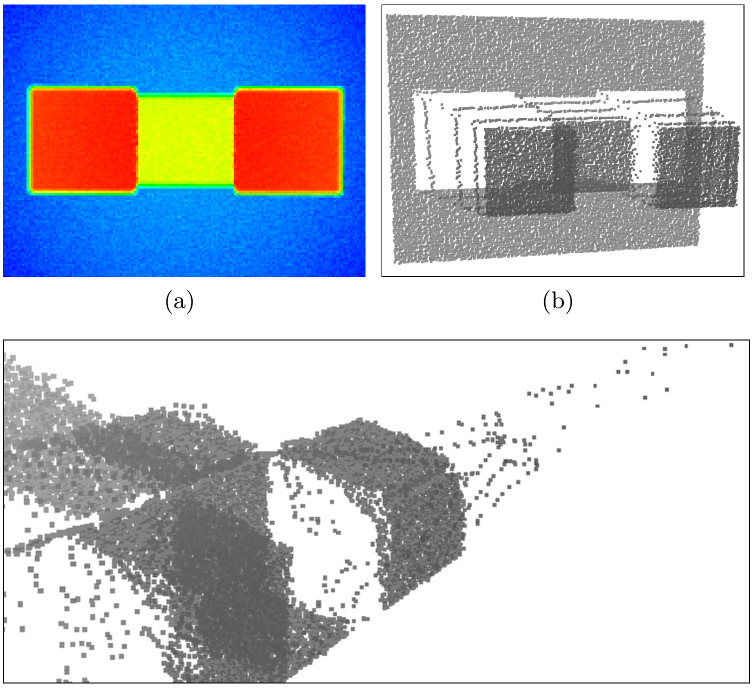
image of the two illumination models is illustrated in Fig. 3.11. The error values are calculated by subtracting the phase image A_0 produced by the simplified point lights approach from the phase image A_0 generated by the area lights approach. It should be noted that the black regions differ less than 0.005% from the area lights approach and the white regions contain even smaller differences. This means that the resulting values are almost zero and thus the two approaches produce nearly the same simulation results. However, the frame-rate of the area light illumination decreases to less than one frame per second because of the high computational costs.

3.6.5 Experimental Scenes

The results presented in this section use a spatial super-sampling which is computed on a scene which has 12 times the size of the simulation's target resolution. This corresponds to 1920×1440 pixels for a ToF resolution of 160×120 pixels. The down-sampling ratio for simulating spatial artifacts is adjusted to 40×40 sub-pixels. Furthermore, the correlation function is based on Fourier-modes (see Sec. 3.5.4) and the noise model uses $\alpha = 0.000035$ and $\beta = 0.01$.

Static and Moving Scene: Cubes Scene

The *cubes scene* displayed in Fig. 3.12 show three static cubes in front of a wall. The virtual ToF sensor is positioned directly in front of the cubes. Fig. 3.12(a) shows the calculated depth information coded in color (small depth values → red, large depth values → blue). A perspective view of the depth data transformed into 3D world coordinates is visualized in Fig. 3.12(b). Here, the flying pixels located between the cube's front faces and the background wall become visible. Fig. 3.12(c) shows the same scene during a movement of the ToF sensor. The temporal artifacts based on the motion blur approach (see Sec. 3.5.3) are indicated by a tail of flying pixels drifting towards the sensor.

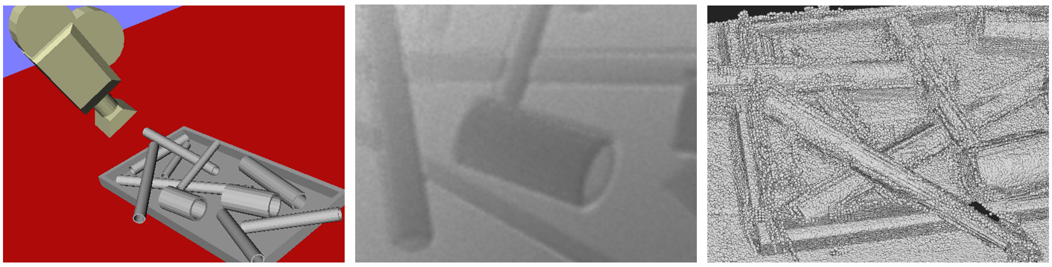


(a)

(b)

(c)

Figure 3.12: Simulation results. The simulated depth information of a PMD sensor is displayed in 3.12(a). In 3.12(b), the scene is shown in 3D world coordinate view. Its calculation is based on simulated phase images. The temporal artifacts of a PMD sensor become visible during the rotational movement of the ToF sensor in 3.12(c)).



(a)

(b)

(c)

Figure 3.13: Tubes box scene. The scene simulates a robot's bin picking application. 3.13(a) displays the scene setup of the simulation framework. A single simulated depth image (noise and motion artifacts are noticeable) is shown in 3.13(b). The accumulation of all depth data of the simulation sequence comprises $\sim$ 7 mio 3D data points (160×120 pixels ×360 frames) as visualized in 3.13(c).

Simulation Sequence: Tubes Box Scene

The *tubes box scene* as shown in Fig. 3.13 consists of a box with multiple tubes of various sizes inside. This scene simulates a real world bin picking application as specified in the Lynkeus 3D project (see Chap. 1 for a brief description regarding the project): a robot autonomously explores the environment and empties the box as demonstrated in [FHK*10] [1]. During the simulation sequence the virtual ToF sensor moves across the scene and acquires the data from various angles. Therefore, various positions and orientations of the sensor are configured in a key-frame sequence. The processing of the simulation data is done in real-time during the playback of the simulation sequence. The full dataset counts 360×4 phase images and comprises the corresponding depth, amplitude and intensity data. In Fig. 3.13(c) all depth images have been accumulated into a consistent model. The motion blur artifacts are clearly visible especially at the object boundaries, i.e. the tubes.

Multi-Sensor Setup: 3D Data Acquisition System

The proposed sensor simulation framework handles the simulation of multiple sensors simultaneously. Thus, not only a single ToF sensor can be placed into a synthetic scene but also multiple sensors at various positions. In Fig. 3.14 a general *3D data acquisition setup* for ToF data is proposed. A car wash system is simulated for the acquisition and 3D reconstruction of a driving car as published for a real world car reconstruction system in [HLK13b] as part of the German Research Foundation (DFG) project *Imaging New Modalities* [2].

The scene consists of several walls and a floor in order to represent realistic background data. Three ToF sensors are placed onto an arch where the objects are passed through (see Fig. 3.14(a)). In detail, the positions of the sensors are chosen with regard to an optimal setup for objects which move through the arch: left-hand side, right-hand side, and the top of the object are acquired at a time. Fig. 3.14(c) - Fig. 3.14(e) show the data acquired from the respective ToF sensors. Motion artifacts are visible at the object's contours while it is moving through the acquisition arch. In Fig. 3.14(b) all depth images from the three sensors have been accumulated into a common model. Motion artifacts are strongly highlighted by the spread data points. However, the illumination interference from multiple sensors is not taken into account by the sensor simulation. This means, each sensor simulation is considered apart from other sensors. A sensor does not interfere with the active illumination of a second sensor as in real world.

[1] For more information about the accumulation of the simulation data into an environment model see the *hierarchical volumetric data structure* approach in Chap. 4.

[2] Research Training Group - Imaging New Modalities, `http://www.grk1564.uni-siegen.de`.

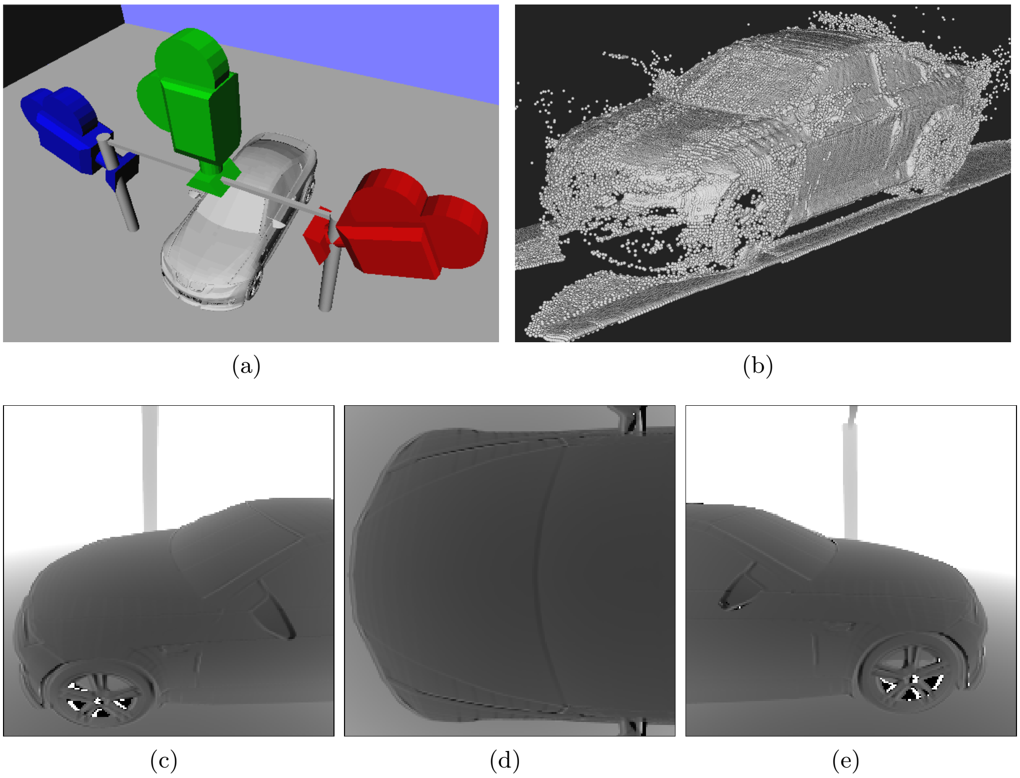

(a)
(b)
(c)
(d)
(e)

Figure 3.14: Data acquisition system with multiple sensor devices. The simulation setup is shown in 3.14(a). In total, $\sim$ 12.5 mio 3D data points (200×200 pixels $\times$ 3 sensors $\times100$ frames) are simulated. All 3D points are accumulated and rendered into a common coordinate system as visualized in 3.14(b). Points which are spread off from the car are caused by the car's motion. Individual simulated depth images of the three sensors of time stamp 43 are displayed in the bottom row showing the left, top, and right-hand side of the car.

The multi-sensor simulation as well as the arch-based acquisition setup for the driving car simulation have been developed in collaboration with Thomas Hoegg, Christ-Elektronik GmbH in Memmingen.

3.7 Discussion

In this chapter a fully equipped simulation model of a ToF sensor has been presented. The simulation is based on a physical model and resembles the sensor's phase image generation. Relevant sensor effects like flying pixels, wiggling and motion blurring have been reproduced by applying this model in connection with the use of spatial and temporal super-sampling techniques. Furthermore, a noise model is integrated into the sensor simulation.

The proposed algorithms are incorporated into a sensor simulation framework which imports 3D scenes and provides tools for object and camera manipulation as well as simulation sequence. The framework is hardware accelerated in order to support interactive simulation feedback, making use of the programmability of GPUs. The resulting data is comparable to real world sensor data exposing the results of dynamic scenes which contain camera and object movements.

The proposed simulation concept is based on research with regard to ToF sensors, in particular PMD technology. Nevertheless, the implementation of other simulation concepts as well as lessons learned from the current model can be applied continuously to the simulation system, e.g. integration of new correlation functions as well as applying sophisticated noise models.

However, the simulation approach has limitations in terms of the utilization of standard rasterization techniques of the GPU. Thus, advanced illumination models are not supported directly which means that ToF sensors' multipath-interference is not covered by the current sensor simulation model. The advanced illumination would require the implementation of effortful ray-tracing algorithms, e.g., and thus real-time constraints are hard to accomplish. The reader is referred to the offline simulation approach by Meister et al. [MNK13] for the simulation of such effects. Furthermore, the current simulation approach deals only with physical units for the final depth maps. Intermediate results such as the computed phase images are normalized. Lambers et al. [LHK15] enhance the current simulation model by using physical units throughout the simulation process. Additionally, their sensor model handles the evaluation of chip layout variants as well as lens parameterization accounting for vignetting effects.

Chapter 4

Hierarchical Volumetric Data Accumulation

The temporal accumulation of sensor data into a consistent data representation is called environment modeling and is of importance for subsequent algorithms which need information about the environment's state. In this chapter a volumetric data structure called Dynamic Volume Trees is introduced for the purpose of environment modeling. An adaptive hierarchical data structure is proposed being updated from depth sensor data in real-time. Its online capability is achieved by realizing both the hierarchical data structure as well as the manipulation of the structure solely on the GPU. The data is organized in a hierarchical kd-tree-like structure which provides a compact storage of multi-resolution volumes with no redundant memory consumption. Boolean operations are supported, i.e. sub-volumes can be efficiently merged and removed with nearly arbitrary resolution.

The current chapter is structured as follows: a motivational part including challenges and research objectives is given at the beginning. The overview of the related work is found in Sec. 4.2, followed by a conceptual overview of the dynamic volumetric tree structure (Sec. 4.3). A description about the implementation is presented in Sec. 4.4. Then, in Sec. 4.5, the approach is analyzed and evaluated presenting applications as well as experimental scenes featuring a volume drawing application and depth map accumulation examples. Finally, this chapter closes with a discussion about the volumetric data structure in Sec. 4.6.

Publications The approach about Interactive Dynamic Volume Trees on the GPU has been presented in [KCK09]. Furthermore, the application of the environment model utilizing the presented data structure within a robotic bin picking application has been published in cooperation with the German Aerospace Center, Inst. of Robotics and Mechatronics, in [FHK*10].

4.1 Motivation

The presented approach about the new hierarchical volumetric data structure for data accumulation and environment modeling has been developed in the context of the Lynkeus 3D project. The project's environment model demands for the following requirements: a model with a flexible spatial resolution is required since various application scenarios cover a variety of different setups (indoor: $4 \times 6.5 \times 6.5$ m with

0.005 m accuracy; outdoor: up to 2 km^2 with 0.08 m accuracy). Also the exploration of unknown environments needs to be ensured. This means, the environment model is set up starting from scratch by accumulating multiple depth maps into a consistent model. Furthermore, dynamically changing environments are supported since objects which are added to or removed from the scene are represented by the accumulated model in real-time. Finally, subsequent processing algorithms are not restricted to the use of the model's surface representation only. Rather, the project demands for volume carving which means that a full voxel model is required.

The accumulation of data acquired from the environment and fed into a consistent model belongs to the area of 3D reconstruction algorithms which have a long tradition in computer graphics. Unlike methods that focus on reconstruction from a complete set of 3D points [HDD*92, KBH06], online methods require the *fusion* of many overlapping depth maps into a single 3D representation that is continuously refined. A challenging subproblem resides in the real-time capability of these algorithms. One of the first real-time reconstruction systems for hand held scanning was demonstrated by Rusinkiewicz et al. [RHHL02], who still need an offline step for accurate surface reconstruction. The volumetric fusion method of Curless and Levoy [CL96] is also not applicable to the requirements stated above since their system lacks on scalability and real-time performance.

In sum, the requirements lead to the development of a new fast and dynamic data structure. Such data structures are essential in many fields related to sensor data processing. Algorithms in areas such as ray tracing, collision detection, and volume data processing, e.g., require exhaustive memory and computational resources. Thus, these applications organize their data mainly in hierarchical structures for fast and efficient traversals for data sampling and manipulation. Tasks such as real-time data processing and accumulation request for the development of parallel algorithms as supported by today's GPUs. While previous data structures like kd-trees were built by the CPU and finally transferred to the GPU memory for traversal and sampling tasks [FS05], today's techniques generate even complex data hierarchies directly on the GPU [ZHWG08]. This way, expensive latency for copying data structures is avoided.

Challenges Ensuring the data structures' online capability is the main challenge. About 60 million depth samples per minute need to be handled in real-time since in the Lynkeus 3D project a ToF chip has been utilized with a lateral resolution of max. 200x200 pixels at a frame-rate of 25 Hz.

Objectives This chapter focuses on the development of a hierarchical volumetric data structure with online processing capabilities. The structure is built and managed by the GPU for the reason of real-time performance. Furthermore, the support of merging and subtraction of sub-volumes is essential.

4.2 Related Work

This section focuses on approaches about dynamically changing data structures rather than on known accumulation systems, such as [CL96] and [RHHL02] since existing systems do not meet the project's requirements of real-time 3D reconstructions as stated in Sec. 4.1. Data structures such as kd-trees, grids, or bounding volume hierarchies (BVHs) are especially required in areas of high performance algorithms but the relative effectiveness of these structures varies tremendously depending on the application.

Any approaches in the context of dynamic data structures can be classified into three categories. The first category is characterized by reconstructing the entire data structure whenever the scene has changed, e.g. Shevtsov et al. [SSK07] rebuild a kd-tree every frame. The primary challenge of this approach is that it can be expensive to rebuild the full data structure especially for large scenes and sophisticated structures. Secondly, if the majority of the scene remains static, a significant coherence exists among subsequent frames. A full reconstruction of the data structure is omitted by updating only the parts which have changed, as proposed by Larsson and Akenine-Möller [LAM06], for instance. Finally, the last category contains approaches which are applicable for scenes consisting of a large set of (rather) static objects. The idea is to pre-compute an acceleration structure for the static parts and to separate it from the dynamic geometry (see [LAM01, WBS03]).

One of the early approaches proposing a dynamic data structure for ray tracing applications has been presented by Reinhard et al. [RSH00], which allows the insertion and the deletion of objects in constant time. The structure is based on hierarchical grids which entirely ran on the CPU. While some methods were examined in order to avoid the performance limiting reconstruction of such structures [LAM01], first approaches started to use the GPU for graphics hardware accelerated data structures [CHH02] but were still limited due to architectural constraints. Modern approaches process hierarchical acceleration structures such as kd-trees [FS05, HSHH07, HMHB06] and BVHs [GPSS07]. However, these approaches essentially built their structures on the CPU, and do not support any GPU-based updates. The efficient GPU-based data structures proposed by [CHL04] and [LKHW05] still use the CPU as a memory manager instance. In fact, Purcell et al. [PDC*03] were the first to describe an entirely GPU-updated, dynamic sparse data structure. Newer GPU approaches, build ocrees for point cloud handling [ZGHG08], and construct kd-trees [ZHWG08] or use BVH [LGS*09] for ray tracing.

More recently, interesting work has been published in the area of hierarchical data structures with regard to environment modeling: octomap [HWB*13] implements a probabilistic 3D map representation of the environment which enables robots for collision free path-planning since voxels are classified into unknown, free and occupied categories. The data structure is completely CPU-based. Depending on the length of the measurement beams the integration of a single depth map takes approximately 1-7 seconds. In contrast to this, the approach presented in this chapter manages complete hierarchical volumes on the GPU and introduces further processing capacities, e.g. merging and subtraction of sub-volumes at interactive frame-rates.

4.3 Overview of Concept

In the remainder of this chapter, the proposed dynamic data structure is called Dynamic Volume Tree (DVT). The DVT resides completely on the GPU and it is only updated in those regions where changes have been made. The concept of DVTs involves a representation suited for the GPU and a set of operations which allow to modify the structure in an efficient way. In the following, the DVT is introduced by providing a conceptual overview of the method.

4.3.1 Tree Topology

The DVT subdivides space similar to a kd-tree. However, the splitting scheme is uniform and axis-aligned: each splitting plane halves the previous block into a pair of two equal-sized sub-blocks. Subdivision is repeatedly performed along the x, y and z-axis, where the depth level of the tree may vary locally. By using a constant bounding box this spatial division scheme assigns a fixed subspace (voxel) to each node in the binary tree. In principle, this rule can be applied to a spatial subdivision in arbitrary dimensions (see Fig. 4.1).

As mentioned before, the representation of geometry is of interest. Thus, only the leafs of the tree, which are defined as voxels, contain spatial information and values are stored to mark the object's interior (0) and exterior (1) as leaf attributes. To improve the accuracy float values can be stored within the range of $[0, 1]$ specifying the distance of the voxel to the respective surface of the object (see Sec. 4.4.5 for rendering aspects). For reasons of simplification, in the following binary values are assumed to be stored in the leafs.

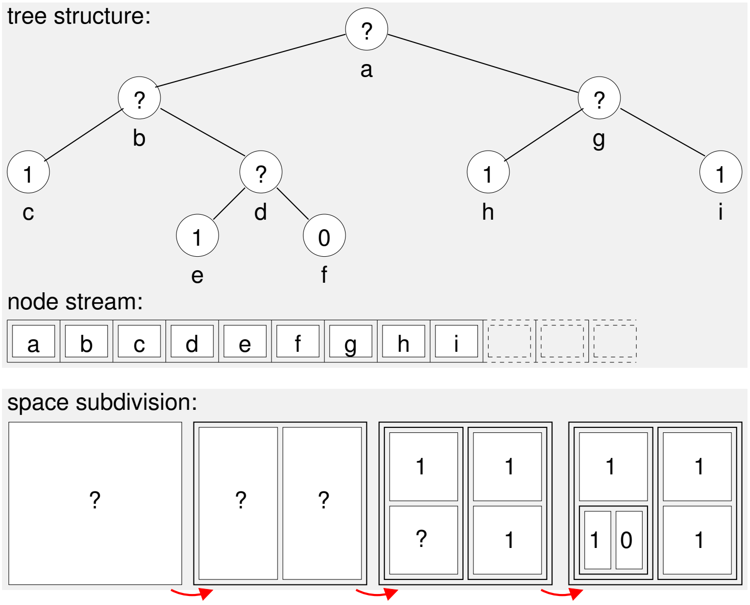


Figure 4.1: Tree structure. An example (in 2D) consisting of a specific tree and its corresponding spatial representation. The node stream results from a pre-order tree-traversal.

4.3.2 Tree Manipulation

The modification of the DVT can be seen as an operation which changes the structure and the values in the tree. Given a current tree and an input tree, which specifies an implicit geometry, e.g. a sphere, or a polygonal mesh (see Sec. 4.4.4 for details on hierarchical geometry rasterization). The task is to “insert” the new tree into the current one. Two basic boolean operations are supported, i.e.

Merge The input geometry is merged with the geometry represented in the current tree; this relates to a union operation.

Subtract The input geometry is subtracted from the geometry represented in the current tree; this relates to the relative complement.

It should be noted that the processing of leafs is sufficient, i.e. both operations can be realized by writing new leafs into the current DVT-structure. Merging and subtracting are realized by reading *1*s from the input stream and writing them to the current DVT as *1*s and *0*s respectively in case of subtracting geometry (see Fig. 4.2.) The resulting DVT has the same topological structure in both cases, but it may contain redundancies. An optimization pass is proposed which removes redundant subtrees (Sec. 4.4.3). No restrictions are put on the voxels of the input tree with regard to their locations within the hierarchy which results in voxels of various sizes.

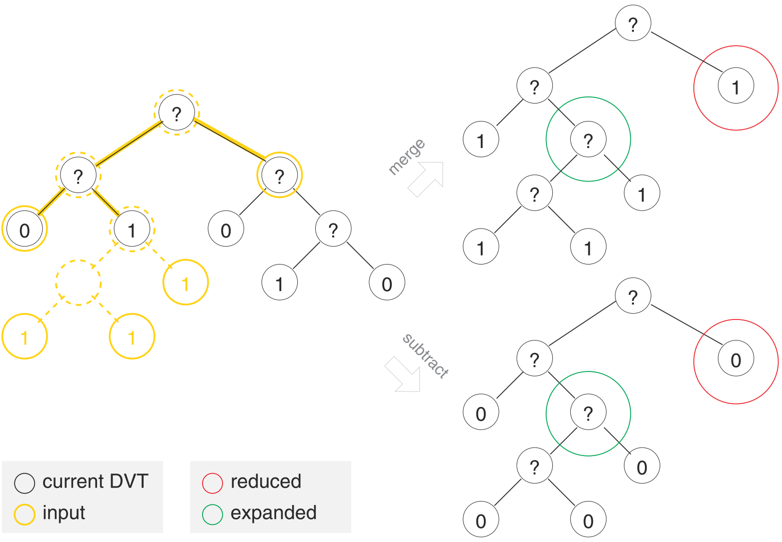


Figure 4.2: Merge and subtract. An example showing the result after a merge or subtract operation. The yellow-marked nodes form the input.

This means that the locations in the tree which may be affected by a modification process are scattered arbitrarily. Thus, depending on the topology of the current tree, it may be necessary to expand or reduce complete subtrees in order to write the respective input voxels into the tree (see Fig. 4.2). This is achieved by performing the tree modifications iteratively, namely by adding and removing node levels locally. The iteration terminates, if no modifications are pending and if the node stream of the current DVT does not change anymore.

The iteration is split up in three data-parallel passes: the *mark*, the *restructure* and the *remap* pass:

Mark Pass Nodes are selected within the current DVT which need to be expanded or reduced (see Sec. 4.4.2). The depth of these marked nodes corresponds to the resolution of the respective input voxel. This information needs to be stored together with the input voxels and will later be referred to as the *target depth*.

Restructure Pass This pass interprets the marked nodes and adds or removes them (see Sec. 4.4.2). Both operations handle subtrees despite the fact that one single level is added/removed per iteration. Adding subtrees is realized by marking newly added nodes in the next mark pass; removing subtrees is realized by removing subsequent children during the next restructure pass.

Remap Pass Finally, a correctly pointered tree structure is ensured by resetting all pointers. This involves a two-pass routine using a temporary look-up table (see Sec. 4.4.2).

Algorithm 1: Overview

```
do
  // pass 1 (mark):
  for each input voxel
    if  appropriate node existent in tree:
      update value
    else if  subtree needs to be expanded/reduced:
      mark node for pass 2

  // pass 2 (restructure):
  for each marked node $n$
    if  $n$ is marked for expansion
      add children
    if  ($n$ marked for reduction)
          or ($n$ has invalid parent)
      set $n$'s childrens' parent ptrs. invalid
      remove $n$

  // pass 3 (remap):
  for each node $n$:
    add node to look−up table
  for each node $n$:
    read pointers from look−up table

while (∃ marked nodes)
        or (stream has been altered)
```

The overall modification process is described in the algorithmic overview given in Alg. 1. A more detailed and implementation-driven explanation of the single passes involved in the algorithm, including technical aspects, is given in the next section.

4.4 Implementation

This section provides an implementation specific view of the strategies presented in the previous section. Alg. 1 is explained in detail including technical aspects which are necessary to realize the data structure on the GPU. A shader API-oriented terminology has been chosen as the DVT method fits perfectly to the use of geometry shaders.

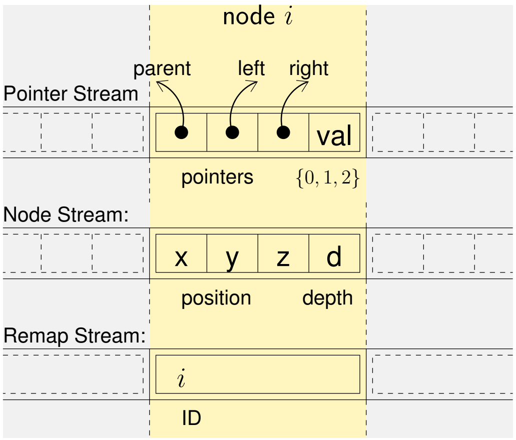


Figure 4.3: Data streams. The tree structure is represented by streams. The cells belonging to one node are marked yellow.

4.4.1 Storage of Dynamic Volume Trees

The tree is stored in three vertex streams in graphics memory. These streams can be propagated through the graphics pipeline in a single render pass. The layout of the node information is shown in Fig. 4.3.

Pointer Stream This stream contains the pointer structure, i.e. the node's parent and children as node IDs, and the node's value. A pointer can be set to -1, meaning that it is invalid, e.g. the parent in the case of the root and the children in the case of leaf nodes do not exist.

Node Stream This stream provides a node's spatial position and a depth value implying its level in the hierarchy. This stream information can be used for an efficient rendering without always being forced to determine each node's position.

Remap Stream The third stream is used as a temporary container for coding the look-up table which is necessary when remapping pointers.

These streams are stored in `gl_Position`, `gl_TexCoord[0]` and `gl_TexCoord[1]` respectively. During processing, stream information can be transferred to a texture object and back by using the pixel buffer extension. This allows the algorithm to use stream processing (e.g. using a geometry shader) and rendering for scattering operations. This strategy is used to read the pointer stream through texture-fetching. To achieve a fragment-based process, the texture object is double-buffered, thus being accessible for writing operations. From now it is assumed that all streams can be read and written as streams *or* textures, assuming that the data has been transferred appropriately before processing. The nodes of the binary tree are stored in *pre-order*. This ensures that the root node is located at the beginning of the stream and does not change its location when modifying the tree.

4.4.2 Modification of Dynamic Volume Trees

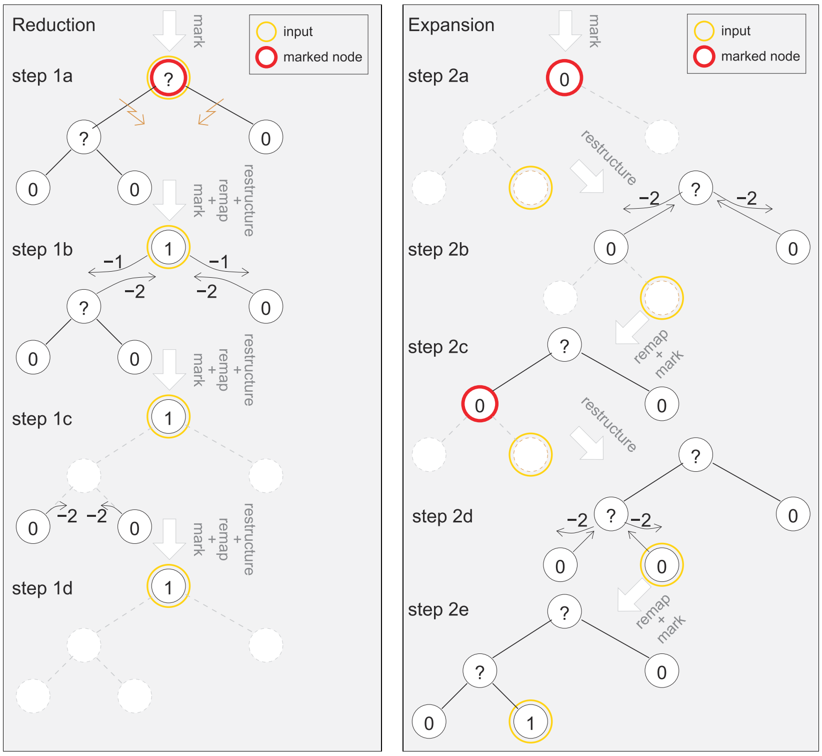
The *write* operation is the central algorithm of the DVT approach. It solves the problem of adding and removing nodes when merging a new input volume to the tree. As mentioned previously, depending on the location of input voxels (e.g. leaf of the input DVT) the tree structure must be expanded in regions where its depth is insufficient, whereas it must be reduced at oversampled regions in order to eliminate superfluous nodes. Alg. 1 contains the steps necessary to realize modifications in parallel. The approach is reformulated to meet the specifics of the GPU, including a workflow which can be realized by using stages of the graphics pipeline.

The workflow consists of a loop with three passes: *mark*, *restructure* and *remap*. Not forgetting that the input consists of a stream of input voxels together with their target depths. The *mark* pass selects a set of tree nodes which are affected by the input by writing special marker values into the pointer stream. The marked stream is then processed in a geometry shader pass: the restructure pass. Finally, the *remap* pass takes care of correct pointers. The three passes are repeated until nothing has to be changed any longer, i.e. if no node is marked and no node is added or removed.

Mark Pass

The idea of the *mark* pass is to select tree nodes which are affected by a write operation. This selection is realized by a scattering approach which is implemented in a vertex shader. In order to simultaneously read from and write to the pointer stream, a double-buffered access is necessary. As only a subset of the stream nodes is changed, the output as well as the input must be initialized to the current state of the pointer stream. The actual marking is done using a scattering approach implemented in a vertex shader: The tree is traversed for each input voxel until the target depth or a leaf node is reached. There are three cases where nodes are marked when a voxel, i.e. leaf node of the input stream at a given level (“target level”), is processed:

1. If a leaf node at target depth is reached in the current DVT and the value of this leaf node is different from the value of the input voxel, then the input voxel’s value is written into the leaf node of the current DVT.
2. If in the current DVT a leaf node is reached at an insufficient depth, i.e. the DVT needs to be expanded, then the node is marked.



(a) (b)

Figure 4.4: Tree reduction and expansion. In 4.4(a) the reduction of DVTs is illustrated by removing superfluous nodes below the marked intermediate node. The example in 4.4(b) displays the expansion of DVTs as new nodes are added.

3. If an intermediate node is reached in the current DVT at the target depth, i.e. the tree needs to be reduced, then the node is marked.

In all other cases the current vertex is clipped by moving it out of the output stream and no value is written. Nodes are marked by subtracting a high number from the node's value. Such nodes are later unmarked by adding back the same high number to the value.

The Restructure Pass

The *restructure* pass processes the node stream in a geometry shader. Nodes are removed and added where it is necessary. Fig. 4.4 shows how removing and adding of nodes works by means of two examples. Both operations use a two-pass strategy: turn pointers to a special value (in the following the value -2 is used) and update them in a subsequent pass.

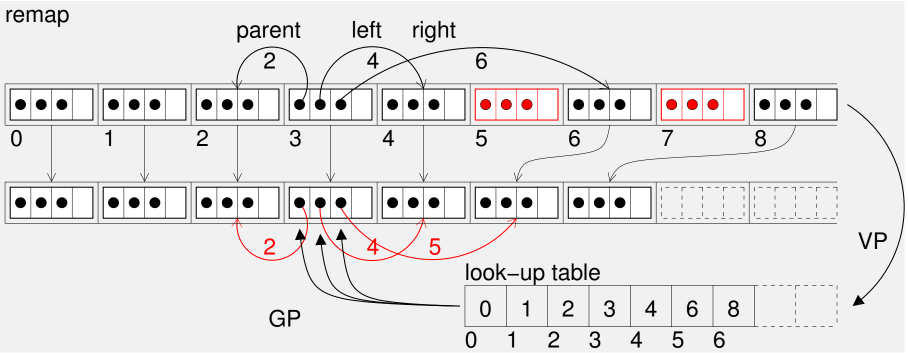


Figure 4.5: The remap operation. Two nodes (marked red) are removed and the respective pointers are updated.

The root of subtrees which have to be removed in the current DVT is indicated by marked intermediate nodes. Starting from the marked nodes the reduction is performed iteratively. Since the subtree's root node remains in the tree as leaf, its children are made invalid, which is indicated by setting their parent pointers to -2. Invalid nodes can be removed safely in the next restructure pass. By removing these nodes in the next pass, their children are transformed into invalid nodes again. Essentially, this is realized by the iteration. In the example given in Fig. 4.4(a), two levels of nodes need to be removed underneath the marked intermediate node. The iteration stops as soon as changes do not occur anymore (after step 1d). Nodes are added at marked leaf nodes which imply insufficient depth. Fig. 4.4(b) illustrates how children are added below the marked node. The geometry program emits two new vertices which, due to the pre-order sequence, are located directly after the current node. The parent-pointers of the two children nodes can be set directly as the parent is exactly the node processed currently in the shader. On the other hand, the newly added nodes do not possess a valid ID yet since the parallel stream processing does not allow to create a globally unique ID sequence. Thus, the parent pointers of the new nodes are temporarily set to -2. The next *remap* pass takes care of remapping the pointers and updating those pointers set to -2. This is an easy task since the children are located next to the parent node in the stream (at offsets $+1$ and $+2$).

The Remap Pass

Each time the size of the node stream is altered, the pointer structure needs to be updated accordingly. The problem is that moving nodes to other stream locations implies that all pointers to these nodes must be redirected (see Fig. 4.5) which can be done in parallel using the following scattering approach. Assume that each node possesses an ID corresponding to its location in the stream. One node moves from an old ID to a new ID if the stream is modified:

1. Render new node IDs as vertices into the remap stream which serves as a look-up table in the subsequent step. The output location of each vertex is determined by the old ID.
2. Process the new stream and perform the following operation: update the pointers (parent, left, right) for each node by reading the remap stream.

4.4.3 Redundancy Optimization

After a write operation on the tree there are cases which can occur where two children or even a whole subtree hold the same value (compare with Fig. 4.2). These cases can be collapsed without changing the represented volume. Accordingly, the *collapse* operation solves the problem of joining two leaves at the same depth with equal value. The operator is implemented in a geometry pass which detects superfluous successors. The parent node is set to the value of the successors and the successors themselves are removed. In order to speed-up the shader, a window of four adjacent nodes is processed by using the adjacency capability of OpenGL in connection with line strips. This avoids texture fetch and ensures best caching behavior. After *collapse*, a pointer remap is performed according to Sec. 4.4.2.

4.4.4 Generation of Dynamic Volume Trees

Initially, in Sec. 4.3 it is assumed that the input geometry is already given as DVT. Even though the rasterization of 3D objects is a field by its own, in the following the generation of DVTs for objects is outlined briefly.

Voxelization of Polygon Meshes A simple, yet potentially exhaustive technique is to rasterize an object on a certain tree level and apply the optimization technique described in Sec. 4.4.3 (see also [FL00]). Possibly, this requires a huge amount of data and time since the DVT has to be fully instantiated at the predefined level. Similar to [WE98] clipping planes as well as front- and back-face rendering is used in order to generate slices which are rendered to a 3D texture. The result is a binary solid voxel model, the quality of which is depending on the volume resolution of the 3D texture.

Hierarchical Rasterization A more effective approach directly works in a hierarchical manner. The hierarchical rasterization provided in Alg. 2 creates a temporary node stream containing a hierarchical description of the object to be written into the

Algorithm 2: Hierarchical rasterization

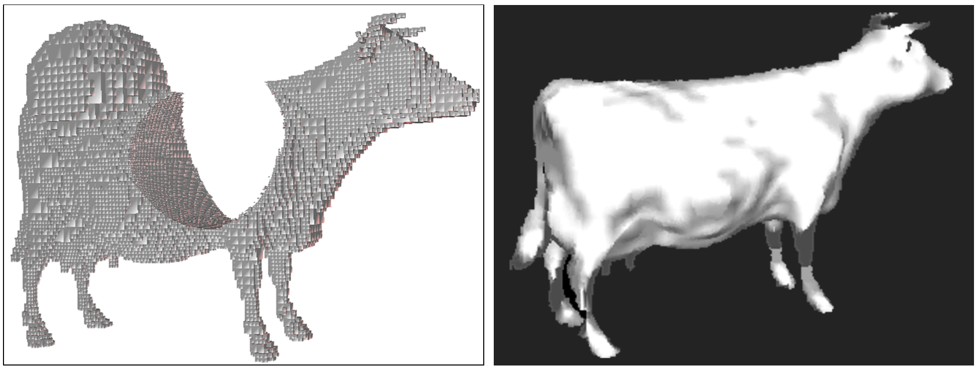
```
initialize a stream with root node of value 1
for each depth $d \in \{0, \ldots, d_{\text{target}}\}$
  for each non-*finished* stream node $n$
    if  $n$ is outside the object:
      discard $n$
    else if  $n$ is inside the object, or $d = d_{\text{target}}$:
      mark $n$ as *finished*
    else if  $n$ crosses the object's surface
      subdivide $n$ by adding 2 children to stream
process the new stream according to Sec. 4.3.2
```

global structure. The stream is constructed in a loop where each iteration adds one depth level. Thus, the strategy can be seen as a refinement of the hierarchy from coarse (only the root node) to a specific target depth which is given as input parameter. A special marker is used to identify nodes which do not need to be refined again, so-called *finished* nodes. It has to be noted that this algorithm heavily depends on the inside/outside/boundary detection for a given geometry in order to be rasterized hierarchically. For implicit geometries, this functionality can easily be realized using the inherent distance measure, whereas polygonal meshes require more sophisticated techniques in order to determine this classification.

Sub-Voxel Accuracy As mentioned in Sec. 4.3.1, voxels are capable of storing float values which specify the distance of the voxel's center to the exact object's surface location. The way of computing this distance depends on the type of geometry, e.g. for implicit geometries it is rather straight forward. To keep the memory consumption low, this is applied only to voxels close to the surface, i.e. within a narrow band. This leads to a representation of the object as a signed distance field and allows for rendering of the surface at sub-voxel accuracy, e.g. when ray casting is performed.

4.4.5 Rendering Taversals

In order to access the information of the data structure the DVTs are traversed down to the leaf nodes since these voxels contain the actual spatial information. The information is visualized using various rendering approaches. In this section the visualization based on primitives and ray casting are proposed. Due to the hierarchical structure, both approaches implicitly implement an empty-space skipping similar to [RSK05].



(a) (b)

Figure 4.6: Rendering of leaf nodes. In 4.6(a) the rendering based on primitives is shown (Fig. courtesy of [KCK09]). The various sizes of the primitives represent the respective levels of the tree hierarchy: the smallest primitive size is defined by the minimum voxel size of the tree. 4.6(b) illustrates the tree structure rendered by ray casting.

Primitive Rendering This rendering approach uses the graphics hardware's geometry shader. The node stream is processed by a geometry program which renders a geometric primitive, e.g. a cube or a screen aligned quad, for each voxel which is marked as the object's interior (see Fig. 4.6(a)). The size of the rendering primitive is calculated by identifying the voxel's position in the DVT, i.e. the depth level of the node stream. If a voxel's orientation is available (e.g. by pre-processing the input data and calculating normals) then rendering of surface splats is useful in order to represent the surface's orientation. The geometry shader's processing speed is affected by the increasing number of nodes which can be reduced by the redundancy optimization. The approach skips the empty space as the geometric primitives adapt exactly their spatial regions corresponding to the voxel size. Its complexity scales with the number of nodes n which are to process with $O(n)$ rather than with the number of pixels of the output image.

Ray Casting Visualization based on ray casting implements the tree traversal as well as its interpretation as a fragment program (similar to [HSS*05]). The DVT is traversed for each pixel of the target image. All the voxels along a viewing ray are traced for further evaluation. If a voxel does not fit the criterion for the object's interior, it is skipped and the next voxel along the ray is processed. The first hit of a voxel of the object's interior represents its surface. Its complexity with $O(p)$ depends on the number of pixels p of the resulting image which initiate the DVT-traversals along the viewing rays.

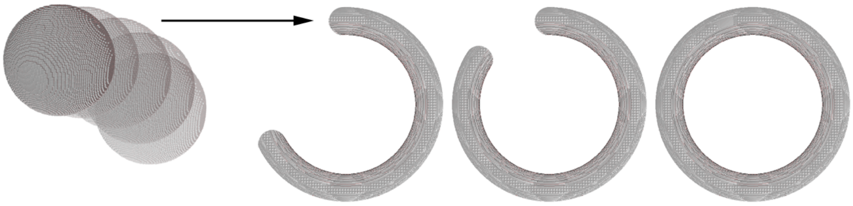


Figure 4.7: Performance analysis scene: a 3D torus is drawn automatically by inserting multiple solid spheres into the tree data structure.

The assignment of float values as leaf-attributes (see *sub-voxel accuracy* in the previous section) is useful to improve the visual appearance of the tree's rendering. The information can be applied to the calculation of shading algorithms, e.g. the computation of gradients at sub-voxel level improves the phong shading (see Fig. 4.6(b)).

4.5 Results and Analysis

In this section the proposed volumetric data structure is analyzed in terms of performance and scalability. The application of DVTs is presented by processing various scenes demonstrating its online merging capabilities.

4.5.1 Performance Analysis

A test scene is processed by the volumetric data structure in order to analyze the performance as well as the memory consumption. The test is executed on an Intel Dual Core 2.67GHz CPU with a Nvidia GeForce GTX 280 (1024MB) graphics card. The maximum depth level for the number of nodes is set to 2^{21}. The test setup automatically draws a fine grained, solid sphere which occupies around 98.000 nodes in the data structure. Then the sphere is moved in a circular shape with a step-width of 3°. After each step the sphere is written to the data structure which finally results in a 3D torus (see Fig. 4.7). The torus is drawn in four variations:

1. Drawing with no redundancy optimization and no hierarchical rasterization (*NO*),
2. drawing with redundancy optimization enabled and no hierarchical rasterization (*RO*),
3. drawing with hierarchical rasterization enabled (*HR*),
4. and drawing with both redundancy optimization and hierarchical rasterization enabled (*ROHR*).

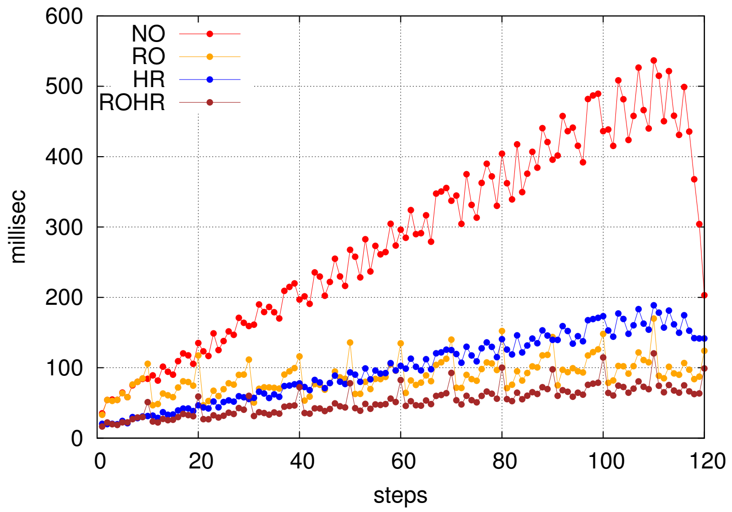


Figure 4.8: Performance plot: timings. Variations of the write and collapse operations are plotted (see Sec. 4.5.1 for legend's abbreviations). A torus is drawn in 120 steps (x-axis). The time for writing the new data into the DVT is measured in milliseconds (y-axis).

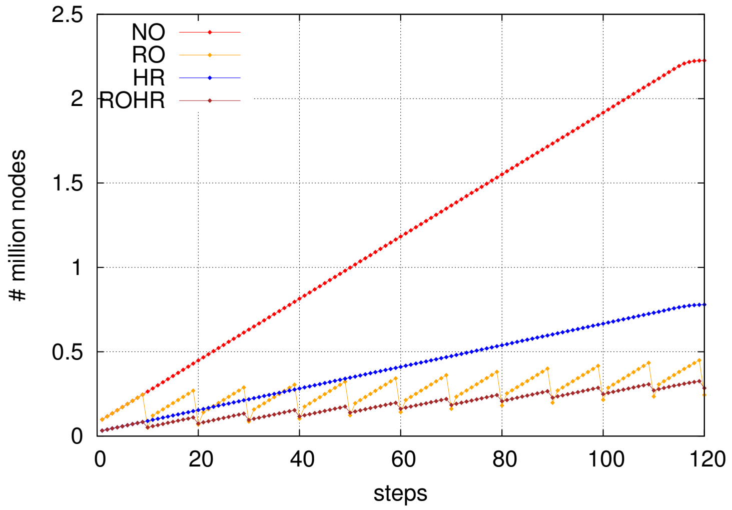


Figure 4.9: Performance plot: number of nodes. The total number of nodes (fill-level) during the drawing of the torus is displayed.

#procs	16	32	48	64	80	96	112	128	speed-up
171.751 nodes	474ms	254ms	185ms	149ms	131ms	120ms	113ms	107ms	4.43
1.232.589 nodes	4086ms	2100ms	1444ms	1120ms	938ms	816ms	773ms	669ms	6.11

Table 4.1: Write operation scalability. The last column shows the speed-up of the use from 16 to 128 processors while inserting a certain number of nodes into the DVT (GeForce 8800 GTX graphics card).

The results of the measurements of tree modifications, i.e. performing write and collapse operations (rendering is switched off during performance measurements), are displayed in the plots shown in Fig. 4.8 and 4.9. The best performance is achieved by enabling the redundancy optimization as well as the hierarchical rasterization which performs the collapse operation every 10th step as indicated by the respective peaks in the time measurement (see RO and ROHR in Fig. 4.8). The effect of the redundancy optimization is also illustrated by the decreasing number of nodes after each iteration (see RO and ROHR in Fig. 4.9). The continuous ups and downs of each plot in Fig. 4.8 are explained by the fact that the tree does not need to be expanded in certain regions during each write operation as previous write operations may have expanded the tree already close to the specific target depth. This results in a better performance and thus in a better time measurement.

4.5.2 Scalability

The scalability of the parallel DVT implementation is examined by disabling processing units of the graphics hardware. Similar to [ZHWG08] the NVStrap-driver in RivaTuner [nic] is used in order to reduce the number of processors of a Geforce 8800 GTX (since the NVStrap cannot be applied to latest Nvidia GPUs). The running time of the write operation is scalable to a great extent. However, its scalability is sublinear due to the constant overhead in API management. Two test-cases are listed in Table 4.1, i.e. inserting a medium and a high number of nodes into the DVT. Each column illustrates the timings for the specific number of active processing units.

4.5.3 Comparison

In contrast to other spatial data structures the DVT focuses on a multi-resolution spatial volume representation for depth map processing as well as object voxelization. It is rather difficult to do a direct comparison with approaches utilizing acceleration structures mostly applied in the area of ray tracing without integrating these data structures into the same application. However, Zhou et al. [ZHWG08] construct a kd-tree on graphics hardware for ray tracing as well as photon mapping for dy-

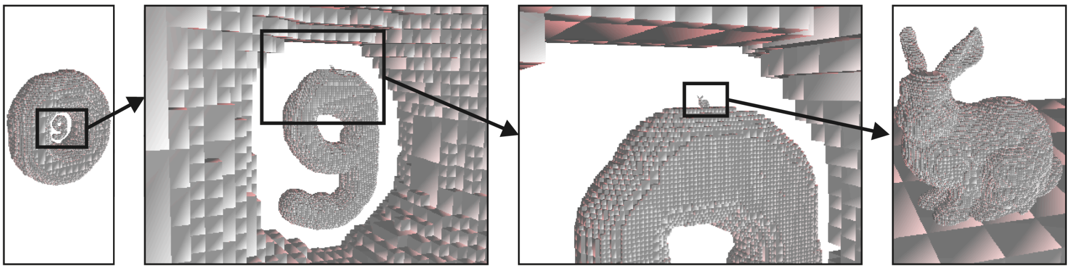


Figure 4.10: Volume drawing results. The tree's high spatial resolution is illustrated by the volume drawing application. The objects are drawn interactively in real-time in solid mode (data structure: $2.5M$ nodes, 25 fps, primitives rendering approach).

namic scenes. The kd-tree is built from scratch for every frame, similar to Lauterbach et al. [LGS*09] who rebuild their BVH for each frame. By the time when DVTs have been published [KCK09], Zhou et al. [ZGHG08] proposed an octree-structure which is build in real-time on the GPU in order to handle point clouds. New points can be inserted into the hierarchy and the respective object surface is reconstructed. Basic boolean operations are also supported by computing implicit functions for the surface. However, in contrast to Zhou et al. DVTs can handle data in a nearly arbitrary resolution, i.e. target depth, furthermore the values of the data structure are not sorted before processing. In general, a quantitative comparison of the data structures is difficult due to the different fields of applications.

4.5.4 Experimental Scenes

The creation of DVTs has been tested for various scenes: a volume drawing application demos online drawing and object-rasterization in 3D space. The capabilities of range data accumulation are shown in several sceneries handling synthetic data as well as real camera data.

High Resolution Volume Drawing

An interactive volume drawing application is demonstrated that stores the drawing into a DVT (see Fig. 4.10). The application is well suited to show the interactive performance of the hierarchical data structure. The data can be *drawn* into the DVT with a nearly arbitrary effective spatial resolution that is only constrained by the hardware's floating point precision. Its size is bounded by the size of the GPU memory. The brush mode (*solid* or *surface*), the brush size as well as the brush's voxel size are adjustable by the user. The voxel size defines the spatial resolution, i.e. the target depth of the tree where the current drawing is written to. This also defines the minimum voxel size. The application catches drawing events at 60 fps and processes the data immediately. Thus, an interactive feedback is provided while drawing in 3D space. Additionally, the application allows for the voxelization of polygonal meshes which can be processed by the DVT, e.g. the voxelized rabbit mesh in Fig. 4.10. Interactive frame-rates are maintained during the drawing process. In particular, the frame-rates depend on the number of nodes which are currently to be processed as well as the fill level of the DVT. With regard to the boolean operations *merge* and *subtract*, DVTs cannot only *draw* new data into the structure but also carve out data.

Range Data Accumulation

Primarily, DVTs have been developed in order to accumulate range images into a consistent data basis. Therefore, depth maps acquired from various scenes have been processed. While the synthetic reference scene *Sim* illustrates the accumulation of hundreds of range images, the *Tubes Box* scene simulates a robot's bin picking application. Finally, the bin picking application is also performed with real sensor data. All accumulation scenes are processed by the DVT with redundancy optimization as well as the hierarchical rasterization enabled. A 6 DoF camera pose transformation comprising rotation and translation information is assigned to each depth map for transforming the depth data into a common 3D world coordinate basis.

Synthetic Reference Scene Sim The data structure's ability of range data accumulation has been tested with a simulation sequence recorded by a camera rotating around the simple synthetic reference scene *Sim*[1]. The range data is computed by using the sensor simulation framework described in Chap. 3 which has been configured in order to generate ideal depth data, i.e. typical ToF effects are not present in this dataset. The final range images are stored in two sizes: 160 × 120 pixels and 640 × 480 pixels. The total simulation sequence consists of 950 depth maps acquired by a camera rotating around a scene comprising two teapots and two box objects. One full turn of the virtual camera is reached after 520 images. The accumulated result is displayed in Fig. 4.11.

The processing of low resolution images with 160×120 pixels as well as the high resolution images with 640×480 pixels are compared to each other as displayed in the plots in Fig. 4.13 and 4.14. Obviously, the processing of the low resolution sequence (thin orange curve) with a mean processing time of 44 ms ($\sim$ 23 fps) outperforms the high resolution sequence (bold curve) with a mean of 455 ms ($\sim$ 2.2 fps) which is no

[1] The same simulation scene is also used for evaluation purpose in Chap. 5 presenting a *point-based fusion* algorithm.

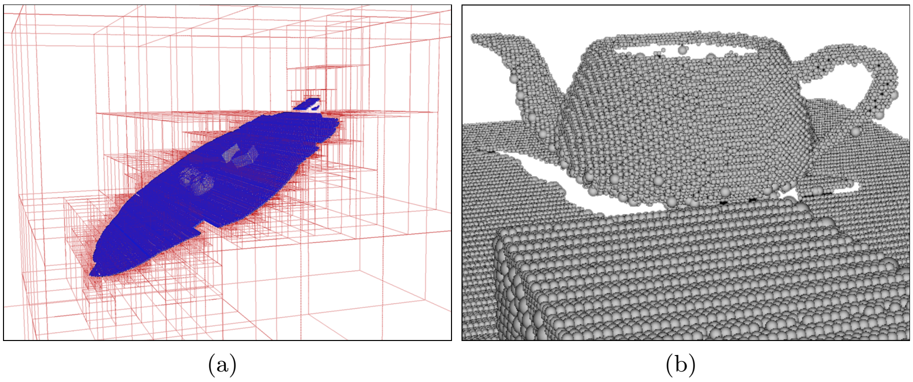
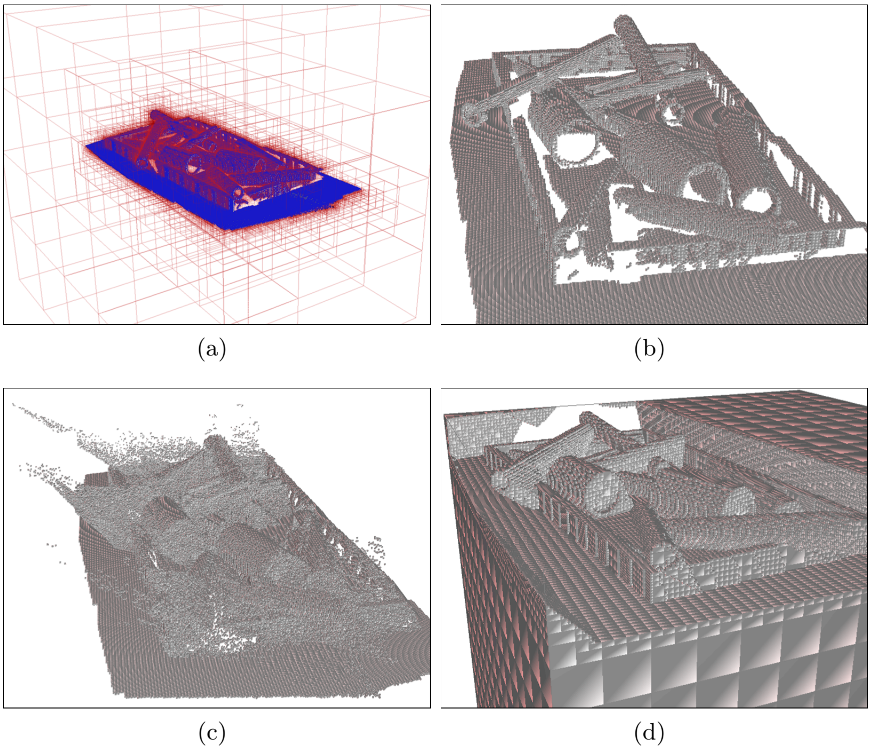


Figure 4.11: Visualization of the synthetic reference scene *Sim*. 4.11(a) displays the tree structure colored in red. The blue colored voxels indicate the leaf nodes of the tree containing the actual information. In 4.11(b) the leaf nodes of the partially accumulated scene are rendered with point sprites visualized as spherical primitives.

longer feasible for real-time accumulation of depth sensor data. The processing time of an individual depth map strongly depends on the balancing of the DVT's nodes. In suboptimal situations the processing lasts over 650 ms in contrast to 300 ms in well suited situations. The significant ups and downs are also caused by the GPU's overall workload which requires a high number of parallel threads for the processing of the high resolution depth maps while both sequences occupy approximately the same number of total nodes in the DVT ($\sim$ 200.000 nodes).

Tubes Box Scene The synthetic *Tubes Box* scene consists of a box with multiple tubes of different sizes inside (see Fig. 4.12). A virtual camera moves over the scene and acquires data from various angles which simulates the real world bin picking application where a robot is equipped with a depth sensor and explores the scenery. The sequence of 360 depth maps is acquired by the simulation framework as explained in Sec. 3.6.5. Depth maps are simulated in several variants such as 160×120 pixels which contain ideal depth values as well as 160×120 pixels which include typical ToF artifacts such as motion blur, flying pixels, and sensor noise (see Fig. 4.12(c)). Additionally, a sequence with high resolution images of 640×480 pixels is produced which contains ideal depth values with no ToF specific senor artifacts. Fig. 4.12(d) displays the sequence's volume carving result. Therefore, the individual depth maps are voxelized comprising their full spatial volumetric representation. Then, the depths maps' solid voxel representation is subtracted from an initially occupied volume.

The plots of the *Tubes Box* scene (also displayed in Fig. 4.13 and 4.14) expose a similar behavior as the previously presented reference scene *Sim*: the processing of low resolution images (thin green curve) with a mean of 160 ms ($\sim$ 6.2 fps) is



(a)

(b)

(c)

(d)

Figure 4.12: Visualization of the *Tubes Box* scene. 4.12(a) displays the tree structure colored in red (leaf nodes: blue). 4.12(b) shows the reconstructed surface after accumulating the ideal depth maps while in 4.12(c) the result of a ToF simulated dataset is shown including typical artifacts such as motion blur and noise. 4.12(d) displays the scene's volume carving result which is obtained by subtracting the individual depth maps from the initially occupied volume.

significantly faster than the processing of high resolution images (bold curve) with a mean of 750 ms ($\sim$ 1.3 fps). The characteristics of ToF data in contrast to ideal data are illustrated by the large number of $\sim$ 1 mio nodes which is more than twice the number of nodes as of the ideal dataset. The large number of nodes is caused by the noisy character of ToF data which affects more nodes in the DVT than noise free data.

The overall computation time of the *Tubes Box* scene is increased in comparison to the reference scene data. This is due to the fact that the *Tubes Box* scene is processed into the DVT with higher details, namely with a normalized target voxel size of $\frac{1}{512}$ in contrast to the coarser voxel size of the reference scene, i.e. $\frac{1}{256}$. The computation time of the *Tubes Box* scene with a decreased voxel size of $\frac{1}{256}$ is displayed as a dotted green curve in Fig. 4.13. Here, the mean processing time is 76 ms ($\sim$ 13 fps) which is of the same magnitude as its equivalent of the reference scene.

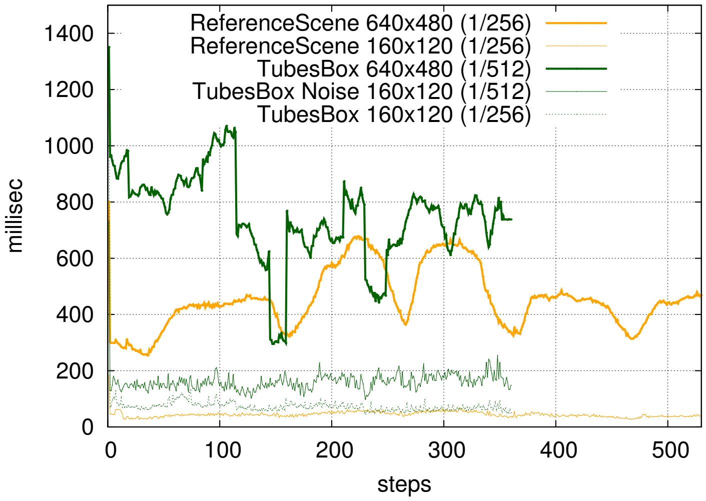


Figure 4.13: Range data accumulation: timings. The timings for depth maps with various sizes (160 × 120 pixels → thin curves vs. 640 × 480 pixels → bold curves) are displayed. The performance measured in milliseconds for each time step is plotted for the synthetic reference scene *Sim* (orange colored curves) as well as for the *Tubes Box* scene (green colored curves).

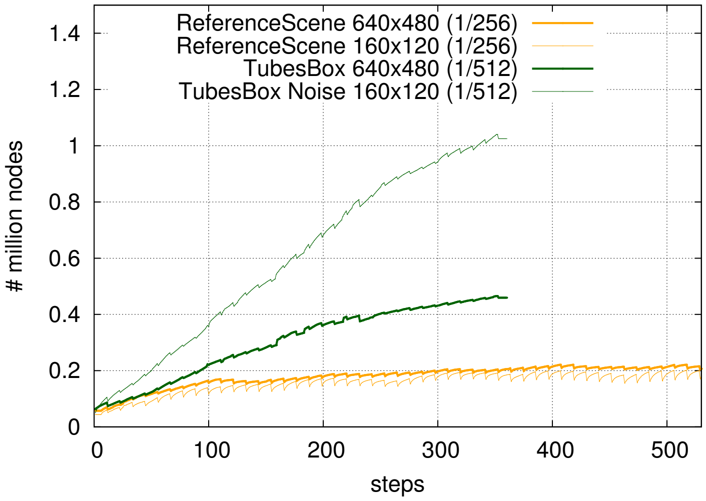
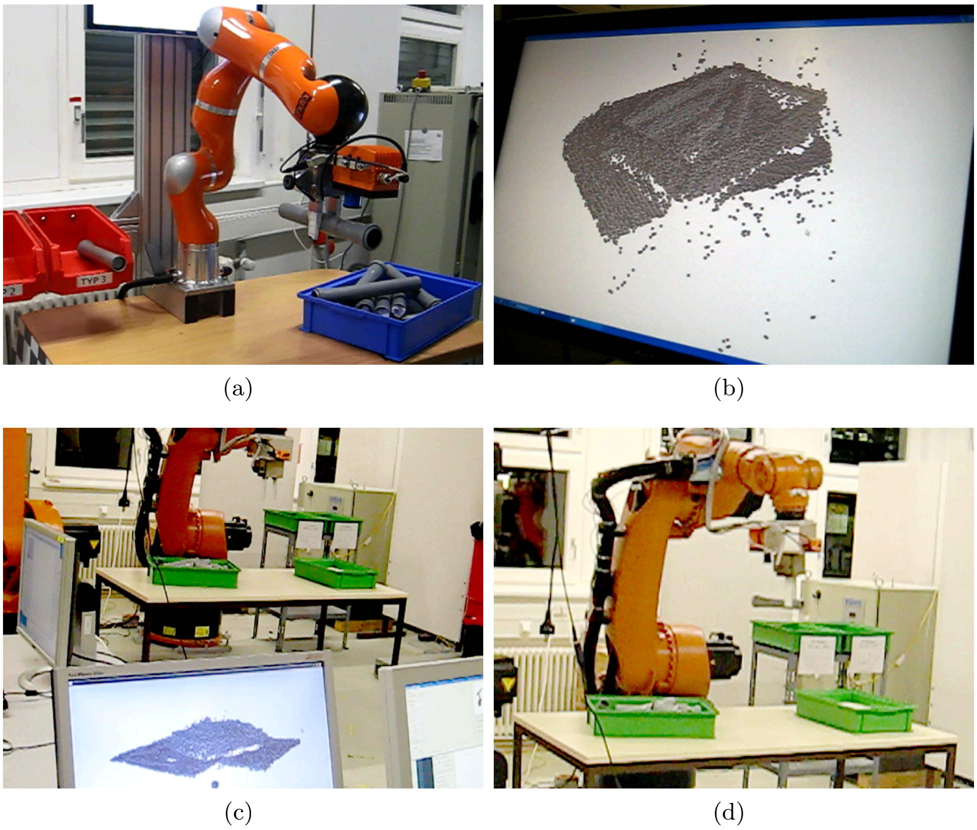


Figure 4.14: Range data accumulation: number of nodes. The total number of nodes of the accumulation is displayed for the synthetic reference scene *Sim* and the *Tubes Box* scene.


(a)

(b)

(c)

(d)

Figure 4.15: Robot's bin picking application. The DVT structure is incorporated into the environment model of the robot's bin picking application system. 4.15(a) shows the DLR lightweight robot which allows for cooperative bin picking tasks with a human co-worker. In 4.15(b) a screenshot of the accumulated data in the environment model is displayed. 4.15(c) and 4.15(d) show the fully automated bin picking application performed by an industrial robot at Karlsruhe Institute of Technology (KIT), utilizing the DVT featured environment model.

Bin Picking Application The bin-picking application is a showcase application in robotics which combines several robotic challenges such as object recognition and localization, grasp planning, path planning, and collision avoidance. The DVT is incorporated into an environment model which allows industrial robots to efficiently explore their environment and to fulfill new classes of picking tasks. Three key components set up the bin picking application: a 14 kg lightweight robot (German Aerospace Center, DLR LWR-III) performs the bin picking tasks as well as allows for direct interaction with humans. Secondly, a ToF camera which is mounted to the robot's front most joint continuously observes the scene. Thirdly, a software system evaluates the observed scene which is accumulated in the environment model. This is the basis for further computation tasks such as object localization, tracking and path-planning. The communication between the components is based on network protocols which ensure a high performance of the overall system.

The robot's task is to pick the tubes out of the box autonomously. The localization of small objects is feasible only at close distances with the utilized depth sensor which provides a small field of view ($\sim 45°$) and a lateral resolution of 200×200 pixels. Furthermore, the working range of the robot does not allow for the observation of the complete workbench at one point in time. Thus, capturing the entire scene is achieved by moving the robot to various positions in order to explore the overall workspace. All acquired depth maps and the respective robot poses are then fed into the environment model which accumulates the data into a consistent data basis. Finally, the object localization and path planning modules request virtual views from the environment model by providing a virtual pose along with image properties (such as image resolution) in order to run segmentation and object localization algorithms on the data. Fig. 4.15 shows the lightweight robot performing the bin picking application as well as a large industrial robot utilizing DVTs incorporated into the system's environment model.

4.6 Discussion

In this chapter an adaptive hierarchical volume data structure has been presented, namely the Dynamic Volume Tree (DVT), which runs on the GPU and can be modified interactively. The kd-tree-like hierarchical structure is built and managed on the GPU and supports boolean operations for merging and subtraction of sub-volumes with nearly arbitrary resolution. The tree is integrated into a real-time volume drawing application for multi-resolution drawings. Furthermore, the accumulation of hundreds of depth maps has been demonstrated by DVTs. The integration of the data structure into a robot's bin picking application has also been presented.

During the practical use of the data structure the following observations have been made:

Real-time Capability DVTs have been proven to successfully handle the accumulation of input data at interactive frame-rates (see Fig 4.8 and 4.13). However, standard real-world scenes result in unbalanced trees which progressively slows down the accumulation. While the processing of depth maps of 160 × 120 pixels looses its online capability for scenes with high details, the accumulation of high resolution depth maps, e.g. 640x480 pixels, could not reach an interactive frame-rate from the start (see Fig. 4.13).

Lean Data Structure Although the development of DVTs targets explicitly the field of lean and efficient data structures, the current implementation still occupies a certain amount of GPU memory and spends a significant part of processing resources for managing the data structure. Thus, computation overhead causes a slower processing of the data.

Registration The current implementation does not compensate for tracking errors. Depth maps are transformed into a common coordinate basis for accumulation purpose only. The transformation matrix is delivered along with the depth maps into the DVT system (e.g. by simulation or robot pose) without accounting for slightly misaligned data. An internal registration step, e.g. pairwise ICP, is not implemented. Finally, this results in an inaccurate accumulation model.

Data Pre-processing The current approach does not account for pre-processing of noisy input data. For example, filtering of ToF artifacts (noise, outliers, flying pixels) is not applied since prefiltering would further decrease the performance of the system. Thus, the current accumulation results are rather noisy.

Spatial Representation The voxel data structure is a spatial representation of the data in the proposed examples, i.e. the extension of a leaf-voxel represents a data point's spatial extension in space. Due to the hierarchical approximation the approach loses data precision. The precision is defined by the tree structure's target depth.

Alternative Approaches The observations mentioned before open up several directions regarding alternative approaches. Potential work areas are: to ensure the real-time capability of the system even for sensors with high lateral resolution; to pre-process and filter noisy input data, to compensate tracking-errors, and to gain a higher precision in the data's spatial representation. The point-based fusion approach, which is proposed in the next chapter, improves the volumetric approach in terms of processing speed, accuracy, and rendering quality.

Chapter 5

Point-based Fusion of Range Data

The accumulation system proposed in the previous chapter is limited by the computational overhead of the underlying volumetric data structure. Interactive frame-rates are hard to reach for larger scenes as well as for depth sensors with higher resolution, such as Microsoft's Kinect with VGA resolution of 640x480 pixels. In this chapter, a point-based fusion algorithm is presented which accounts for the limitations of the previous approach. The method focuses on real-time dense reconstruction of 3D environments with equivalent quality to existing online methods, but with support for additional spatial scale and robustness in dynamic scenes. The approach is designed around a simple and flat point-based representation, which directly works with the input acquired from depth sensors, without the overhead of converting between representations. The use of points enables speed and memory efficiency.

After the introductory sections about motivation and related work, the chapter proceeds with the basic point-based fusion approach in Sec. 5.3. Sec. 5.4 describes the support of dynamically changing scenes, followed by a section about the integration of occupancy grids (Sec. 5.5) which addresses the topic of environment models for safe robot navigation and autonomous exploration in unknown areas. The chapter closes with results and their analysis (Sec. 5.6.3) as well as a conclusive discussion in Sec. 5.7.

Publications The proposed work on real-time 3D reconstruction using point-based fusion has been published in collaboration with Microsoft Research, Cambridge, and the University College London in [KLL*13]. The underlying GPU framework has been proposed in [OKK09]. The part about sketching an alternative fusion technique based on MLS instead of computing a running average has not been published yet. The section about the integration of occupancy grids into the point-based system remains also unpublished so far.

5.1 Motivation

The availability of consumer depth cameras, such as the structured light version of Microsoft's Kinect (2011) as well as their ToF version (2014), has made real-time depth map acquisition a commodity. This has brought the topic of real-time 3D reconstruction further to the forefront. KinectFusion [IKH*11, NIH*11] has been an inspiration to the approach presented in this chapter. KinectFusion fuses multiple overlapping depth maps into a single 3D representation which is continuously refined over time while acquiring new depth maps. The system deals with the registration of depth maps to the sensor's ego-motion, takes care of outlier removal, and uses a simple weighted averaging approach in order to fuse the data. However, existing grid-based methods such as KinectFusion either trade scale to achieve higher quality real-time reconstructions. Or support larger scenes at the expense of real-time performance and/or quality.

The point-based fusion approach presented in this thesis accounts for spatial limitations which also became apparent in the volumetric volume trees method presented in the previous chapter: a purely point-based algorithm removes the need for a spatial data structure and thus eliminates spatial dependencies.

Challenges The approach's online capability needs to be ensured even for large reconstruction scenes as well as for high lateral resolution depth maps and high temporal data acquisition: depth sensors such as the Kinect with 640×480 pixels at 30 Hz acquire about 550 million depth samples per minute. Also high quality surface rendering should be achieved which requires the precise computation for a point's spatial orientation. This is challenging due to noisy input data from depth sensors. Furthermore, the handling of dynamically changing scenes demands for a robust object segmentation algorithm in real-time.

Objectives This chapter targets the development of a real-time 3D reconstruction approach using a point-based representation, rather than any spatial data structure. The method deals with incremental reconstruction from noisy depth maps without converting between different geometry representations.

5.2 Related Work

When Microsoft's Kinect was brought to market a new era of real-time 3D reconstruction approaches began. Izadi and Newcombe [IKH*11, NIH*11] achieve high-quality results by adopting the original volumetric fusion method of Curless and Levoy [CL96] in order to perform real-time processing. Their approach supports incremental updates, exploits redundant samples, makes no topological assumptions. Furthermore, sensor uncertainty is partially incorporated, and fusion is performed using a simple weighted average. For active sensors, this method produces very compelling results [CL96, LPC*00, IKH*11, NIH*11]. The drawback of this method is the computational overhead needed to continuously transition between different data representations: where point-based input is converted to a continuous implicit function, discretized within a regular grid data structure, and converted back to an (explicit) form using computationally expensive polygonization [LC87] or raycasting [PSL*98] methods. As well as the memory overheads imposed by using a regular voxel grid, which represents both empty space and surfaces densely, and thus significantly limits the size of the reconstruction volume due to computational resources such as GPU memory.

These memory limitations have led to *moving-volume* systems [RV12, WKF*12], which still operate on a very restricted volume, but free-up voxels as the sensor moves; or hierarchical volumetric data structures [ZZZL13], which incur additional computational and data structure complexity for limited gains in terms of spatial extent. Recently, Nießner et al. [NZIS13] presented an efficient voxel-based hashing data structure for volumetric real-time 3D reconstruction which produces promising high quality reconstruction results at larger scales.

Beyond volumetric methods, simpler representations have also been explored. Height-map representations [GPF10] work with compact data structures allowing scalability, especially suited for modeling large buildings with floors and walls, since these appear as clear discontinuities in the height-map. Multi-layered height-maps support reconstruction of more complex 3D scenes such as balconies, doorways, and arches [GPF10]. While these methods support compression of surface data for a specific class of scenes, the 2.5D representation fails to model complex 3D environments efficiently.

Point-based representations are more amenable to the input acquired from depth sensors. Rusinkiewicz et al. [RHHL02] use a point-based method and a custom structured light sensor to demonstrate in-hand online 3D scanning. Online model rendering requires an intermediate volumetric data structure. Interestingly, an offline volumetric method [CL96] was used for higher quality final output, which nicely highlights the computational and quality trade-offs between point-based and volumetric methods. Weise et al. [WWLVG09] took this one step further, demonstrating higher quality scanning of small objects using a higher resolution custom structured light camera, sensor drift correction, and higher quality surfel-based [PZVBG00] rendering. These systems however focus on single small object scanning. Further, the sensors produce less noise than consumer depth cameras making model denoising less challenging.

Beyond reducing computational complexity, point-based methods lower the memory overhead associated with volumetric (regular grid) approaches, as long as overlapping points are merged. Such methods have therefore been used in larger sized reconstructions [HKH*12, SB12]. However, a clear trade-off becomes apparent in terms of scale versus speed and quality. For example, Henry et al. [HKH*12] allow for reconstructions of entire floors of a building (with support for loop closure and bundle adjustment), but the frame-rate is limited ($\sim$ 3 Hz) and an unoptimized surfel map representation for merging 3D points can take seconds to compute. Stückler and Behnke [SB12] use a multi-level surfel representation that achieves interactive rates ($\sim$ 10 Hz) but requires an intermediate octree representation which limits scalability and adds computational complexity.

The approach proposed in this chapter tackles also the topic of dynamically changing scenes: while previous systems assume a scene to be static or treat dynamic content as outliers [RHHL02, WWLVG09], KinectFusion [IKH*11] is capable of reconstructing moving objects in a scene, providing a preliminary acquired static pre-scan of the background. In contrast to this, the current approach presents a robust segmentation algorithm for dynamic objects without assuming strong requirements about the scene layout.

5.3 Basic Fusion System

In this section the basic fusion approach is presented and its algorithmic components are described. The following high-level approach shares commonalities with existing incremental reconstruction systems: samples are used from a moving depth sensor; first the depth data is pre-processed; then the current 6 DoF pose of sensor relative to the scene is estimated; and finally this pose is used to convert depth samples into a unified coordinate space. The depth samples are fused into an accumulated global model. Unlike other systems, a purely point-based representation is adopted throughout the processing, designed to support data fusion with quality comparable to online volumetric methods, whilst enabling real-time reconstructions at extended scales.

5.3.1 Overview of Concept

The main system pipeline as shown in Fig. 5.1 is similar to the one proposed by Newcombe et al. [NIH*11] for volumetric fusion. The pipeline is based on the following steps:

Depth Map Pre-processing Using the intrinsic parameters of the camera, each input depth map from the depth sensor is transformed into a set of 3D points, stored in a 2D *vertex map*. Corresponding normals are computed from central-differences of the denoised vertex positions, and per-point radii are computed as a function of depth and gradient (stored in respective *normal* and *radius* maps). In Sec. 5.3.2 the pre-processing is outlined.

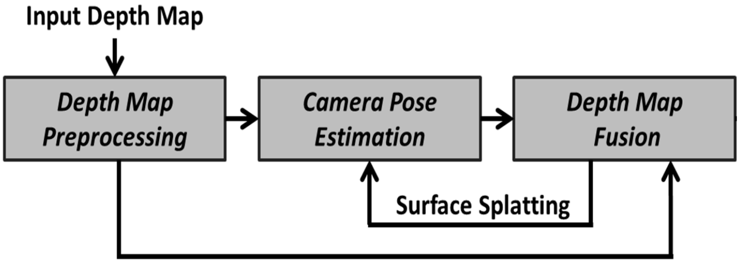


Figure 5.1: Main system pipeline.

Depth Map Fusion Given a valid camera pose, input points are fused into the *global model.* The global model is a list of 3D points with associated attributes. Points evolve from *unstable* to *stable* status based on the confidence they gathered (essentially a function of how often they are observed by the sensor). Data fusion first projectively associates each point in the input depth map with the set of points in the global model, by rendering the model as an *index map*. If corresponding points are found, the most reliable point is merged with the new point estimate using a weighted average. If no reliable corresponding points are found, the new point estimate is added to the global model as an unstable point. The global model is cleaned up over time to remove outliers due to visibility and temporal constraints. Sec. 5.3.3 discusses the point-based data fusion in detail.

Camera Pose Estimation and Rendering All established (high confidence) model points are passed to the visualization stage, which reconstructs dense surfaces using a surface splatting technique (see Sec. 5.3.4). To estimate the 6 DoF camera pose, the model points are projected from the previous camera pose, and a pyramid-based dense iterative closest point (ICP) [NIH*11] alignment is performed using this rendered *model map* and input depth map. This provides a new relative rigid 6 DoF transformation that maps from the previous to new global camera pose. Pose estimation occurs prior to data fusion, to ensure the correct projection during data association.

5.3.2 Depth Map Pre-processing

A 2D pixel is denoted as $\boldsymbol{u} = (x, y)^\top \in \mathbb{R}^2$. $\mathcal{D}_i \in \mathbb{R}$ is the raw depth map at time frame i. Given the intrinsic camera calibration matrix $\boldsymbol{K}_i$, $\mathcal{D}_i$ is transformed into a corresponding vertex map $\mathcal{V}_i$, by converting each depth sample $\mathcal{D}_i(\boldsymbol{u})$ into a vertex position $\boldsymbol{v}_i(\boldsymbol{u}) = \mathcal{D}_i(\boldsymbol{u})\boldsymbol{K}_i^{-1}(\boldsymbol{u}^\top, 1)^\top \in \mathbb{R}^3$ in camera space. A copy of the depth map (and hence associated vertices) is also denoised using a *bilateral filter* [TM98] (for camera pose estimation later). A corresponding normal map $\mathcal{N}_i$ is determined from

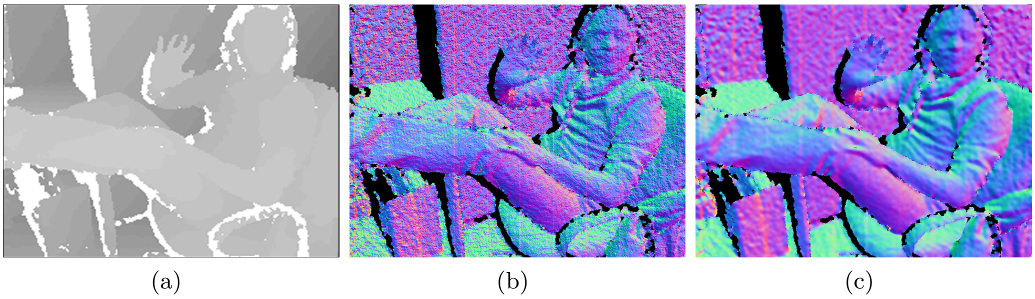


Figure 5.2: Filtered normal map. 5.2(a) displays the greyscale depth image from a Kinect sensor. The unfiltered normal map is shown in 5.2(b) whereas a pre-processed normal map is displayed in 5.2(c) using the bilateral filtered depth image.

central-differences of the bilateral filtered vertex map:

$$\boldsymbol{n}_i(\boldsymbol{u}) = \big(\boldsymbol{v}_i(x+1,y) - \boldsymbol{v}_i(x-1,y)\big) \times \big(\boldsymbol{v}_i(x,y+1) - \boldsymbol{v}_i(x,y-1)\big) \in \mathbb{R}^3 \ , \qquad (5.1)$$

and normalized to unit length (see Fig. 5.2).

The 6 DoF camera pose transformation comprises rotation matrix ($\mathrm{R}_i \in \mathbb{SO}_3$) and translation vector ($\boldsymbol{t}_i \in \mathbb{R}^3$), computed per frame i as $\boldsymbol{T}_i = [\mathrm{R}_i, \boldsymbol{t}_i] \in \mathbb{SE}_3$. A vertex is converted to global coordinates as $\boldsymbol{v}_i^{\mathrm{g}} = \boldsymbol{T}_i \boldsymbol{v}_i$. The associated normal is converted to global coordinates as $\boldsymbol{n}_i^{\mathrm{g}}(\boldsymbol{u}) = \mathrm{R}_i \, \boldsymbol{n}_i(\boldsymbol{u})$. Multi-scale pyramids $\mathcal{V}_i^l$ and $\mathcal{N}_i^l$ are computed from vertex and normal maps for hierarchical ICP, where $l \in \{0, 1, 2\}$ and $l = 0$ denotes the original input resolution (e.g. 640×480 pixels for Kinect or 160×120/200×200 pixels for PMD Camboard devices respectively).

Each input vertex also has an associated radius $r_i(\boldsymbol{u}) \in \mathbb{R}$ (collectively stored in a radius map $\mathcal{R}_i \in \mathbb{R}$), determined as in [WWLVG09]:

$$r_i(\boldsymbol{u}) = \frac{1}{\sqrt{2}} \frac{\boldsymbol{v}_i(\boldsymbol{u})(z)/f}{\boldsymbol{n}_i(\boldsymbol{u})(z)} \in \mathbb{R}, \qquad (5.2)$$

where f is the focal length of the depth sensor and z denotes the vertex's cartesian depth value and the normal's z component respectively. To prevent arbitrarily large radii from oblique views, radii for grazing observations exceeding 75° are clamped.

In the remainder of this chapter, time frame indices i are omitted for clarity, unless two different time frames are addressed at once.

5.3.3 Depth Map Fusion

The approach maintains a single global model, which is simply an unstructured set of points $\bar{M}_k$ each with associated position $\bar{\boldsymbol{v}}_k \in \mathbb{R}^3$, normal $\bar{\boldsymbol{n}}_k \in \mathbb{R}^3$, radius $\bar{r}_k \in \mathbb{R}$,

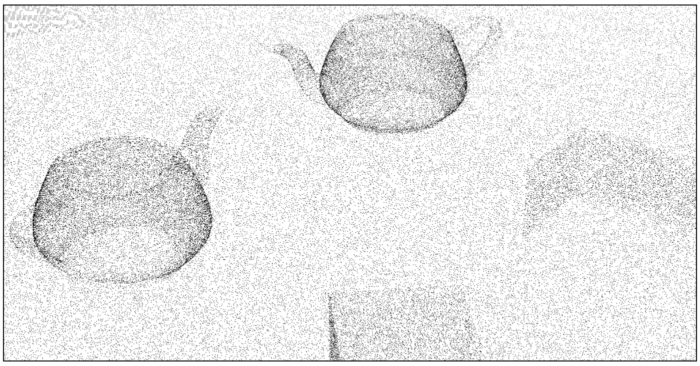


Figure 5.3: Index map: all global model points are rendered from the previous camera perspective into the index map $\mathcal{I}$; encoding the points' position in the global array. This way, the data is accessed fast and efficiently without using an explicit data structure.

confidence counter $\bar{c}_k \in \mathbb{R}$, and time stamp $\bar{t}_k \in \mathbb{N}$, stored in a flat array indexed by $k \in \mathbb{N}$.

New measurements $\boldsymbol{v}$ are either added as or merged with unstable points, or they get merged with stable model points. Merging $\boldsymbol{v}$ with a point $\bar{M}_k$ in the global model increases the confidence counter $\bar{c}_k$. Eventually an unstable point changes its status to stable: points with $\bar{c}_k \geq c_{\text{stable}}$ are considered stable (in practice $c_{\text{stable}} = 10$). In specific temporal or geometric conditions, points are removed from the global model.

Data Association

After estimating the camera pose of the current input frame (see Sec. 5.3.4), each vertex $\boldsymbol{v}^{\mathrm{g}}$ and associated normal and radius are integrated into the global model. In a first step, for each valid vertex $\boldsymbol{v}^{\mathrm{g}}$ potential corresponding points are found in the global model. Given the inverse global camera pose $\boldsymbol{T}^{-1}$ and intrinsics $\boldsymbol{K}$, each point $\bar{M}_k$ in the global model can be projected onto the image plane of the current physical camera view, where the respective point index k is stored: all model points are rendered into a sparse *index map* $\mathcal{I}$ (see Fig. 5.3). Unlike the splat-based dense surface reconstruction renderer used in other parts of the approach (see Sec. 5.3.4), this stage renders each point index into a single pixel to reveal the actual surface sample distribution. Generating this index map is performed efficiently using the standard graphics pipeline (allowing parallel processing of points and features such as efficient frustum culling). As nearby model points may project onto the same pixel, the resolution of $\mathcal{I}$ is increased by supersampling, representing $\mathcal{I}$ at 4×4 the resolution of the input depth map. While the surface sample distribution is mostly sparse in the index map, regions of high curvature as well as the object's silhouette are rather dense. This is due to the fact that the objects' front- and backfacing points are projected onto nearby pixels.

Firstly, the model points near $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ are identified by collecting point indices within the 4×4-neighborhood around each input pixel location $\boldsymbol{u}$ (suitably coordinate-transformed from $\mathcal{D}$ to $\mathcal{I}$). Amongst those points, a single corresponding model point is determined by applying the following criteria:

1. Discard points larger than $\pm\delta_{\text{depth}}$ distance from the viewing ray $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ (the sensor line of sight), with δ_{depth} adapted according to sensor uncertainty (i.e. as a function of depth for triangulation-based methods [NIL12]).
2. Discard points whose normals have an angle larger than δ_{norm} to the normal $\boldsymbol{n}^{\mathrm{g}}(\boldsymbol{u})$. In particular, $\delta_{\text{norm}} = 20°$ is used.
3. From the remaining points, select the ones with the highest confidence count.
4. If multiple such points exist, select the one closest to the viewing ray through $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$.

Point Averaging with Sensor Uncertainty

If a corresponding model point $\bar{M}_k$ is found during data association, this is averaged with the input vertex $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ and normal $\boldsymbol{n}^{\mathrm{g}}(\boldsymbol{u})$ as follows:

$$\bar{\boldsymbol{v}}_k \leftarrow \frac{\bar{c}_k \bar{\boldsymbol{v}}_k + \alpha \boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})}{\bar{c}_k + \alpha}, \quad \bar{\boldsymbol{n}}_k \leftarrow \frac{\bar{c}_k \bar{\boldsymbol{n}}_k + \alpha \boldsymbol{n}^{\mathrm{g}}(\boldsymbol{u})}{\bar{c}_k + \alpha}, \quad \bar{r}_k \leftarrow \frac{\bar{c}_k \bar{r}_k + \alpha r(\boldsymbol{u})}{\bar{c}_k + \alpha}, \tag{5.3}$$

$$\bar{c}_k \leftarrow \bar{c}_k + \alpha\,, \quad \bar{t}_k \leftarrow t\,, \tag{5.4}$$

where t is a new time stamp. The weighted average is distinct from the original KinectFusion system [NIH*11], as an explicit sample confidence α is introduced. This applies a Gaussian weight on the current depth measurement as $\alpha = \mathrm{e}^{-\gamma^2/2\sigma^2}$, where γ is the normalized radial distance of the current depth measurement from the camera center, and $\sigma = 0.6$ is derived empirically. This approach weights measurements based on the assumption that measurements closer to the sensor center will increase in accuracy [CL96]. As shown in Fig. 5.4, modeling this sensor uncertainty leads to higher quality denoising.

Since the noise level of the input measurement increases as a function of depth [NIL12], Eqs. (5.3) is only applied if the radius of the new point is not significantly larger than the radius of the model point, i.e., if $r(\boldsymbol{u}) \leq (1 + \delta_r)\bar{r}$; $\delta_r = 1/2$ is empirically chosen. This ensures that details are always refined, but the global model is never coarsened. The time stamp and the confidence counter updates are applied according to Eqs. (5.4) irrespectively.

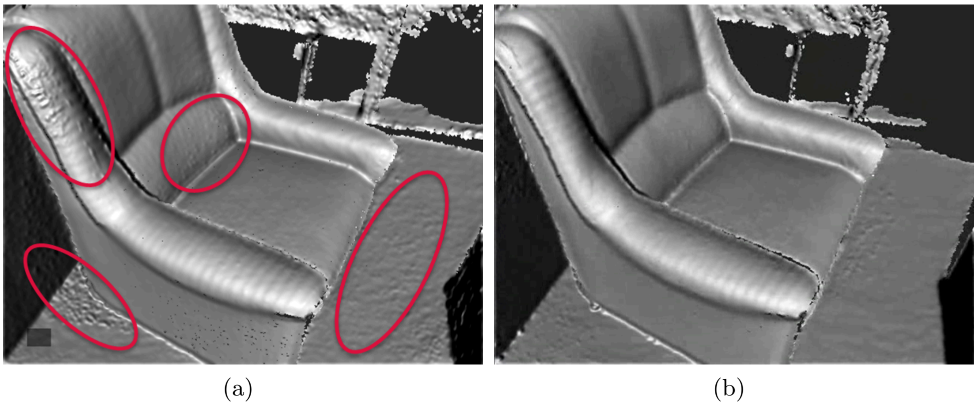


Figure 5.4: Sensor uncertainty. Weighted averaging of points using the method of [NIH*11] is shown in 5.4(a). The new uncertainty model is applied to the scene displayed in 5.4(b) which leads to better denoising results.

If no corresponding model point has been identified, a new unstable point is added to the global model with $\bar{c}_k = \alpha$, containing the input vertex, normal and radius.

Extended Data Fusion using Moving Least Squares

The process of data fusion as described in the prior section is solely based on preprocessed input depth maps and on averaging the global model data. In this section the approach is extended by applying a robust moving least squares (MLS) algorithm to the input data similar to Rusu et al. [RBM*07, RMB*08]. This way, the input data better approximates the actual geometry which reduces the noise significantly. In detail, the following steps are applied in order to update the input depth map $\mathcal{D}$:

1. For each input point $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ a set Q is selected which contains a local neighborhood of model points within a certain radius h. The sampling of the points is based on the look-up mechanism described previously by using the index map $\mathcal{I}$.

2. The model points in Q are still influenced by noise and measuring errors. Therefore, a local reference plane is robustly computed by taking the largest number n of inliers into account, with $\bar{\boldsymbol{v}}_k \in Q_{\text{inlier}} \subset Q$ which are obtained by performing a random sample consensus algorithm (RANSAC) [FB81]. The equation of the plane is computed by using eigenanalysis on the covariance matrix of the inlier points.

3. The polynomial fitting step of the MLS procedure is performed by transforming the inlier model points into the local coordinate system which is defined by the

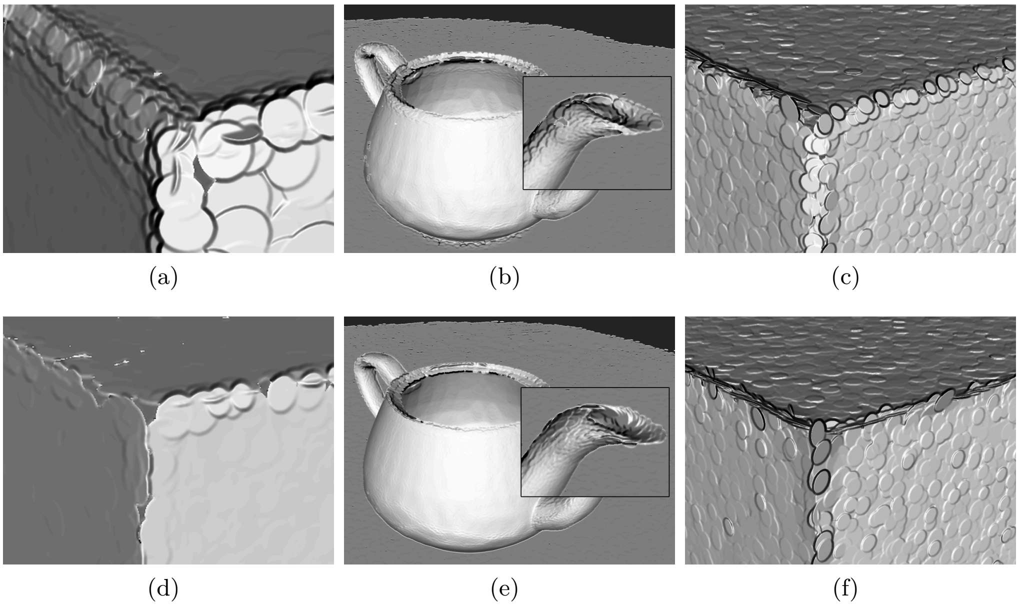


Figure 5.5: MLS results. The top row displays simple weighted averaging results, and the bottom row shows improvements based on MLS. Edges are rounded off caused by the weighted averaging as shown in 5.5(a), whereas MLS improves the reconstruction quality with sharp object edges as illustrated in 5.5(d). A better result is also achieved for spherical objects as well as for data with a significant amount of noise, see teapot and noisy box respectively.

local reference plane. A high-order bivariate polynomial (here a 3rd order polynomial is chosen) is fitted to the heights of the model points above the plane. The height of the fitted input point is then recalculated which yields the MLS result $\boldsymbol{v}^{\mathrm{mls}}(\boldsymbol{u})$. During the fitting step each of the model points $\bar{\boldsymbol{v}}_k$ is explicitly assigned with a weighting value $\bar{w}_k$. This way, the model points' distances to the current input point as well as their confidence is taken into account: $\bar{w} = \psi \tilde{c}_k$, where $\psi = \mathrm{e}^{-\|\boldsymbol{v}^{\mathrm{g}} - \bar{\boldsymbol{v}}_k\|^2/h}$ and $\tilde{c}_k$ is the weight based on the normalized confidence counter with $\tilde{c}_k = \bar{c}_k/c_{\mathrm{stable}}$. Thus, the input point is moved *towards* the fitting result with respect to the weighting of inliers.

4. The influence of the input point $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ is ensured by weighting its current position with $\boldsymbol{v}^{\mathrm{mls}}(\boldsymbol{u})$:

$$\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u}) \leftarrow \frac{\alpha \boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u}) + \hat{w} \boldsymbol{v}^{\mathrm{mls}}(\boldsymbol{u})}{\alpha + \hat{w}}, \quad \text{with} \quad \hat{w} = \frac{1}{n} \sum_{j=1}^{n} \bar{w}_{k\,j} \tag{5.5}$$

5. The input point's normal $\bar{\boldsymbol{n}}_k$ is set to the normal of the polynomial. Its radius $\bar{r}_k$ is calculated as the radius of the re-computed input point $\boldsymbol{v}(\boldsymbol{u})$.

This algorithm ensures that sharp edges are preserved since the reference plane is robustly computed. Additionally, the resulting quality is improved by considering the recalculated point normals (see Fig. 5.5).

Finally, further processing of the recalculated input points and normal vectors follows the procedure described in the previous sections about *data association* and *point averaging*.

Removing Points

So far new measurements have been merged or added to the global model. Another key step is to remove points from the global model due to various conditions:

1. Points that remain in the unstable state for a long time are likely outliers or artifacts from moving objects and will be removed after $t_{\max}$ time steps.
2. For stable model points that are merged with new data, all model points are removed that lie in front of these newly merged points, as these are free-space violations. To find these points to remove, the index map is used again and the neighborhood is searched around the pixel location that the merged point projects onto[1]. This is similar in spirit to the free-space carving method of [CL96], but avoids expensive voxel space traversals.
3. If after averaging, a stable point has neighboring points (identified again via the index map) with very similar position and normal direction and their radii overlap, then these redundant neighboring points are merged to further simplify the model.

Points are first marked to be removed from $\bar{M}_k$, and in a second pass, the list is sorted (using a fast radix sort implementation), moving all marked points to the end, and finally items are deleted. This way, the global model only contains relevant model points. Furthermore, the global model is kept lightweight.

5.3.4 Camera Pose Estimation and Rendering

Following the approach of KinectFusion [NIH*11], the camera pose estimation uses dense hierarchical ICP to align the bilateral filtered input depth map $\mathcal{D}_i$ (of the current frame i) with the reconstructed model as frame-to-model camera pose estimation has been shown to be superior to frame-to-frame methods [NIH*11].

[1] Backfacing points that are close to the merged points remain protected - such points may occur in regions of high curvature or around thin geometry in the presence of noise and slight registration errors. Furthermore, points are protected that would be consistent with direct neighbor pixels in $\mathcal{D}$, to avoid spurious removal of points around depth discontinuities.

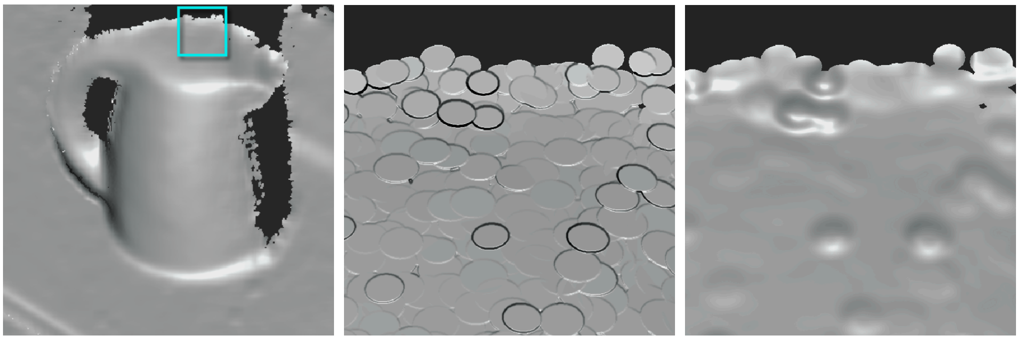

(a) (b) (c)

Figure 5.6: Surface splatting. Model points are rendered as disk-shaped splats as shown in 5.6(a). Phong shading as well as a bilateral filter is added for visual user feedback. In the close-ups of 5.6(b) and 5.6(c) various filter settings are applied.

Given the previous camera pose $\boldsymbol{T}_{i-1}$ and camera intrinsics $\boldsymbol{K}$, first a virtual depth map $\hat{\mathcal{D}}_{i-1}$ of all stable model points is rendered. Using $\hat{\mathcal{D}}_{i-1}$ and the current input depth map $\mathcal{D}_i$, vertex and normal pyramids $\mathcal{V}_i^l$ and $\mathcal{N}_i^l$ are generated with the finest level at the camera's resolution; unstable model points are ignored - except when a new accumulation sequence is started based on an empty global model: in the first couple of seconds all points are taken into account for pyramids generation. These maps are then passed to a hierarchical ICP step, based on the original KinectFusion method, which iteratively aligns the two sets of point clouds, outputting a single relative 6 DoF transformation providing the relative change from $\boldsymbol{T}_{i-1}$ to $\boldsymbol{T}_i$.

While KinectFusion employs raycasting of the implicit voxel-based reconstruction, the current approach renders the explicit, point-based representation using a simple surface-splatting technique: overlapping, disk-shaped *surface splats* are rendered that are spanned by the model point's position $\bar{\boldsymbol{v}}$, radius $\bar{r}$ and orientation $\bar{\boldsymbol{n}}$. Unlike more refined surface-splatting techniques, such as EWA Surface Splatting [ZPBG01], the proposed approach does not perform blending and analytical prefiltering of splats but trades local surface reconstruction quality for performance by simply rendering opaque splats.

The same point-based renderer is used for visual user feedback, but Phong shading of surface splats is added, and also the dynamic regions of the input depth map (see next section) are overlayed. In order to improve the overall visual quality the virtual depth map is filtered using low bilateral filter settings. This results in a smoother representation of the surface by the disk-shaped splats (see Fig. 5.6).

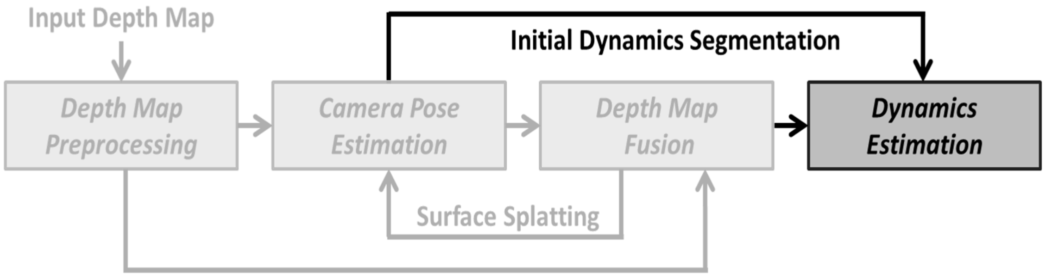


Figure 5.7: Main system pipeline with dynamics extension.

5.4 Advanced Fusion System: Scene Dynamics

This section describes an extension to the basic system that allows support for dynamically changing scenes, i.e. moving objects (see Fig. 5.7). The method automatically detects dynamic changes in the scene, the global reconstruction is updated accordingly which results in a more robust camera tracking. The idea is based on the re-utilization of the ICP result of the camera tracking stage. Starting from areas where no point correspondences are found, a point-based region growing procedure is performed to identify dynamic regions.

The work presented in this section has been developed in collaboration with Damien Lefloch, member of the Computer Graphics and Multimedia Systems Group at the University of Siegen, who has been responsible for real-time implementation of the dynamics segmentation algorithm, i.e. region growing method based on hierarchical connected components.

5.4.1 Overview of Concept

The basic system as described previously already has limited support for dynamic objects, in that unstable points must gain confidence to be promoted to stable model points, and thus fast moving objects will be added and then deleted from the global model. This basic behavior is extended by additional steps that lead to an explicit classification of observed points as being part of a dynamic object. Furthermore, the concept of handling dynamically changing scenes aims at segmenting entire objects whose surface is partially moving and remove them from the global point model.

The method builds upon an observation by Izadi et al. [IKH*11]: when performing ICP, failure of data association to find model correspondences for input points is a strong indication that these points are depth samples belonging to dynamic objects. Accordingly, this information is retrieved by constructing an ICP status map $\mathcal{S}$ (see Sec. 5.4.2). Next, this map is used for creating a *dynamics map* $\mathcal{X}$ which segments the dynamic parts of the current input frame by using a region growing algorithm. This method aims at marking entire objects as dynamic even if only parts of them actually move. This high-level view on dynamics is an improvement over the handling of dynamics in [IKH*11]. Finally, dynamics classification affects the depth map fusion stage in order to reflect the input points' dynamic status also in the global model data.

The complete world view of the system consists of *two* point sets: the global model points, which represent the reliable static parts of the environment, and the set of input points marked as dynamic, which represent moving objects.

5.4.2 Algorithmic Details

The ICP status map $\mathcal{S}$ with its elements $s_i(\boldsymbol{u})$ encodes for each depth sample the return state of ICP's search for a corresponding model point:

`no_input`: $\boldsymbol{v}_k(\boldsymbol{u})$ is invalid or missing.

`no_cand`: No stable model points in proximity of $\boldsymbol{v}_k(\boldsymbol{u})$.

`no_corr`: Stable model points in proximity of, but no valid ICP correspondence for $\boldsymbol{v}_k(\boldsymbol{u})$.

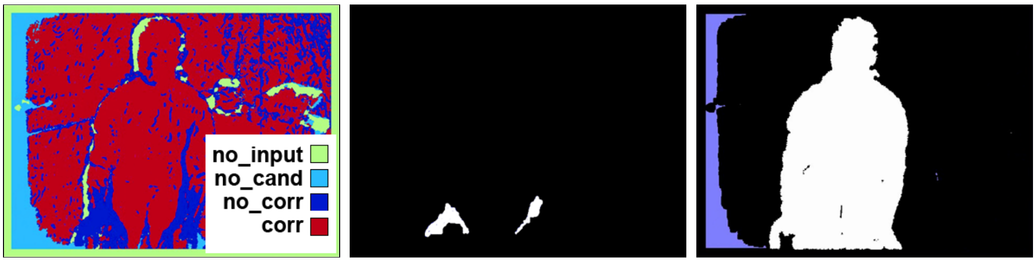
`corr`: Otherwise ICP found a correspondence.

Input points which are marked as `no_corr` are a strong initial estimate of parts of the scene that move independent of camera motion, i.e. dynamic objects in the scene. These points are used to seed the segmentation method based on region growing.

Building the Dynamics Map $\mathcal{X}$ The goal of the segmentation is essentially to find connected components in $\mathcal{D}$ belonging to potentially moving objects. In the absence of explicit neighborhood relations in the point data, the region growing algorithm is performed on the input depth map based on point attribute similarity. Starting from the seed points, points are agglomerated whose position and normal are within given thresholds of vertex $\boldsymbol{v}(\boldsymbol{u})$ and normal $\boldsymbol{n}(\boldsymbol{u})$ of a neighbor. This results in the dynamics map $\mathcal{X}$, storing the connected components $x_i(\boldsymbol{u})$, that segments the current input frame into static and dynamic points. Fig. 5.8 illustrates the various steps from the initial ICP output of the camera tracking stage to a fully segmented object. Details about the region growing approach have been published in [KLL*13].

Updating the Global Model In the depth map fusion stage, model points that are merged with input points marked as dynamic are potentially demoted to unstable points using the following rule:

$$\text{if} \quad x_i(\boldsymbol{u}) \;\wedge\; \bar{c}_k \geq c_{\text{stable}} + 1 \quad \text{then} \quad \bar{c}_k \leftarrow 1 \tag{5.6}$$



(a) (b) (c)

Figure 5.8: Dynamics segmentation. 5.8(a) displays the initial ICP output. The white regions in 5.8(b) indicate the seed points for region growing after applying an erosion step in order to eliminate noise. In 5.8(c) the final segmentation result is displayed showing the full person who initially started moving hands.

Thus, the state change from static to dynamic is reflected immediately in the model. A critical aspect is the offset of +1 in Eq. (5.6): it ensures that any dynamic point that sufficiently grew in confidence (potentially because it is now static) is allowed to be added to the global model for at least one iteration; otherwise, a surface that has once been classified as dynamic would never be able to be re-added to the global model, as it would always be inconsistent with the model, leading to `no_corr` classification. Most often, however, dynamic points remain *unstable* and as such are not considered for camera pose estimation (see Sec. 5.3.4), which improves accuracy and robustness of $\boldsymbol{T}$ (see *moving person scene* in Sec. 5.6.3).

Conversely, input points of previously moving objects are merged into the model and gradually increase confidence counters until the object becomes part of the global model again. Thus, the state change from dynamic to static naturally requires a few input frames until enough confidence is gained, similar to the acquisition of new scene parts.

5.5 Occupancy Grid Integration

The point-based fusion approach presented so far produces high quality surface reconstruction results. However, in the area of robotics this type of environment model is often not sufficient and occupancy grids are required. Occupancy grids address the problem of generating consistent maps from noisy and uncertain measurement data. These 3D models represent not only occupied areas of the surrounding (i.e. the global model containing the 3D surface points) but also free and unknown parts. While the distinction between occupied and free space is crucial for safe navigation, the knowledge about unknown areas (i.e. areas where no information is available yet) is important for autonomous exploration.

5.5.1 Overview of Concept

The approach proposed by Hornung et al. [HWB*13] about OctoMap, a probabilistic 3D map representation, is adapted to point-based fusion. The OctoMap approach converts point clouds into 3D occupancy grids representing occupied, free, and unknown areas which allows for a collision-free navigation planning and the exploration of unknown areas. Hornung's approach is extended by integrating the dynamics classification of dynamically changing scenes (see Sec. 5.4) into the sensor model utilized in the occupancy grid representation. By doing this, the point-based fusion's surface reconstruction quality is combined with the advantages of acquiring the environment's state into a full voxel model. The integration comprises the following three main aspects: the data structure, its traversal, and the application of the sensor model.

Regular Grid Structure The occupancy grid represents its content in terms of a map of binary random variables, arranged in a regular spaced 3D grid. Each variable corresponds to the occupancy of the location it covers, i.e. voxels. A rather coarse voxel size ($\sim$ 2cm - 10cm) is preferred for covering the various states of the environment roughly since the surface is already represented in high quality by the point-based representation (e.g. see millimeter scale of telephone keys in Fig. 5.16(c) in results section).

Update Traversal A voxel's state is modeled probabilistically, this means the probability of each grid cell is estimated as being occupied. Since the environment is subject to change and new noisy depth maps are permanently accumulated into the system the occupancy grid is synchronously updated in order to reflect the real world situation. A ray casting approach is performed by traversing the occupancy grid along the viewing rays of $\boldsymbol{v}^{\mathrm{g}}(\boldsymbol{u})$ from the current (pre-processed) input depth map. Thus, the voxels along the sensor measurements of the current view are determined. Each voxel that the ray passes through lowers the voxel's occupancy probability, while the probability is raised for the voxel that the ray ends in. A voxel is considered as being *occupied* if a probability close to 1 is reached, whereas a voxel's state is *free* for a probability close to 0.

Probabilistic Sensor Model The sensor measurements are integrated into the occupancy grid by applying the traditional occupancy grid mapping approach by Moravec and Elfes [ME85]. Analogous to [ME85] and [TBF05] the recursive update rule for a voxel's probability P is transformed to the commonly used logOdds (L) notation:

$$\begin{aligned} \mathrm{L}(c \mid z_{1:i}) &= \mathrm{L}(c \mid z_{1:i-1}) + \mathrm{L}(c \mid z_i) && (5.7) \\ \text{with} \quad \mathrm{L}(c) &= \log\left[\frac{P(c)}{1-P(c)}\right], && (5.8) \end{aligned}$$

being $\mathrm{L}(c \mid z_{1:i})$ the probability of a voxel c to be occupied in logOdds notation given the set of all sensor measurements up to frame i, whereas z_i and $z_{1:i-1}$ denote the measurements of the i-th frame and the previous estimate respectively. logOdds values can be converted back into probability values and the other way round. However, logOdds values allow for faster updates due to less computational overhead for pre-computed sensor models and are therefore used directly in the occupancy grid.

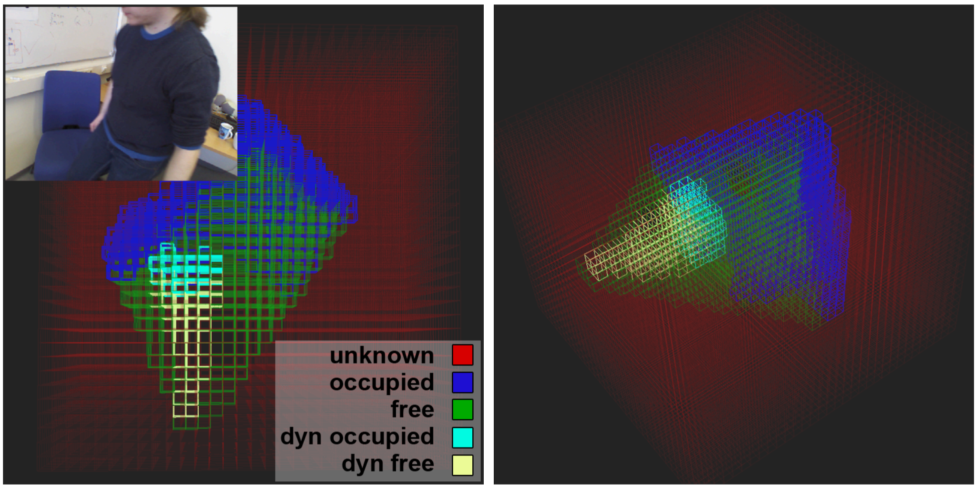
The inverse sensor model (in logOdds notation) defines how the voxels are updated along the viewing rays of the current input depth map:

$$\mathrm{L}(c \mid z_i) = \begin{cases} l_{\text{dynOcc}}, & \text{if ray ends in voxel} \wedge x(\boldsymbol{u}) \neq 0 \\ l_{\text{dynFree}}, & \text{else if ray passes through voxel} \wedge x(\boldsymbol{u}) \neq 0 \\ l_{\text{occ}}, & \text{else if ray ends in voxel} \\ l_{\text{free}}, & \text{else if ray passes through voxel} \end{cases}, \qquad (5.9)$$

with using explicit update values l_{dynOcc} and l_{dynFree} for dynamic data as marked in the dynamics map $\mathcal{X}$ (see Sec. 5.4.2). This ensures that the occupancy grid immediately reflects the dynamic parts of the current frame.

5.5.2 Implementation

The integration of the occupancy grid requires that the spatial extend of the scene and its boundaries are defined a priori when starting data acquisition. This is necessary for a proper setup of the regularly spaced grid structure. All voxels are initialized with a given uniform prior probability $P(c) = 0.5$, which corresponds to $L(c) = 0$ in logOdds notation. The inverse sensor model is based on the values proposed by [HWB*13], which has been originally developed for laser scanners, and is extended with regard to scene dynamics: $l_{\text{dynOcc}} = 4.6$ (0.99), $l_{\text{dynFree}} = -4.6$ (0.01), $l_{\text{occ}} = 0.85$ (0.7) and $l_{\text{free}} = -0.4$ (0.4). All values are in logOdds notation, the respective probability values are denoted in parentheses. The occupancy grid is updated performing the following steps:



(a) (b)

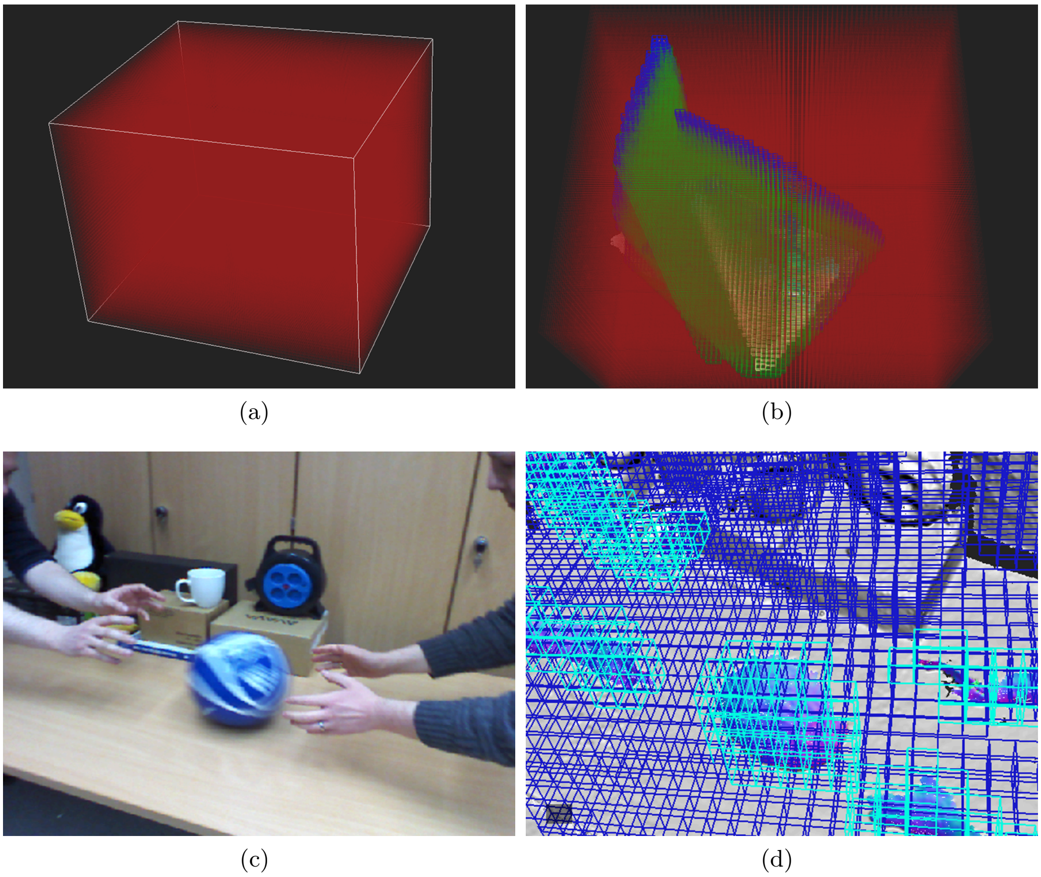
Figure 5.9: Occupancy grid states. A person is moving in an office room (see RGB image). The colored grid cells illustrate the occupancy grid's various states after the accumulation of several hundred range images (5.9(a): top view, 5.9(b): perspective view). For visualization purpose the cell size is enlarged to 10 cm.

1. A temporary grid structure $\text{tmpGrid}_{\boldsymbol{v}}$ (aligned to the occupancy grid) is computed for all vertices $\boldsymbol{v}^{\text{g}}(\boldsymbol{u})$ of the current input depth map according to [Gre12] using hash values and sorting. This way all measurements are assigned to grid cells which allows a fast spatial access.

2. The occupancy grid is traversed by adapting a fast 3D variant of the Bresenham algorithm [Bre65, AW87] to GPU processing. The casting of rays is performed in parallel and therefore atomic operations [NVI11] are utilized which ensures a proper integration of the individual measurements. If rays pass through voxels then voxels are updated by applying the inverse sensor model with values l_{free} and l_{dynFree}. If rays end within voxels then values l_{occ} and l_{dynOcc} are applied[1]. Voxels which are measured as *occupied* are preserved from receiving further updates of being *free* (invoked by neighboring rays which just pass through). Therefore, the respective cells in $\text{tmpGrid}_{\boldsymbol{v}}$ are checked as to whether they are free or contain a measurement from the current depth map.

3. Instead of using Eq. 5.7 directly the logOdds values of voxels which have been updated by non-dynamic data are clamped to thresholds $l_{\text{min}} = -2$ (0.12) and $l_{\text{max}} = 3.5$ (0.97) according to Yguel et al. [YAL07]. This lower and upper

[1] In practice, values l_{occ} and l_{free} are only applied to voxels which are traversed by viewing rays of $\boldsymbol{v}^{\text{g}}(\boldsymbol{u})$ associated with stable model points, i.e. $\bar{c}_k \geq c_{\text{stable}}$ (see Sec. 5.3.3). Thus, the occupancy grid's update strategy is rather conservative.


(a)

(b)

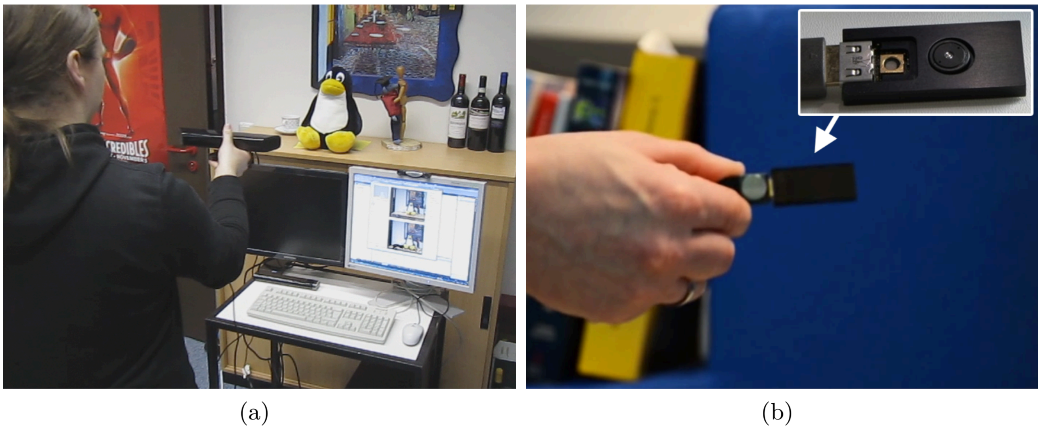
(c)

(d)

Figure 5.10: Occupancy grid overlay and 3D reconstruction (grid size: 4 cm). 5.10(a) and 5.10(b) show the initially unknown area as well as the grid's state after processing multiple range images during a ballgame. 5.10(d) displays a close-up of the 3D reconstruction with occupied grid cells overlayed. Cyan grid cells reflect dynamic parts of the currently processed scene (see RGB image).

> bound of a voxel's probability ensures that the voxel needs only a limited number of updates in order to change its state and thus it reflects changes in the environment quickly. Whereas voxels which have been affected by dynamic data are reset to their previous state before processing the next input depth map. This way, the dynamic data is represented by the occupancy grid only for a single frame.

In Fig. 5.9 and 5.10 the states of the occupancy grid are illustrated by various colors: red (unknown area), blue (occupied), and green (free). Voxels are considered *occupied* or *free* if their values have reached the upper or the lower bound, respectively. Dynamic data is reflected immediately in the occupancy grid by cyan (occupied) and yellow (free) if a voxel's estimate has been assigned with l_{dynOcc} or l_{dynFree}.



(a) (b)

Figure 5.11: Scene acquisition devices. In 5.11(a) Microsoft Kinect is shown and in 5.11(b) PMD Camboard XS is displayed during scene acquisition.

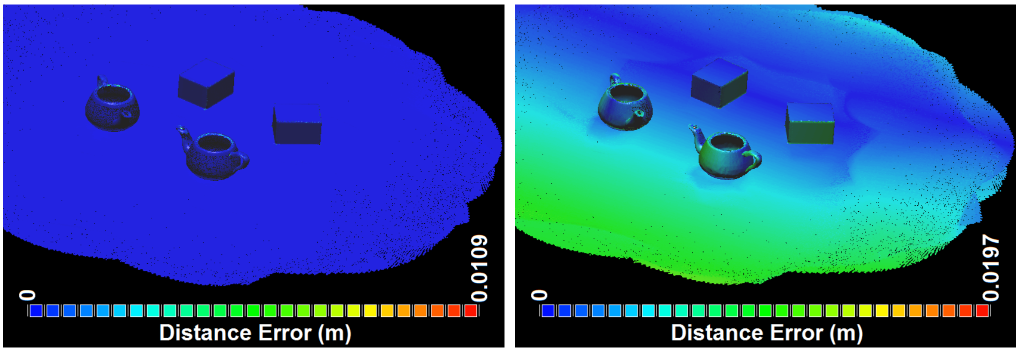
5.6 Results and Analysis

The proposed point-based fusion system has been tested on a variety of scenes. In this section qualitative and quantitative results are presented that show robust tracking and high quality reconstructions of a diverse set of scenes at varying scales (see Table 5.1). Also an experimental comparison to KinectFusion is depicted.

Scene Acquisition Setup The proposed point-based fusion algorithm has been integrated into an application with an underlying client-server architecture. Depth sensors are connected to the system and the acquisition of depth data is performed independently from its processing by using multiple threads. Thus, a good workload of CPU cores as well as the graphics unit is ensured. The system handles hard disks' data-saving and loading procedures with almost no latency. The playback of acquired range images allows for parameter changes under reproducible conditions. See Fig. 5.11 for an illustration of the acquisition setup. The GPU algorithms are programmed using [OKK09]. As shown in Sec. 5.3.3, the data fusion algorithm using MLS achieves better results in areas with sharp geometrical edges. However, the current implementation of the MLS method is not applicable to real-time processing. Therefore, the standard fusion method has been used for performance reasons implementing weighted averaging of points.

	#frames input/processed	#model-	Avg. timings [ms]		
	(fps acquisition/processed)	points	ICP	Dyn- Seg.	Fusion
Sim	950/950 (15/15)	467,200	18.90	2.03	11.50
Flowerpot	600/480 (30/24)	496,260	15.87	1.90	6.89
Teapot	1000/923 (30/27)	191,459	15.20	1.60	5.56
Large Office	11892/6704 (30/17)	4,610,800	21.75	2.39	13.90
Moving Person	912/623 (30/20)	210,500	15.92	3.23	16.61
Ballgame	1886/1273 (30/21)	350,940	16.74	3.15	17.66
PMD	4101/4101 (27/27)	280,050	10.70	0.73	3.06

Table 5.1: Acquisition scenes and timings. The table displays results from test scenes obtained on a PC equipped with an Intel i7 8-core CPU and an Nvidia GTX 680 GPU. Input frames have a size of 640×480 pixels, except for the *PMD* scene which uses a frame size of 200×200 pixels.



(a)

(b)

Figure 5.12: The synthetic reference scene *Sim*. 5.12(a) illustrates the error in the final global model based on ground truth camera transformations. 5.12(b) displays the final error based on ICP pose estimation.

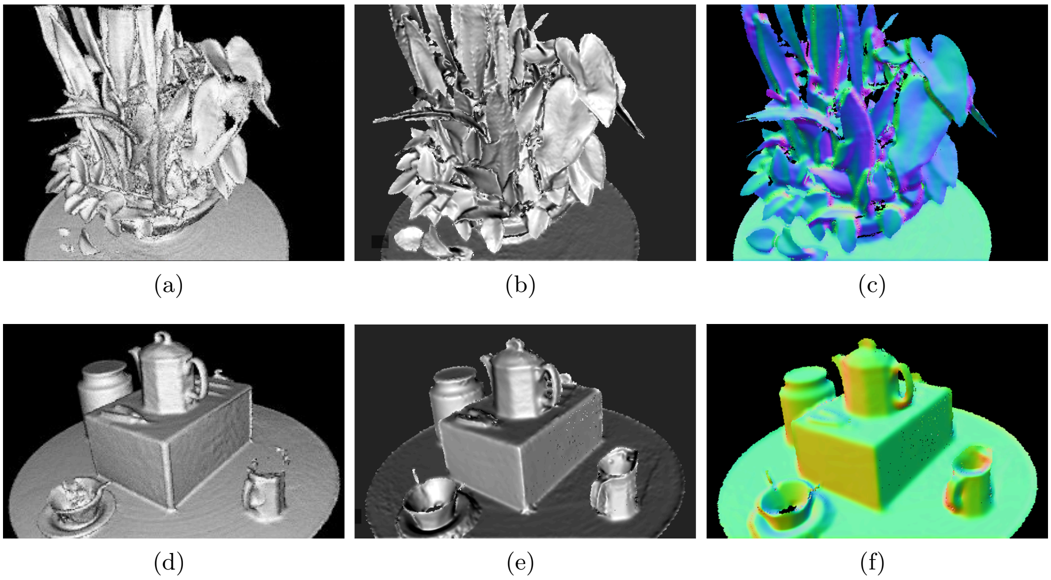


Figure 5.13: The scenes *Flowerpot* (top row) and *Teapot* (bottom row). 5.13(a) and 5.13(d) show reconstruction results of the original KinectFusion system. The other images show the proposed point-based fusion method: 5.13(b) and 5.13(e) display phong-shaded surfels; 5.13(c) and 5.13(f) show model points colored with surface normals.

5.6.1 Ground Truth Data Evaluation

Fig. 5.12 shows the synthetic reference scene *Sim*[1] which has been presented in Sec. 4.5.4 already. Ground truth camera transformations $\boldsymbol{T}_i^{\mathrm{GT}}$ is assigned to each depth map as well as ground truth scene geometry is provided. Using $\boldsymbol{T}_i^{\mathrm{GT}}$, the points in the resulting global model have a mean position error of 0.019 mm. This demonstrates only minimal error for the point-based data fusion approach. The camera transformations $\boldsymbol{T}_i$ obtained from ICP have a mean position error of 0.87 cm and a mean viewing direction error of 0.1°. This results in a mean position error of 0.20 cm for global model points.

5.6.2 Comparison

In this section the point-based approach is compared to the original KinectFusion algorithm [NIH*11]. The *Flowerpot* and *Teapot* scenes shown in Fig. 5.13 were recorded by Nguyen et al. [NIL12]. The objects are placed on a turntable which is rotated in front of a stationary Kinect camera. A Vicon optical tracking system is used

[1] The comparison is rendered using CloudCompare, `http://www.danielgm.net/cc/`.

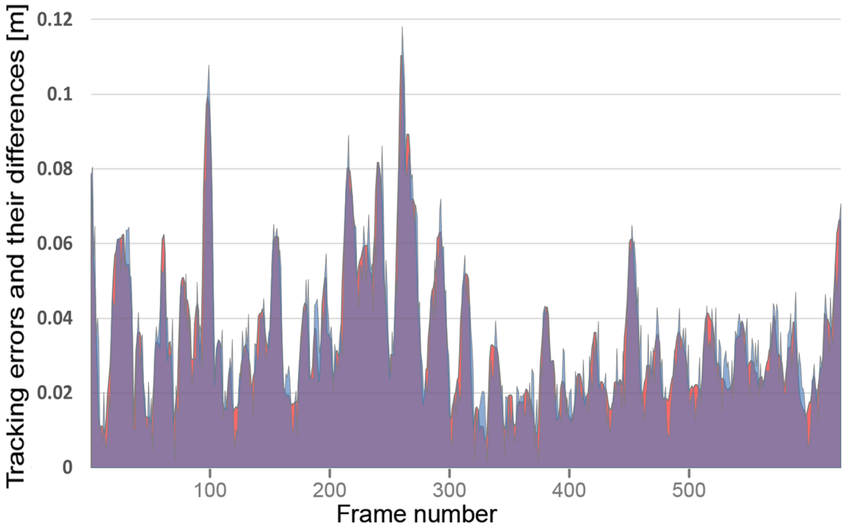


Figure 5.14: Tracking comparison. Tracking errors for the original KinectFusion system compared to the proposed point-based approach. Blue indicates that the error of KinectFusion exceeds the point-based approach. Where the error of point-based fusion exceeds the original approach, the gap is colored red.

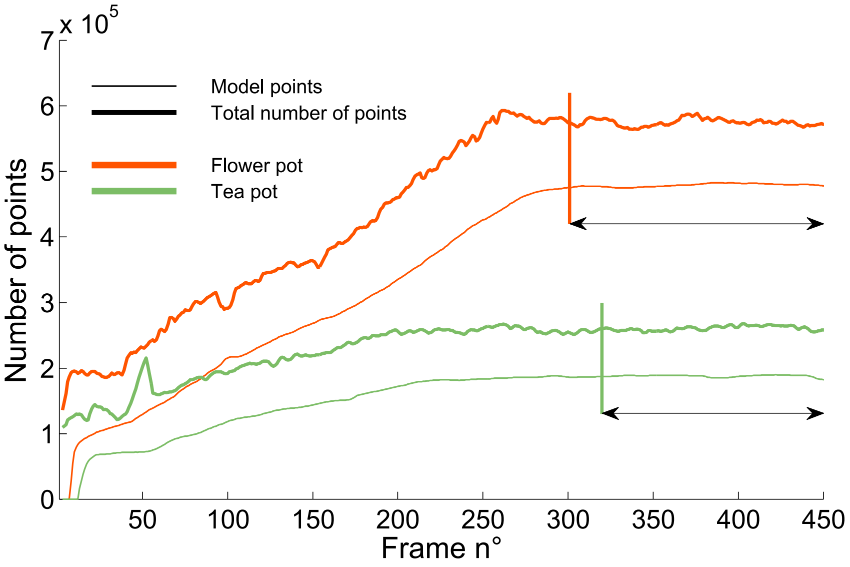


Figure 5.15: Turntable performance plot. The number of global model points stored on the GPU plotted over time for the *Flowerpot* and *Teapot* scenes. Note after the completion of one full turn of the turntable, the number of points converges instead of continuously growing.

for ground truth pose estimation of the Kinect, which are compared to ICP for the point-based method as well as the original KinectFusion system (see Fig. 5.14). The tracking results were computed on the *Flowerpot* sequence, by subtracting Vicon ground truth data from the resulting per frame 3D camera position. For each system, the error is computed as the absolute distance between the estimated camera position and the ground truth position (after aligning both coordinate spaces manually). Where the error of the original KinectFusion exceeds that of the new approach, the gap is colored blue. Where the error of the new method exceeds the original, the gap is colored red. The point-based method is similar in performance with the largest delta being $\sim$ 1 cm. Fig. 5.15 shows that the number of global model points for these scenes remains roughly constant after one full turn of the turntable. This demonstrates that new points are not continuously added; and the global model is refined but kept compact. It should be noted that a single Kinect camera input frame provides up to $307,200$ input points, but the total number of points in the final global teapot model is less than $300,000$.

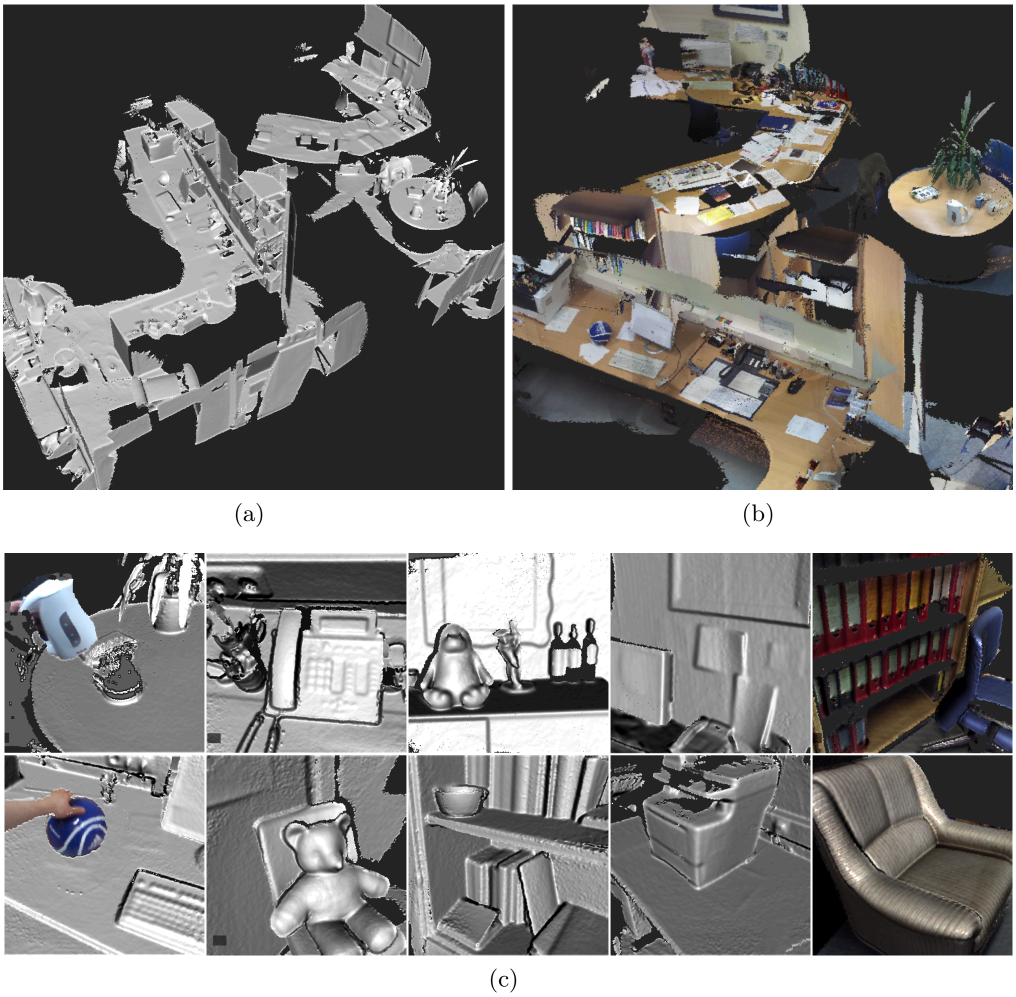
5.6.3 Experimental Scenes

Spatial extent: Large Office Scene

The *Large Office* scene shown in Fig. 5.16 consists of two rooms with a total spatial extent of approximately 10 m × 6 m × 2.5 m. A predefined volumetric grid with 32-bit voxels and 512 MB of GPU memory would result in a voxel size of more than 1 cm^3. In contrast, the point-based system does not define the spatial extent of the scene in advance: the global model grows as required. Furthermore, it does not limit the size of representable details; Fig. 5.16(c) shows close-ups of details on the millimeter scale (e.g. the telephone keys). The 4.6 million global model points reported in Tab. 5.1 can be stored in 110 MB of GPU memory using 3 floating point values for the point position, 2 for the normalized point normal, 1 for the radius, and one extra byte for a confidence counter. Additionally, RGB colors can be stored for each global point, to texture the final model (see Fig. 5.16(b) and 5.16(c) far right). Rather than merge RGB samples, the last one is stored currently.

Dynamics Detection: Moving Person and Ballgame Scenes

The detection and handling of scene dynamics is demonstrated by the *Moving Person* and the *Ballgame* scenes. In the *Moving Person* scene shown in Fig. 5.17, the person first sits in front of the sensor and is reconstructed before moving out of view. Since the moving person occupies a large part of the sensor's field of view, leaving only few reliable points for ICP, camera tracking fails with previous approaches (e.g. see



(a)

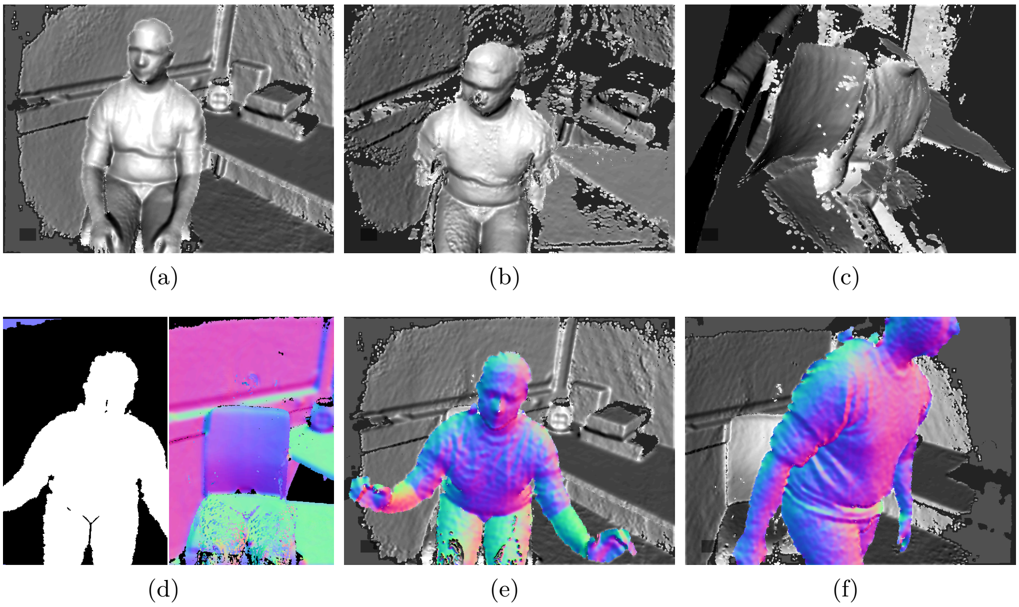
(b)

(c)

Figure 5.16: The *Large Office* scene, consisting of two large rooms and connecting corridors as displayed in 5.16(a) and 5.16(b). Close-ups are shown in 5.16(c) highlighting the high spatial resolution at millimeter scale, e.g. keys of the telephone.

Fig. 5.17(c) and Izadi et al. Fig. 8 [IKH*11]). However, the point-based fusion system segments the moving person and ignores dynamic scene parts in the ICP stage, thereby ensuring robustness to dynamic motion (see 5.17(e) and 5.17(f), dynamic parts are colored with surface normals).

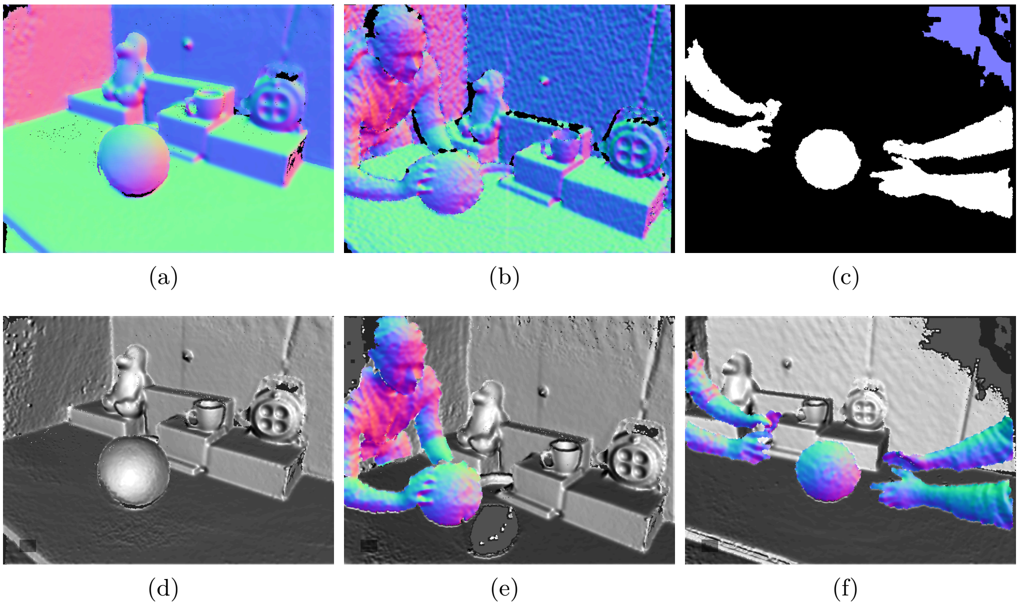
The *Ballgame* scene shown in Fig. 5.18 shows two people playing with a ball across a table. The region growing approach segments dynamics on the object level instead of just the point level: each person is recognized as dynamic even if only parts of their bodies are actually moving. This high-level dynamics information combined with the reliable global model is useful for applications requiring user interaction. Static objects that start moving are marked as dynamic and their model points are demoted



(a) (b) (c)

(d) (e) (f)

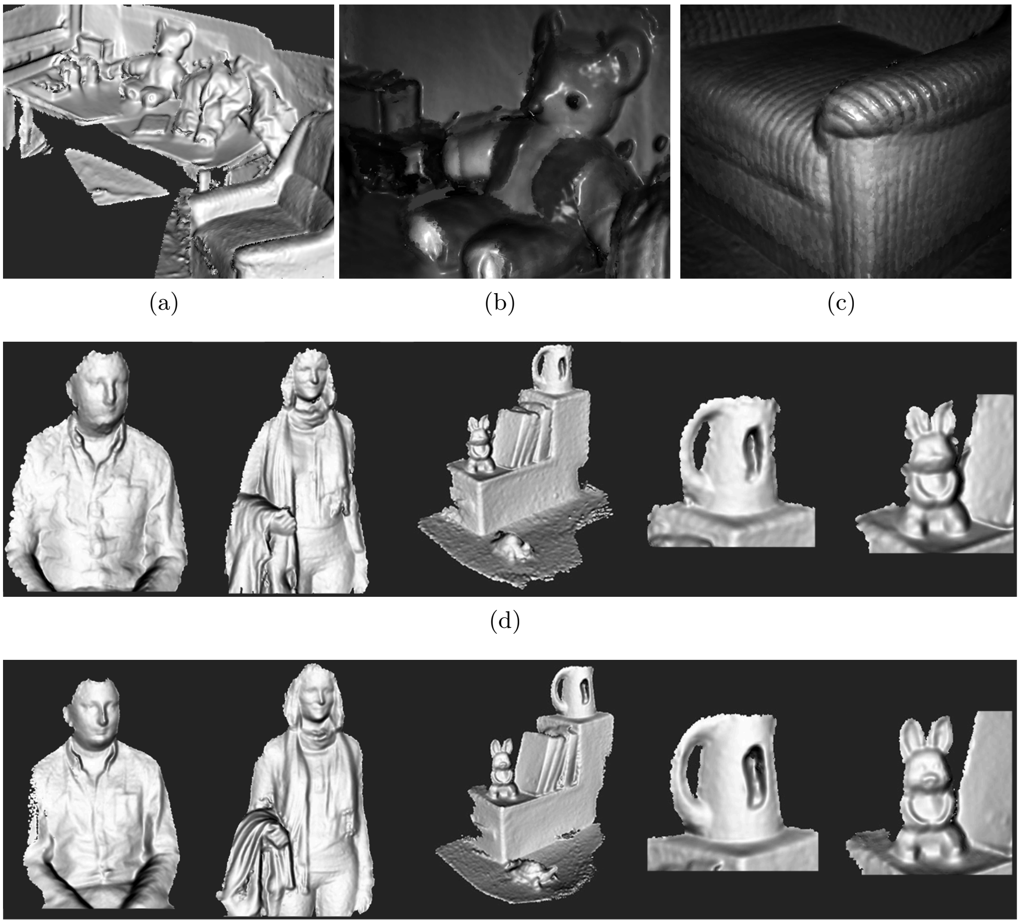
Figure 5.17: *Moving Person* scene. Person sits on a chair, is reconstructed, and then moves. Dynamic parts occupy the field-of-view and cause ICP errors with previous approaches, 5.17(b) and 5.17(c). Segmenting the dynamics and ignoring them during pose estimation (see 5.17(d)) allows increased robustness, 5.17(e) and 5.17(f).



(a) (b) (c)

(d) (e) (f)

Figure 5.18: *Ballgame* scene. Two people moving a ball across a table. 5.18(a): global model colored with surface normals; 5.18(b): raw input data of the previously static ball being picked up; 5.18(c): segmentation of dynamic parts; Bottom row shows reconstructed results (model points and dynamic parts).



(a) (b) (c)

(d)

(e)

Figure 5.19: *PMD ToF* scenes. The first row illustrates the scene listed in Table 5.1. 5.19(d) is recorded by a Camboard XS sensor with 160×120 pixels whereas the same scenes in 5.19(e) are acquired using a 100k prototype camera with 352×288 pixels.

to unstable status, while dynamic objects that stop moving eventually reach stable status in the global model when the observed points gain enough confidence.

Time-of-Flight Sensor Scenes

Most scenes shown throughout this chapter were acquired with a Microsoft Kinect camera configured in near mode, but the proposed point-based fusion method is agnostic to the type of sensor used. Fig. 5.19 shows experimental scenes recorded with a PMD ToF camera, which exhibits significantly different noise and error characteristics [KBKL10]. In these examples, the per-pixel amplitude information provided by PMD sensors is used in the computation of the sample confidence α (see Sec. 5.3.3):

a high amplitude value results in a high confidence of the respective depth sample whereas a low amplitude value implies less reliability. The close-ups shown in 5.19(b) and 5.19(c) use per-pixel amplitude values also for coloring.

The scenes in Fig. 5.19(d) are recorded with a PMD Camboard XS providing a resolution of 160×120 pixels whereas in Fig. 5.19(e) the same scenes are acquired using a PMD prototype camera which provides 100k depth samples per frame (352×288 pixels). Please note, the fine details of the facial contours and the clothing of the people in the high resolution scenes in comparison to the low resolution scenes. However, the point-based fusion approach is still capable of fusing low resolution depth maps with only $\sim 19,000$ depth samples each into a consistent model without considerable drift errors.

5.7 Discussion

A system for online 3D reconstruction has been presented in this chapter which demonstrates state-of-the-art reconstruction capabilities. The system is designed to allow for a single point-based representation to be used throughout the full processing pipeline, which closely fits the sensor input.

Despite the lack of a spatial data structure, the system still captures many benefits of volumetric data fusion (see KinectFusion [NIH*11]), allowing for accumulation of denoised 3D models over time that exploit redundant samples, model measurement uncertainty, and make no topological assumptions. This is achieved using the point-based fusion method based on [CL96]. In comparison to other point-based methods the proposed approach has its strengths in scene reconstructions at the presented scale, quality and speed. The basic system is extended by dealing with scene motion and dynamic updates of the global model. Furthermore, the integration of an occupancy grid allows for autonomous robot navigation in unknown areas. Experimental results have shown that the presented approach does not rely on a specific kind of depth sensor, rather the accumulation of data from structured light (e.g. Microsoft Kinect) as well as ToF cameras (e.g. PMD Camboard) leads to promising results.

However, the current implementation of the point-based fusion approach loses performance in large scenes with multi-million points (see large office scene in Table 5.1). Streaming the data which is currently not in use from GPU memory to CPU could help to balance the overall system. Additionally, the topic of sensor drift is not addressed yet by the presented approach which can become an issue in considerably large scenes.

Chapter 6

Conclusion and Outlook

3D range imaging systems gain more and more importance these days. Initially started out with Microsoft's Kinect sensor based on structured-light technology, a new generation of inexpensive depth sensors has entered the consumer market which deliver reliable depth data. Today's consumer depth sensors increasingly focus on the Time-of-Flight principle, e.g. Kinect's second generation and pmdtechnologies' Camboard series. Traditional application areas such as mobile robotics and also trendy application fields such as virtual and augmented reality incorporate range image data acquired by modern depth sensors. Recently Microsoft presented the Hololens which essentially mixes real world with holograms in a wearable glasses-like display device featuring a Time-of-Flight sensor.

The work presented in this thesis proposes a simulation approach for Time-of-Flight sensors as well as techniques for environment modeling. These approaches allow for the development of new algorithms which rely on realistic depth data under reproducible conditions as well as high quality environment capturing.

6.1 Summary and Conclusion

Chapter 2 presented a rough overview about current range imaging techniques. A more detailed introduction has been given to the Time-of-Flight principle and the relating photo mixing device (PMD) technology. The major sensor effects have been discussed, such as wiggling, noise, motion blur, and mixed phases. The second main topic of this thesis, namely environment modeling, has been introduced by presenting an overview about the processing steps in the area of 3D reconstruction and the respective approaches. Finally, the chapter closed with a review about the programmable graphics pipeline and its application to real-time data processing.

In Chapter 3 an approach for real-time simulation of Time-of-Flight sensors has been proposed. The major sensor artifacts such as wiggling, flying pixels, and motion blur are simulated by a physically-based sensor model which aims at the generation of phase images. The theoretical model has been integrated into a simulation framework which targets real-time simulation by the use of graphics processing units (GPUs). In particular, a PMD sensor has been implemented and the simulation results in static as well as dynamic scenes corresponded to a great extent to real sensor data. Various scenes have been simulated and the results have been processed by volumetric environment modeling (for simulating the bin-picking application) as well as point-based fusion (for ground truth data alignment in order to evaluate ICP quality). However, artifacts like multipath-interference, which are successfully simulated by Meister et al. [MNK13] in an offline approach, are not feasible. These effects demand for more sophisticated rendering techniques such as ray-tracing algorithms.

In Chapter 4 the hierarchical volume data structure has been presented. While supporting boolean operations for merging and subtraction of sub-volumes the data structure is completely managed by the GPU. The accumulation of hundreds of range images into the data structure has been demonstrated. Furthermore, the approach has been integrated into a robot's bin picking application, which demos the robot's ability to empty a box of tubes autonomously. However, runtime observations revealed weak spots of the approach: the data structure's real-time applicability strongly depends on the lateral resolution of the range images, e.g. the processing of 640x480 pixels could not reach interactive frame-rates anymore. The proposed implementation uses shader based GPGPU techniques. However, an experimental comparison of CUDA's stream expansion and compaction algorithms [HSO07] and the geometry program usage with transform feedback does not show a significant performance difference. Nevertheless, a CUDA implementation would decrease the complexity of the program structure. In fact, the management of the tree structure itself causes computational overhead. Furthermore, high quality reconstruction results have not been achieved by solely applying the volumetric data structure during rendering. Due to the hierarchical approximation the approach lost data precision and thus a smooth surface representation is hardly feasible.

The point-based fusion approach presented in Chapter 5 basically made up for the drawbacks of the volumetric accumulation approach. Depth sensors with VGA resolution and scenes of larger scales, e.g. the office scene with dimensions of 10 m × 6 m × 2.5 m, have been addressed at interactive frame-rates. The point-based representation throughout the full processing pipeline allowed for high quality reconstruction results. The basic system has been extended in order to handle dynamically changing scenes which improved the ICP-based tracking significantly. Dynamic changes in the scene are automatically detected and the global reconstruction model is updated accordingly. Additionally, occupancy grids have been integrated into the point-based fusion system which allowed for autonomous robot navigation in unknown areas while simultaneously preserving detailed surface representation. Experimental results have shown that high quality reconstruction results are not only obtained by depth sensors providing high lateral resolution and reduced sensor noise but also by sensors with low lateral resolution, such as ToF devices. However, the processing of scenes which contain several million model points has slowed down. While interactive frame-rates are still feasible real-time processing could not be guaranteed. This is mainly due to an unoptimized implementation which maintained the global model solely on the GPU and which did not support CPU-GPU data streaming in order to keep the GPU data processing compact. It should be noted that the topic of sensor drift has not been addressed yet. Although considerable drift errors did no become obvious in small scenes the drift reduction in large environments would increase the overall quality.

6.2 Outlook

The upcoming generation of depth sensors will be even more miniaturized compared to today's sensors clearly targeting mobile applications on devices such as smartphones and tablets. One important aspect is the reduction of the overall power consumption on mobile devices. This can be achieved by decreasing the chip size and the frame-rate as well as the exposure time of the sensor's illumination unit which may have a significant impact on the quality of the data. Algorithms need to learn how this data can be handled successfully.

The ToF sensor simulation approach proposed in this thesis is a starting point for simulating the next generation of depth sensors in order to prepare the applications and algorithms appropriately. Reduced frame-rates at a diminished lateral resolution as well as a higher amount of depth noise can be easily accomplished by the ToF sensor simulation approach. However, the extensions presented by Lambers et al. [LHK15], who enhance the current simulation model by using physical units throughout the simulation process, would significantly upscale the benefit of the simulation framework. Furthermore, the integration of their lens vignetting effects would help to make the simulation data look as real as possible. Another topic which is worth to be addressed is the simulation of data acquired by multiple frequencies since this will help to detect and to eliminate multipath effects [Fuc10, DGC*11, FSK*14].

In the context of range data accumulation various techniques could be applied to further improve the performance and 3D reconstruction quality of the work presented in this thesis. The current implementation of the point-based fusion approach still suffers from performance losses in considerable large scenes with several million points. Mechanisms for streaming subsets of points (from GPU to CPU) especially once they are significantly far away from the current pose would help to increase performance. Clearly the point-based data would be low overhead in terms of CPU-GPU bandwidth.

A further way to reduce the overall computation time would be the reutilization of the rendering pass: for camera pose estimation as well as for user feedback visualization the model points are rendered as surfels. The processing of a large number of model points is a time consuming operation in the vertex shader stage and re-using the rendering results would significantly reduce the processing time. One possibility would be to render the various attributes for the ICP algorithm as well as for the visual feedback into separate render targets. Furthermore, the rendering quality could be improved by incorporating the work presented by Lefloch et al. [LWK15] about anisotropic point-based fusion which features also a reduced memory footprint of point attributes and thus allows for the storage of additional properties such as accumulated anisotropic noise.

Another issue is the occurrence of sensor drift: the presented accumulation approach does not tackle this topic explicitly and instead focuses on the data representation itself. Drift in larger environments can become an issue and remains an interesting direction for improvements. Anyway, after loop closure detection the point-based representation might be more amenable to correction and post processing optimization, rather than resampling a dense voxel grid. Recently, Whelan et al. [WLSM*15] published ElasticFusion, an approach which extends the basic point-based fusion technique with regard to online loop closure on point data.

As it has been experimentally shown: the application of a moving least squares algorithm for the correction of the input data has a positive impact on the data quality of the global model especially in areas with sharp edges. A GPU based version of the current CPU implementation would be necessary for its real-time applicability. Thus, the input data would approximate the actual scene geometry in a better way which reduces the noise in the data significantly.

Finally it should be noted that the solutions presented in this thesis are not limited to the specific setting they have been applied to. In fact, the ToF sensor simulation is not restricted to the simulation of PMD sensors. The working principle should be applicable also to other ToF manufacturers. Furthermore, the implementation of alternative simulation concepts is also possible in the sensor simulation framework. The same flexibility applies to the accumulation techniques for environment modeling: the algorithms are not restricted to the utilized depth sensors. Any kind of range images should be qualified for being processed by the proposed 3D reconstruction methods.

Bibliography

[AF87] Ayache N., Faverjon B.: Efficient registration of stereo images by matching graph descriptions of edge segments. In *J. of Computer Vision* (1987), pp. 107–131.

[AMH02] Akenine-Moller T., Haines E.: *Real-Time Rendering.* A. K. Peters, Ltd., Natick, MA, USA, 2002.

[ASG*01] Allen P. K., Stamos I., Gueorguiev A., Gold E., Blaer P.: AVENUE: Automated site modeling in urban environments. In *Int. Conf. on 3D Digital Imaging and Modeling* (2001), pp. 357–364.

[Aut06] Autodesk: Maya and 3D Studio Max. `http://www.alias.com/`, 2006.

[AW87] Amanatides J., Woo A.: A fast voxel traversal algorithm for ray tracing. In *Eurographics* (1987), pp. 3–10.

[BDL95] Blais G., D. Levine M.: Registering multiview range data to create 3D computer objects. *IEEE Trans. Pattern Anal. Mach. Intell. 17*, 8 (Aug. 1995), 820–824.

[Ben75] Bentley J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept. 1975), 509–517.

[BF82] Barnard S. T., Fischler M. A.: Computational stereo. *ACM Computing Surveys 14*, 4 (Dec. 1982), 553–572.

[Bli77] Blinn J. F.: Models of light reflection for computer synthesized pictures. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (1977), ACM, pp. 192–198.

[BM92] Besl P. J., McKay N. D.: A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (Feb. 1992), 239–256.

[BMR*99] Bernardini F., Mittleman J., Rushmeier H., Silva C., Taubin G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Computer Graphics 5*, 4 (Oct. 1999), 349–359.

[BR02] Bernardini F., Rushmeier H. E.: The 3D model acquisition pipeline. *Computer Graphics Forum 21*, 2 (2002), 149–172.

[Bre65] Bresenham J. E.: Algorithm for computer control of a digital plotter. *IBM Syst. J. 4*, 1 (Mar. 1965), 25–30.

[BS97] Benjemaa R., Schmitt F.: Fast global registration of 3D sampled surfaces using a mini-buffer technique. In *3DIM97* (1997).

[CHH02] Carr N., Hall J., Hart J.: The ray engine. In *Proc. Conf. on Graphics Hardware* (2002), pp. 37–46.

[CHL04] Coombe G., Harris M., Lastra A.: Radiosity on graphics hardware. In *Proc. Graphics Interface* (2004), pp. 161–168.

[CL96] Curless B., Levoy M.: A volumetric method for building complex models from range images. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (1996), pp. 303–312.

[CM91] Chen Y., Medioni G.: Object modeling by registration of multiple range images. In *Proc. IEEE Int. Conf. on Robotics and Automation* (1991), pp. 2724–2729.

[DGC*11] Dorrington A., Godbaz J., Cree M., Payne A., Streeter L.: Separating true range measurements from multi-path and scattering interference in commercial range cameras. In *SPIE Volume 7864, Three-Dimensional Imaging, Interaction, and Measurement* (2011).

[FB81] Fischler M. A., Bolles R. C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM 24*, 6 (June 1981), 381–395.

[FHK*10] Fuchs S., Haddadin S., Keller M., Parusel S., Kolb A., Suppa M.: Cooperative bin picking with ToF-camera and impedance controlled DLR light-weight robot III. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)* (October 2010), IEEE/RSJ, pp. 4862–4867.

[FK03] Fernando R., Kilgard M. J.: *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[FL00] Fang S., Liao D.: Fast CSG voxelization by frame buffer pixel mapping. In *Proc. IEEE Symp. Volume Visualization* (2000), pp. 43–48.

[FS05] Foley T., Sugerman J.: Kd-tree acceleration structures for a GPU raytracer. In *Proc. Graphics Hardware* (2005), pp. 15–22.

[FSK*14] Freedman D., Smolin Y., Krupka E., Leichter I., Schmidt M.: SRA: Fast removal of general multipath for tof sensors. In *Proc. of the European Conf. on Computer Vision (ECCV)* (2014).

[Fuc10] Fuchs S.: Multipath interference compensation in Time-of-Flight camera images. In *Int. Conf. on Pattern Recognition, ICPR* (2010), pp. 3583–3586.

[FZ01] Früh C., Zakhor A.: 3D model generation for cities using aerial photographs and ground level laser scans. In *IEEE Conf. on Computer Vision and Pattern Recognition* (2001), pp. 31–38.

[GKS00] Gopi M., Krishnan S., Silva C.: Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum* (2000).

[GPF10] Gallup D., Pollefeys M., Frahm J.-M.: 3D reconstruction using an n-layer heightmap. In *Pattern Recognition*. Springer, 2010, pp. 1–10.

[GPSS07] Günther J., Popov S., Seidel H.-P., Slusallek P.: Realtime ray tracing on GPU with BVH-based packet traversal. In *Proc. Interactive Ray Tracing* (2007), pp. 113–118.

[Gre12] Green S.: *Particle Simulation using CUDA*. Tech. rep., Santa Clara, CA, USA, 2012.

[GY06] G. Yahav G. Iddan D. M.: 3D imaging camera for gaming application. In *Int. Conf. on Consumer Electronics(ICCE)* (2006).

[GYB04] Gokturk S. B., Yalcin H., Bamji C.: A Time-of-Flight depth sensor - system description, issues and solutions. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshop* (2004).

[HBT03] Hähnel D., Burgard W., Thrun S.: Learning compact 3D models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems 44*, 1 (2003), 15–27.

[HDD*92] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.: Surface reconstruction from unorganized points. *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH) 26*, 2 (1992).

[HDH*01] Hebert M., Deans M., Huber D., Nabbe B., Vandapel N.: Progress in 3-D mapping and localization. In *In. Symposium on Intelligent Robotic Systems* (July 2001).

[HKH*12] Henry P., Krainin M., Herbst E., Ren X., Fox D.: RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *Int. J. Robotics Research 31* (Apr. 2012), 647–663.

[HKRs*06] Hadwiger M., Kniss J. M., Rezk-salama C., Weiskopf D., Engel K.: *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[HLK13a] Hoegg T., Lefloch D., Kolb A.: Real-time motion artifact compensation for PMD-ToF images. In *Time-of-Flight and Depth Imaging, LNCS – INMWorkshop* (2013), vol. 8200, p. 273.

[HLK13b] Hoegg T., Lefloch D., Kolb A.: Time-of-Flight camera based 3D point cloud reconstruction of a car. *Computers in Industry (3D Imaging) 64*, 9 (2013), 1099 – 1144.

[HMHB06] Hubo E., Mertens T., Haber T., Bekaert P.: The quantized kd-tree: Efficient ray tracing of compressed point clouds. *Symp. on Interactive Ray Tracing* (2006), 105–113.

[HSHH07] Horn D., Sugerman J., Houston M., Hanrahan P.: Interactive k-d tree GPU raytracing. In *Proc. I3D* (2007), pp. 167–174.

[HSO07] Harris M., Sengupta S., Owens J.: Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*. Addison Wesley, 2007.

[HSS*05] Hadwiger M., Sigg C., Scharsach H., Bühler K., Gross M.: Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum 24*, 3 (2005), 303–312.

[HWB*13] Hornung A., Wurm K. M., Bennewitz M., Stachniss C., Burgard W.: Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots* (2013).

[IKH*11] Izadi S., Kim D., Hilliges O., Molyneaux D., Newcombe R., Kohli P., Shotton J., Hodges S., Freeman D., Davison A., Fitzgibbon A.: KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. ACM Symp. User Interface Softw. & Tech.* (2011), pp. 559–568.

[JVC] JVC ZCam 3D. `http://pro.jvc.com/prof/attributes/features.jsp?model_id=MDL101309`. Accessed: 2015-03-30.

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: Poisson surface reconstruction. In *Proc. EG Symp. Geom. Proc.* (2006).

[KBKL10] Kolb A., Barth E., Koch R., Larsen R.: Time-of-Flight cameras in computer graphics. *Computer Graphics Forum 29*, 1 (2010), 141–159.

[KCK09] Keller M., Cuntz N., Kolb A.: Interactive dynamic volume trees on the GPU. In *Proc. Vision, Modeling and Visualization* (November 2009).

[KE12] Khoshelham K., Elberink S. O.: Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors 12*, 2 (2012), 1437–1454.

[KH10] Kirk D. B., Hwu W.-m. W.: *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[Kin] Mircosoft Kinect for Windows. `http://www.microsoft.com/en-us/kinectforwindows`. Accessed: 2015-03-30.

[KK09] Keller M., Kolb A.: Real-time simulation of Time-of-Flight sensors. *J. Simulation Practice & Theory 17* (2009), 967–978.

[Kle08] Klein S.: *Entwurf und Untersuchung von integrierten Ausleseschaltungen für hochauflösende 3D-Bildsensoren auf PMD-Basis zur Unterdrückung von Fixed-Pattern-Noise (FPN).* Master's thesis, University of Siegen, 2008.

[KLL*13] Keller M., Lefloch D., Lambers M., Izadi S., Weyrich T., Kolb A.: Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)* (June 2013).

[KOKP07] Keller M., Orthmann J., Kolb A., Peters V.: A simulation framework for Time-Of-Flight sensors. In *Proc. of the Int. IEEE Symp. on Signals, Circuits & Systems (ISSCS)* (July 2007), vol. 1, IEEE CAS Society, pp. 125 – 128.

[KS04] Kuzu Y., Sinram O.: Volumetric model refinement by shell carving. In *Proceedings of XXth ISPRS Congress* (2004), pp. 148 – 153.

[KS06] Kuhnert K.-D., Stommel M.: Fusion of stereo-camera and PMD-camera data for real-time suited precise 3D environment reconstruction. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)* (2006), pp. 4780–4785.

[KVN07] Kuijk M., Van Nieuwenhove D.: TOF rangefinding with large dynamic range and enhanced background radiation suppression, 2007. US Patent 7,268,858.

[KW03] Kruger J., Westermann R.: Acceleration techniques for GPU-based volume rendering. In *Proc. IEEE Conf. on Visualization* (2003).

[LAM01] Lext J., Akenine-Möller T.: Towards rapid reconstruction for animated ray tracing. In *Proc. Eurographics, Short-Paper* (2001).

[LAM06] Larsson T., Akenine-Möller T.: A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics 30*, 3 (2006), 451–460.

[Lan00] Lange R.: *3D Time-Of-Flight Distance Measurement with Custom Solid-State Image Sensors in CMOS/CCD-Technology.* PhD thesis, University of Siegen, 2000.

[LC87] Lorensen W., Cline H.: Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (1987), ACM, pp. 163–169.

[LC94] Li A., Crebbin G.: Octree encoding of objects from range images. *Pattern Recognition 27*, 5 (1994), 727 – 739.

[LEC*01] Liu Y., Emery R., Chakrabarti D., Burgard W., Thrun S.: Using EM to learn 3D models of indoor environments with mobile robots. In *Proc. of Int. Conf. on Machine Learning (ICML)* (2001), pp. 329–336.

[Lev88] Levoy M.: Display of surfaces from volume data. *IEEE Comput. Graph. Appl. 8*, 3 (May 1988), 29–37.

[LGS*09] Lauterbach C., Garland M., Sengupta S., Luebke D., Manocha D.: Fast BVH Construction on GPUs. *Computer Graphics Forum 28*, 2 (2009), 375–384.

[LHK13] Lefloch D., Hoegg T., Kolb A.: Real-time motion artifacts compensation of ToF sensors data on GPU. In *SPIE Defense, Security – Three-Dimensional Imaging, Visualization, and Display* (May 2013), pp. 87380U – 87380U–7.

[LHK15] Lambers M., Hoberg S., Kolb A.: Simulation of Time-of-Flight sensors for evaluation of chip layout variants. *IEEE Sensors Journal 15*, 7 (July 2015), 4019–4026.

[Lin10] Lindner M.: *Calibration and Realtime Processing of Time-of-Flight Range Data.* Phd thesis, Computer Graphics Group, University of Siegen, December 2010.

[LK06] Lindner M., Kolb A.: Lateral and depth calibration of PMD-distance sensors. In *Advances in Visual Computing* (2006), vol. 2, Springer, pp. 524–533.

[LK07] Lindner M., Kolb A.: Calibration of the intensity-related distance error of the PMD TOF-Camera. In *SPIE: Intelligent Robots and Computer Vision XXV* (2007), vol. 6764, p. 67640W.

[LK09] Lindner M., Kolb A.: Compensation of motion artifacts for Time-of-Flight cameras. In *Proc. Dynamic 3D Imaging* (2009), Springer, pp. 16–27.

[LKHW05] Lefohn A., Kniss J., Hansen C., Whitaker R.: A streaming narrow-band algorithm: interactive computation and visualization of level sets. In *ACM SIGGRAPH Courses Notes* (2005), p. 243.

[LKR08] Lindner M., Kolb A., Ringbeck T.: New insights into the calibration of ToF sensors. In *IEEE Conf. on Computer Vision and Pattern*

Recognition, Workshop on ToF Camera based Computer Vision (TOF-CV) (May 2008), IEEE, pp. 1–5.

[LLK08] Lindner M., Lambers M., Kolb A.: Data fusion and edge-enhanced distance refinement for 2D rgb and 3D range images. *Int. J. on Intell. Systems and Techn. and App. (IJISTA), Issue on Dynamic 3D Imaging 5*, 1 (2008), 344 – 354.

[LPC*00] Levoy M., Pulli K., Curless B., Rusinkiewicz S., Koller D., Pereira L., Ginzton M., Anderson S., Davis J., Ginsberg J., et al.: The digital michelangelo project: 3D scanning of large statues. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (2000), pp. 131–144.

[LWK15] Lefloch D., Weyrich T., Kolb A.: Anisotropic point-based fusion. In *Fusion* (2015).

[MBR*00] Matusik W., Buehler C., Raskar R., Gortler S. J., McMillan L.: Image-based visual hulls. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (2000), pp. 369–374.

[ME85] Moravec H., Elfes A. E.: High resolution maps from wide angle sonar. In *Proc. of IEEE Int. Conf. on Robotics and Automation* (March 1985), pp. 116 – 121.

[Mea82] Meagher D.: Geometric modeling using octree encoding. In *Computer Graphics and Image Processing* (1982), vol. 19, pp. 129 – 147.

[MES] MESA Imaging AG: Sr4000 user manual. version 2.0. `http://downloads.mesa-imaging.ch/dlm.php?fname=customer/Customer_CD/SR4000_Manual.pdf`. Accessed: 2015-03-30.

[MKF*05] Möller T., Kraft H., Frey J., Albrecht M., Lange R.: Robust 3D measurement with PMD sensors. In *Proc. of Range Imaging Research Day at ETH* (2005), pp. 3–906467.

[MNK13] Meister S., Nair R., Kondermann D.: Simulation of Time-of-Flight sensors using global illumination. In *Proc. Vision, Modeling and Visualization* (2013), pp. 33–40.

[MP79] Marr D., Poggio T.: A computational theory of human stereo vision. *Proc. of the Royal Society of London. Series B, Biological Sciences 204*, 1156 (1979), 301–328.

[Neu97] Neugebauer P. J.: Geometrical cloning of 3D objects via simultaneous registration of multiple range images. In *Proc. of Int. Conf. on Shape Modeling and Applications* (1997).

[nic] RivaTuner - A. Nicolaychuk. `www.guru3d.com/rivatuner/`. Accessed: 2015-03-30.

[NIH*11] Newcombe R., Izadi S., Hilliges O., Molyneaux D., Kim D., Davison A., Kohli P., Shotton J., Hodges S., Fitzgibbon A.: KinectFusion: Real-time dense surface mapping and tracking. In *Proc. IEEE Int. Symp. Mixed and Augm. Reality* (2011), pp. 127–136.

[NIL12] Nguyen C., Izadi S., Lovell D.: Modeling Kinect sensor noise for improved 3D reconstruction and tracking. In *Proc. Int. Conf. 3D Imaging, Modeling, Processing, Vis. & Transmission* (2012), pp. 524–530.

[NSH03] Nüchter A., Surmann H., Hertzberg J.: Automatic model refinement for 3D reconstruction with mobile robots. In *3DIM* (2003), IEEE Computer Society, pp. 394–401.

[NVI11] NVIDIA: *NVIDIA CUDA C Programming Guide*, June 2011.

[NZIS13] Nießner M., Zollhöfer M., Izadi S., Stamminger M.: Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 169:1–169:11.

[OHL*08] Owens J., Houston M., Luebke D., Green S., Stone J., Phillips J.: GPU computing. *Proceedings of the IEEE 96*, 5 (May 2008), 879–899.

[OK83] Ohta Y., Kanade T.: *Stereo by intra - and inter-scanline search using dynamic programming.* Tech. Rep. CMU-CS-83-162, Carnegie-Mellon University. Computer science. Pittsburgh, 1983.

[OKK09] Orthmann J., Keller M., Kolb A.: Integrating GPGPU functionality into scene graphs. In *Proc. Vision, Modeling and Visualization* (November 2009).

[OLG*07] Owens J., Luebke D., Govindaraju N., Harris M., Krüger J., Lefohn A., Purcell T.: A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum 26*, 1 (2007), 80–113.

[OLK*04] Oggier T., Lehmann M., Kaufmann R., Schweizer M., Richter M.: An all-solid-state optical range camera for 3D real-time imaging with sub-centimeter depth resolution (swissranger). In *SPIE 5249, Optical Design and Engineering* (2004).

[Owe05] Owens J.: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation.* Addison-Wesley Professional, 2005, ch. Streaming Architectures and Technology Trends, pp. 457–470.

[PDC*03] Purcell T., Donner C., Cammarano ., Jensen H., Hanrahan P.: Photon mapping on programmable graphics hardware. In *Proc. Graphics Hardware* (2003), pp. 41–50.

[PF05] Pharr M., Fernando R.: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation.* Addison-Wesley Professional, 2005.

[PHKL06] Peters V., Hasouneh F., Knedlik S., Loffeld O.: Simulation of PMD based self-localization of mobile sensor nodes or robots. In *Symp. on Simulation Technique (ASIM))* (2006).

[PLHK07] Peters V., Loffeld O., Hartmann K., Knedlik S.: Modeling and bistatic simulation of a high resolution 3D PMD-camera. In *Proc. Congress on Modelling and Simulation (EUROSIM)* (2007).

[PMD] PMD Technologies: Camboard pico s - spec sheet. `http://www.pmdtec.com/html/pdf/PMD_RD_Brief_CB_pico_71.19k_V0103.pdf`. Accessed: 2015-03-30.

[PMF85] Pollard S., Mayhew J., Frisby J.: PMF: A stereo correspondence algorithm using a disparity gradient limit. *Perception 4*, 14 (1985), 449–470.

[Pro] Google Project Tango. `https://developers.google.com/project-tango`. Accessed: 2015-03-30.

[PSL*98] Parker S., Shirley P., Livnat Y., Hansen C., Sloan P.-P.: Interactive ray tracing for isosurface rendering. In *Proc. IEEE Conf. on Visualization* (1998), IEEE, pp. 233–238.

[Pul99] Pulli K.: Multiview registration for large data sets. In *Proc. of Int. Conf. on 3-D Digital Imaging and Modeling* (1999), pp. 160–168.

[PZVBG00] Pfister H., Zwicker M., Van Baar J., Gross M.: Surfels: Surface elements as rendering primitives. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (2000), pp. 335–342.

[RA99] Reed M. K., Allen P. K.: 3-d modeling from range imagery: An incremental method with a planning component. In *Image and Vision Computing* (1999), pp. 76–83.

[Rap07] Rapp H.: *Experimental and Theoretical Investigation of Correlating TOF-Camera Systems.* Master's thesis, University of Heidelberg, Germany, 2007.

[RBM*07] Rusu R., Blodow N., Marton Z., Soos M., Beetz M.: Towards 3D object maps for autonomous household robots. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS)* (2007).

[RFHJ07] Rapp H., Frank M., Hamprecht F., Jähne B.: A theoretical and experimental investigation of the systematic errors and statistical uncertainties of Time-of-Flight cameras. In *Int. J. on Intell. Systems and Techn. and App. (IJISTA), Issue on Dynamic 3D Imaging* (2007).

[RGW*03] Röttger S., Guthe S., Weiskopf D., Ertl T., Straßer W.: Smart hardware-accelerated volume rendering. In *EG/IEEE Symopsium on Visualization* (2003), pp. 231–238.

[RH07] Ringbeck T., Hagebeuker B.: A 3D time of flight camera of object detection. In *Optical 3D Measurement Techniques* (Zurich, 2007).

[RHHL02] Rusinkiewicz S., Hall-Holt O., Levoy M.: Real-time 3D model acquisition. *ACM Trans. Graph. (Proc. SIGGRAPH) 21*, 3 (2002), 438–446.

[RL00] Rusinkiewicz S., Levoy M.: QSplat: A multiresolution point rendering system for large meshes. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (2000), pp. 343–352.

[RMB*08] Rusu R. B., Marton Z. C., Blodow N., Dolha M., Beetz M.: Towards 3D point cloud based object maps for household environments. *Robot. Auton. Syst. 56*, 11 (Nov. 2008), 927–941.

[RPZ02] Ren L., Pfister H., Zwicker M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *J. Computer Graphics Forum (Proc. Eurographics)* (Sept. 2002), vol. 21, pp. 461–470.

[RSH00] Reinhard E., Smits B., Hansen C.: Dynamic acceleration structures for interactive ray tracing. In *Eurographics Workshop on Rendering* (2000), pp. 299–306.

[RSK05] Rezk-Salama C., Kolb A.: A vertex program for efficient box-plane intersection. In *Proc. Vision, Modeling and Visualization* (2005), pp. 115–122.

[RV12] Roth H., Vona M.: Moving volume KinectFusion. In *British Machine Vision Conf.* (2012).

[SB12] Stückler J., Behnke S.: Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D cameras. In *Proc. IEEE Int. Conf. Multisensor Fusion & Information Integration* (2012), pp. 162–167.

[SBKK07] Streckel B., Bartczak B., Koch R., Kolb A.: Supporting structure from motion with a 3D-range-camera. In *Proc. Scandinavian Conf. Image Analysis (SCIA)* (2007), pp. 233–242.

[SDK05] Strzodka R., Doggett M., Kolb A.: Scientific computation for simulations on programmable graphics hardware. *J. Simulation Practice & Theory 13*, 8 (2005), 667–680.

[SJ09] Schmidt M., Jähne B.: A physical model of Time-of-Flight 3D imaging systems, including suppression of ambient light. In *3rd Workshop on Dynamic 3-D Imaging* (2009), vol. 5742, Springer, pp. 1–15.

[SK09] Schiller I., Koch R.: Datastructures for capturing dynamic scenes with a Time-of-Flight camera. In *Dynamic 3D imaging. DAGM Workshop* (2009), Springer, pp. 42–57.

[SK10] Sanders J., Kandrot E.: *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.

[SS02] Scharstein D., Szeliski R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision 47*, 1-3 (Apr. 2002), 7–42.

[SSK07] Shevtsov M., Soupikov A., Kapustin A.: Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum 26*, 3 (2007), 395–404.

[Sto11] Stolte N.: Visualizing remote sensing depth maps using voxels. In *Proceedings of Machine Vision and Advanced Image Processing in Remote Sensing* (2011), Springer, pp. 170–180.

[SZ08] Shpunt A., Zalevsky Z.: Depth-varying light fields for three dimensional sensing, 2008. US Patent 20080106746 A1.

[TBF05] Thrun S., Burgard W., Fox D.: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[Tem96] Temes G. C.: Analog circuit design. Kluwer Academic Publishers, Norwell, MA, USA, 1996, ch. Autozeroing and Correlated Double Sampling Techniques, pp. 45–64.

[TL94] Turk G., Levoy M.: Zippered polygon meshes from range images. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (1994), pp. 311–318.

[TM98] Tomasi C., Manduchi R.: Bilateral filtering for gray and color images. In *Proc. Int. Conf. Computer Vision* (1998), pp. 839–846.

[TSK02] Tosovic S., Sablatnig R., Kampel M.: On combining shape from silhouette and shape from structured light. In *Proc. of 7th Computer Vision Winter Workshop, Bad Aussee* (Feb. 2002), PRIP, TU-Wien, pp. 108–118.

[WBS03] Wald I., Benthin C., Slusallek P.: Distributed interactive ray tracing of dynamic scenes. In *Proc. IEEE Symp. on Parallel and Large-Data Visualization and Graphics (PVG)* (2003).

[WE98] Westermann R., Ertl T.: Efficiently using graphics hardware in volume rendering applications. In *ACM Trans. Graph. (Proc. SIGGRAPH)* (1998), pp. 169–177.

[Web97] Web-3D-Consortium: VRML specification. http://www.web3d.org/x3d/specifications/vrml/, 1997.

[Wes90] Westover L.: Footprint evaluation for volume rendering. In *Proc. Int. Conf. Computer Graphics and Interactive Techniques (SIGGRPAH)* (1990), pp. 367–376.

[WFF11] Whitehead N., Fit-Florea A.: Precision & performance: Floating point and ieee 754 compliance for NVIDIA GPUs. *NVIDIA technical white paper* (2011).

[WKF*12] Whelan T., Kaess M., Fallon M., Johannsson H., Leonard J., McDonald J.: *Kintinuous: Spatially Extended KinectFusion.* Tech. rep., CSAIL, MIT, 2012.

[WLSM*15] Whelan T., Leutenegger S., Salas-Moreno R. F., Glocker B., Davison A. J.: ElasticFusion: Dense SLAM without a pose graph. In *Robotics: Science and Systems (RSS)* (Rome, Italy, July 2015).

[WWLVG09] Weise T., Wismer T., Leibe B., Van Gool L.: In-hand scanning with online loop closure. In *Proc. IEEE Int. Conf. Computer Vision Workshops* (2009), pp. 1630–1637.

[XSH*98] Xu Z., Schwarte R., Heinol H., Buxbaum B., Ringbeck T.: Smart pixel – photonic mixer device (PMD). In *Proc. Int. Conf. on Mechatron. & Machine Vision* (1998), pp. 259–264.

[YAL07] Yguel M., Aycard O., Laugier C.: Update policy of dense maps: Efficient algorithms and sparse representation. In *FSR* (2007), Laugier C., Siegwart R., (Eds.), vol. 42 of *Springer Tracts in Advanced Robotics*, Springer, pp. 23–33.

[ZGHG08] Zhou K., Gong M., Huang X., Guo B.: *Highly Parallel Surface Reconstruction.* Research Report MSR-TR-2008-53, Microsoft Research, 2008.

[ZHWG08] Zhou K., Hou Q., Wang R., Guo B.: Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5 (2008), 1–11.

[ZPBG01] Zwicker M., Pfister. H., Baar J. V., Gross M.: Surface splatting. In *ACM Trans. Graph. (Proc. SIGGRAPH)* (2001), pp. 371–378.

[ZZZL13] Zeng M., Zhao F., Zheng J., Liu X.: Octree-based fusion for realtime 3D reconstruction. *Graph. Models 75*, 3 (2013), 126 – 136.

List of Figures

List of Tables