DISSERTATION

Depth Sensing and Mesh-Based 3D Reconstruction

conducted in partial fulfillment of the requirements for the degree of a

Doktor der technischen Wissenschaften (Dr. techn.)

supervised by

Ao.Univ.-Prof. Dipl.-Ing. Dr. techn. Markus Vincze
E376 Automation and Control Institute

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology

by

Simon Schreiberhuber, MSc
DOB 21.02.1989
Matr. Nr.: 0927622

Vienna, April 2023

Simon Schreiberhuber

Acknowledgment

Finishing things is a challenge, and so are the final strides of my strikingly long Ph.D. studies. The beginning, my admission into a Ph.D. program, seems easy in comparison: an outburst of youthful energy and determination helped in leaving the right impression on my former supervisor Thomas Mörwald and, ultimately, Prof. Markus Vincze. However, my studies soon turned out to be challenging and were often enough filled with the feeling of being lost. Identifying scientifically impactful research directions that were personally fulfilling and beneficial to our group required skills I lacked. This struggle continued despite Prof. Markus Vincze keeping unfulfilling tasks away from me and despite all the support I received from the kind group of researchers I was surrounded with. Ultimately, I have to thank Johannes Prankl and Georg Halmetschlager-Funek for guiding me in this phase and through my first research topic. Without the occasional "Don't worry, everything is going to be fine.", I would have lost hope on one or the other occasion. My deepest gratitude goes to Jean-Baptiste Weibel and Timothy Patten, who helped me with tough decisions and brushing up my papers. I cannot help but imagine this to be an unfulfilling job that cannot possibly be repaid by a mere mention in this section.

After expressing my gratitude to people performing actual and deliberate labor supporting my work, I want to praise the friendships that have formed in these times as they were even more important and kept me not only sane, but also made my time a pure joy. I got to enjoy long discussions with J-B whose endless stream of ideas contains enough material to entertain a whole research division. He has such a kind character and a humor that is so versatile that he could adapt to meet the taste of virtually any person he meets. I miss Mohammad Loghmani who kept interacting with me despite me constantly misinterpreting his recitals about the Italian cuisine. His work morale is exemplary, but yet he knew how to fool around and have discussions that span from entertaining to absurd. Stefan Thalhammer I could always count on to have a strong opinion about everything, but also a way of motivating me (not exclusively) during our climbing-gym visits. Matthias Hirschmanner, who I studied with for many years, I thank for all the life advice and the emotional wisdom. He has proven to be an enduring friend I could look up to in the last 13 years, and who has, more than any other person, shaped my notion of what kind of person I aim to be. There are more people to mention like Edith Langer, Michael Koller, Bernhard Neuberger, Kiru Park or Dominik Bauer who are no less responsible for making my journey a pleasant one.

While the people named (and many more) had the most profound impact on my work and needed to endure most of my crazy interventions aimed at trading some of

their productivity with joy and foam dart induced injuries, it was my partner Stefanie Kropfreiter who held me together when everything seemed too much.

The research leading to these results has received funding from EC Horizon 2020 for Research and Innovation under grant agreements No. 101017089 - TraceBot, No. 645376 - FLOBOT, the Vienna Science and Technology Fund (WWTF) under agreement No. ICT15-045 - RALLI and the Austrian Science Foundation (FWF) under grant agreement No. I3969-N30 - InDex.

Abstract

Spatial understanding and sensing is a cornerstone of modern computer vision and the enabler of a vast field of applications in robotics, augmented reality, or self-driving by providing a geometrically interpretable foundation. However, this comes with a number of challenges whose two most fundamental ones are the acquisition of range data and the coherent integration of said data into a reconstruction. These are the main topics of this work and receive dedicated chapters each.

The first major chapter of this work, Chapter 3, is focused on depth sensing and takes on the challenge of improving existing structured light principles. By projecting encoded light onto the measured surfaces and decoding the pattern position from a captured camera image, simple triangulation can give dense depth values on a per-pixel basis. Spatial neighborhood encoding, in particular, is a popular structured light approach for off-the-shelf hardware. However, it suffers from the distortion and fragmentation of the projected pattern by the scene's geometry in the vicinity of a pixel. This forces algorithms to find a delicate balance between depth prediction accuracy and robustness to pattern fragmentation or appearance change. While stereo matching provides more robustness at the expense of accuracy, we show that learning to regress a pixel's position within the projected pattern is not only more accurate when combined with classification but can be made equally robust. We propose splitting the regression problem into smaller classification subproblems in a coarse-to-fine manner with the use of a weight-adaptive layer that efficiently implements branching per-pixel Multilayer Perceptrons applied to features extracted by a Convolutional Neural Network. As our approach requires full supervision, we train our algorithm on a rendered dataset sufficiently close to the real-world domain. On a separately captured real-world dataset, we show that our network outperforms state of the art and is significantly more robust than other regression-based approaches.

The second major topic, presented in Chapter 4, discusses the challenges and benefits of 3D reconstruction while focusing on the influence of the utilized data structure. Dense 3D reconstructions generate globally consistent data of the environment suitable for many robot applications. Current RGB-D-based real-time reconstructions, however, only maintain the color resolution equal to the depth resolution of the used sensor. This firmly limits the precision and realism of the generated reconstructions. We present a real-time approach for creating and maintaining a surface reconstruction in as high as possible a geometrical fidelity with full sensor resolution for its colorization.By performing the reconstruction directly on a dense triangle mesh, we overcome the lossy and inflexible nature of voxel-based reconstructions that, for many purposes, need to be transformed to such triangle mesh. A multi-scale memory management process and a Level of Detail scheme enable equally detailed reconstructions to be generated at small scales, such as objects, as well as large scales, such as rooms or buildings. We showcase the benefit of this novel pipeline with a PrimeSense RGB-D camera as well as by combining the depth channel of this camera with a high resolution global shutter camera. Further experiments show that our memory management approach allows us to scale up to larger domains that are not achievable with current state-of-the-art methods.

Kurzzusammenfassung

Räumliche Wahrnehmung und Analyse sind Fundamente von moderner Computer Vision und ermöglichen ein breites Feld an Anwendungen in Robotik, autonomen Fahren und Augmented Reality. Damit einher geht eine Vielzahl an Herausforderungen, wobei die beiden grundlegendsten die Erfassung von Entfernungsdaten und Kombination dieser Daten in eine konsistente Rekonstruktion sind. Diese zwei Aufgaben bilden den Kern dieser Arbeit.

Das erste große Kapitel, Kapitel 3, widmet sich der maschinellen Tiefenwahrnehmung und Verbesserung existierender “structured light”-Methoden. Bei diesen Methoden wird codiertes Licht auf die zu messenden Oberflächen projeziert und mit räumlich versetzten Kameras aufgenommen. Indem man das Signal dieser Pixel entschlüsselt, lassen sich die ursprünglichen Projektionsrichtungen ableiten und mittels Triangulierung die Tiefe der einzelnen Pixel errechnen.

In diesem speziellen Fall bedienen wir uns der “spatial neighborhood”-Verschlüsselung, einem verbreiteten “structured light”-Ansatz, der in kommerzieller Hardware genutzt wird. Um eine erfolgreiche Dekodierung zu gewährleisten muss das projezierte Muster in einer räumlichen Nachbarschaft intakt bleiben. Verzerrung, Abdämpfung und Fragmentierung aufgrund der Oberflächengeometrie und -beschaffenheit stellen dabei eine Herausforderung dar. Die verwendeten Dekodierungsalgorithmen müssen deshalb einen Kompromiss zwischen Präzision der resultierenden Messung und Robustheit gegenüber den soeben genannten Effekten bieten. Klassische “stereo matching”-Methoden liefern zuverlässige Resultate, indem das aufgenommene Bild mit einem Referenzbild des projezierten Musters verglichen wird. Robustheit bezüglich Verzerrungen und akkurate Ergebnisse sind dabei gegensätzlich, sodass eine Verbesserung in einem Aspekt oft nur mit Kompromissen in anderen Aspekten möglich ist. Wir zeigen, dass Robustheit und Genauigkeit entscheidend gesteigert werden können, indem man die Positionen der einzelnen Pixel im Referenzmuster direkt schätzt. Der von uns vorgestellte “weight-adaptive layer” ermöglicht es Informationen zu verarbeiten, die zuvor durch ein “convolutional neural network” aus den rohen Bilddaten extrahiert worden sind. Dies geschieht in einem mehrstufigen Klassifikations- und Regressionsmodell: Ein auf “multilayer perceptrons” basierter Entscheidungsbaum, der den Lösungsraum in kleinerwerdende Intervalle unterteilt, bis ein finaler Regressionsschritt subpixelgenaue Resultate liefert. Da unser Ansatz während der Trainingsphase auf absolute “ground-truth”-Tiefendaten angewiesen ist, führen wir das Training mit Hilfe eines künstlichen Datensatzes aus, der nahe an der Realität und den Anwendungsfällen eines solchen Sensors liegt. Anhand eines separaten Datensatzes, der mit einem “structured light”-Sensor und einem professionellen Laserscanner aufgenommen worden ist, zeigen wir auf, dass unser Ansatz aktuellen konkurrierenden Methoden überlegen ist: Wir demonstrieren starke Unempfindlichkeit hinsichtlich des Wechsels von der Trainings- zur Anwendungsdomäne, sowie hohe Genauigkeit auf subpixel-Niveau.

Das zweite zentrale Kapitel, Kapitel 4, behandelt die Herausforderungen und Vorteile von 3D-Rekonstruktionen anhand des genutzten, auf Dreiecken basierten Oberflächenmodells. Dichte 3D-Rekonstruktionen akkumulieren einzelne Aufnahmen in ein global konsistentes Modell der Umgebung, welches in Anwendungen genutzt werden kann, wie zum Beispiel der Robotik. Gegenwärtige, auf RGB-D Daten operierende Systeme sind dabei jedoch eingeschränkt, da die gängigen volumetrischen Datenstrukturen die Auflösung der Farbinformationen an die geometrische Auflösung koppeln – Dies ist ein limitierender Faktor dieser Methoden. Wir präsentieren deshalb einen echtzeitfähigen Algorithmus, der die Erstellung und Aktualisierung dreiecksbasierter Oberflächenrekonstruktionen ermöglicht. Die vorgestellten Methoden, um Speicherverbrauch und Detailgrad zu steuern, erlauben sowohl kleinste Details von Objekten akkurat zu rekonstruieren, als auch große Areale oder Gebäude in Echtzeit zu verwalten und darzustellen. Um die Vorteile dieses Algorithmus zu demonstrieren, erweitern wir eine handelsübliche PrimeSense basierte RGB-D Kamera mit einer hochauflösenden "global shutter" RGB Kamera. Die resultierenden Rekonstruktionen zeigen nicht nur mehr Details, sondern erfassen auch größere Areale als typische surfel- oder volumenbasierte Rekonstruktionen.

Contents

Abstract **III**

Kurzzusammenfassung **V**

1 Introduction **1**
1.1 Problem Statement . . . 4
1.1.1 Depth Estimation and Acquisition . . . 4
1.1.2 3D Reconstruction . . . 5
1.2 Contribution and Outline . . . 8
1.2.1 GigaDepth: Learning Depth from Structured Light with Branching Neural Networks . . . 8
1.2.2 ScalableFusion: High-resolution mesh-based real-time 3D reconstruction . . . 9
1.3 Related Publications . . . 10

2 Related Work **11**
2.1 Depth Estimation . . . 11
2.2 3D Reconstruction . . . 14

3 GigaDepth: Learning Depth from Structured Light with Branching Neural Networks **18**
3.1 Method . . . 19
3.1.1 Backbone CNN . . . 21
3.1.2 MLP Tree . . . 21
3.1.3 Training . . . 22
3.1.4 Invalidation Network . . . 24
3.2 Experiments . . . 26
3.2.1 Baseline Methods . . . 26
3.2.2 Dataset . . . 27
3.2.3 Results . . . 29
3.2.4 Ablation . . . 37
3.2.5 Invalidation Network . . . 40
3.3 Discussion . . . 41

4 ScalableFusion: High-resolution Mesh-based Real-time 3D Reconstruction 45
4.1 Method . . . 47
4.1.1 Geometry Refinement Update . . . 47
4.1.2 Expand Update . . . 50
4.1.3 Meshing . . . 50
4.1.4 Stitching . . . 50
4.2 Implementation . . . 54
4.2.1 Lookup Texture . . . 54
4.2.2 Texture Atlas and Driver Overhead . . . 55
4.3 Evaluation . . . 56
4.3.1 Geometry Comparison . . . 56
4.3.2 Texture Comparison . . . 58
4.3.3 Memory Consumption and Computational Performance . . . 58
4.4 Discussion . . . 59

5 Conclusion 62
5.1 Summary . . . 62
5.2 Outlook . . . 63
5.2.1 GigaDepth: Learning Depth from Structured Light with Branching Neural Networks . . . 63
5.2.2 Scalablefusion: High-resolution mesh-based real-time 3D reconstruction . . . 64

Bibliography 66

Chapter 1

Introduction

Long before the industrial revolution, visionaries described machines performing tasks that previously were the exclusive domain of humans. Automatons were an expression of this desire and therefore often came in human form to reenact simple movements. The actions performed by these devices were mechanically encoded and stored onto drums and gave no room for deviations from simple motion sequences (Figure 1.1a).

Before the dream of truly interactive machinery came to fruition the industrial revolution brought a cost effective split of tasks between machines, performing small specialized tasks, and humans operating those machines. These machines, driven by steam, belts and later electricity often replaced just one operation in a long production process. Humans were required for operating or feeding these machines as well as performing tedious work that required adaptability (Figure 1.1b).

While the concept of automatons has accompanied mankind ever since, the complexity of perceiving the environment and deriving reasonable actions to fulfill the intended tasks hindered such developments. As a result, production pipelines are organized to streamline the material flow and make the state of the product known at every point. This facilitates manufacturing, that is, except for a few human interventions, entirely based on machines performing simple and repeating operations.

The advancements in sensors, imaging technology, computers and algorithms brought by the recent decades have sparked hopes of human-like abilities in artificial machines, but also unearthed the true extent of the challenge. While these artificial systems exceed human capabilities in isolated tasks, they lack the ability to perform common compound operations. Humans possess the ability to continuously adapt and refine their motions during execution, based on multiple senses to cope with uncertainties. Engineered systems (robots), on the other hand, commonly split up the tasks of estimating the task-critical states, planning and execution, into discrete steps. This is owed to the fact that deeply integrating multiple modules into (robotic) systems results in exponentially growing complexity. In order to address this challenge, the principles of the subsumption architecture is applied, splitting tasks into low-level components that only consider a subset of the inputs, and high-level components that rely on those. This hierarchical scheme commonly extends over multiple layers, reducing the complexity of the entire system, but bears a loss in flexibility in the integration and communication of modules.

To illustrate the difference between the biological and the engineered approach, we

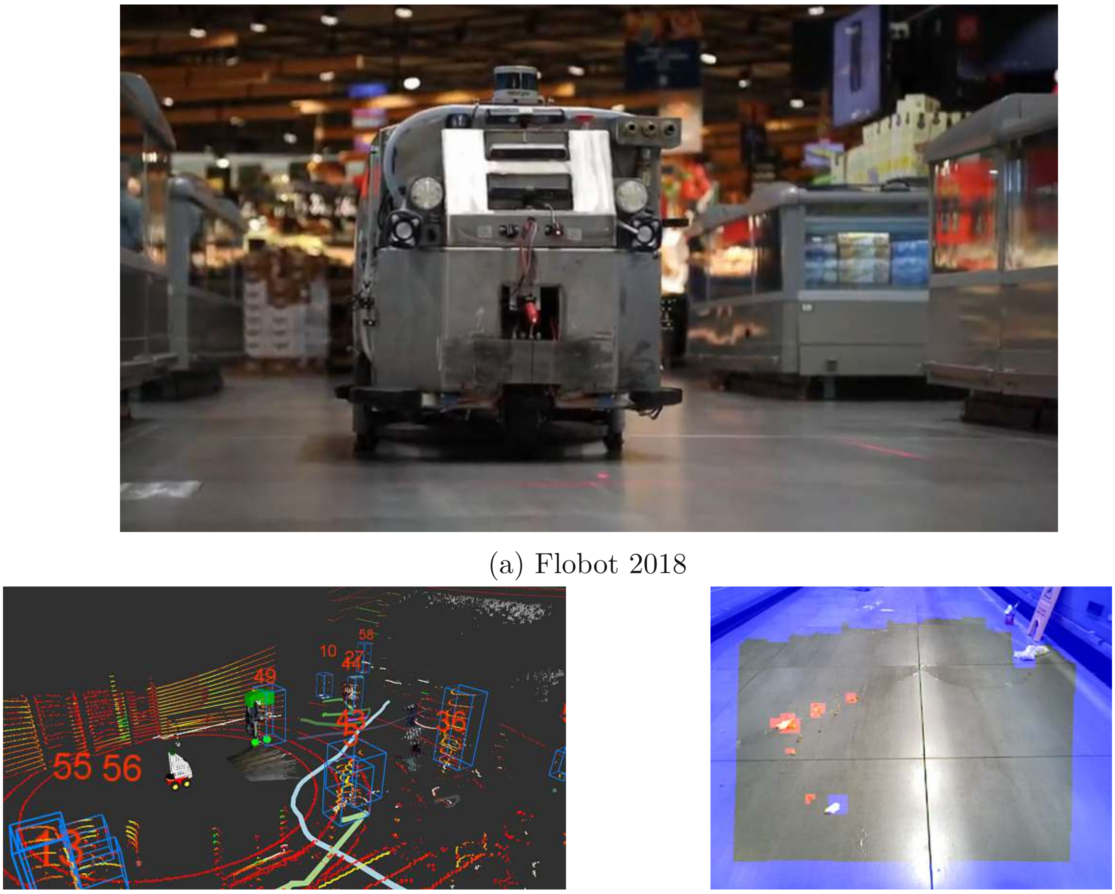
(a) Automaton ca. 1770 by Jaquet-Droz

(b) Weaving machine ca 1840. A belt is distributing power mechanically.

Figure 1.1: While the automaton by Jaquet-Droz (a) was capable of writing down text, they did so by executing movements pre-recorded on drums. The mechanism might be complex and by means of replaceable drums to a limited extent adjustable, but wide-ranging adoption of machines only came with the cruder machines of the industrial revolution (b) replacing only few of the manual actions during production. Machines reacting to their environment would not surface until the age of computers.

can imagine the task of picking up small objects from a surface. A human will, at least briefly, slide the fingers across the surface and use tactile feedback to compensate for inaccuracies in vision and motor skills, and thus, essentially, act on all involved senses simultaneously. A typical robotic system, on the other hand, will determine the free space in the environment and the precise pose of the object of interest before computing and executing a trajectory. While research is ongoing in order to develop systems closer to humans, the prevalent paradigm strongly depends on a very precise understanding of the environment surrounding the robot and, in particular, its geometry because components need to exhibit superhuman performance for the entire system to achieve human capabilities.

For the purpose of illustrating the importance of geometry and spatial understanding, we consider a floor-cleaning robot platform system developed by various partners from academia and industry. Similar to the widespread vacuum-cleaning robots, the complexity of this task is manageable for current robotic systems, but also illustrates their limitations. The FLOBOT (Figure 1.2) project demonstrated an autonomous floor-scrubbing cleaner capable of performing its task without intervention of the usually required human operator by relying on depth sensing technologies. Contrary to simpler vacuum-cleaning robots, industrial floor-cleaning systems are significantly larger and more powerful, making it all the more important to work from an accurate understanding of the environment to avoid accidents. The utilized sensors allow most of the behaviors and heuristics to be developed by following geometrical considerations and thus ensure the reliability of the whole system in complex dynamic environments. A deeper discussion can be found in the scientific publication [Yan, ISR 13, 2020] ([1]).



(a) Flobot 2018

(b) Human tracking (LIDAR)

(c) Dirt detection (RGB-D)

Figure 1.2: A prototype of FLOBOT, an autonomous scrubber cleaner. Distance sensors of different types allow it to react to its environment and set its actions accordingly. Among the sensors are a 3D Light Detection And Ranging (LIDAR), downward facing RGB-D sensors, a stereo camera, upward facing RGB-D sensors and a 2D LIDAR.

Navigation and human tracking are operating on highly accurate 3D LIDAR sensors delivering sparse but robust distance measurements, while a small-obstacle detection system is relying on dense but short ranged depth data to detect physical objects on the floor that would interfere with the scrubbing process. While the obstacle detection aspect can be managed based on the currently observed part of the scene, higher levels of planning require a more complete knowledge of the scene. This process, known as mapping, can be performed at a different level of detail depending on the needs, and will depend on the resolution of the depth sensing. A more complete and accurate knowledge of the scene is always beneficial to downstream tasks, but comes with many trade-offs in terms of computational cost and memory. A second challenge arrives in the form of the complexity to update this representation based on the perception of dynamic environments. For example, the detection of dirt on the floor is crucial to guide and optimize the floor-cleaning process, and is performed based on the RGB input in the described system. This could, however, be achieved by a simple outlier detection in the RGB input after masking out non-floor pixels by geometrical considerations on the scene geometry.

1.1 Problem Statement

In the previous section, we introduced perception as the key enabler for intelligent robotic systems and postulate that sensing and maintaining a consistent scene geometry will lead to more robust and intuitive algorithms.

Obtaining and maintaining this geometric information does, however, bring its own set of challenges that necessitate (separate) consideration. We thus, in a first step, discuss the challenges of depth-estimation while focusing on a structured light sensor setup. In the second step, we address the challenges of integrating multiple perceptions into a consistent surface model. Such a split is justified by the analogous separation found in real-world systems.

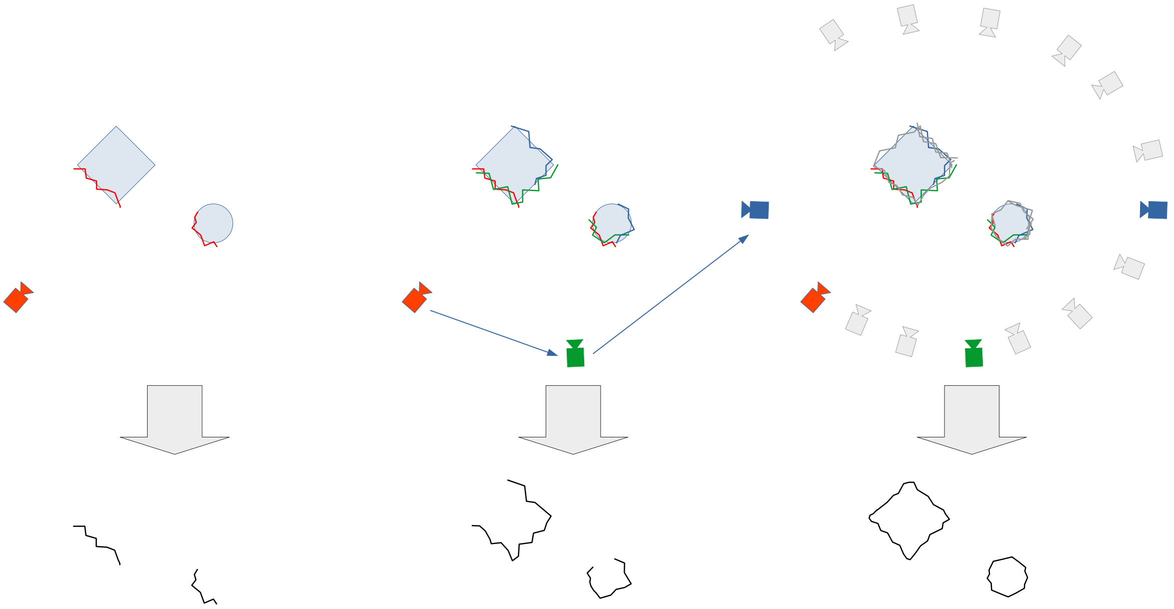
1.1.1 Depth Estimation and Acquisition

As stated in the previous section, visual sensing of geometry can facilitate robot applications and lower the complexity of algorithms by allowing geometric considerations to flow into control algorithms. We will discuss several challenges faced by these approaches.

The speed of acquisition (e.g. to support real-time control in dynamic scenes) stands in direct competition with the accuracy of measurements. This stems from the fact that the fundamental measurements used to derive distances will feature better signal-to-noise ratios with longer exposure times. This relationship actually goes further when considering sensing based on a light emitter, as stronger signal-to-noise ratios also require more energy to be emitted. Similarly, a higher spatial resolution requires more bandwidth and measurements leading to even higher power demands.

Factors inherent to the environment will challenge the sensing principle as surface properties will reflect, refract, absorb, or even outshine the sensor's light source. Similarly, fast-moving objects could prevent the sensor from collecting enough information for a sensible measurement and leave ambiguous interpretations.

Considering these challenges and the abundance of different sensing principles, it is sensible to take a closer look at one specific sensor category that is common in the robotic context. Structured light depth sensors with spatial neighborhood encoding emit a pattern into the scene that is captured by an offset camera. By decoding the captured pattern in the neighborhood of each pixel, the direction of emission is inferred and depth triangulated. This measurement principle is challenged by the scene geometry distorting the pattern, scene lighting competing with the pattern intensity, and the surface itself interacting with the light. Most sensors tackle these challenges by stereo matching the captured image against a stored reference pattern. Such algorithms can deliver subpixel accuracy but often lack the ability to cope with the changes induced by the interaction of the emitted pattern with the scene. On the other hand, modern deep learning-based



(a) One single depth frame (b) 3 captured frames (c) Full reconstruction

Figure 1.3: With increasing number of observations the reconstruction becomes more complete, whereas the surfaces gain accuracy.

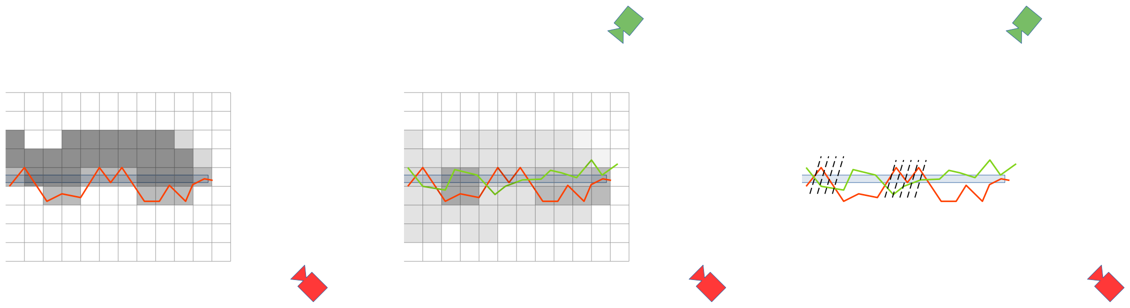
approaches internalize knowledge about the pattern and the composition of scenes leading to seemingly complete and dense depth measurements but fail at delivering accurate results. However, applications require accurate results in conjunction with robustness and error awareness to drive algorithms. These challenges are not adequately solved by state-of-the-art approaches that adopt network architectures successful in similar but decisively different tasks. It requires neural networks tailored to the specifics of this sensor category to achieve the desired combination of accuracy, completeness, robustness, and error awareness.

1.1.2 3D Reconstruction

By fusing dense depth measurements from multiple views into a consistent representation, a 3D reconstruction can support numerous operations commonly found in robotics and augmented reality. Motion planning, as applied in a robotic context, requires scene geometry to ensure the trajectories do not cause collisions. Augmented reality applications commonly rely on synthesized views from novel perspectives to aid user interaction.

The representation can be volumetric (e.g., describing some form of occupancy by a 3D grid or function) or a description of the scene's surfaces. The latter model the surface with a set of small primitives, such as disks in the case of surfels, or even model their connectivity as is the case with triangle meshes. Each of these solutions comes with different advantages and drawbacks.

The fundamental purpose is the same for all reconstruction methods, irrespective of



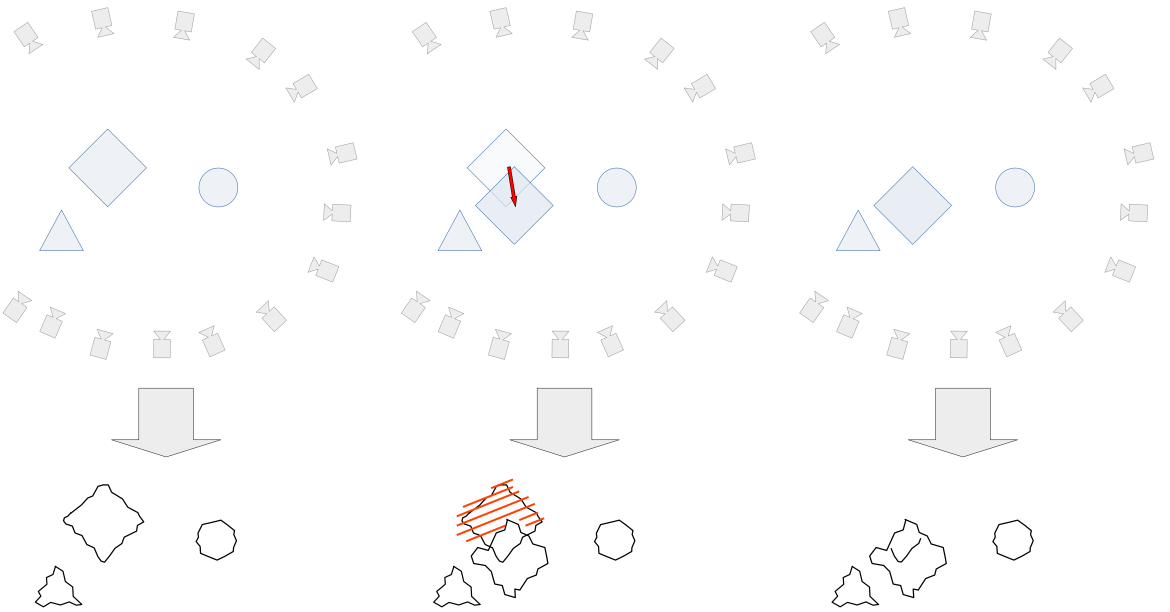
(a) Occupancy grid of surface captured from one side. (b) Occupancy after capture from other side. (c) Intersecting measurements (dashed lines).

Figure 1.4: Capturing thin objects is a challenge. As occupancy grids are updated not only in the free space observable by the sensor but also in the space behind the observed surface, the first observation leads to a shadow making the surface appear thicker (a). With the second observation, the contradictory measurements will cut holes in this observation and reduce the occupancy in most grid positions as each cell is smaller than the object's thickness (b). As shown in (c), this is a general problem not only for coarse occupancy grids, but any surface reconstruction attempting to reconstruct objects with a thickness lower than the sensor's noise level.

the underlying data structure. All methods seek to integrate multiple measurements into representations that show maximum agreement with the available data (Figure 1.3). Sensor noise, inaccurate odometry, and other partially unknown sensor characteristics make this a complex task. In practice, the data structure plays a significant role in approaching this challenge. It is sufficient to update a volume's observed empty/occupied elements to integrate new measurements because the surfaces are implicitly defined as the interface between unoccupied and occupied space. With surface-based reconstructions, the algorithm needs to make deliberate decisions about where to instantiate a surface and when and how to attribute new measurements to existing surfaces to perform updates. These design decisions have implications for an algorithm's performance when handling, e.g., the capture of thin objects as seen in Figure 1.4 or dynamic scenes as seen in Figure 1.5.

A major challenge for all systems performing Simultaneous Localization And Mapping (SLAM) is the detection and handling of loops illustrated in Figure 1.6. As the error of the odometry system accumulates, and parts of the scene are revisited during normal operations, geometry that should belong to the same surfaces will not align and appear offset and duplicated. Restoring the integrity of the reconstruction requires updates to vast portions of the geometry and, thus, is the most expensive process of a reconstruction pipeline.

Considering the challenges outlined above, it becomes clear that the underlying representation determines the performance of a reconstruction algorithm. The prevalent voxel grid has several disadvantages, such as its inability to scale the resolution depending

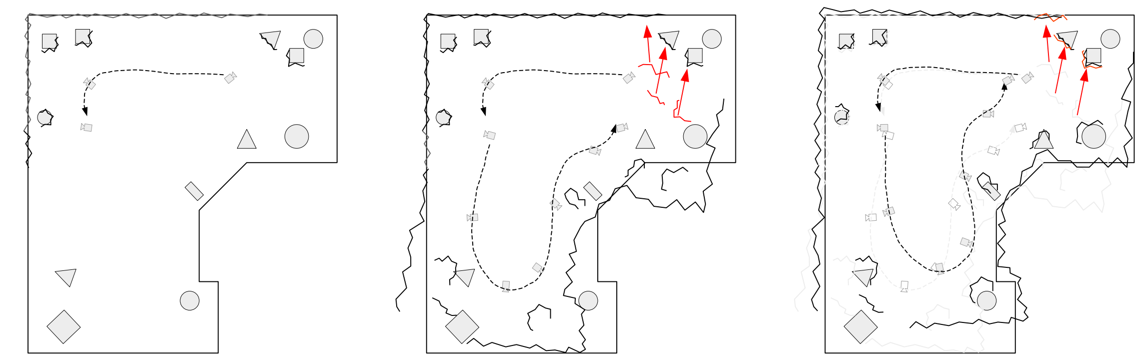


(a) Full reconstruction. (b) Moving one object. (c) Updated reconstruction with surface artifact.

Figure 1.5: After completely reconstructing a scene (a), one of the objects is moved, triggering an update (b). The space formerly inhabited by the moved object is now observed to be free, leading to the deletion of the corresponding surfaces (red scratch trough). This, however, does not apply to the parts of the old surface that now reside within the object. Sensors cannot observe this volume and thus not trigger the deletion of the according surface (c).

on the geometric complexity or its fixed coupling of geometrical and color resolution. Performing geometrical transformations (e.g., translation, scale, rotation) to maintain consistency is similarly expensive and introduces losses due to re-sampling. Surface representations such as triangle meshes, on the other hand, provide variable-scale resolution, decoupled geometry and texture resolution, and cost-efficient means of transforming the geometry by operating on the positions of primitives. However, they come with different significant algorithmic challenges.

Some of those challenges stem from the specifics of established rendering architectures, e.g., textures cannot be generated at an arbitrarily small (e.g., triangle-level) granularity due to the involved GPU-driver overhead. Still, they can only be performed for part of the mesh at once, as mapping is an incremental process. Commonplace operations such as removing geometry from a mesh will lead to memory fragmentation if no effort is made to keep the geometry primitives contiguous. During update steps, vertex positions of mapped geometry need to be updated, leading to potential self-intersection of the triangle mesh. Precautions must be taken to detect and prevent these cases. Finally, an efficient strategy to cheaply find which vertices should be used to attach newly mapped surfaces is required to maintain a complete and accurate surface reconstruction.



(a) The beginning of the reconstruction process. Camera tracking exhibits no visible drift.

(b) After a full loop, the pose estimation has drifted significantly.

(c) Moving all the surfaces to make the surfaces at the beginning and ending of the loop meet.

Figure 1.6: When simultaneously mapping the scene (straight lines) while tracking the sensor position, errors accumulate (a-b) due to various factors. Revisiting already captured surfaces (red), the positions do not overlap indicating tracking drift (b). As the observations at the endpoints are required to overlap, the whole trajectory together with the captured surfaces between these points need to be transformed to achieve consistency (c). Note that while the reconstruction can achieve consistency, it does not necessarily align with the environment, as seen in the bottom part of (c).

1.2 Contribution and Outline

This work aims to extend the capabilities of spatial sensing and reconstruction geared towards real-time applications such as robotics and augmented reality. To this end, we focus on the improvement of existing structured light-based depth sensors in Chapter 3 and the introduction of a reconstruction algorithm that is both lightweight and versatile in Chapter 4. But first, Chapter 2 will discuss state-of-the-art methods to provide the reader with the broader context for both tasks. Chapter 5 will summarize the results, discuss their relevance and provide several paths for future improvements.

1.2.1 GigaDepth: Learning Depth from Structured Light with Branching Neural Networks

To tackle the challenge outlined in Section 1.1.1 and provide robust real-time depth measurement in challenging environments, many robotic applications have adopted structured light-based sensors. With the projectors employed by these sensors capable of covering surfaces of the observed scenes with encoded light, it is possible to derive the shape of surfaces that would lack the necessary features in a passive multi-view approach.

After capturing the reflected pattern with a camera, the challenge becomes algorithmic as decoding the projected information reveals the angle of emission, facilitating triangulation. In the case of spatial neighborhood encoding, this means that the pattern position needs to be derived for each pixel by processing the information captured in the adjacent pixels, provided that enough of the pattern/information is captured by the camera after it was bounced back by the scene.

GigaDepth, the algorithm discussed in Chapter 3, takes on exactly this task, aiming for superior accuracy and robustness. While the commonly available hardware employs stereo matching against a stored reference pattern to solve this encoding task, we show with our work that directly regressing a pixel's position can be made more accurate and equally robust. Regression has been the underlying principle used by other algorithms, that receive extensive analysis in our experiments section. However, the method we propose uses a novel combination of classification and neural networks to split the regression problem into smaller subproblems in a coarse-to-fine manner. A decision tree based on Multilayer Perceptrons (MLPs) processes features extracted by a Convolutional Neural Network (CNN) in a per-pixel manner. The leaf nodes of this tree feature MLPs specialized for small pattern regions each and demonstrate the ability to regress subpixel-accurate pattern-positions. As our approach requires full supervision, we train our algorithm on a rendered dataset sufficiently close to the real-world domain. We evaluate on a separately captured real-world dataset and confirm that our network outperforms state-of-the-art and is significantly more robust than other regression-based approaches.

1.2.2 ScalableFusion: High-resolution mesh-based real-time 3D reconstruction

Dense 3D reconstructions generate globally consistent data of the environment suitable for a diverse set of applications from augmented reality to robotics. However, current RGB-D-based real-time reconstructions only maintain the color resolution equal to the depth resolution of the used sensor. This firmly limits the precision and realism of the generated reconstructions. Chapter 4 presents a real-time approach for creating and maintaining a surface reconstruction that matches both the depth sensor's resolution in its geometry representation and that of the RGB camera in the form of textures attached to this geometry. A multi-scale memory management process and a Level of Detail scheme enable equally detailed reconstructions to be generated at small scales, such as objects, as well as large scales, such as rooms or buildings. We showcase the benefit of this novel pipeline with a PrimeSense RGB-D camera as well as combining the depth channel of this camera with a high resolution global shutter camera. Further experiments show that our memory management approach allows us to scale up to larger domains that are not achievable with current state-of-the-art methods.

1.3 Related Publications

[Yan, ISR 13, 2020] Z. Yan[1], S. Schreiberhuber[1], G. Halmetschlager-Funek, T. Duckett, M. Vincze, N. Bellotto **Robot Perception of Static and Dynamic Objects with an Autonomous Floor Scrubber**, in *Intelligent Service Robotics* (ISR) 13 pp. 403-417, 2020.

[Schreiberhuber, ICRA, 2019] S. Schreiberhuber, J.Prankl, T. Patten, M. Vincze, **ScalableFusion: High-resolution Mesh-based Real-time 3D Reconstruction**, *International Conference on Robotics and Automation* (ICRA) 2019.

[Schreiberhuber, CVPR, 2022] S. Schreiberhuber, JB. Weibel, T. Patten, M. Vincze, **GigaDepth: Learning Depth from Structured Light with Branching Neural Networks**, *European Converence on Computer Vision* (ECCV) 2022.

[Schreiberhuber, OAGM, 2018] S. Schreiberhuber, J. Prankl, and M. Vincze, **Towards ScalableFusion: Feasibility Analysis of a Mesh Based 3D Reconstruction**, *Austrian Association for Pattern Recognition* (OAGM) 2018.

[Schreiberhuber, OAGM, 2017] S. Schreiberhuber, T. Mörwald, M. Vincze, **2.5D Plane Segmentation done quick: An analiysis of the Bilateral Tangential Filter**, *Austrian Association for Pattern Recognition* (OAGM) 2017.

[1] First two authors with equal contribution.

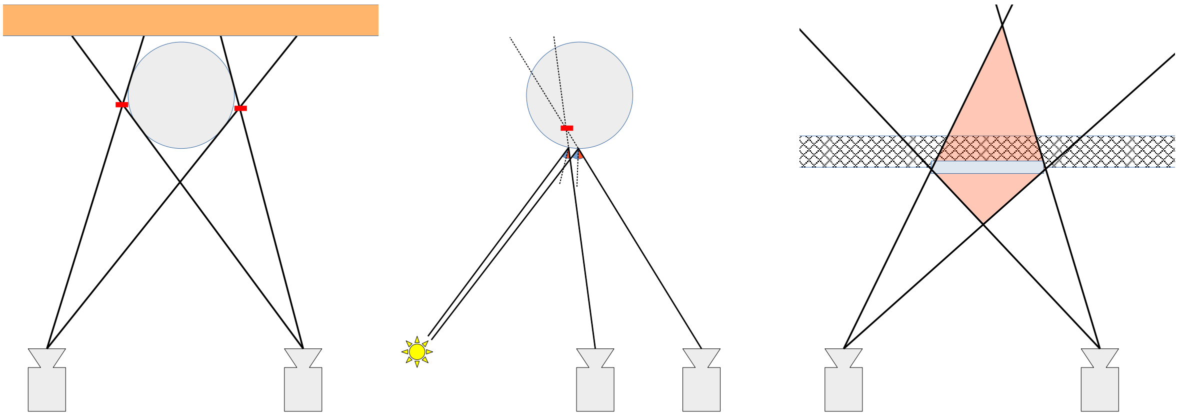
Chapter 2

Related Work

The contributions in this work take inspiration from existing research and are in a sense children of their time. As the contribution to depth-sensing is more recent, it is e.g. leveraging deep learning-based algorithms. The discussed 3D reconstruction, on the other hand, is mostly relying on hand-crafted heuristics, as it was less common to integrate learning-based methods into reconstruction pipelines. Here we, nonetheless, strive to give an account of current developments non-withstanding their lack of influence on our method.

2.1 Depth Estimation

To motivate the work in Chapter 3 and the choice of sensors, we have to take a wider view at depth perception methods applied in computer vision. With different approaches to depth-sensing delivering different mixes of accuracy, robustness and complexity, it is important to select algorithms developed to the requirements of the intended applications. Current research on depth sensing thus is split into multiple tracks that only show limited overlap but, nonetheless, often produce findings applicable to different concepts. For passive methods such as stereoscopic or multi-view depth sensing, one of the major challenges (Figure 2.1) lies in featureless ill-lit surfaces which demand coping mechanisms that follow sensible assumptions for said regions. In order to still achieve dense depth maps with most pixels containing sensible depth data, Hirschmüller et al. [2] demonstrated that a semi-global optimization approach can efficiently enforce smoothness along featureless regions in a stereo setting. Kuschk et al. [3] showed that fully global optimization can be achieved with variational optimization techniques that further generalize to multi-view approaches. Some modern CNN-based approaches still resort to similar techniques in their training phase but have the capacity to incorporate domain knowledge into their estimates improving performance. The applications of neural networks reach from stereo matching [4]–[6], multi-view geometry [7]–[10], to depth completion [11]–[13] where sparse sets of depth points are used in tandem with an intensity image to generate dense depth maps. Works such as [14] even showcase high-level reasoning as depth can be extracted from single views without using any conventional principle like triangulation. While the results of CNN-based approaches indeed are impressive with respect to their handling of featureless regions



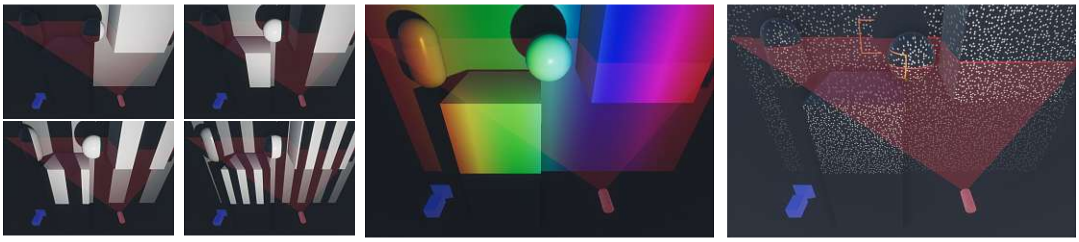
(a) Curved object edges, (b) Specular highlights. (c) Untextured surface

Figure 2.1: Some of the challenges of stereo sensing. Silhouettes along curved surfaces lead to strong but wrong matching results (red). In actuality, the edges perceived by both cameras do not belong to the same part of the surface (a). Specular reflections lead to false depth estimates (red) as they are not fixed at the surface of objects as they are dependant on the positions of observer as well as the light source (b). Surfaces without texture (blue tinted bar) lead to ambiguous depth estimates (red) that could lie anywhere in a volume (c).

and challenging surfaces/lighting conditions, obtaining high-resolution depth estimates with sharp and accurate object boundaries still poses a significant challenge. For some multi-view and stereo approaches this is a property of the used cost-volumes, modeling different depth levels in a tensor that is significantly downsampled for memory consumption and execution time purposes.

In cases where reliability is a greater concern and cost, space, and power budget permits the use of active sensor components, methods like Time of Flight (ToF), laser ranging and structured light can alleviate some of the challenges that come with passive methods. Low-cost ToF sensors, e.g., flood the scene with strong modulated infrared light while capturing the reflected light with intensity sensors of modulated sensitivity or switchable filters. This allows for phase demodulation but requires multiple captures at different frequencies to eliminate ambiguities for the signal run time. Although these sensors offer individual depth measurements for each pixel by only a few captures, they suffer from multi-path artifacts, low depth resolution and are susceptible to motion artifacts caused by fast movements. LIDAR, on the other hand, sweeps the scene with fewer rays and thus yields stronger signals in fewer directions, resulting in reliable, highly accurate, long range measurements while greatly reducing the chance for multi-path artifacts. An advantage that is offset by the relative sparsity of depth measurements that requires elaborate depth-completion algorithms to achieve dense depth maps.

Going back to the idea of triangulation, structured light-based depth projects encoded



(a) Temporal multiplexing (b) Direct encoding (c) Spatial neighborhood encoding

Figure 2.2: Different encoding methods for the light emitted by the projector (red cylinder on the right of each image). The camera (blue, left) picks up the light reflected off the surface to decode the emitted direction and facilitate triangulation. Temporal multiplexing (a) as, e.g., binary encoding utilizes multiple patterns encoding the projected projection on a per-pixel and time-slice basis. Similarly, direct encoding (b) as, e.g., color-coding allows for per-pixel evaluation of depth but on single frames and greater sensitivity to surface and lighting conditions. Spatial neighborhood encoding (c) encodes the projected position in the pattern found in the vicinity of each pixel enabling robust single-capture measurements at the expense of added susceptibility against distortion, fragmentation, and occlusion.

light onto the scene that is reflected and picked up by an imaging sensor. The encoding needs to be robust enough to recover the direction each ray is emitted at, after being subject to the scene's geometry, surface, and lighting conditions. The three major coding strategies cater to different tasks, depending on accuracy and time constraints and are displayed in Figure 2.2.

- **Temporal multiplexing** changes the projected pattern over multiple images (multi-shot) and allows decoding on a per-pixel basis. While the encoding of the image can be as simple as binary or Gray-coding as demonstrated by Altschuler et al. [15], the temporal aspect can be combined with other encoding methods to, e.g., achieve similar resolution with fewer captures.

- **Direct encoding** embeds information about the pattern position in the form of intensity, first demonstrated by Carrihill et al. [16], or color but is challenged by factors within the scene that change the appearance of the pattern. Additional patterns/captures are often required to retrieve influences like surface colors, multi-path reflections and ambient light (multi-shot).

- **Spatial neighborhood encoding** enables the extraction of depth maps from singular images by encoding the position in a narrow pixel neighborhood. The decoding process is more elaborate than the often simple decoding schemes utilized for the direct or temporal variant as scene geometry and surface parameters strongly influence the patterns appearance.

To tackle the problems unique to the structured light principle, researchers have approached the problem from multiple directions. Given the hardware to adjust the projected pattern, approaches presented in e.g. [17], [18] demonstrate performance improvements by selecting patterns according to an optimality criterion. In a similar track, the work in [19] describes the design of Hamiltonian encodings to either improve quality or drastically reduce the amount of required images. This is a stark improvement from the beginnings of temporal multiplexing by Altschuler et al. [15] who applied binary/Gray-coding or the direct encoding by Carrihill et al. [16] where position is encoded in the intensity.

On the other hand, many works improve upon the decoding step while adhering to patterns that are easy to generate. Recent work in profilometry uses CNNs to regress the topography of a surface from only a single exposure with one standard sinusoidal fringe pattern [20], [21]. This is especially noteworthy as the employed high-frequency pattern might be well suited to resolve small details but does not feature an absolute encoding as the utilized fringes are repetitive.

Although the use of neural networks can enable certain applications to cope without absolute encoding, adequate encoding still plays a crucial role for achieving robustness. Spatial neighborhood encoding like the dot pattern used with popular sensors as the Kinect v1 allow to derive pattern positions from only small patches of the pattern. In the case of the Kinect v1 and other PrimeSense-based sensors, simple block matching is executed between a captured frame and a stored reference pattern [22]. Learning-based methods can internalize knowledge about the pattern and the effects of the scene on its appearance to directly regress pattern positions or depth without look-ups in a reference pattern. This is demonstrated in the work of Fanello et al. [23] where random forests are employed to directly regress the pattern position without a matching step by traversing trees based on intensity values in a small neighborhood around each pixel. Similarly, Riegler et al. [24] as well as Johari et al. [25] used CNNs to directly regress disparity without searching in a reference pattern at run-time.

Albeit Chapter 3 and [26] mainly focus on the decoding of spatial neighborhood patterns, the sensor used in the test-bed features infrared stereo cameras together with a dot-projector. It thus is suitable to compare structured light methods relying on only one IR camera + dot-projector with active stereo methods like [27]. Zhang et al. [27] optimized a CNN-based stereo algorithm [6] for active stereo and demonstrated improved performance on Intel RealSense hardware which, by default, uses Semi-Global Matching (SGM) [2].

2.2 3D Reconstruction

Similar to the depth estimation, the type of 3D reconstruction vastly depends on the intended application, sensors and other requirements such as real-time capabilities, ability to capture dynamic scenes, resolution as well as the physical extent of the reconstruction. This has implications not only for the underlying data structures but also for the methods required to maintain and integrate new measurements into them.

Geometrical primitives known from the computer graphics domain like points, vertices, triangles and voxels were the first to be adopted by the reconstruction domain as they benefit from decades of research and established methods for rendering and manipulation.

Uniform grids of voxels (voxel grids) storing some form of occupancy, color and often weight or another form of certainty measure can easily be rendered by e.g. ray marching or transformation to triangle meshes. One algorithm building a voxel grid from input from RGB-D sensors was introduced by Newcombe et al. as KinectFusion [28] and tackled the main questions of maintenance, rendering and pose estimation. Each voxel contains color, weight (as a measure of confidence), and an estimate for the signed distance to the next surface. By aligning camera poses to the reconstruction via the projective Iterative Closest Point (ICP) algorithm, updates can be applied to the confidence, distance, and color of voxels according to the measurements. As most of the processing, like the ICP computation and updating of the volume, is already executed on the GPU, the rendering can easily make use of the rasterization hardware by first transforming the volume into a triangle mesh via the marching cubes algorithm [29]. The initial implementation is only able to map volumes of fixed position, size, and resolution by maintaining this Truncated Signed Distance Field (TSDF). By dynamically changing the position of the active reconstruction volume, Kintinuous [30] extends the base algorithm and enables the reconstruction of larger scenes. The use of voxel hashing [31] allows for larger and higher-resolution reconstructions by reducing the memory footprint required for the reconstruction volume. KinectFusion spawned further notable expansions such as DynamicFusion [32] which introduced a warp-able volume to reconstruct non-rigid objects.

While the RGB-D methods above require depth inputs and operate in real-time, optimization-based methods can optimize whole volumes in unison taking a multi-view approach. By combining photometric consistency formulations with regularization constraints, Kolev et al. [33] demonstrated how reconstructions can be performed without explicit depth measurements, albeit at the cost of computational complexity. Operating on volumes does provide the advantage of not only maintaining surfaces but also the empty space around those. However, they provide challenges when updating large surfaces and volumes as is required when keeping larger scans globally consistent. When, e.g., encountering a loop by revisiting an already mapped space, the reconstruction based on drifted sensor-poses must be corrected so the geometry stays consistent. The mechanisms to update the geometry potentially are compute intensive and lossy, especially when directly applied to the reconstruction volume itself. Imagining the most naive approach of re-sampling the volume according to a transformation derived from a bundle-adjustment algorithm, one can see why this is true. An algorithm acting on this idea would need to attribute each voxel of the volume to sensor-frames, solve a bundle-adjustment optimization problem and re-sample each voxel according to updated sensor poses. Especially the process or re-sampling will introduce losses by washing out details as the sampling at sub-voxel positions will blend together values of the initial voxels. Whelan et al. [34] showed that connecting a pose-graph with a deformable mesh generated from the TSDF can handle loop closures. It is notable that the necessary transformations are applied only after the type of representation has been transformed into one that is less costly to manipulate than raw volumes. Other algorithms such as [35] subdivide the reconstruction volume into smaller, overlapping voxel grids that can be translated and rotated to achieve consistency defined by surface alignment constraints. Closest to the idea of re-sampling a volume is the approach of de-integration older depth-measurements at their old pose and re-integration at their updated, bundle-adjusted sensor poses as described by Dai et al. [36]. This approach only performs updates on voxels belonging to updated keyframes and does not exhibit the same information loss as re-sampling, but requires older frames to be stored and still is compute-intensive.

Accumulating and maintaining loose points in a point cloud is another suitable 3D reconstruction for certain applications. These points can, e.g., be considered keypoints when augmented with feature vectors delivering distinctive descriptions for SLAM (e.g. ORB-SLAM [37]). Instead, when a surface representation is required, it is not sufficient to maintain sparse point clouds containing only keypoints meaningful to localization and tracking algorithms. A much denser point cloud is required to achieve the appearance of a dense surface model as, e.g., introduced in a work by Keller et al. [38] that maintains a reconstruction comprised of points augmented with physical extent and color, but also manages to reconstruct dynamic elements. By replacing points with discs (surfels) described by size, orientation, position and color, Whelan et al. introduced a sufficiently dense surface reconstruction (ElasticFusion [39]). Together with several heuristics generating, deleting and updating these surfels, the authors described an RGB-D reconstruction system that set itself apart from the volumetric approaches at that time.

While the RGB-D reconstructions mentioned above track color for each primitive (voxel, point, and surfel), this also means that any color information captured at resolutions higher than the geometrical resolution will be lost. This holds true for point-based but also surfel-based reconstructions as augmenting these with higher resolution color information is impractical. For triangle meshes, texturing has been the long-serving solution for augmenting low-resolution geometry with high-resolution detail such that reconstructions operating on compatible data-structures can benefit from existing rasterization techniques as demonstrated in offline methods as e.g. [40]–[42].

The approach by Fu et al. [40], e.g., enhances a volumetric reconstruction with textures. Following a TSDF-based reconstruction phase, a mesh is exported from the reconstruction volume. This mesh is partitioned into regions that are textured by RGB frames, which have been collected during the capture phase. This operation by itself would introduce stitching artifacts into the regions boundaries. Therefore, the presented algorithm ensures that all of these textured regions contain overlapping color information for their bordering regions. A global optimization step then aligns these overlaps by adjusting the intrinsic and extrinsic parameters of the keyframes. Since photoconsistency cannot be fully achieved due to the limited knowledge of the scene geometry and other factors, a second local optimization step is required to optimize texture coordinates. This combination of global and local optimization generates high-fidelity results but comes at the cost of an extensive offline optimization phase.

Most of the works mentioned above contributed concepts that were either applied in, or served as inspiration for the algorithm described in Chapter 4. The time frame of the developments behind Chapter 4 was dominated by reconstructions footing on rather classic, heuristics-driven computer vision and rendering solutions. With the advent of machine learning, neural networks, and deep learning, a new paradigm was introduced into the field of reconstruction, leading to a shift of focus in reconstruction research in the time after we concluded our work. Early examples of generating views with neural networks, e.g., with the Generative Adversarial Network (GAN) introduced by Goodfellow et al. [43], showed that neural networks can generate artificial images to mimic the training data. These GANs are, for example, also applied to augment a dataset with additional samples [44], [45], or in an auxiliary view generation task for pose estimation (Park et al. [46]). While these generative networks learn to generate views similar to the training data, they exhibit a tendency to generating spatially infeasible/irrational output. However, Eslami et al. [47] demonstrated just that, albeit only for rather simplistic scenes and low resolutions. The introduced Generative Query Networks show the ability to generate condensed scene representations from just a few images, as well as the capacity to render novel views from pose queries.

With the introduction of Neural Radiance Fields (NeRF), Midenhall et al. [48] demonstrated how current commodity GPU hardware was capable of ray marching volumes whose density and color is entirely defined by MLPs. The MLPs take position and viewing direction as inputs to compute density and color. By evaluating these MLPs along view rays originating from pixels and blending their results, images are rendered for novel views. Training these MLPs on a scene and rendering frames at a decent resolution did not reach acceptable speeds initially, but the quality of the resulting appearance models sparked further developments. Neural Reflectance Decompositions (NeRDs) introduced by Boss et al. [49] extended the pipeline by the Bidirectional Reflectance Distribution Function (BRDF) to explicitly model the surface properties and a spherical Gaussian illumination model. Instead of just learning the appearance of a point from the captured directions, the BRDF model allows for optimizing physical surface parameters and thus enables renderings of the model in different lighting conditions. Hedman et al. [50] drastically reduced the number of points sampled for each ray by baking a trained NeRF into a sparse voxel grid, thereby facilitating real-time rendering. Mueller et al. demonstrated a similar real-time capability with instant neural graphics primitives [51] by employing voxel hashing at multiple resolutions to attribute features to spatial locations. Decoding said features, the MLP can be smaller and thus fit into the on-chip GPU cache, but must resolve hash conflicts. Effectively, this allows to train appearance models with similar quality as NeRFs within seconds and to render them at a high frame-rate. But it is not only volumetric approaches that achieve impressive results, as proven by Ruckert et al. [52], demonstrating that the combination of point clouds augmented with additional features and neural image completion can achieve impressive scene representation.

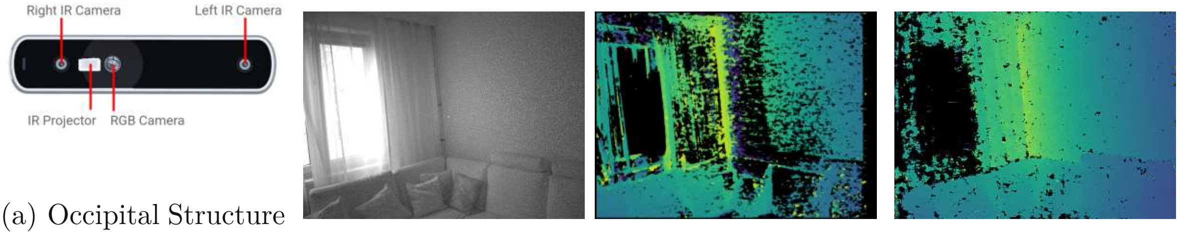
Chapter 3

GigaDepth: Learning Depth from Structured Light with Branching Neural Networks

Depth sensing is essential for safe interactions in augmented and virtual reality applications as well as mobile robotics. Structured light sensors are a particularly appealing solution for these indoor applications. These sensors predict depth from the alterations of the light patterns they project in the scene rather than the scene appearance, thus overcoming the issue of featureless areas. Spatial neighborhood encoding (which encodes the pattern position of every pixel by creating identifiable structures in the neighborhood of the pixel of interest) provides a valuable trade-off between accuracy, cost, and power consumption among structured light methods. The projector only needs to project a fixed sparse dot pattern once on the scene rather than encode every captured pixel separately (Figure 2.2c), or require multiple acquisitions like temporal multiplexing.

Such approaches are, however, very sensitive to pattern distortion, fragmentation and attenuation. In practice, the correspondence problem between the captured image and the projected pattern is solved by stereo matching to a pre-stored reference pattern as used in the Kinect v1 or PrimeSense Carmine [22], trading some robustness and accuracy with the simplicity of matching algorithms. More modern machine learning-based approaches spark the hope to directly, robustly and accurately decode these spatial encodings, but currently only deliver either accurate (e.g. [23]) or dense (e.g. [24], [25]) depth maps.

With GigaDepth, this work contributes a novel neural network architecture that decodes spatial neighborhood encodings even if the projected pattern is distorted, fragmented or attenuated by the scene's geometry and surface properties. By combining the strengths of CNNs for feature encoding with those of regression trees, we are able to extract pattern positions from captured images with higher accuracy and produce much denser output. Regression trees are implemented with MLPs in a novel, weight-selective layer for which we provide an efficient CUDA implementation. Novel datasets for training and testing this approach are artificially rendered and captured with the Occipital Structure Core (Figure 3.1) as well as an industrial-grade 3D Scanner for ground-truth. Our method outperforms state-of-the-art structured light approaches and active stereo in applications where disparity in subpixel accuracy is required.



(a) Occipital Structure Core (b) Left IR image (c) Default disparity (d) GigaDepth (ours)

Figure 3.1: The proposed algorithm (d) is capable of deriving depth by only considering one infrared image (b) and knowledge about the projected pattern. The sensor's (a) internal processing on the other hand uses both cameras, but often fails to pick up the pattern and densify its output (c). The non-repeating nature of the pattern and the availability of two cameras makes it an ideal testbed to contrast active stereo matching with structured light depth sensing.

3.1 Method

Precise depth estimates rely on determining pattern positions at subpixel accuracy for the entire pattern. While it is possible to solve this problem by means of a similarity search within a reference pattern, it is more suitable to use a priori knowledge about the pattern's structure to directly derive the pattern position.

Given a rectified image $I(x)$ and pattern $P(x)$ intensities along a given epipolar line, it is possible to extract the pattern position x_P, where the projected pattern $P(x_P)$ resembles the captured intensities $I(x_I)$ around pixel region x_I. The difference between these two positions (along the horizontal axis x) $d = x_P - x_I$ is called disparity and can express depth $z = {}^{fb}\!/_{d}$ given the baseline b and focal length f.

To estimate depth for a given pixel $I(x_I)$, both estimating the disparity d or the position x_P suffices. However, directly regressing the position x_P poses a challenge for pure CNNs as they exhibit too much noise over such a large output range. This is a motivating factor for [24] to limit the output range to 128 pixel and estimate disparity d instead. If, however, short-ranged depth estimates are needed, it is necessary to increase the range of disparity, which leads to the same aforementioned challenge. Another approach of splitting the range of pattern positions x_P into classes leads, together with our requirement of subpixel accuracy, to thousands of classes. While it is unmanageable to use one-hot encoded outputs, as typically done in CNNs for every pixel, it is a task gracefully managed by the decision trees employed in [23].

Our approach, GigaDepth, first introduced in [26] thus employs a similar hierarchical principle of splitting the output range into smaller, easier to regress regions. Instead of directly deriving decisions for tree-traversal from intensity differences as in [23], we employ a CNN to extract features allowing for more robust decision functions based on MLPs. Figure 3.2 shows an overview of our architecture.

To accentuate the features of the projected pattern as well as dampen any low frequency signal, we add the option to employ Local Contrast Normalization (LCN) seen in Figure 3.3 as a pre-processing step (see Figure 3.2). The works of Riegler et

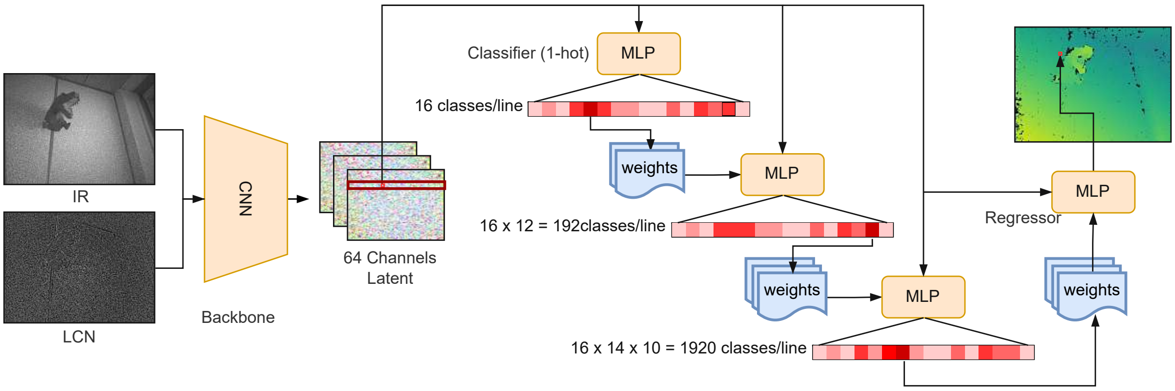
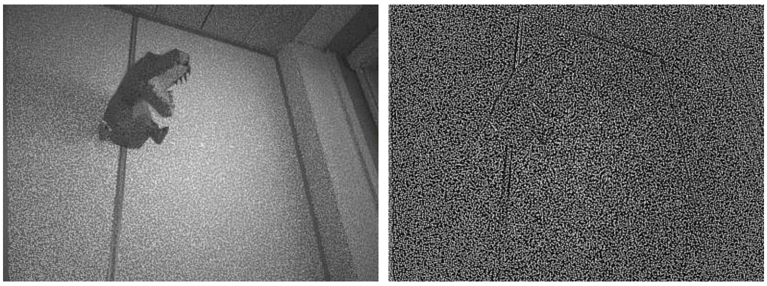


Figure 3.2: The proposed network architecture: While the backbone is a relatively shallow CNN, the regressor is a hierarchy of adaptive MLPs operating on individual pixels. The stage 1 MLP features one set of weights for each line and splits it up into 16 classes. Stage 2 has 16 sets of weights for each line to classify into 12 subclasses. Stage 3 further splits these up into 10 categories. Only at the last stage, one of 1920 weights/regressors is selected to perform the regression.



(a) IR input (b) LCN

Figure 3.3: Local Contrast Normalization (LCN) is used to accentuate the pattern of the IR projector.

al. [24], Johari et al. [25], and Zhang et al. [27] employ this technique to emphasize the utilized dot pattern during both training and runtime. The intensity I of the input image is adjusted around the mean $\mu_i(\boldsymbol{x})$ at each pixel position $\boldsymbol{x}$ and then normalized by the standard deviation $\sigma_I(\boldsymbol{x})$.

$$LCN(I, \boldsymbol{x}) = \frac{I(\boldsymbol{x}) - \mu_I(\boldsymbol{x})}{\sigma_I(\boldsymbol{x}) + \epsilon} \tag{3.1}$$

Mean and standard deviation ($\mu_I(\boldsymbol{x})$, $\sigma_I(\boldsymbol{x})$) are calculated in a small neighborhood with an 11 pixel radius around the center $\boldsymbol{x}$, and thus local measures that reflect the diverse lighting situations that appear within frames.

In Section 3.2.4, we demonstrate that the influence of the ablation is not essential but measurable and almost without a cost in execution time.

	In							Out
k	5	3	3	3	3	5	3	3
s	2	1	1	1	1	1	1	1
C_{in}	2	16	24	32	40	64	64	96
C_{out}	16	24	32	40	64	64	96	160

Table 3.1: The layers of the used backbone CNN. Layers are described by their kernel sizes k, stride s and in/output channels C_{in} and C_{out}. Each layer is combined with BatchNorm and ReLU.

stage	C_{in} [*start*, *stop*]	layers [*in*, *hidden*, $^{c}/_{o}$]
class1	[0, 64]	[64, 64, 32, 16]
class2	[16, 80]	[64, 32, $^{12}/_{2}$]
class3	[80, 144]	[64, 32, $^{10}/_{3}$]
reg(out)	[128, 160]	[32, 32, $^{1}/_{1}$]

Table 3.2: The used regression tree structure. The classification stages of our MLP tree split the output region into ever smaller subregions until specialized MLP regressors take over. The classifiers have multiple layers with the one-hot encoded output being split in output classes c and overlap o to the neighboring group of classes

3.1.1 Backbone CNN

The first step in our pipeline is to condense the local image data such that the features concerning the pattern and the scene can be efficiently processed by the MLP tree. Compared to current CNNs, this backbone is implemented as a shallow CNN (Table 3.1) with a receptive field radius of only 21 pixel, which makes the number of considered pixels similar to [23] ($(21 \cdot 2 + 1)^2 = 1849$ vs. $32^2 = 1024$ for [23]). Other networks (e.g. [14], [24]) employed in similar scenarios are multiple times deeper and use a U-shaped structure such that their receptive field spans the entire image and high-level perception of the scene can be learned. For this pattern detection task, however, we found it sufficient to only consider regions large enough to capture uniquely identifiable pattern segments and to detect cues about depth discontinuities around object boundaries.

Furthermore, we halve the resolution of our feature maps ($1216 \times 896 \rightarrow 608 \times 448$) by convolution with a stride of two early in the network as high-resolution depth maps are impractical for real-time purposes.

3.1.2 MLP Tree

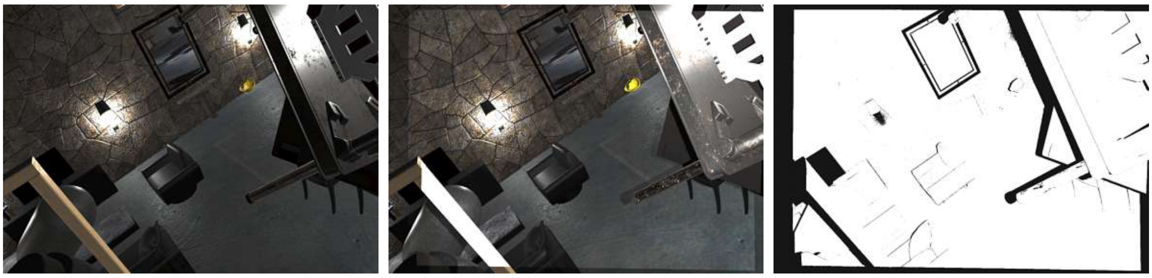
For every line of our output image we maintain one specialized tree consisting of one-hot encoding decision MLPs at each node and specialized regression MLPs at each leaf. In contrast to [23], directly comparing two intensity values for a binary decision function, the MLPs constitute much more expressive but also heavier decision functions leading to more robust performance. However, as even the minimally feasible perceptrons are relatively compute intensive, we are forced to use a much shallower tree ($10-15$ for [23] vs. 3) with each node splitting into more branches to reach similar width.

Each of these trees is evaluated on a per-pixel basis, meaning that the root node processes the features of one individual pixel to split the output range $x_{P,x}$ into $c_1 = 16$ consecutive, equally spaced regions X_{i_1}. These are further split by consecutive nodes into $c_2 = 12$ regions $\mathcal{X}_{i_1,i_2}$ each. A third stage follows, splitting each of these regions into $c_3 = 10$ regions $\mathcal{X}_{i_1,i_2,i_3}$, yielding a total of $c = c_1c_2c_3$ classes/regions. Finally, at the leaves of this tree structure sit specialized regressors that are tuned to estimate $x_{P,x}$ in the small regions $\mathcal{X}_{i_1,i_2,i_3}$.

A few modifications are employed that deviate from this simplified description: We share the trees for two consecutive rows to reduce the overall parameter count. For the same purpose, we have regressor MLPs share weights for four consecutive classes while having per-class weights for the final regression. We also allow some overlap between the classification results of the classifiers of stage 2 and 3 as the preceding classifications might be inaccurate at the boundaries between neighboring classes/regions e.g. $\mathcal{X}_{i_1}$, $\mathcal{X}_{i_1+1}$. Given e.g. $c_2 = 14$ with an overlap of $o_2 = 1$ means that the MLPs at stage 2 has 16 raw output classes with indices $i'_2 = i_2 + o_2$. In our example, raw results as $i'_2 = 0$ or $i'_2 = 15$ mean that the previous classification result will likely fall in one of the neighboring regions $\mathcal{X}'_{i'_1,0} \equiv \mathcal{X}'_{i'_1-1,14}$, $\mathcal{X}'_{i'_1,15} \equiv \mathcal{X}'_{i'_1+1,0}$. Similarly, the regressors are trained to cover for their neighbors. See Table 3.2 for a detailed account of the involved MLPs.

The described structure requires branching on a per-pixel basis and thus is ill-suited for an efficient implementation on the basis of high-level functionality of popular deep learning frameworks. Therefore, we provide a CUDA implementation of a weight-selective layer for pytorch to keep execution time and, more importantly, the memory footprint of our architecture manageable.

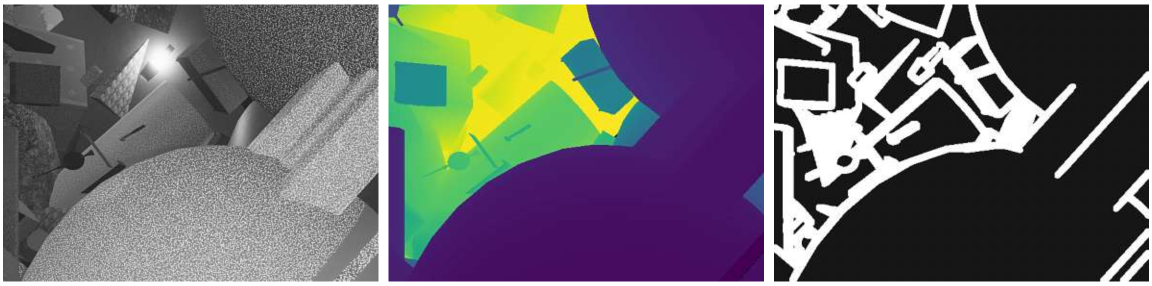
3.1.3 Training



(a) Projector off (b) Projector on (rectangle) (c) Resulting mask

Figure 3.4: We mask all pixels that show enough of the pattern to reasonably attempt depth estimation. This is done by comparing two images: one taken with an active IR illuminator and one without thresholding.

When traversing the MLP tree to reach a leaf, each node takes in the features extracted by the backbone to derive a decision about the path to be taken. If we only apply a loss



(a) Simulated IR (b) Depth (c) Edge mask

Figure 3.5: Starting with the ground-truth depth, we use a Sobel filter followed by thresholding and dilation to extract masks for depth discontinuities. We use those to emphasize the loss on the edges, which increases sharpness at object boundaries and depth discontinuities.

on the regression predicted by a leaf MLP and apply backpropagation starting there, we would not be able to update the weights of the non-leaf node MLPs. The class indices i_1, i_2, i_3 stemming from these only act to select weights of the corresponding MLPs and do not allow for the gradient to propagate back, making supervision based on principles, such as photoconsistency, a serious challenge.

Thus, we employ a training modality that allows full supervision for each of the tree's nodes individually, shifting the focus away from elegant means of self-supervision towards the benefits of our branching architecture. We design our system around the availability of complete ground-truth data which is provided in the form of a novel artificial dataset simulating the sensor. The classification stages of the MLP trees are supervised by class indices i_1, i_2, i_3 generated by discretizing the horizontal position x of the dot pattern into hierarchical regions $\mathcal{X}_{i_1}$, $\mathcal{X}_{i_1,i_2}$, $\mathcal{X}_{i_1,i_2,i_3}$ that are equally sized at each level. We utilize the cross entropy loss

$$\mathcal{L}_{class}(\boldsymbol{x}, \bar{i}_1, \bar{i}_2 \dots \bar{i}_l) = \sum_{i_l \in 1 \dots c_l} -C_{\bar{i}_1, \bar{i}_2 \dots \bar{i}_{l-1}}(\boldsymbol{x}, i_l) log(\delta(i_l - \bar{i}_l)) \tag{3.2}$$

with the classifiers C_0, C_{i_1}, C_{i_1,i_2} at the 3 stages estimating probabilities for a given class index i_l at stage l and pixel $\boldsymbol{x}$. Note that the ground-truth indices are dependent on the pixel position $\boldsymbol{x}$ and are written $\bar{i}_l$ instead of $\bar{i}_l(\boldsymbol{x})$ for the sake of brevity. As the classifiers are less reliable at the edges between adjacent regions, we strengthen the MLPs of stages 2, 3, and our leaf nodes for the cases where the preceding classifier is off by one index by additionally applying the cross-entropy loss for these regions as well:

$$\mathcal{L}^*_{class}(\boldsymbol{x}, \bar{i}_1, \bar{i}_2 \dots \bar{i}_l) = \sum_{i \in \{-1,0,1\}} \mathcal{L}_{class}(\boldsymbol{x}, \bar{i}_1, \bar{i}_2 \dots \bar{i}_{l-1} + i, \bar{i}_l^{(i)}) \tag{3.3}$$

Here, the index $\bar{i}_l^{(i)}$ denotes the ground-truth class index in case the preceding classification is off by i whereby the corresponding regions are still identical ($\mathcal{X}_{\bar{i}_1, \bar{i}_2, \dots, \bar{i}_{l-1}+i, \bar{i}_l^{(i)}} = \mathcal{X}_{\bar{i}_1, \bar{i}_2, \dots, \bar{i}_l}$).

To train the regressor MLPs at the leaves, we use the $\mathcal{L}_1$ loss and further employ the same strategy as before to let each regressor R_{i_1,i_2,i_3} cover for its neighbors:

$$\mathcal{L}_{reg}(\boldsymbol{x}, \bar{i}_1, \bar{i}_2, \bar{i}_3, \overline{x}_{P,x}) = |R_{\bar{i}_1, \bar{i}_2, \bar{i}_3}(\boldsymbol{x}) - \overline{x}_{P,x}| \tag{3.4}$$

$$\mathcal{L}^*_{reg}(\boldsymbol{x},\bar{i}_1,\bar{i}_2,\bar{i}_3,\bar{x}_{P,x}) = \sum_{i\in\{-1,0,1\}} \mathcal{L}_{reg}(\boldsymbol{x},\bar{i}_1,\bar{i}_2,\bar{i}_3+i,\bar{x}_{P,x})) \tag{3.5}$$

In order to mix and weigh these losses, we further utilize an edge mask m_{edge} (Figure 3.5) that marks regions around depth discontinuities to emphasize sharpness around edges by increased loss. Another mask m_{vis} is used to remove the loss where the training signal is too ambiguous for spatial neighborhood encodings (Figure 3.4). This mask covers pixels whose surface do not sufficiently reflect the projector's light due to albedo, distance, or occlusion.

Weighing the edge mask with λ_{edge} and the regression with λ_{reg}, we can now combine our loss functions to a combined loss function

$$\begin{aligned}\mathcal{L}(\boldsymbol{x},\bar{i}_1,\bar{i}_2,\bar{i}_3,\bar{x}_{P,x}) = \Big(&\mathcal{L}^*_{class}(\boldsymbol{x},\bar{i}_1)+\mathcal{L}^*_{class}(\boldsymbol{x},\bar{i}_1,\bar{i}_2)+\mathcal{L}^*_{class}(\boldsymbol{x},\bar{i}_1,\bar{i}_2,\overline{i_3})+\\ &\lambda_{reg}\mathcal{L}^*_{reg}(\boldsymbol{x},\bar{i}_1,\bar{i}_2,\overline{i_3},\bar{x}_{P,x})\Big)m_{vis}(1+\lambda_{edge}m_{edge})\end{aligned} \tag{3.6}$$

The gradients for this loss are propagated all the way back to the backbone CNN and used to update weights via classical Stochastic Gradient Descent using a curriculum outlined in Table 3.3. Training the whole system on our artificial dataset takes $\sim 10h$ utilizing an NVIDIA RTX 3090 graphics card.

Augmentation of the data with noise and a slight vertical jitter of 4 pixel are used to introduce robustness against slight changes of the sensor geometry and intrinsics that occur when the sensor is objected to temperature changes and mechanical manipulation. The impact of this strategy is measured and discussed in Section 3.2.4.

epoch	lr.	$\lambda_{reg.}$	λ_{edge}
1-2	4	500	0
3-4	2	1000	1
5-6	1	2000	2
7-8	0.5	2000	4
8-12	0.25	2000	20

Table 3.3: The training of the network focuses on the classification stages initially before increasing the emphasis on the regressor stages and sharpness on edges and corners by increasing $\lambda_{reg.}$ and λ_{edge}. The learning rate (lr.) steadily decreases, resulting in shorter steps towards an optimum.

3.1.4 Invalidation Network

As we seek to employ our approach in reconstruction applications, it is vital to develop the means to predict the validity of our depth estimations. For this purpose, we utilize a network estimating a (binary) mask m_{inv} flagging potentially invalid pixels in our disparity estimates. This is very similar to our backbone CNN but differing in the layer count and the total lack of strided convolutions (Table 3.4). As seen in Figure 3.6, the input for this network is comprised of a concatenation of the raw disparity from our main network and the probability outputs of its classification stages. By forgoing

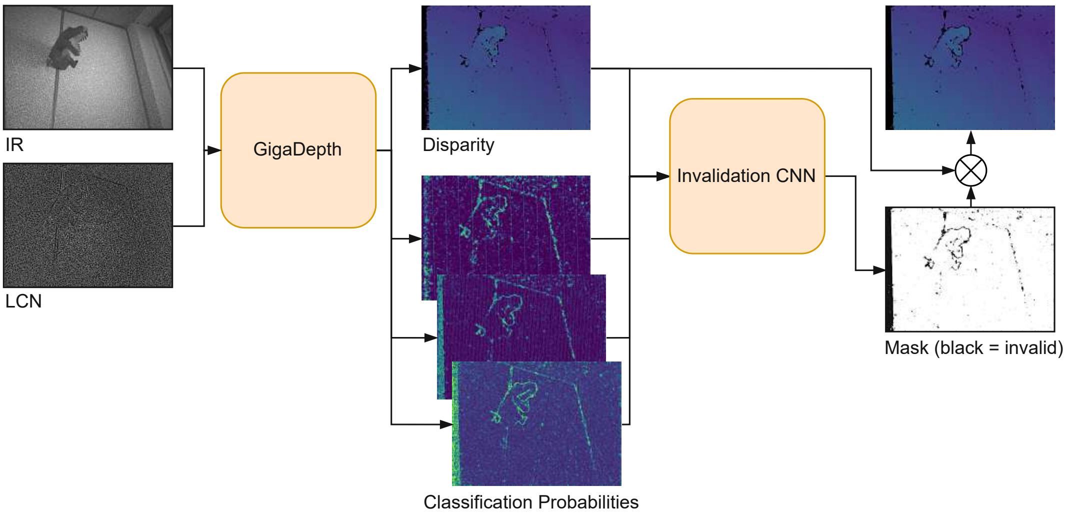


Figure 3.6: The proposed invalidation network processes the output of GigaDepth to generate a mask indicating invalid pixels. By concatenating the raw output disparity and the classification probabilities of the three classification stages in our MLP tree, we enable the network to operate without relying on GigaDepths latent feature-maps.

the use of raw image data or directly extracted features, the network is capable of predicting invalid pixels while harnessing any domain agnosticism exhibited by the main predictor. It can also be expected that this network is agnostic to retraining or smaller changes to the GigaDepth network and thus would not need to be retrained each time the architecture changes. We use the binary cross entropy loss

$$\mathcal{L}_{inv}(m_{inv}, \overline{m}_{inv}) = \log(m_{inv}) + (1 - \overline{m}_{inv}) \log(1 - m_{inv}) \tag{3.7}$$

to supervise and train the network while keeping weights of the main GigaDepth network constant. The ground-truth mask $\overline{m}_{inv}$ is generated by thresholding (0.5 pixel) the difference between the estimation and rendered ground-truth disparity. To further emphasize a more conservative strategy, we weigh the loss of invalid pixels with a factor of $\lambda_{inv} = 5$ compared to valid pixels.

$$\mathcal{L}^*_{inv}(m_{inv}, \overline{m}_{inv}) = \mathcal{L}_{inv}(m_{inv}, \overline{m}_{inv}) \begin{cases} \lambda_{inv}, & \text{if } \overline{m}_{inv} \\ 1, & \text{otherwise} \end{cases} \tag{3.8}$$

It is noteworthy that GigaDepth does overfit on the ground-truth data and thus not exhibit all possible failure cases on its training dataset. Training for the invalidation network thus has to be performed on another dataset that has to be distinct enough.

	In			Out
k	5	5	5	5
s	1	1	1	1
C_{in}	4	8	32	32
C_{out}	8	32	32	1

Table 3.4: The invalidation network. Layers are described by their kernel sizes k, stride s and in/output channels C_{in} and C_{out}. Each layer is combined with BatchNorm and ReLU.

3.2 Experiments

Real-world applications require accurate disparity in a subpixel range as well as depth measurements that cover challenging surfaces. Similar to [24], we report the outlier ratio $o(th)$, describing the ratio between the number of pixels that feature a disparity error higher than a given threshold th and the overall number of pixels.

We also present the Root Mean Square Error ($RMSE$) of the depth measurements derived by the different algorithms on the Occipital Structure Core. Pixels with a disparity error greater than one pixel are excluded as errors of this magnitude will lead to problematic depth measurements. Including outliers of this kind would also distort the evaluation in favor of algorithms capable of filling in for regions where little to no pattern could be captured as strong outliers have disproportional influence on the $RMSE$.

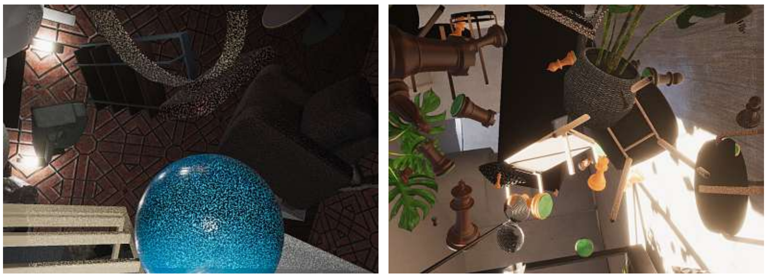
A thorough performance comparison to existing methods is conducted using an artificial dataset. The rendering process provides ground-truth, which allows us to investigate the upper limits of the evaluated algorithms. Finally, we compare on real-world data with ground-truth captured by an industrial grade 3D scanner.

3.2.1 Baseline Methods

The main reference points for our method are HyperDepth [23], Connecting The Dots [24], and DepthInSpace [25], all of which directly regress a pixel's position within the pattern or the disparity. To contrast with these methods that need a reference image to operate, we also compare to ActiveStereoNet [27] which extends [6] by modifying the loss to emphasize the dot pattern. It is expected that the usage of the second IR camera gives ActiveStereoNet an advantage wherever the pattern is too weak and classical stereo matching can pick up scene features. Fittingly, all of these methods operate at similar execution times when tested on our RTX 2070 Max-Q with HyperDepth $20-70ms$, Connecting The Dots $40ms$, DepthInSpace $20ms$, ActiveStereoNet $45ms$ and ours $60ms$.

While our rendered dataset enables training of these methods, we deviated from the original methods in a few aspects:

- **Resolution:** For the purpose of comparability, we upscale the results of algorithms operating on a lower resolution than the input resolution. The algorithms



(a) Training image (b) Test image

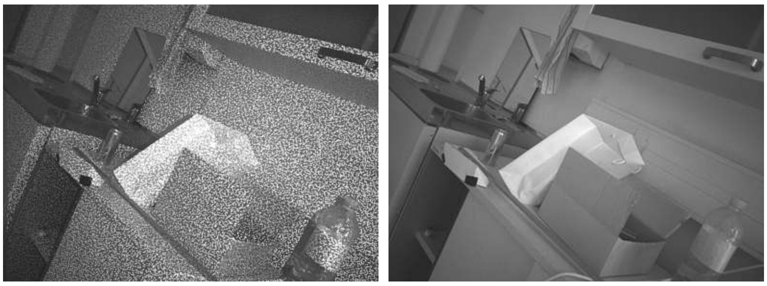
Figure 3.7: Images for training as well as testing use different objects, textures and lighting conditions.

themselves operate at or close to their intended resolution.

- **Jitter:** As our sensor hardware exhibits vertical drift (Section 3.2.4), we train all approaches with the appropriately jittered inputs.
- **HyperDepth:** Unlike the original authors [23], Riegler et al. [24] published an implementation of HyperDepth that was used for their baseline comparison. We ported their implementation to CUDA for faster experiments and added k-means clustering to improve the regression accuracy at the leaf nodes. We further utilize deeper trees (16 vs. 14 levels) for improved results.
- **Connecting The Dots:** The original approach of matching with the reference pattern did not converge on our dataset. We therefore use the image captured/rendered by the right camera to have a stereo matching approach during training time. During runtime, we operate the algorithm as intended by the authors, not utilizing the second camera. Despite these efforts, we are unable to bring Connecting The Dots to the same performance levels as on the originally intended dataset. Hence, we also include a comparison of our method on the dataset presented in the original publication [24].
- **DepthInSpace:** As this method is similar to Connecting The Dots in many aspects and thus faces similar challenges during training, we adopt the same adjustments, which leads to good success. The training requires optical flow [53], which is why only 241 of our captured sequences can be utilized as only those have additional captures with a disabled IR-projector. This is more than the 148 training sequences used in the original work [25], but puts it at a disadvantage to the other baseline methods trained on 967 sequences.

3.2.2 Dataset

The Structure Core allows us to run the algorithms of [23]–[25] and [27], thus we render a new dataset based on this sensor. The artificial data is rendered via Unity3D using the High Definition Rendering Pipeline and free assets found on the asset store. Unlike the methods in [24], [25] that rely on ShapeNet [54] objects without textures, our objects are



(a) IR projector on (b) IR projector off

Figure 3.8: For 241 of 967 images our dataset provides an "ambient" image. Images taken with the same camera at the same pose but with deactivated IR projector.

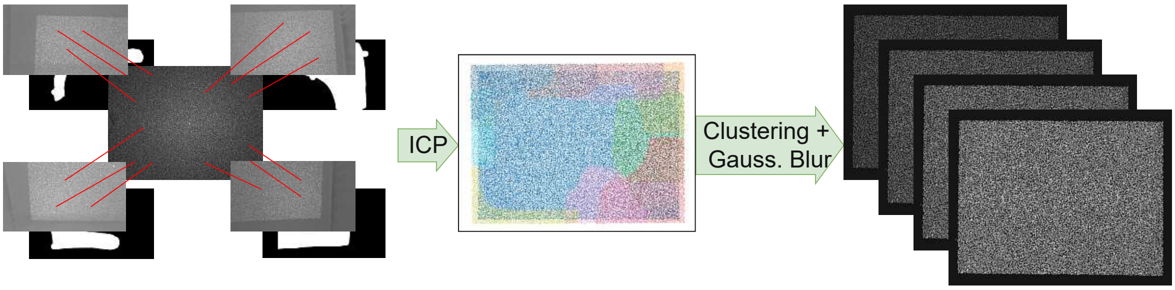
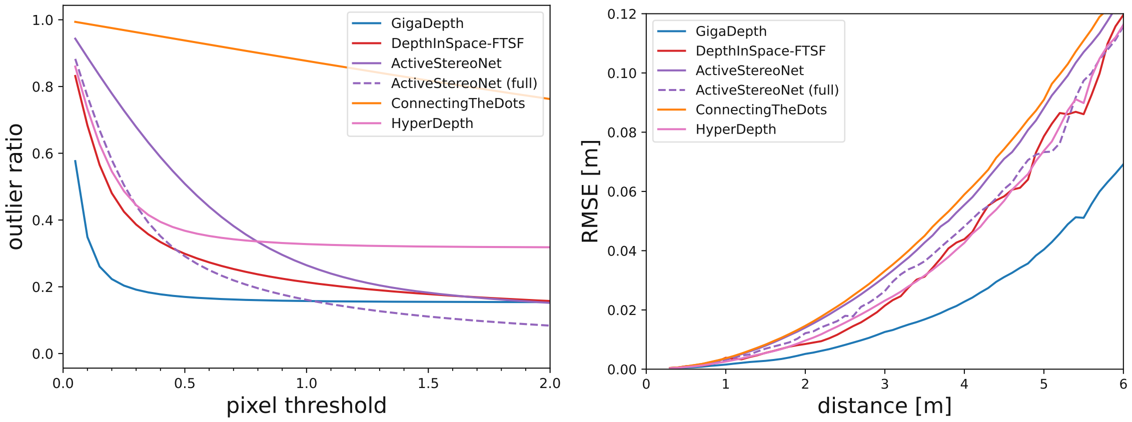


Figure 3.9: While the center of the dot pattern is captured via the Structure Core, the peripheral regions are captured with a RealSense (D415) camera (left). Each of these images is accompanied by a hand-drawn mask, marking the region of interest. Accumulating the dots on the image plane of the dot-projector (center), and finally clustering them allows us to reconstruct the pattern (right). To increase the variety of the resulting texture, we generate the textures with different speckle sizes defined by the radius of a Gaussian filter.

textured with partial randomization for selected surfaces[1]. Aside from the ground-truth depth/disparity, we render stereo images as well as pixel-level masks corresponding to areas where the pattern projector does not have enough influence. 15k sequences of four frames each are rendered this way. The test set features a different set of objects and textures and offers 9k frames.

To extract a texture for the pattern, we point the Structure Core as well as a RealSense sensor with disabled projector towards a wall and capture multiple IR frames. While the center of the pattern is covered by the Structure Core itself, the RealSense captures the fringes as seen in Figure 3.9. In the processing phase, we manually match three points between each pair of Realsense and Structure Core IR frames, and create a mask to cut out all parts of the image that either are too distorted, not properly lit, or do show a non-planar surface. From these images, we extract the center points for all valid speckles, calculate a rough alignment based on the manually selected correspondences

[1] Randomly selected textures on planes used in walls as well as cube, sphere, cylinder and pill shapes.



(a) Outlier ratios over a range of thresholds (b) $RMSE$ over distance

Figure 3.10: Outlier ratios over different thresholds (a) and $RMSE$ over different distances (b) evaluated on synthetic data. For the $RMSE$, only pixels with a disparity error smaller than 1 pixel are evaluated.

and perform ICP alignment in the three dimensional space. The resulting point clouds are projected onto the image plane of the projector where the measurements for the individual speckles are clustered together and averaged. In a final step, we generate a set of textures from these center points with varying degrees of blur. This allows for greater variety in our rendered dataset.

We further captured a real-world dataset consisting of 967 scenes with four frames each to finetune the baseline algorithms. 241 of these scenes have a second set of images with the dot-projector disabled. This is essential to train the edge detector required by Connecting The Dots [24] and precompute the optical flow required by DepthInSpace [25]. To evaluate the performance, we collect 11 scenes with ground-truth data captured by a Photoneo MotionCam-3D M in scanning mode. With an accuracy stated as $< 0.250mm$ at a distance of $0.65m$, this sensor is magnitudes more accurate than the Structure Core. Translating these accuracy numbers to the geometry and intrinsics of the Structure Core, results in disparity errors of ~ 0.05 pixel. Provided the alignment between both sensors via the ICP algorithm is similarly accurate, we can expect ground-truth disparity adequate for our evaluations.

3.2.3 Results

The experiments on rendered and real data show that we have a real-time method capable of capturing dense and highly accurate disparity maps. In addition to the increased accuracy and sensitivity to comparable methods, we show that our approach bridges the domain gap much more gracefully.

3.2.3.1 Rendered Data

The synthetic data allows us to assess each method in terms of accuracy by measuring $RMSE$ and outlier ratios, robustness against signal attenuation and, at last, by their

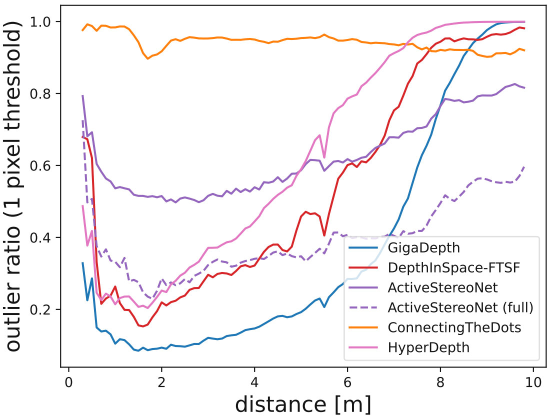


Figure 3.11: The outlier ratio with a 1 pixel threshold $o(1)$ over distance performed on our synthetic dataset. While these results will not translate to a real-world application, they demonstrate the algorithm's relative sensitivity to the dot pattern. ActiveStereoNet performs better at higher distances as the stereo modality can triangulate features that are indigenous to the scene.

ability to handle sudden changes in depth. To further magnify the focus on the model architectures themselves as opposed to the training strategy, we train versions of ActiveStereoNet [27] and Connecting The Dots [24] with full supervision by utilizing the L1 loss.

As presented in Figure 3.10 as well as the qualitative results in Figures 3.13 and 3.24, it is evident that the branching approaches of HyperDepth [23] and GigaDepth can deliver more precise results but cannot always reach the level of completeness of the CNN-based ActiveStereoNet [27] and DepthInSpace [25]. Plotting the outlier ratios with a 1 pixel threshold over distances in Figure 3.11, we can infer that our algorithm has a higher sensitivity towards the dot pattern than any of the baseline methods. ActiveStereoNet [27], which utilizes the second camera, performs stereo matching and therefore shows better performance at higher distances that are otherwise insufficiently lit by the projector.

To substantiate the claim about our algorithm's robustness in situations of a fragmented pattern, we evaluate its performance in regions around depth discontinuities. After extracting the depth discontinuities from the ground-truth, we increasingly dilate these edges to obtain regions of different proximity. In Figure 3.12, we plot the outliers in these regions along the a range of radii around edges. It becomes evident that our method exceeds the remaining methods when high pixel accuracy is needed. If accuracy is not of utmost importance, ActiveStereoNet [27] can achieve a lower outlier ratio as it actually benefits from the strong intensity gradients that often coincide with object boundaries.

As reflected in Figures 3.10 to 3.12, it is challenging to achieve acceptable results for the Connecting The Dots [24] algorithm on our dataset. To still include a truthful comparison, we perform a comparison of HyperDepth [23], Connecting The Dots [24], and our algorithm on the dataset provided by Riegler et al. [24] in Section 3.2.3.3.

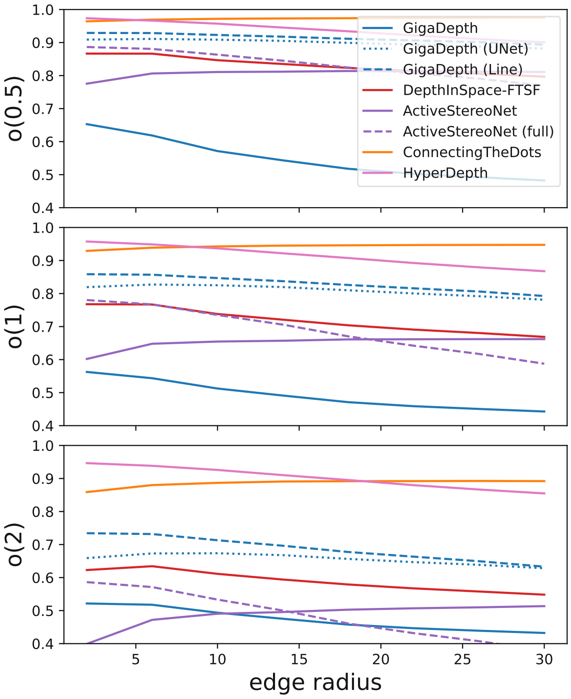
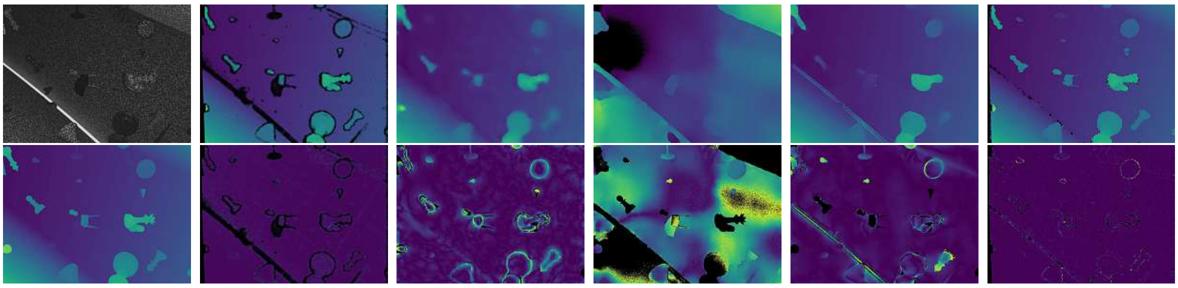


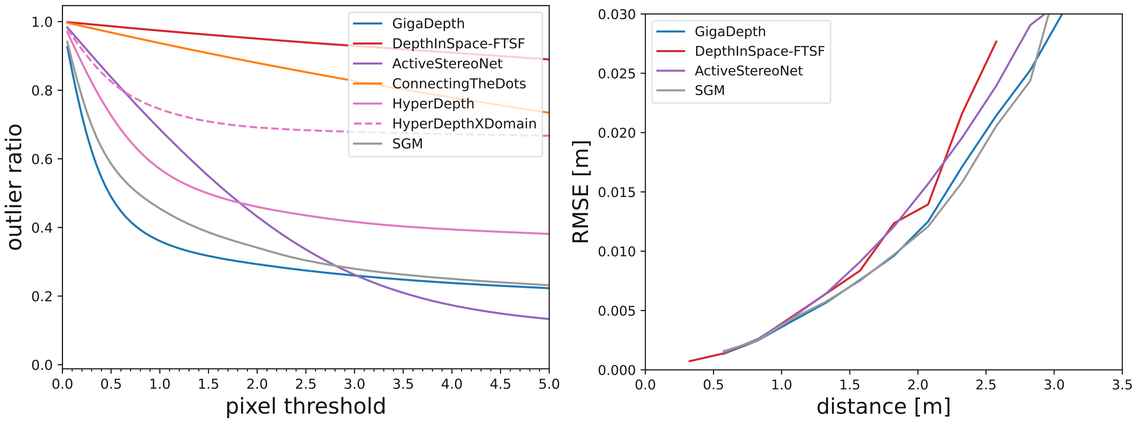
Figure 3.12: Outlier ratios on regions around depth discontinuities with our rendered dataset. Evaluated for different outlier thresholds over a growing region around object edges. Note that only estimates by ActiveStereoNet [27] benefit from a proximity to the edges as they often coincide with strong intensity gradients.



(a) IR + GT (b) HD (c) ASN (d) CTD (e) DIS (f) GD (ours)

Figure 3.13: Disparities (top b - f) of different algorithms applied on a scene rendered with Unity3D. Color-coding is applied for disparity errors (bottom b - f) of 0-5 pixel with outliers (> 5 pixel) being black. Algorithms are: HyperDepth (b, HD), ActiveStereoNet (c, ASN), Connecting The Dots (d, CTD), DepthInSpace (e, DIS) and GigaDepth (e, GD). (more in Figure 3.24)

3.2.3.2 Real-world Data



(a) Outlier ratio over a range of thresholds

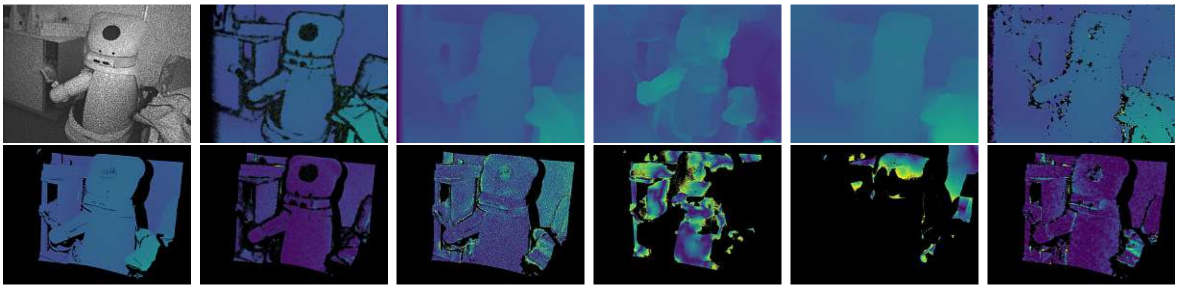
(b) $RMSE$ over distance

Figure 3.14: Experiments on real-world data. Outlier ratios over different thresholds (a) and $RMSE$ over different distances (b) evaluated on captured data. For the $RMSE$, only pixels with a disparity error smaller than 1 pixel are evaluated.

For our real-world evaluation, we align the data from the Photoneo MotionCam-3D M and the point clouds derived from each algorithm using ICP. Projecting the ground-truth point cloud to the respective camera frames yields the disparity maps we compare against (Figure 3.16). We plot the outlier ratios for our set of algorithms in Figure 3.14a and show favorable results compared to all baselines. Only ActiveStereoNet [27] achieves superior outlier ratios above thresholds of ∼3 pixel, which we attribute to this method's ability to fall back to its stereo matching roots when the pattern is absent. Evaluating the $RMSE$ in line with the experiment on artificial data is challenging as not all methods produce enough usable depth samples at the full depth range. Compared to the remaining methods (HyperDepth [23], DepthInSpace [25] and SGM [2]), we demonstrate equivalent to favorable performance for our method (Figure 3.14b). Note that basing a comparison on the depth $RMSE$ leads to a distorted view due to the influence of errors in rectification/calibration and (mis)alignment between the captured frame and ground-truth data. It is advisable to focus on the pixel-metrics as they depend less on sensor geometry and, to some extent, even allow to compare algorithms across sensors.

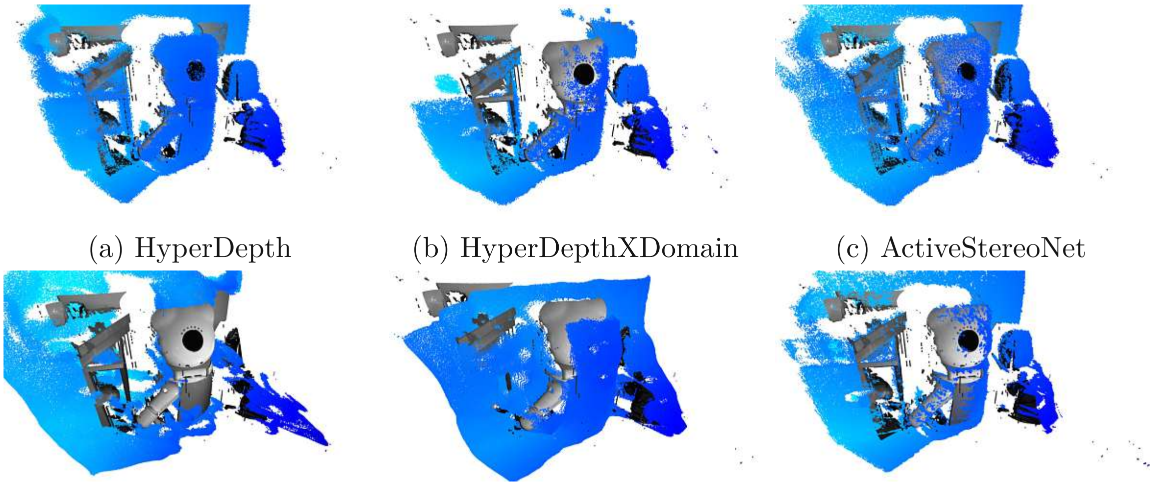
We also include a comparison to HyperDepth [23] when trained on artificial data (marked as XDomain) and a variant trained on SGM [2] to eliminate the influence of the different training strategies. Similar to the results on the synthetic data, the results cannot convince and further demonstrate that, for these approaches, the sim-to-real domain gap can only be crossed with semi-supervised fine-tuning on the target domain.

A qualitative assessment in Figures 3.15 and 3.25 reveals that GigaDepth delivers notably lower disparity errors compared to the baseline methods, with measurements mostly being omitted at object fringes and pixels that are shadowed from the pattern projector. Despite these positive results, the domain transfer remains challenging and



(a) IR + GT (b) HD (c) ASN (d) CTD (e) DIS (f) GD (ours)

Figure 3.15: Disparities (top b - f) of different algorithms applied on scenes captured by the Occipital Structure Core with ground-truth (GT) captured by a Photoneo MotionCam-3D M. Color-coding is applied for disparity errors (bottom b - f) of 0-5 pixel with outliers (> 5 pixel) being black. Algorithms are: HyperDepth (b, HD), ActiveStereoNet (c, ASN), Connecting The Dots (d, CTD), DepthInSpace (e, DIS) and GigaDepth (e, GD). (more in Figure 3.25)

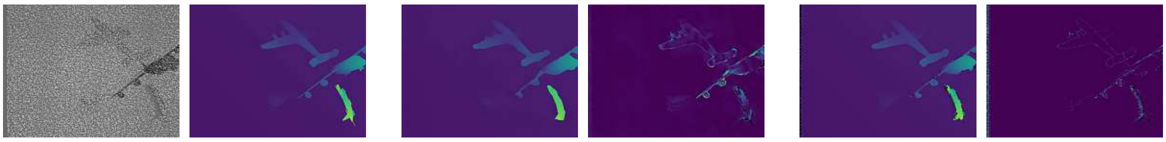


(a) HyperDepth (b) HyperDepthXDomain (c) ActiveStereoNet

(d) Connecting The Dots (e) DepthInSpace-FTSF (f) GigaDepth (ours)

Figure 3.16: The point cloud captured with the Photoneo Scanner (greyscale) aligned with the cloud captured by the Structure Core (blue). The point clouds resulting from the algorithms (blue) are aligned with the ground-truth (greyscale) via ICP.

leads to artifacts in otherwise non-challenging situations. Surfaces that are comparable between the synthetic and real-world domains lead to slightly different results by our algorithm. While synthetic data would almost always result in a dense and smooth estimate, an equivalent scene captured by our sensor would lead to depth estimates with small, infrequent, and arbitrary holes.



(a) IR + GT (b) Connecting The Dots (c) GigaDepth (ours)

Figure 3.17: Disparities (left b, c) of the different algorithms on an artificial dataset based on models from [54]. Color-coding is applied for disparity errors (right b, c) of 0-5 pixel with outliers (> 5 pixel) being black. (a) shows the input infrared image (IR) and ground-truth (GT).

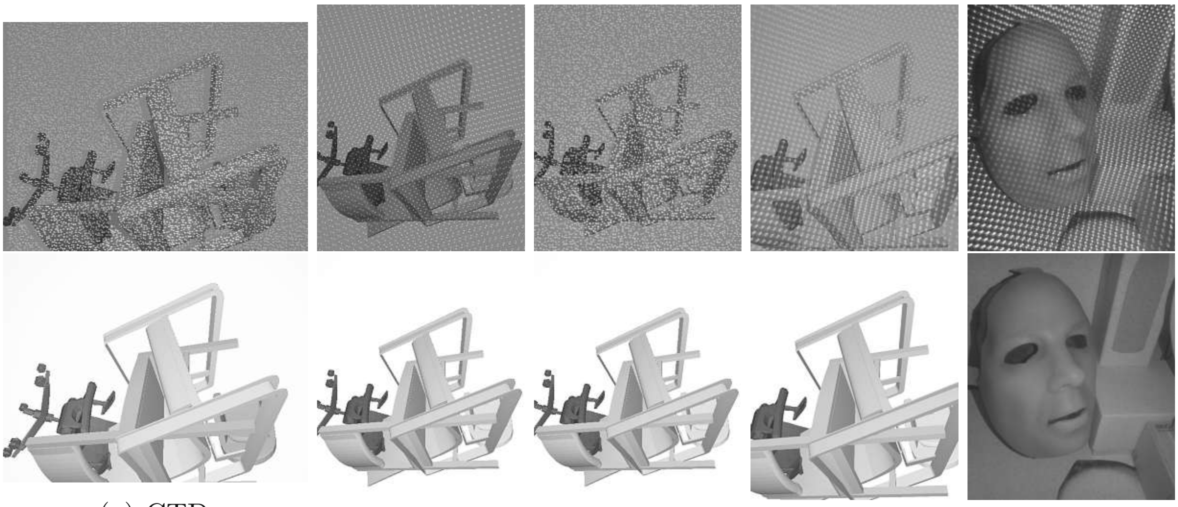
3.2.3.3 Isolated Comarisons on Native Baseline Datasets

While the datasets used in [24], [25] are perfectly suitable to demonstrate the respective abilities for semi-supervision, they are not well suited to prepare our fully supervised method for the simulation-to-real domain gap. To prepare our algorithm for real-world application we rely on cluttered renderings, with textures and diverse lighting conditions that seem to be too challenging for the training strategies of [24], [25]. This is despite our attempts of adapting the cost function, fully supervised pre-training or capturing the ambient images (Figure 3.8) required by [24], [25]. We thus also train and test our method on the datasets native to [24], [25].

Connecting The Dots: Riegler et al. [24] use a synthetic as well as a captured dataset to train and test their method. The synthetic dataset is comprised of 9216 scenes featuring 4 frames each (640 × 480) and objects of the "chair" class taken from ShapeNet [54]. These objects are put in front of a single untextured plane and are lit by environment light and a projector that emits a crop of a pattern that is used by a Microsoft Kinect v1 or PrimeSense Carmine (see Figure 3.18a). In the original publication, these renders are used to train (index 1024 to 9216) as well as validate/test (index 0 to 1024) the algorithm.

Instead of the full (Kinect/PrimeSense) pattern with a resolution of 1280 × 1024, a 640 × 480 pixel crop is used. This preserves the pixel density compared to data captured by the original sensor as the raw frames are captured at 1280 × 1024 while the renderings happen at 640 × 480. For the real-world experiments, it is not entirely clear which training data is used as the training regime requires some captures with deactivated projector to train the edge-detection network. This makes recreating the results in [24] challenging even though the testing is conducted on a PrimeSense-based dataset provided in [55]. Training seems to happen on this dataset as well as additional captured, unpublished data. It is noteworthy that these images are downsampled from 1280 × 1024 to 640 × 480 leading to an input with different pattern density compared to the cropping approach used during rendering.

While we cannot train our algorithm for the real-world data provided in [55] due to a lack of precise ground-truth and low sample-count, we do provide a comparison on synthetic data in Section 3.2.3.1. However, we deviate from [24] by rendering a test set consisting of 1024 scenes with objects of the "airplane", "car" and "watercraft" to reduce the similarity to the data the algorithm is trained on. We also modify our backbone CNN for this experiment to not downsample but to operate at the full input resolution.

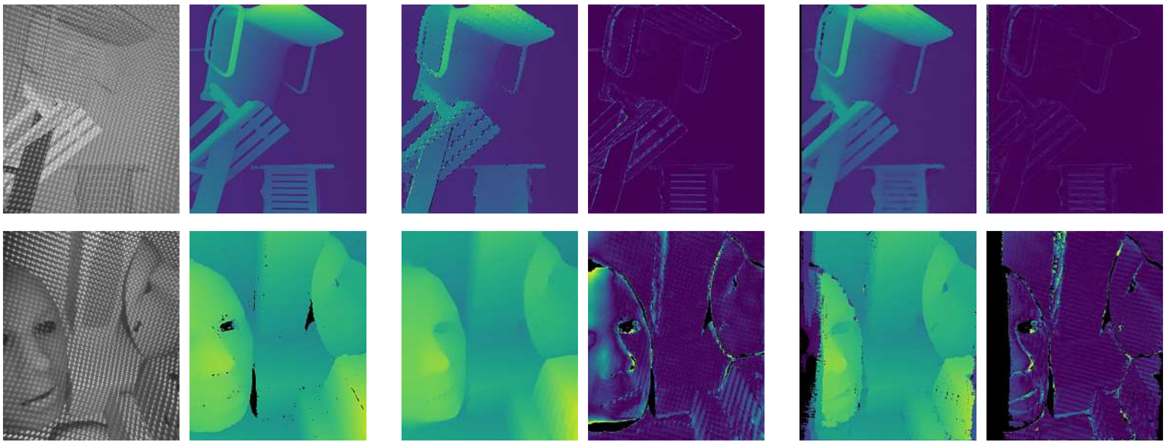


(a) CTD (b) DIS default (c) DIS Kinect (d) DIS real (e) DIS

Figure 3.18: The training datasets of Connecting The Dots (CTD) compared to the one of Depth In Space (DIS). The rendered data (a - d) completely relies on textureless models drawn from the "chair" class of the ShapeNet [54]. The used Kinect pattern is a crop of the original pattern (a, d) to achieve a speckles per pixel ratio similar to the original sensor. The completely artificial pattern (b), as well as the captured real pattern (d, e) in comparison show a lower density of speckles. Real-world data captured with the real hardware (e) only capture few sequences, and do not strictly split between training, validation and test set. See the last row of Figure 3.19 for a frame of the test set.

DepthInSpace: The synthetic datasets used in [25] uses the same methodology of [24] but differs in the aspect ratio of the images (512 × 432 vs. 640 × 480) and the used patterns (see Figure 3.18). In comparison to [24], the authors have been more clear about the data used to train the real-world variant of their network. The public dataset contains 148 sequences, each consisting of 4 frames, that have been sampled from 4 longer continuous video captures mostly showing close-range masks and simple scenes. Unfortunately, the sampling for the validation data is the same as the one for testing and consists of every 8th frame in those sequences. As a result, the training data is very similar to the target/test domain. We further note that the utilized patterns are sparser and more regular than the ones used in our dataset or the one used in [24]. We thus modify our backbone network to not downsample and increase the receptive field to compensate for the lower pattern density. For HyperDepth [23], whose random forests operate on a hardcoded window of 32 × 32 pixels, this might be a factor contributing to decreased performance.

Despite these circumstances, diminishing the significance of these comparisons, we extend table 1 of [25] to demonstrate our algorithm's competitive performance for



(a) IR + GT (b) DepthInSpace (c) GigaDepth (ours)

Figure 3.19: Disparity (left b, c) estimates by DepthInSpace and GigaDepth applied on the different test datasets native to DepthInSpace. Color-coding is applied for disparity errors (right b, c) of 0-5 pixel with outliers (> 5 pixel) being black. The top row shows the results on rendered data based on the captured pattern. The bottom row shows the results on captured real-world data.

applications where accuracy better than 0.5 pixel is required and ground-truth training data can be supplied (Table 3.5). Qualitative results in Figure 3.19 show that GigaDepth performs well on captured data, even though we do not perform any fine-tuning. On the left side of the captured images, however, we register a decline in performance. We assume this is because the synthetic training data is barely showing this region of the pattern, mainly because the objects are rendered in a narrow range of distances.

data	method	$o(0.5)$	$o(1)$	$o(2)$	$o(5)$
Synthetic Kinect	SGM	10.36	9.13	8.76	2.45
	HD	4.38	3.22	2.69	2.39
	CTD	2.74	1.45	0.77	0.24
	DIS-SF	2.11	1.13	0.59	0.16
	DIS-FTSF	1.92	**1.00**	**0.51**	**0.14**
	DIS-MF	1.59	0.72	0.33	0.10
	GD (ours)	**1.88**	1.63	1.39	0.82
Synthetic default	SGM	12.93	11.64	11.22	4.06
	HD	7.35	6.48	6.11	5.86
	CTD	3.38	1.71	0.85	0.28
	DIS-SF	2.31	1.24	0.62	0.19
	DIS-FTSF	1.96	**0.95**	**0.45**	**0.12**
	DIS-MF	1.58	0.71	0.32	0.10
	GD (ours)	**1.75**	1.48	1.20	0.82
Synthetic captured	SGM	12.45	10.37	9.55	4.83
	HD	6.13	4.92	4.34	4.00
	CTD	3.76	2.25	1.03	0.37
	DIS-SF	3.66	2.16	1.00	0.23
	DIS-FTSF	2.87	**1.48**	**0.66**	**0.17**
	DIS-MF	2.46	1.24	0.54	0.14
	GD (ours)	**2.41**	2.82	1.24	0.86
real	SGM	25.54	19.23	17.75	16.96
	HD	34.62	25.09	22.49	21.77
	CTD	22.74	9.26	3.79	**1.00**
	DIS-SF	17.95	7.93	3.59	1.14
	DIS-FTSF	**17.06**	**7.48**	**3.47**	1.11
	DIS-MF	16.07	7.14	3.41	1.09
	GD (ours)	18.59	12.06	10.34	8.80

Table 3.5: We extended table 1 of [25] with our evaluation. Three of the datasets are rendered with a completely artificial pattern (default), a pattern captured from a Microsoft Kinect v1 (Kinect) and one captured from the intended sensor (captured). The last set of experiments is based on real data. The compared methods are: semi-global matching (SGM), HyperDepth (HD), Connecting The Dots (CTD), Depth In Space for single frames (DIS-SF), Depth In Space on single frames with fine-tuning by its multi-frame variant(DIS-FTSF), Depth In Space with a spatial network capable of consolidating the results of multiple single-frame results (DIS-MF) and finally our GigaDepth (GD). The best single-frame results are presented in bold.

3.2.4 Ablation

Architecture

To corroborate our choice of network architecture we benchmark different combinations of feature extractors and regressors on our synthetic dataset and report outlier ratios in Table 3.6. The first set of experiments is based on a relatively powerful and compute-intensive UNet that takes an order of magnitude more time to execute than all the other baseline methods ($1s$ compared to $\sim 50ms$).

Fully supervising the UNet on our regression task without any additional network does not yield any usable behavior. The same can be concluded when supplying output layers with per-line weights. Using the network as a backbone for our regression network, however, gives superior performance even to our own backbone, albeit it being at much higher cost. Looking at the outlier ratio for low thresholds ($o(0.1)$), we do not see much improvement, but the overall amount of valid pixels ($o(1)$) seems to have increased. We

BB	Reg.	$o(0.1)$	$o(0.5)$	$o(1)$	params	c_1	c_2, o_2	c_3, o_3	w. sharing
UNet	none	89.55	54.78	33.41	81M				
	lines	94.27	72.39	50.23	105M				
	$^{1920}/_{2}$	**34.01**	**14.32**	**12.74**	446M	16	12, 2	10, 3	4
Our Backbone	$^{288}/_{2}$	51.86	19.88	17.16	134M	8	6, 1	6, 1	1
	$^{384}/_{2}$	60.49	18.64	16.29	217M	16	6, 1	4, 2	2
	$^{640}/_{1}$	44.50	19.05	17.39	131M	16	10, 3	4, 2	2
	$^{640}/_{2}$	44.37	17.53	16.02	275M	16	10, 3	4, 2	2
	$^{640}/_{3}$	38.49	18.11	16.62	359M	16	10, 3	4, 2	2
	$^{1280}/_{2}$	**34.91**	**16.97**	**15.72**	379M	16	10, 3	8, 4	3
	$^{1920^a}/_{2}$	41.27	17.20	15.97	388M	16	12, 2	10, 3	4
	$^{1920}/_{2}$	38.08	17.20	15.97	388M	16	12, 2	10, 3	4
	$^{2688}/_{2}$	44.59	17.26	15.93	429M	16	14, 1	12, 2	6

Table 3.6: Different configurations of our architecture tested on synthetic data. We test two backbones, ours as well as a full UNet network both of which dominate execution time with $\sim 70ms$ and $\sim 1s$, respectively (RTX 2070 Max-Q). Regressors are specified as $^{c}/_{l}$ with the number of classes c and MLP layers l. All networks take IR + LCN as input (superscript ^{a} omits the LCN). Note that with increasing class count, we more aggressively apply our weight sharing scheme for hidden layers of the regressor stage. The right half of the table details the used classes (c_1, c_2, c_3) at each stage of the classification tree and the overlap (o_1, o_2, o_3) between the raw classification output of neighboring nodes. We utilize a strategy of weight sharing in final regression MLP that allows to share the weights of hidden layers between neighboring classes.

can attribute this to the capacity of the network to incorporate high-level information without sacrificing the ability to encode local features. Regarding quality, we find in Figure 3.20 that the UNet-based version will output disparity measurements that might not be invalid but seem to be crude and erroneous interpolations.

The second set of experiments operates with the backbone we tailored for this task and aims at analyzing the influence of the MLP tree structure. Varying the amount of output classes between 288 and 2688, we see the return of investment diminishing after 1280 classes. While the runtime is almost unaffected by the class count, the increasing amount of parameters is a cause for concern. Note that the parameter count would rise even more steeply if we were not increasingly aggressive with our strategy of sharing weights between consecutive output classes in the latent regressor layers. The influence of the depth of the MLPs is shown in three variations of the 640 class version, showing diminishing returns for MLPs deeper than two layers. By adding/removing one of the latent 32 channel layers from the classification MLPs described in Table 3.2, we demonstrate that the increased amount of weights that coincide with the added layers is not justified by the increased performance. Finally, we evaluate for possible performance regressions that stem from omitting the LCN filter with a variation (superscript ^{a}) of our 1920 class experiment. While the measured degradation is small, it is still notable

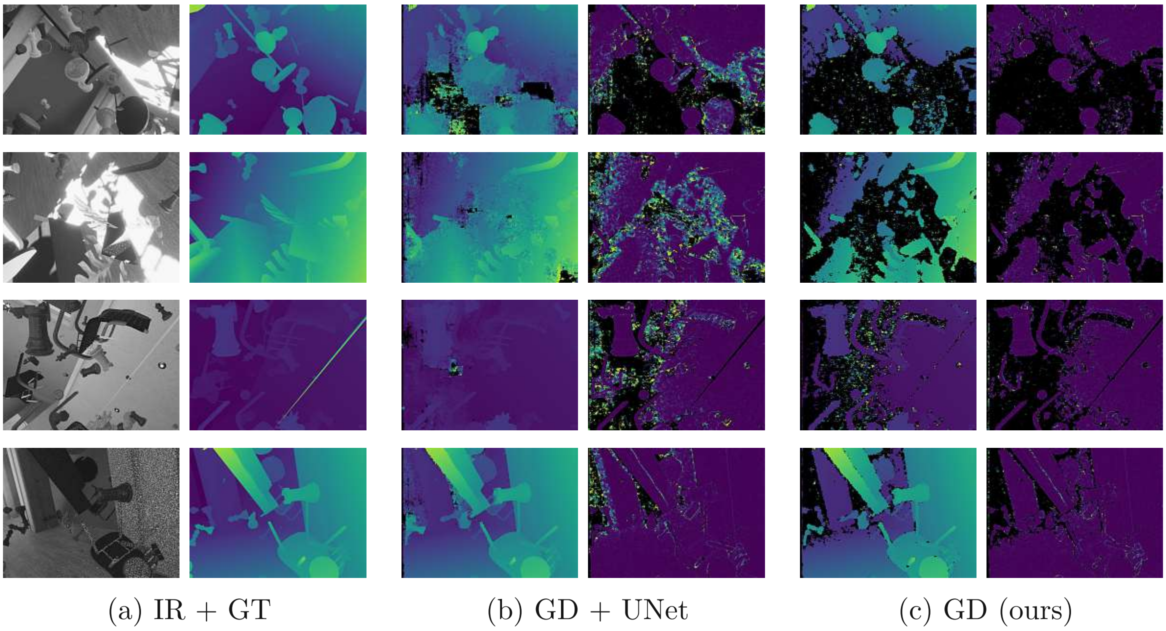

(a) IR + GT (b) GD + UNet (c) GD (ours)

Figure 3.20: Disparities (left b, c) of our algorithm applied on scenes rendered with Unity3D. Color-coding is applied for disparity errors (right b, c) of 0-5 pixel with outliers (> 5 pixel) being black. While our algorithm with a heavier UNet backbone (b) produces more complete results, it requires 1s for computation. Our backbone (c) achieves a better trade-off with an execution time of 60ms.

enough to keep the relatively cheap LCN filter as part of this pipeline. For MLPs with a layer count of one ($l = 1$), we do not have hidden layers, and for two layers MLPs ($l = 2$), we use the network described in Table 3.2 that, despite the difference for the other layers, uses a 3-layer-MLPs at the root nodes. For $l = 3$, MLPs would feature [64, 64, 32, 32, $^c/_o$] channels in the root nodes and [64, 32, 32, $^c/_o$] in the remaining ones ([in, $hidden$, $^c/_o$] notation). The structure of the regressor MLPs was kept the same in all experiments [32, 32, $^1/_1$].

Vertical jitter In Figure 3.21, we explore the effect of omitting the jittering augmentation during training. It shows shifting regions of failing depth estimation even within short sequences. This is an indication that the sensor geometry and components are not entirely rigid or susceptible to temperature. Randomly shifting the training images by just a few vertical pixels robustifies the algorithm.

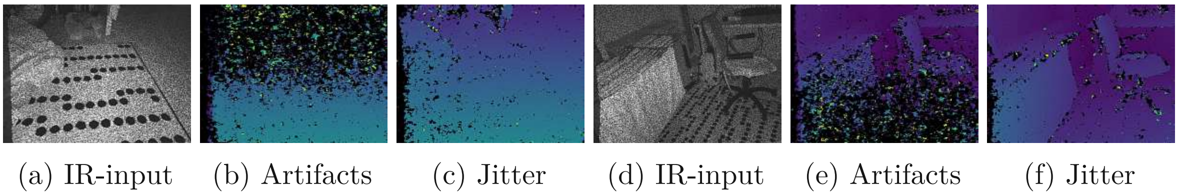

(a) IR-input (b) Artifacts (c) Jitter (d) IR-input (e) Artifacts (f) Jitter

Figure 3.21: Two frames captured in the same sequence. The captured images (a, d) seem to change their vertical alignment with the projector causing the algorithm to fail in shifting regions (b, e). Applying vertical jitter of 4 pixel during the training phase makes the network agnostic to this variability (c, f).

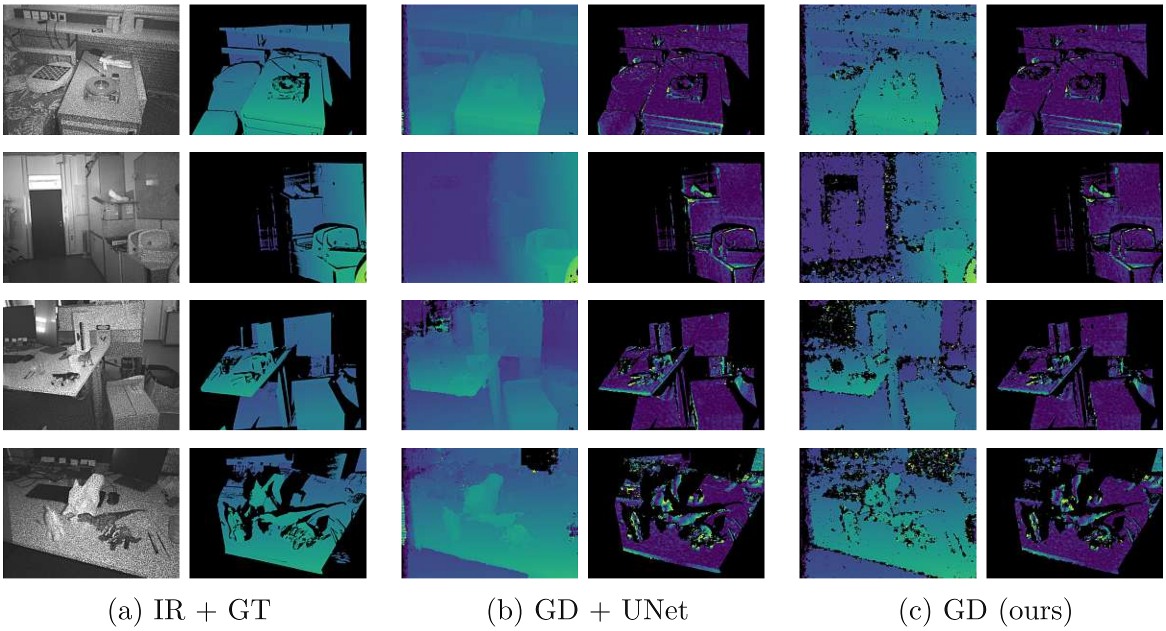
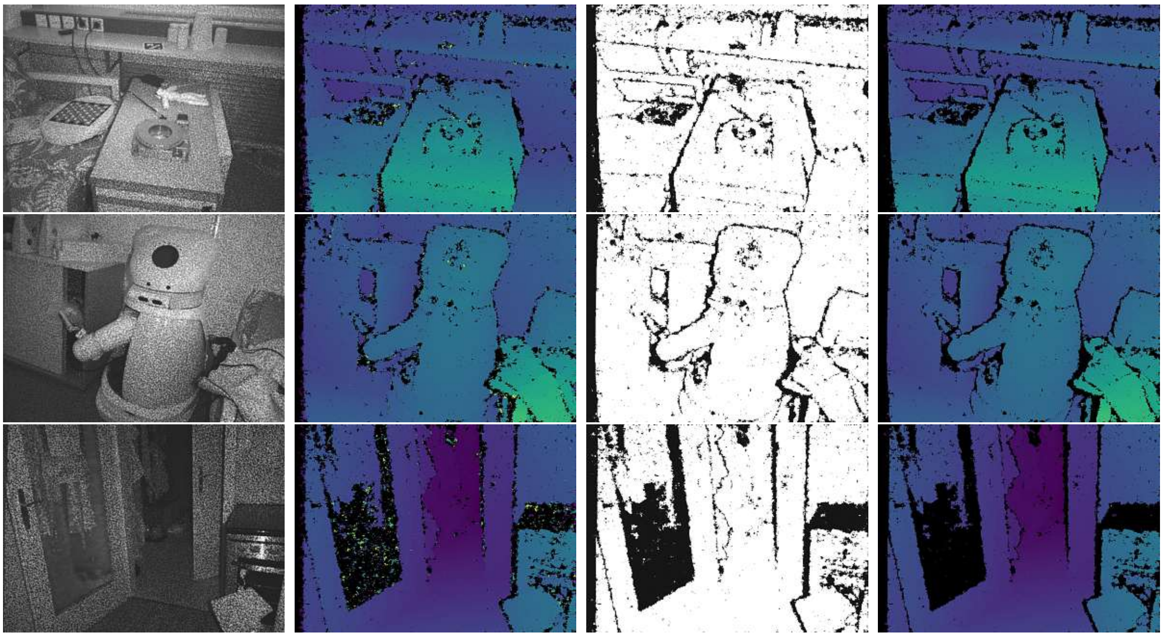

(a) IR + GT (b) GD + UNet (c) GD (ours)

Figure 3.22: Disparities (left b, c) of our algorithm applied on scenes captured with the Occipital Structure Core. Color-coding is applied for disparity errors (right b, c) of 0-5 pixel with outliers (> 5 pixel) being black. While our algorithm with a heavier UNet backbone (b) produces more complete results, it requires $1s$ for computation. Our backbone (c) achieves a better trade-off with an execution time of $60ms$.

3.2.5 Invalidation Network

We train the invalidation network presented in Section 3.1.4 on the synthetic training data as the main network did show signs of over-fitting to the training-set. This is a pragmatic decision after a dataset distinct enough from the training dataset was required to avoid overestimating the quality of disparity maps. However, this diminishes the significance of tests performed on synthetic data leading to this isolated comparison here. The previous experiments thus did not employ the invalidation network. In Figure 3.23 we employ the invalidation network on captured data unseen during the



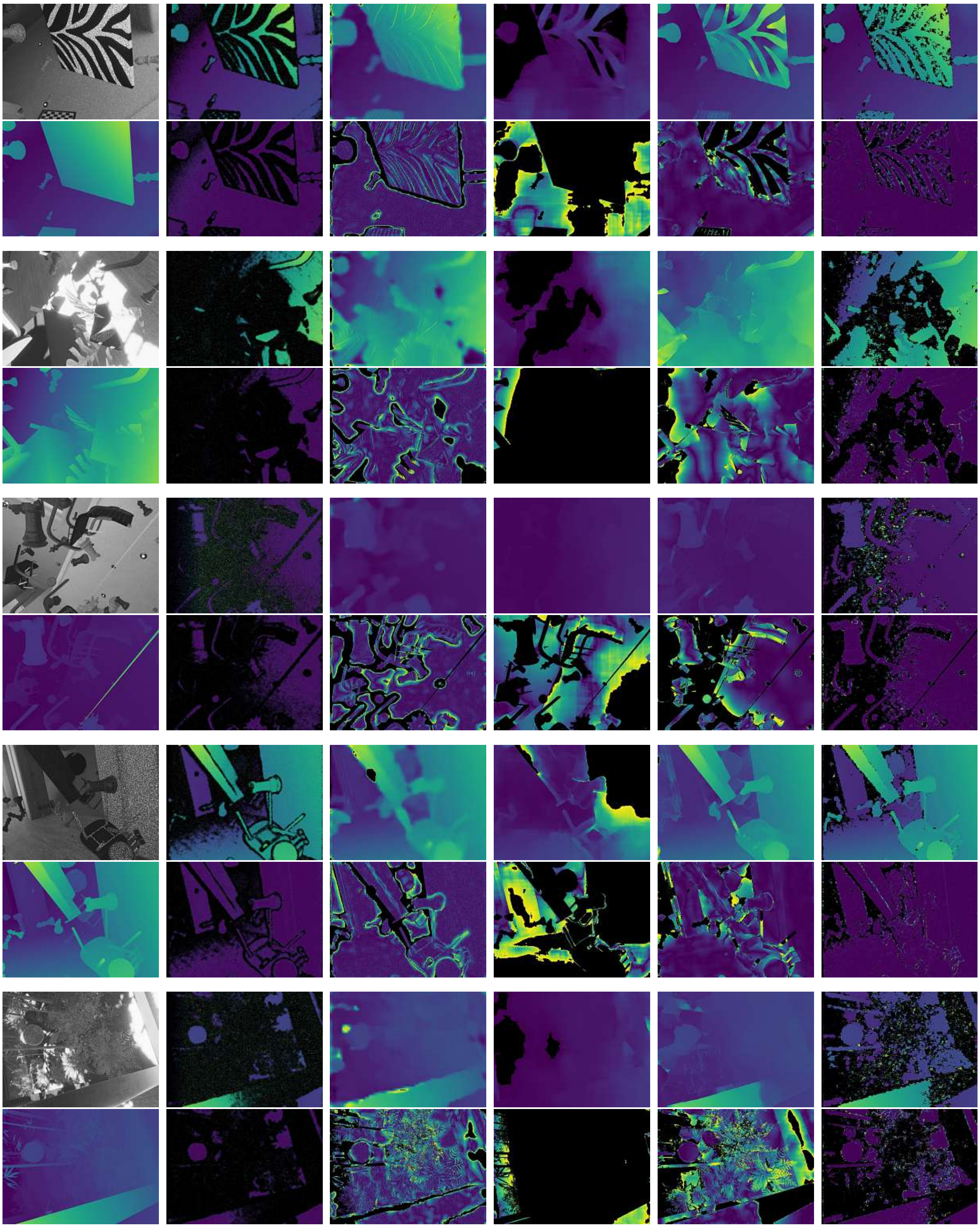
(a) IR (b) Raw disparity (c) Mask (d) Valid disparity

Figure 3.23: Gigadepth applied to real-world IR images. Input (a) is computed to a raw disparity output (b) by GigaDepth. The mask by our invalidation network (c) can be used to remove/invalidate questionable pixels/estimates leading to a sparser but more reliable result (d). Note the exaggerated borders around objects in (d) compared to (b). The left edge of the raw disparity maps yields estimates that make the invalidation network overly optimistic leaving undetected errors.

training processes of both, the main network and this invalidation network. The results demonstrate that while most of the obvious outliers are masked out, some otherwise sensible pixels around the edges of objects become invalidated too. Furthermore, it is notable that GigaDepth itself fails to correctly interpret the pattern on the left side of the image (also pronounced in Figure 3.19) as these are pixels that are underrepresented in the training set. Our invalidation network in turn fails to separate out all the error cases.

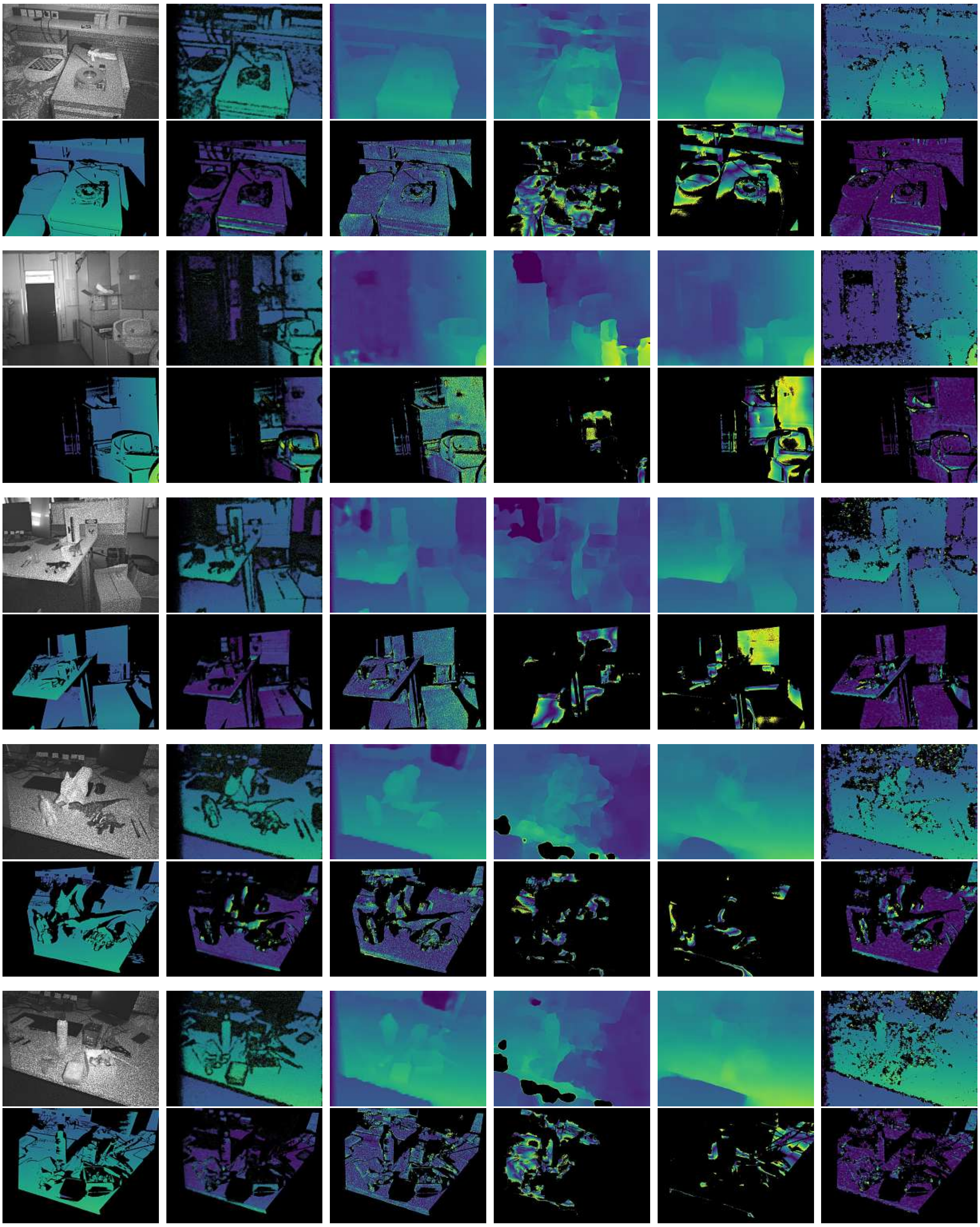
3.3 Discussion

This chapter introduced HyperDepth, an algorithm that outperforms the state of the art on extracting depth from structured light. Benchmarks on artificial as well as real data show precision and sensitivity superior to comparable methods. It is shown that, while pure CNN-based methods struggle to deliver high accuracy for these regression tasks, combining a CNN-based backbone with a regressor consisting of weight-adaptive layers can overcome this challenge. These weight-adaptive layers enable us to implement a neural decision tree with small specialized regressors at the leaf nodes. While the comparable HyperDepth [23] follows a branching approach similar to our regression stage, the decision functions on each node are comparably trivial and thus struggle to model the different influences of scene and surface compositions. The focus on dot patterns and the strategy of keeping the receptive field small allows our large set of small neural networks to specialize for their respective regions within the pattern. Despite the necessity for accurate ground-truth to train the classification part, most easily obtained using artificial data, our approach's resilience to domain shift is demonstrated by the good performance on real-world data.



(a) IR + GT (b) HD (c) ASN (d) CTD (e) DIS (f) GD (ours)

Figure 3.24: Disparities (top b - f) of different algorithms applied on scenes rendered with Unity3D. Color-coding is applied for disparity errors (bottom b - f) of 0-5 pixel with outliers (> 5 pixel) being black. Algorithms are: HyperDepth (b, HD), ActiveStereoNet (c, ASN), Connecting The Dots (d, CTD), DepthInSpace (e, DIS) and GigaDepth (e, GD).



(a) IR + GT (b) HD (c) ASN (d) CTD (e) DIS (f) GD (ours)

Figure 3.25: Disparities (top b - f) of different algorithms applied on scenes captured by the Occipital Structure Core. The ground-truth (GT) is captured with the Photoneo MotionCam-3D M and compared to other algorithms. Color-coding is applied for disparity errors (bottom b - f) of 0-5 pixel with outliers (> 5 pixel) being black. Algorithms are: HyperDepth (b, HD), ActiveStereoNet (c, ASN), Connecting The Dots (d, CTD), DepthInSpace (e, DIS) and GigaDepth (e, GD).

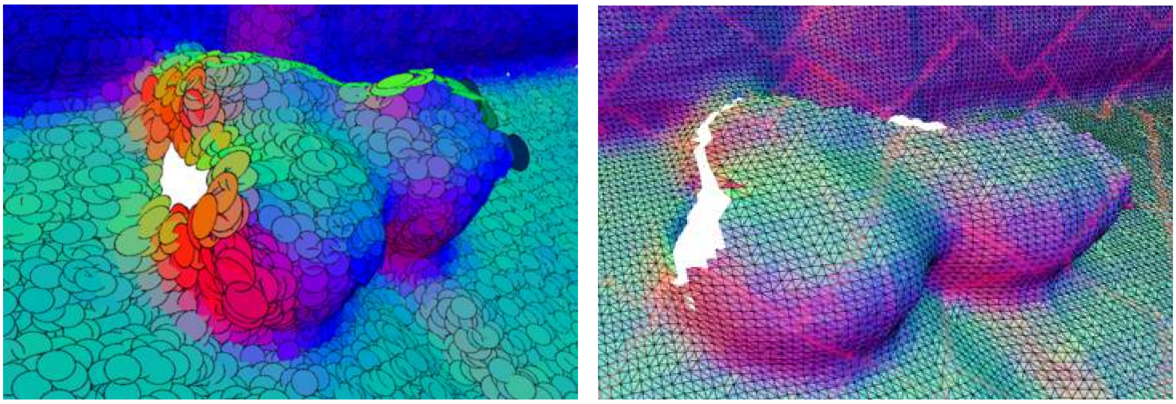
Chapter 4

ScalableFusion: High-resolution Mesh-based Real-time 3D Reconstruction

Sensors similar to the one discussed in the previous chapter have begun flooding the market in the form of affordable RGB-D sensors for the entertainment industry in 2010. The Microsoft Kinect (v1) was meant to be an input device to engage games employing a player's entire body, and due to its low price and availability encouraged a wide adoption of these sensors in the computer vision community. Most notably, the fusion of multiple frames into one consistent 3D reconstruction of scenes has fueled expectations for a large variety of applications. Robotics as well as Augmented Reality, just to name two examples, benefit greatly from a complete representation of environments.

Generating consistent 3D reconstructions based on these sensors is challenging because of two main reasons. Firstly, geometrical consistency is not provided by the depth sensor due to distortions and incorrect depth values, noise and other effects caused by rolling shutter sensors. Secondly, photometric consistency can not be achieved by the RGB sensor owing to limited geometrical resolution of the reconstruction, the dependence of perceived intensity on the camera and object position as well as lighting conditions.

While future sensors may mitigate these problems, present sensors all suffer from the same problems [56], [57]. Therefore, it is the task of RGB-D reconstruction algorithms and the supporting hardware to cope with the challenges. The most common approaches are based on KinectFusion ([28], [32], [30]) and ElasticFusion ([39], [58], [59]). The first class of methods utilize TSDFs to carve out the free space of a regular voxel grid of uniform resolution. The second class of methods stores the geometrical information in Surfels which are small unconnected discs of different size and color. Essentially, both methods attempt to achieve geometrical and photometric consistency by averaging measurements from multiple perspectives. The averaging schemes achieve pleasing results for geometry but they are unsatisfactory for the color. This is, firstly, due to the widespread practice of ignoring exposure and gain information of the sensors [60]. But more importantly, the limitation is due to the underlying data structures that enforce a one-to-one relationship between color and geometric resolution, which leads to a loss of color information. It is also common for these algorithms to steadily aggregate new data on the GPU until it runs out of memory. This disqualifies their use for serious applications in robotics.



(a) ElasticFusion. Surfels are color coded by normal vectors with additional outlines in black for the borders.

(b) ScaleableFusion (direct mesh-based reconstruction) with outlined triangles.

Figure 4.1: A meshed set of three apples (zoomed in from Figure 4.2). Due to not relying on excessive overlapping of surfels to achieve a tight surface model, mesh reconstructions allow surfaces to be described more precisely and efficiently.

Our solution to the described shortcomings is a third paradigm for 3D reconstruction to work directly on meshes. Our contribution is the introduction of *ScaleableFusion*: A triangulated mesh, as shown in Figure 4.1, represents surfaces more accurately by breaking up the direct coupling of geometry and color resolution. This requires extra effort to maintain in comparison to surfels or TSDFs, but it comes with several benefits. (1) The vertices making up the geometry do not need to be evenly spaced, like the voxels in a TSDF, and the triangles spanning the vertices still sufficiently describe tight surfaces unlike unconnected surfels. (2) The ability to handle textures allows the progression from averaging colors of unconnected surface elements to selecting fitting keyframes for entire surface regions. These keyframes are selected with awareness of exposure time [60] and corrected for vignetting [61] to further improve photometric consistency. Real-time performance is achieved through our Level of Detail (LOD) system (example in Figure 4.2) that is novel to RGB-D reconstruction. This scheme offloads the geometry from the GPU to the CPU memory to enable large scale reconstructions.

Our approach, ScalableFusion, is described in more detail in the next section (Section 4.1). Section 4.2 discusses some of the intricacies of implementing the LOD system on current hardware, as well as how some of the mentioned heuristics map onto OpenGL and CUDA programming APIs. Section 4.3 gives a detailed comparison between KinectFusion, ElasticFusion and our approach to highlight where and why ScalableFusion generates higher quality reconstructions. We additionally collect data with a higher resolution RGB sensor to demonstrate the improvements over the classic VGA-resolution RGB-D sensor as well as the benefits of a higher resolution RGB sensor in the high resolution pipeline. Our experiments also establish the limits up to which current methods can reconstruct scenes before running out of GPU memory while demonstrating that our system efficiently offloads unused data to the system memory.

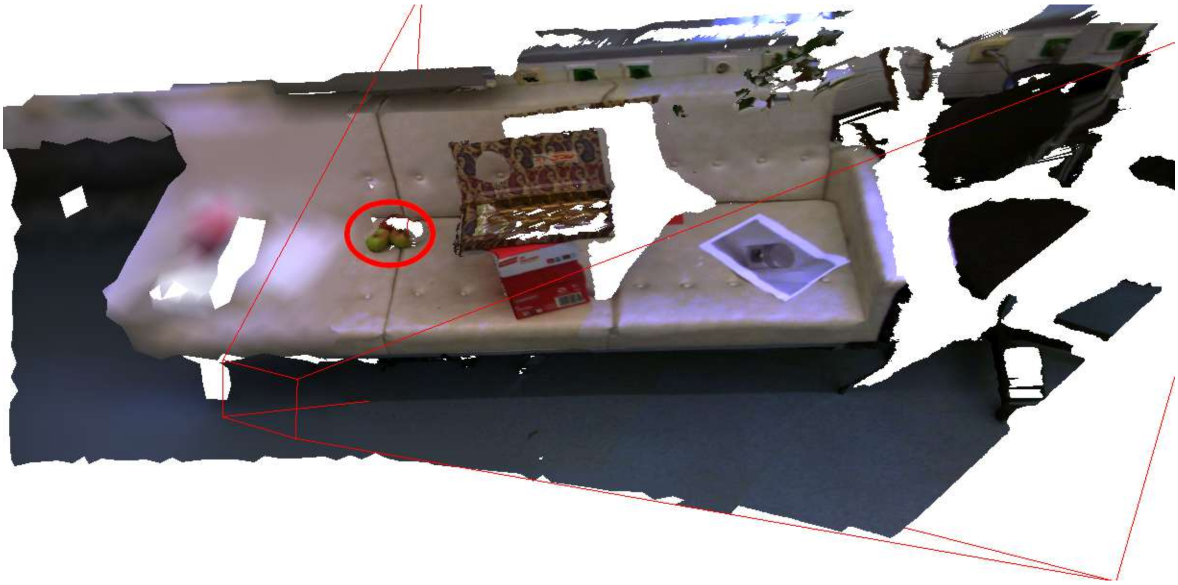


Figure 4.2: Demonstration of the Level of Detail system. The younger part of the reconstruction (right) resides in the memory of the GPU and is processed and displayed at full detail, while the older part (left) is downloaded to system memory and displayed with a low detail replacement. The apples shown in Figure 4.1 are taken from this reconstruction (red circle) and are meant to demonstrate the dynamic range of detail.

4.1 Method

Our approach to 3D reconstruction, initially presented in [62], directly utilizes triangle meshes which, as is their nature, maintain point neighborhoods and have several textures attached. Incoming points are first partitioned into smaller patches. These are then meshed and finally stitched together at their borders. The patches are then enhanced with textures that store geometrical standard deviation and color. To update the reconstruction on the fly, keyframes are selected to donate textures to surface segments. The possible misalignment between segments prior to stitching (as a result of no costly global optimization step) is minimized by processing the captured images with exposure time awareness and vignetting correction.

In the following, we describe the key steps of our framework. We outline the geometry refinement updates, map expansion, meshing and segment stitching.

4.1.1 Geometry Refinement Update

The noise characteristics of depth sensors make it impossible to attain precise geometry at large distances when accumulating multiple measurements. After a certain amount of measurements at a large distance, high quality results can only be achieved by positioning the sensor closer to the surface. This is made possible using current methods by employing a scheme that allows close range measurements to overrule long range measurements, no matter how many samples are taken at long range.

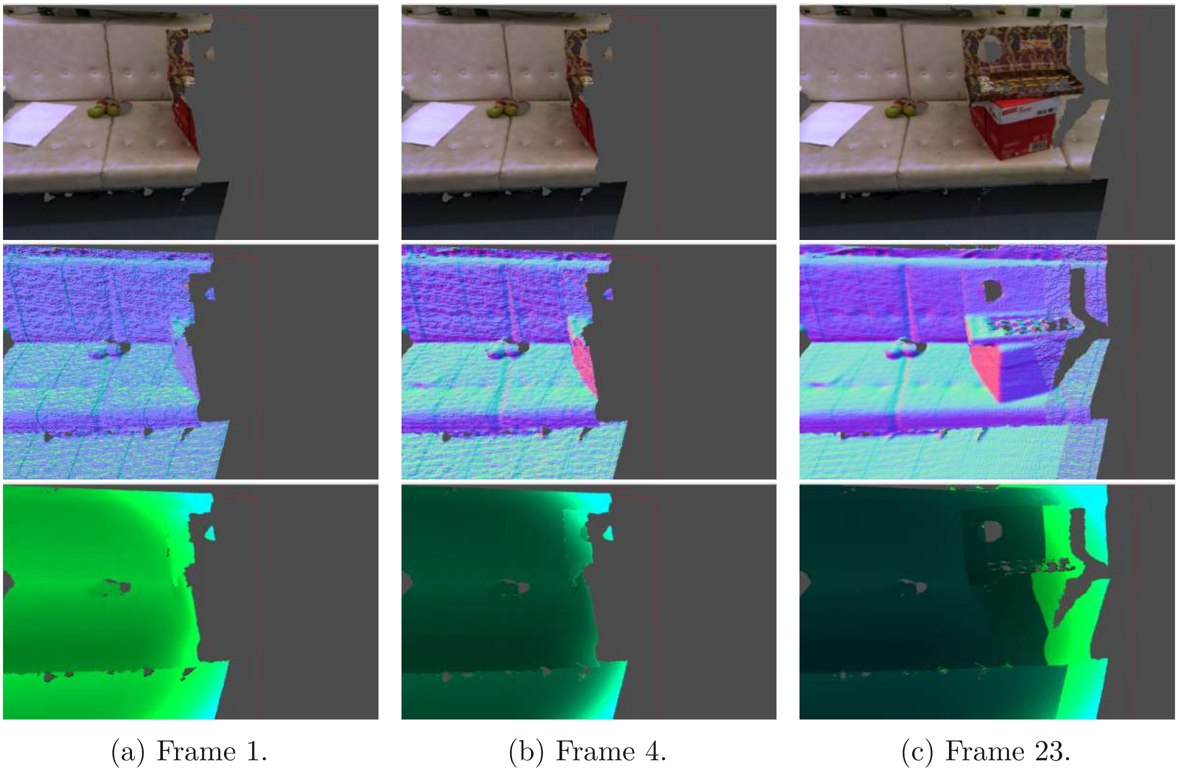


Figure 4.3: A typical geometry update procedure. Displayed are the geometry with RGB (top), rendered surface normals (middle), and an uncertainty measure (bottom). The uncertainty is encoded in the green channel of the geometry texture. After the first frame (a), the geometry shows strong noise and heavy quantization stemming from the Asus Xtion PRO LIVE depth sensor, while the uncertainty is correctly marked as being large (bright green). With additional measurements (b), the geometry is smoothed out reducing the quantization noise and the estimate for uncertainty (indicated by darker green colors). Finally (c), after more frames, the surface is considerably smoothed out and forms a stark constrast to the newly added geometry.

ElasticFusion [39] implicitly handles this problem by introducing a “weight” property for surfels. The weight of a surfel is increased with each observation while position, color and orientation of the surfel are updated. An increasing weight for a surfel implies that it becomes progressively and eventually static while the influence of additional measurements vanishes. Static surfels are not moved but replaced with a denser set of lighter (in terms of weight) surfels when the sensor makes observations from a closer range.

Our approach is seen in Figure 4.3 and takes inspiration from these weights, but instead of spawning new geometry when the sensor approaches a surface, we only do this when it benefits the reconstruction. For example, when approaching a planar surface, ElasticFusion would spawn new surfels just to accommodate the additional color information, whereas our approach directly updates the texture and existing mesh.

This behavior is achieved by storing additional textures containing values for every sampled surface point p. The values contained for each texture pixel (texel) are:

μ_k The average deviation of the k measurements from the actual surface. This is used to indicate where the meshed surface deviates from the sensor's perception.

σ_k An estimate of the noise level that decreases with every additional measurement. The smaller the value, the less influence new measurements have on the geometry. Additionally, $\sigma_{s,k}$ defines the estimated noise level of the sensor measurement k projected onto the surface point p.

$\sigma_{m,k}$ The estimated minimal noise level achievable with all measurements until step k. This assumes quantization is the only limiting factor. Similar to beforehand, $\sigma_{m,s,k}$ refers to the projected value of the sensor. Opposed to $\sigma_{s,k}$, the noise level of a single measurement, $\sigma_{m,s,k}$ indicates the minimum expected error that can be achieved with infinite captures from the sensor's perspective.

The estimated noise level and minimal noise level are updated according to

$$\sigma_{m,k+1} = \min(\sigma_{m,k}, \sigma_{m,s,k}), \tag{4.1}$$

$$\sigma_{k+1} = \sigma'_{k+1} + \sigma_{m,k+1}, \tag{4.2}$$

$$\tag{4.3}$$

where

$$\sigma'_{k+1} = \frac{\sigma'_k \sigma'_{s,k}}{\sigma'_k + \sigma'_{s,k}}, \tag{4.4}$$

$$\sigma'_{s,k} = \sigma_{s,k} - \sigma_{m,k+1}, \tag{4.5}$$

$$\sigma'_k = \sigma_k - \sigma_{m,k+1}. \tag{4.6}$$

Note that σ'_{k+1} is smaller than σ'_k and $\sigma'_{s,k}$, which implies the assumption that every further measurement improves the result. This system guarantees that σ_k approaches $\sigma_{m,k}$ with increasing iteration count k, yet never falls below it. The resulting values $\sigma'_{s,k}$ and σ'_{k+1} are used to update μ by

$$\mu_{k+1} = \left(\frac{\mu_k}{\sigma'_k} + \frac{d_{s,k} - d_k}{\sigma'_{s,k}} \right) \sigma'_{k+1}, \tag{4.7}$$

where $d_{s,k}$ is the distance of the surface point perceived by the sensor and d_k is the distance between the reconstructed point/texel and the sensor. Transcriptions of the texture updates to the vertices are performed by shifting the vertex positions along the view rays such that μ_{k+1} is equal to 0 wherever possible.

These updates do not necessarily need to occur every time new sensor values are available for surface pixels. When the estimated noise level of the sensor data σ_k is higher than $\sigma_{s,k}$ on the surface, no update needs to be conducted. The same applies when the perceived depth values significantly differ from the mapped values. This would indicate either unmapped geometry of an already reconstructed surface or an invalid surface.

The sensor characteristics $\sigma_{s,k}$ and $\sigma_{m,s,k}$ are derived from the work of Halmetschlager-Funek et al. [57] that measures various error metrics over distance. We approximate the error over distance in [57] with a polynomial and extended it with a term that - similar to [39] - attributes a higher standard deviation to measurements in peripheral regions of the depth frame.

4.1.2 Expand Update

As soon as the sensor generates a new frame from a new position, the formerly mapped surface elements are used to render a depth map for the current sensor position. The artificial depth map is compared to the depth values currently perceived by the sensor. Depth values that are in close proximity to the mapped values result in updates to the already existing surfaces. If the captured surface leaves this proximity towards the camera, it is added (meshed) to the current reconstruction. The thresholds for these operations as well as $\sigma_{s,k}$ depend on depth, pixel position and the sensor itself.

4.1.3 Meshing

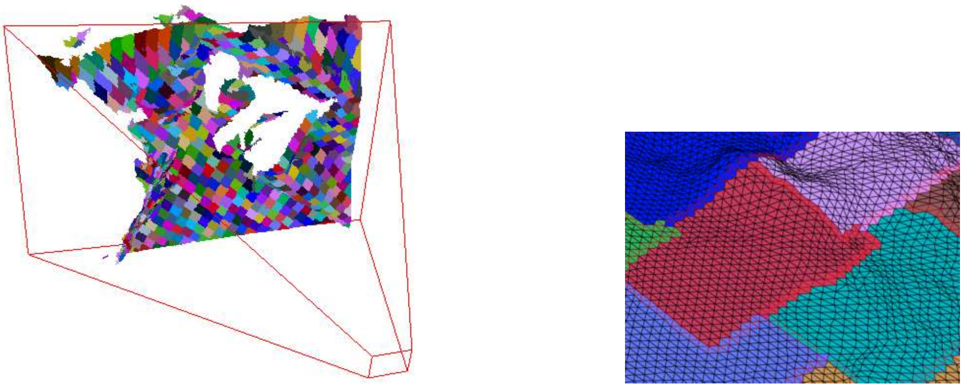
After identifying the novel parts of the captured depth map, 3D points are created by applying the pinhole model to project the depth pixels. These points are segmented into smaller blocks, based on their distance to each other and the estimated normal vector. The neighborhood information derived from the organized point cloud is directly used for this operation and also for spanning triangles between each neighboring set of 3 points. During this process, it is ensured that no triangles are created where neighboring depth values are within the thresholds mentioned in Section 4.1.2. An example of the output from this segmentation and meshing process is shown in Figure 4.4.

4.1.4 Stitching

Generating a mesh on a single organized depth map is computationally undemanding due to the presence of neighborhood information in the 2D image plane. The situation changes, however, as soon as new sensor data is to be integrated into an existing reconstruction.

To address this problem, all visible triangles captured prior to the current frame are searched for open edges. This refers to every edge where a triangle does not border another. The open edges are projected to the pixel space of the current frame's depth map. After this process, finding a potential neighbor for a reconstructed triangle within the set of novel triangles is a simple lookup in the current image plane. The result of expanding the reconstruction is shown in Figure 4.5.

Despite the simple premise leading to the efficient stitching algorithms, the fact that the existing geometry would generally not fit the pixel-grid of the new geometry remains and justifies a deeper discussion. Any movement of the camera will shift the center of reconstructed geometry and its vertices away from the pixel centers of the image plane such that, for example, a reconstructed surface will appear less dense in consecutive frames when the sensor comes closer. In case the sensor increases the distance to the

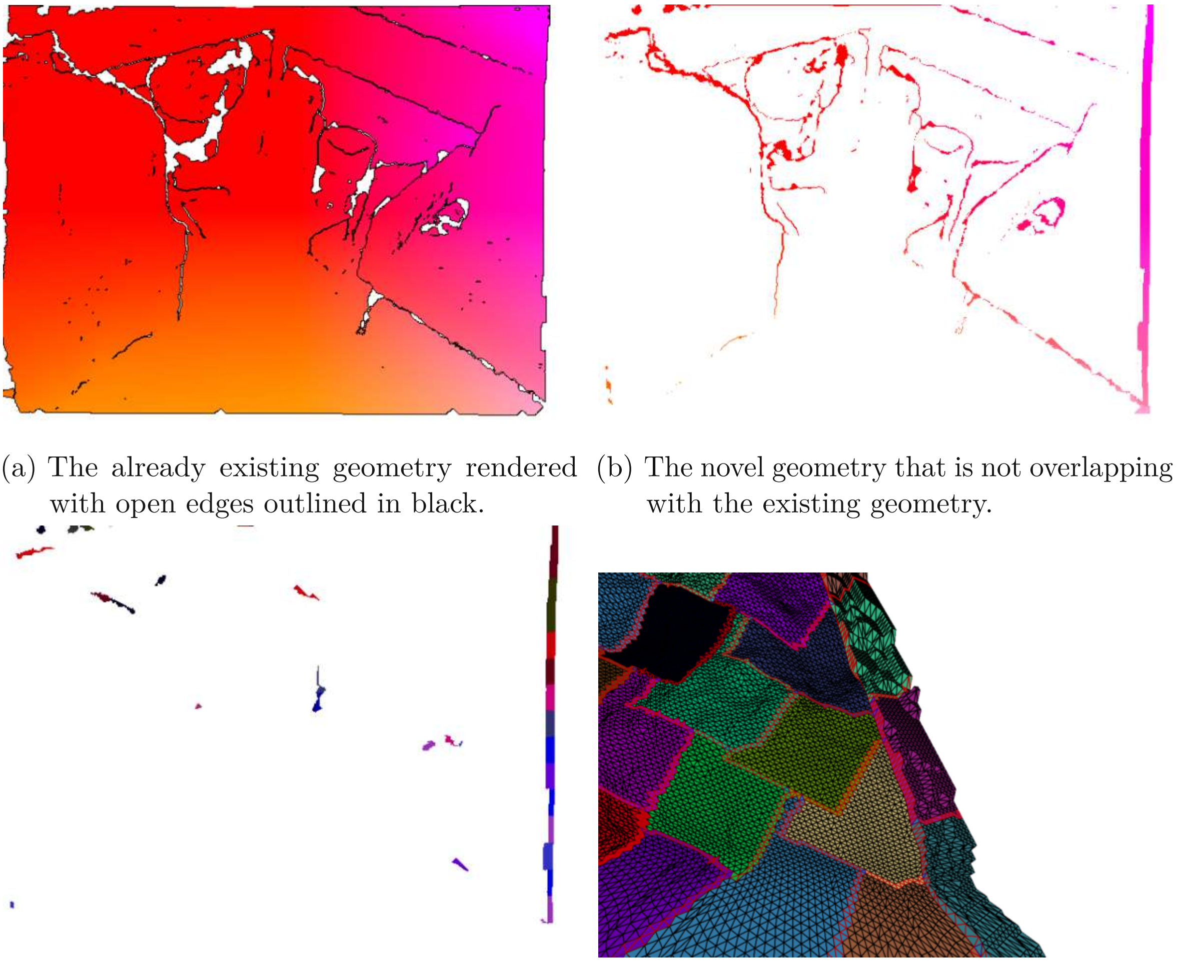


(a) Geometry of a single capture (b) Close-up

Figure 4.4: Start of the meshing process: The depth data is segmented into smaller patches (a). These meshlets are then meshed into grids that are regular on the image plane of the sensor. Shown in (b) is a closeup of the mesh after its vertices have been refined and thus smoothed by additional depth measurements.

reconstructed surface, multiple vertices will end up occupying the same pixel. Due to more complex sensor movements occurring in real-world applications, hybrids of both cases are expected.

Figure 4.6 shows a projection of pre-existing geometry onto the image plane of the current frame to be of lower vertex density than the newly created mesh. Thus, the strategy is to rasterize along the edge of this pre-existing geometry with the Bresenham algorithm and search within the eight neighboring pixels for newly introduced vertices. In case a neighbor was found, and a stitching triangle would meet a set of geometrical criteria, a stitching process is performed, progressing along the edges of existing and new geometry simultaneously. For instances where the existing geometry would project onto the image plane and cause multiple edges to fall into the same pixel, the process looks slightly different. In these cases, we can traverse the edge vertex by vertex as seen in Figure 4.7, while taking proper precautions to not generate self-intersections.



(a) The already existing geometry rendered with open edges outlined in black.

(b) The novel geometry that is not overlapping with the existing geometry.

(c) The coarse segmentation of the novel geometry with smaller regions not being mapped.

(d) The novel and still rough geometry (right) is stitched to the existing geometry.

Figure 4.5: Connecting novel data to existing geometry: Open edges (a) outlined in black are connected to the coarse segmentation (c). The result (d) shows a tight seam of triangles between the denoised old mesh and the noisy new geometry.

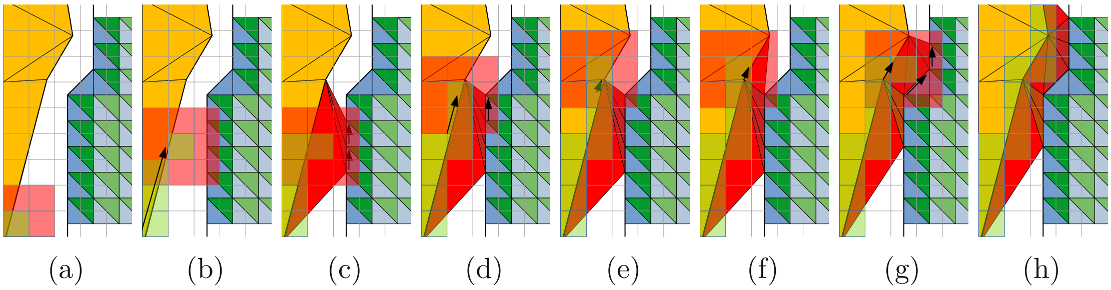


Figure 4.6: The stitching process attaching newly introduced geometry (right of each image) with vertices in the center of pixels (grey grid) and pre-existing geometry (left). Until detecting the first new vertex the edges of pre-existing geometry is followed pixel by pixel without creating new triangles (a-b). Upon detection of the first vertex in the 8-neighborhood (b), triangles are introduced beginning with the closest to the starting vertex (c). Progressing with the rasterization, more triangles are added (d) until the end of the edge is reached (e) continuing with the next edge (on the same pixel) allows for more triangles to be added (f). Following these principles, we end up with a completed stitch (g-h).

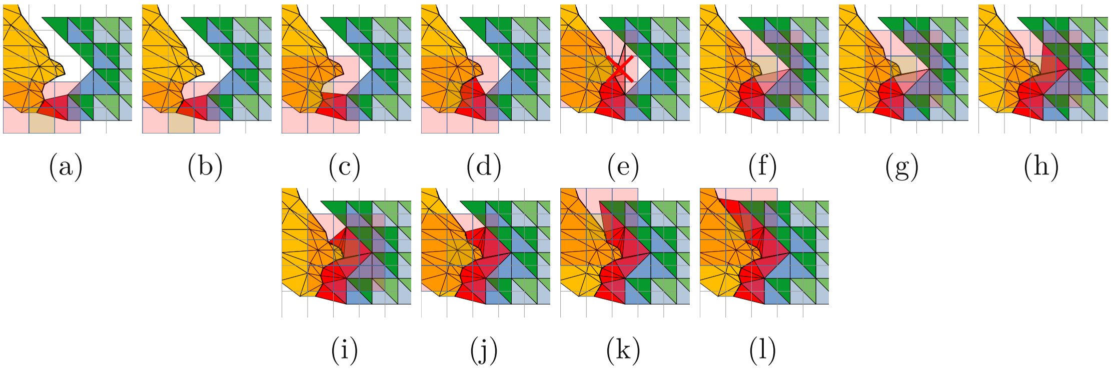


Figure 4.7: The stitching process attaching newly introduced geometry (right of each image) with vertices in the center of pixels (grey grid) and pre-existing geometry (left). In case that the pre-existing mesh is of higher resolution than the newly introduced depth, the algorithm will encounter many more cases where creating a new triangle would lead to self-intersection (e).

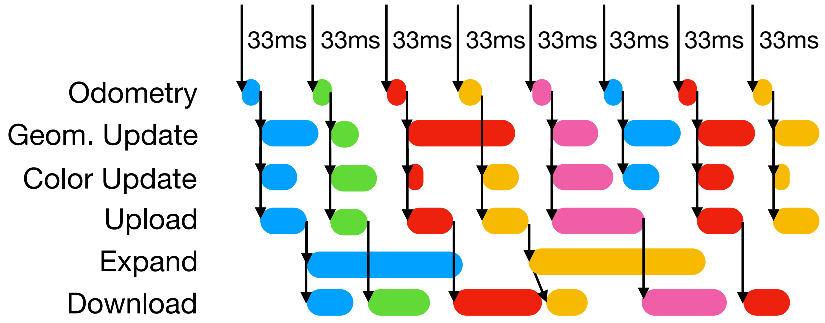


Figure 4.8: The employed threading system. Black arrows depict the flow of data. Colored stripes present actual processing.

4.2 Implementation

Modern desktop hardware distinguishes between memory bound to the GPU and system memory that is bound to the CPU. Access cannot occur across memory spaces without performing expensive data transfers over the PCI-E bus. Therefore, our data structures are designed to mirror the data between CPU and GPU, and are only synchronized when necessary.

Modifications of the geometry occur on either the GPU or the CPU, depending on which processor is better suited for the operation. This has implications on the data structure. While data stored on the CPU is excessively cross-referencing, the structures on the GPU only store very few references between elements.

We introduce a threading system to simultaneously process tasks that are not fully interdependent. For example, while the geometry of one frame is updating the mesh, data not needed can be simultaneously transferred from the GPU to the system memory (download). The odometry, likewise, is not explicitly reliant on the most current version of geometry. The pose estimation of a new frame can therefore be performed while the last frame is still being integrated. An exemplary timeline of this system is shown in Figure 4.8.

4.2.1 Lookup Texture

For every texture attached to a meshlet storing information (color, standard deviation), we also store a texture of equal size pointing to the surface positions of each texel. This surface position is encoded as a triangle-ID as well as barycentric coordinates defining the position of the texel on the triangle.

When we e.g. want to update one texel of a standard deviation texture as shown in Figure 4.9, we look up the triangle by its ID and mix its vertices weighted by the barycentric coordinates to reach a surface position. By projecting this position onto the imaging plane of the sensor we can drive the update strategy defined in Section 4.1.1. To detect occlusions by already reconstructed surfaces a depth check has to be performed against the Z-Buffer of the rendered reconstruction.

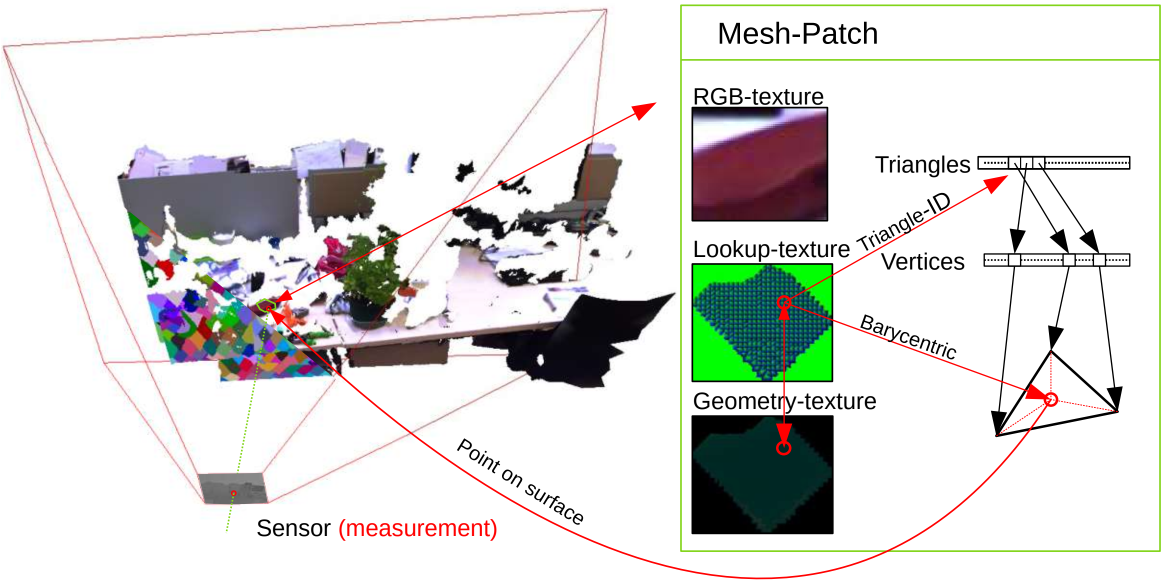


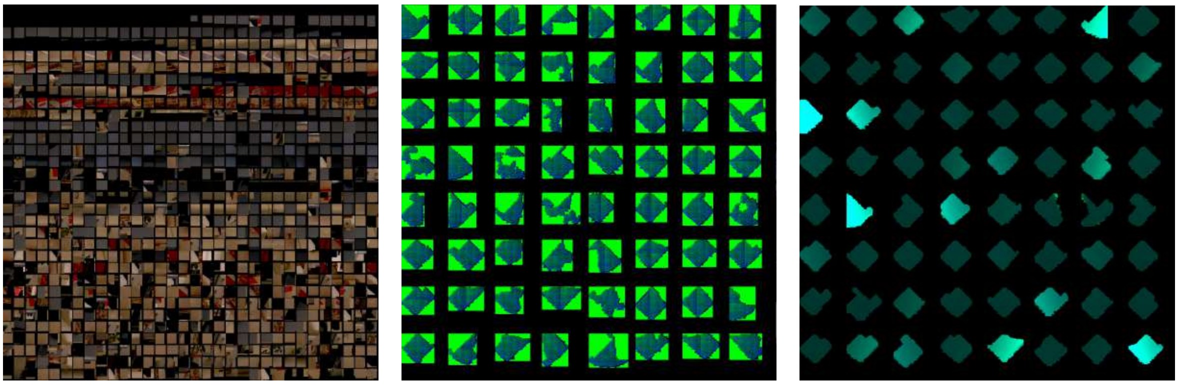
Figure 4.9: When updating the geometry of a mesh patch (green rectangle), each pixel of the geometry texture containing mean and a confidence measure needs to be updated before passing the update to the vertices. To query new measurements for the texels of this texture, the lookup-texture contains references to the triangle ID and the position of each triangle. This enables the calculation of the associated point on the reconstructed surface and, in turn, read out the measurements on the depth sensor after projecting into the sensor frame and testing for occlusion.

To generate this lookup texture we rasterize the UV coordinates as triangles writing out triangle index and barycentric coordinates whenever a texture (color, standard deviation) is initialized or changed in resolution. Padding is performed on pixels bordering on invalid pixels by repeating indices and coordinates and prevents errors when textures are sampled in rendering steps.

4.2.2 Texture Atlas and Driver Overhead

We employ texture atlases as seen in Figure 4.10 to manage the textures associated with the thousands of meshlets. In these atlases, each texture is split up in order to support the surface of multiple meshlets. Primarily, this is a measure to minimize the driver overhead by OpenGL and CUDA, as creating these textures takes $\sim 1ms$ each. Creating smaller individual textures instead of sharing and splitting bigger ones would make real-time mapping with potentially hundreds to thousands of new meshlets during a frame impossible.

We use the NVIDIA bindless texture extension during rendering to avoid the inherent efficiencies of OpenGL in situations with many textures. In fact, this extension allows us to attach references directly to our geometry data structures and draw the entire geometry with a single draw call.



(a) Color textures (b) Geometry lookup textures (c) Uncertainty textures

Figure 4.10: Texture atlas for different types of texture. As the creation of textures causes significant driver overhead, they need to be batched together in bigger textures. The uncertainty textures (c) store uncertainty in the green color channel and the estimate for mean deviation in the blue channel.

Note that modern graphics APIs such as DirectX 12 or Vulkan would offer multiple advantages over the combination of OpenGL and CUDA and potentially reduce complexity. These new APIs are e.g. capable of maintaining and referencing pools of resources with low overhead. This would remove the need for a vendor-specific extension and might even make the use of texture atlases redundant.

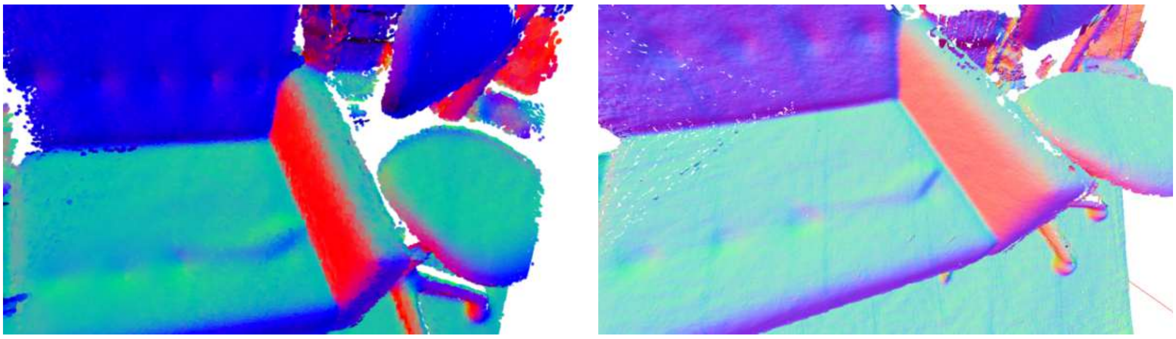
4.3 Evaluation

A comparison of 3D reconstruction pipelines is traditionally done by either comparing the reconstructed point cloud with a ground-truth mesh or by comparing the odometry with the ground-truth trajectory. A ground-truth mesh is inherent to rendered data sets [63], while a trajectory can either be input for the rendering process or captured with external cameras and markers [64].

These benchmarks mainly reflect the quality of odometry instead of the perceived quality. To showcase photometric quality instead of tracking quality, we capture selected scenes with a combination of an Xtion Live Pro and a rigidly attached IDS global shutter camera that delivers a resolution of 1600 × 1200 pixel. While recording depth and color from the Xtion sensor at 30Hz, we also capture high resolution data from the IDS sensor at 60Hz. We evaluate the quality of our texturing algorithm against the output of ElasticFusion in a similar manner as the work on high fidelity texturing presented by Fu et al. [40].

4.3.1 Geometry Comparison

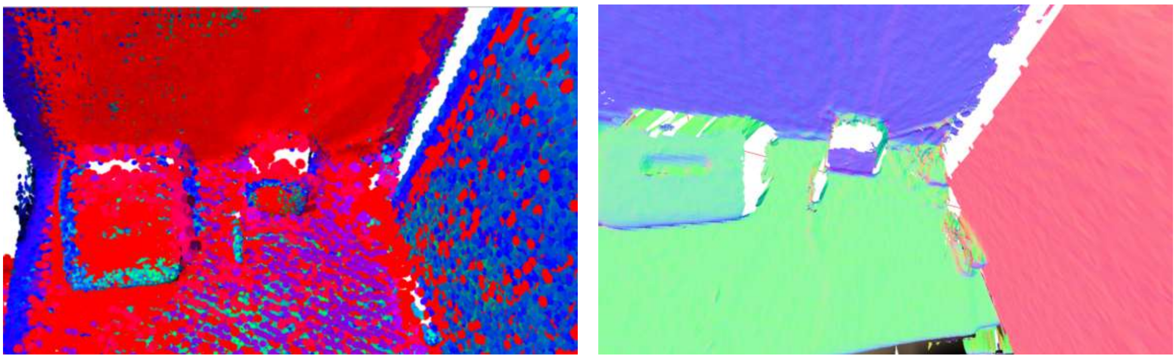
The weighting scheme used by ElasticFusion and our standard deviation-based approach are similar. The results do not differ significantly as underlined by the images in Figure 4.11. It is, however, apparent that the surfels of ElasticFusion are of lower



(a) ElasticFusion.

(b) ScaleableFusion.

Figure 4.11: Normals acquired from ElasticFusion and ScaleableFusion. While both methods deliver similar results, it is notable that our method retains sharper features.



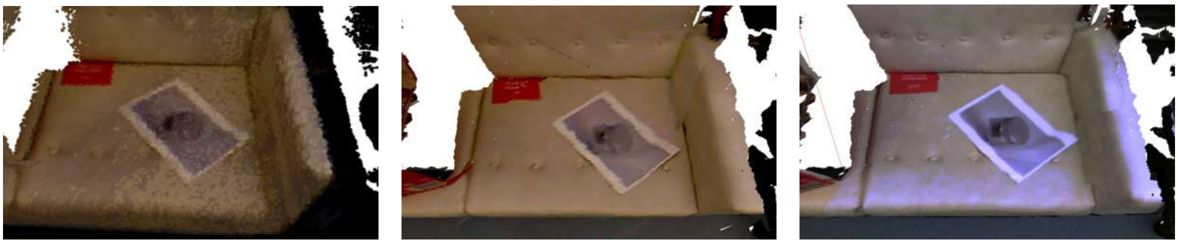
(a) ElasticFusion surfels colorized by number of observations (blue, few vs. red, many).

(b) Our reconstruction colorized by normals.

Figure 4.12: Reconstruction conducted with ElasticFusion and ScalableFusion. (a) Shows how younger surfels (blue) are partially layered on top of older (red) surfaces. Our reconstruction (b) on the other hand maintains smooth non-redundant surfaces.

density than the triangles in our reconstruction. Furthermore, these surfels are intended to overlap, resulting in a gapless surface. A strategy that also leads to redundant and duplicated versions of the surfaces as seen in Figure 4.12.

Methods that directly operate on surfaces are sensitive to tracking errors in a different way. If it is not possible to attribute a captured surface to a reconstructed surface, a duplicate will be mapped. Volumetric approaches, such as KinectFusion, are suitable to handle these situations because they update a reconstruction volume allowing them to derive a correct surface model. We mitigate this problem through our noise-dependent thresholding scheme to achieve more consistent maps than traditional surface-based methods. Despite this effort, these artifacts can sometimes still arise when depth and tracking errors exceed the given thresholds.



(a) ElasticFusion with Xtion-based RGB data.

(b) ScalableFusion with Xtion-based RGB data.

(c) ScalableFusion with high resolution RGB camera.

Figure 4.13: Most methods average the color observations, leading to washed out and speckled results (a). Our texturing-based approach ((b), (c)) samples the color patchwise, therefore, retaining sharper detail. This enables the use of higher resolution cameras (c).

4.3.2 Texture Comparison

Figure 4.13 demonstrates that our method makes better use of the RGB sensors than ElasticFusion which, like other methods, only features one color per geometrical primitive. Given that there is only space for 9 million surfels as seen in Figure 4.14, the entire reconstruction can only store as much color detail as a 9 megapixel image. Furthermore, the averaging schemes for color, together with tracking inaccuracies and no consideration of photometric sensor characteristics leads to washed out and speckled reconstructions. Our texturing approach overcomes many of these shortcomings: The texel density is no longer coupled to the vertex density anymore, but instead taken from optimal camera frames to advance the reconstruction batch-wise. This enables the usage of high resolution cameras as seen in Figure 4.13c. We also handle exposure times and vignetting according to [60] and [61] to reduce seams along segment borders.

The images of our approach in Figure 4.13 still display texture misalignments along segment borders, which is due to tracking inaccuracies. Offline optimization schemes such as presented in [40] yield more consistent results, but suffer from heavy computation and are unusable for real-time reconstruction.

4.3.3 Memory Consumption and Computational Performance

The implementations of TSDF and surfel-based approaches do not offload irrelevant geometry from the GPU to CPU memory. This becomes a limiting factor when reconstructing large scenes. We therefore benchmark our system against ElasticFusion in two large scale scenes.

The first experiment is a live and continuous reconstruction of a large office space consisting of multiple rooms (Figure 4.15). Data is offloaded to the CPU memory and only the currently observed region is maintained on the GPU for immediate processing. This is indicated by the different LOD of the bottom right of the reconstruction in Figure 4.15b. Our scheme vastly reduces the memory consumption for large scale reconstructions because it is regulated by the current demand. As such, the vertex

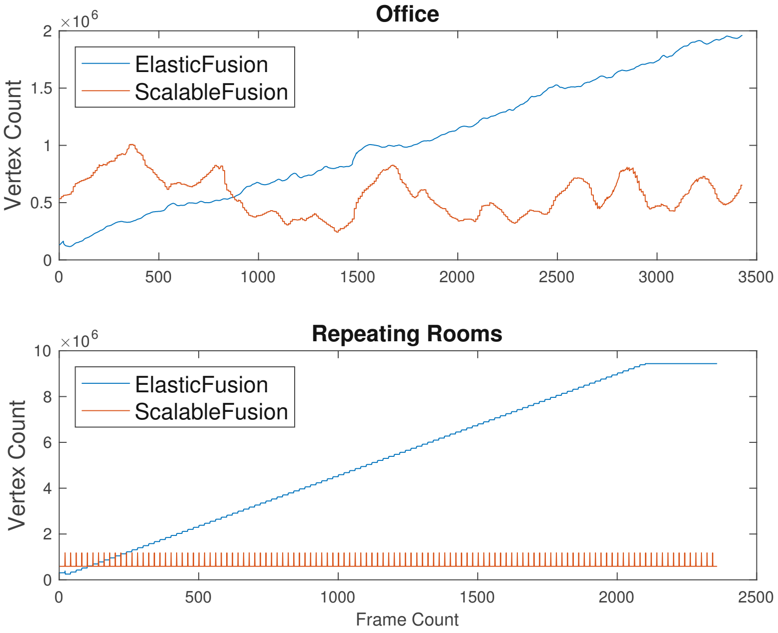


Figure 4.14: Count of vertices residing on the GPU. Top: Reconstruction of the office in Figure 4.15. Bottom: Dataset repeating the same room with pose offsets. ScalableFusion stays below 1 million active vertices, while ElasticFusion accumulates data in correlation to the size of the environment.

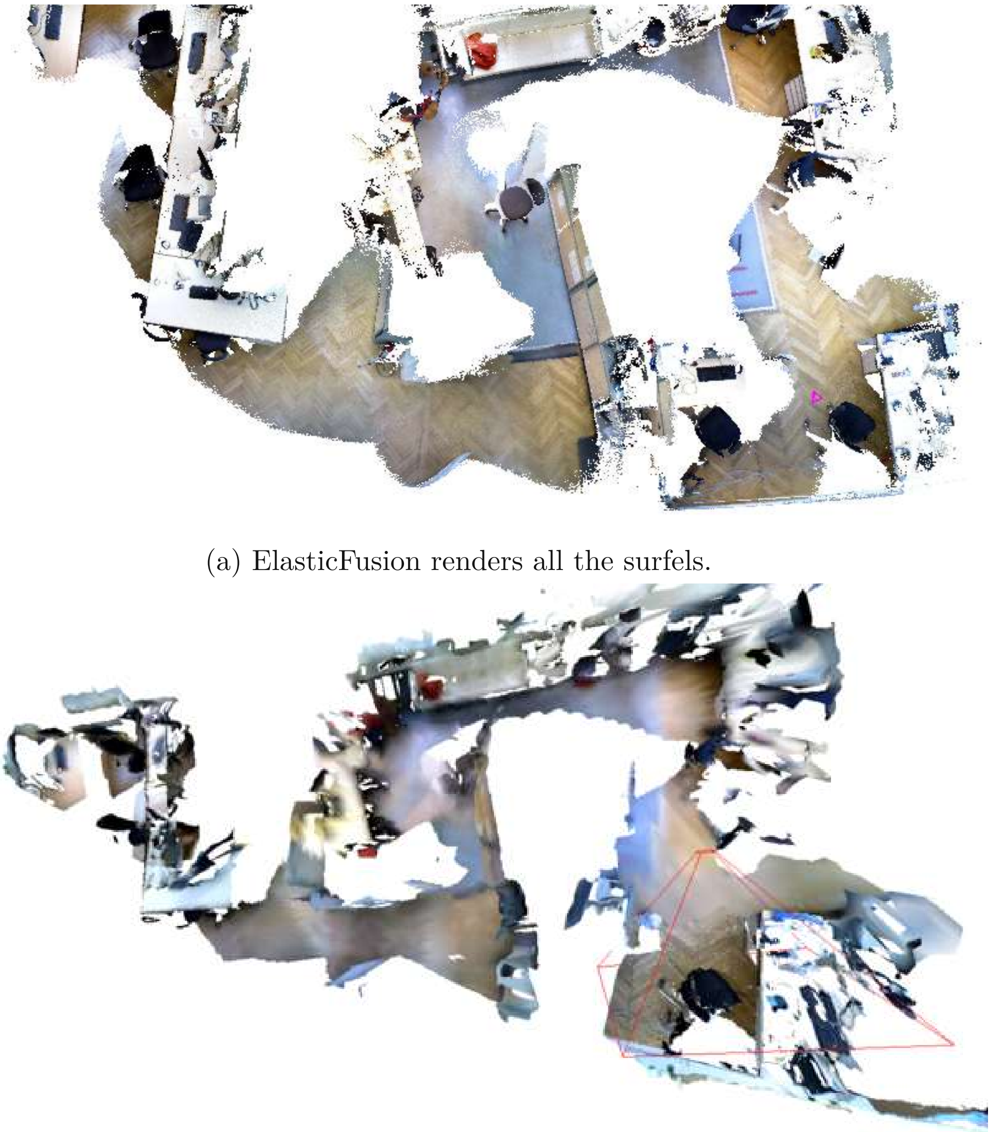
count residing on the GPU is constant on average, only fluctuating depending on the complexity of the reconstructed frames (Figure 4.14). In comparison to ElasticFusion, the surfel count growth is correlated with the overall size of the captured scene.

The second experiment is an extreme stress test for both systems. We repeatedly reconstruct a room dataset [63] while offsetting the pose by 5 m to artificially create a large environment. In this test, ElasticFusion halts at 2200 frames after accumulating 9 million surfels because the capacity of the GPU memory is reached. Our system, on the other hand, seamlessly continues to operate with a constant demand on the same memory.

On our system (Intel Core i7-7700K, Geforce GTX 1070), we achieve a constant framerate of 30 Hz for our texturing and geometry refinement tasks while still allowing the geometry to be expanded, meshed and stitched at 5 Hz. This is independent of the overall reconstruction size.

4.4 Discussion

In this chapter, we introduced ScalableFusion, a novel reconstruction approach for directly texturing triangles and vertices that significantly improves the quality of 3D reconstruction. While the use of a triangulated mesh imposes an overhead in terms of data management, the benefits vastly outweigh the downsides of increased computational effort. This particularly applies to real-world environments that feature very small objects. A mesh-based approach describes these elements with an outstanding level of detail. ScaleableFusion performs the entire meshing process in the pixel space of the camera frame to eliminate the expensive neighborhood search. This method proves to be superior to colorized geometrical primitives because the camera resolution is utilized to its full potential. We also provide a sophisticated memory management scheme to enable real-time operation using a Level of Detail approach. A detailed comparison shows that ScaleableFusion creates better reconstruction for robotic applications. It creates sharper features (Figure 4.13), alleviates the issue of generating double planes (Figure 4.12), maintains the full resolution of color cameras (Figure 4.2) and overcomes memory limits of present methods (Figure 4.14). While results show significant improvements over state-of-the-art methods, the high resolution of the textures exposes shortcomings in the alignment between color and geometry. The overly simplistic color and surface model, inaccurate odometry, poor timestamping/synchronization, and rolling shutter effects of the depth sensor lead to visible transitions at meshlet boundaries and misalignment with the geometry. Such effects are, albeit not hindering for most robot applications, visually displeasing.



(a) ElasticFusion renders all the surfels.

(b) Whereas our system renders the same scene more efficiently.

Figure 4.15: Partial reconstruction of a large office space.

Chapter 5

Conclusion

Despite building depth-sensors capable of delivering super-human accuracy, we still struggle to deliver human-level performance on vision-dependant tasks. Unfortunately, we cannot easily imitate human-like strategies/policies/control laws that commonly involve other sensing modalities but also exhibit variability undesired in certain applications. An alternative to the extremely challenging and uncertain task of imitating these organic mechanisms/strategies is to improve the building blocks of our current systems.

The main contribution of this thesis is the improvement of raw depth-sensing based on structured light and the integration of these depth-measurements into a coherent 3D reconstruction. These two building blocks are typical for computer vision pipelines and can greatly influence the utility of AR and robotic systems.

5.1 Summary

In Chapter 3 we briefly discuss the prevalent depth-sensing modalities before enlarging upon the spatial neighborhood encoding-based structured light sensors and finally introducing GigaDepth, our machine learning-based solution to improve the performance on these devices. It is established early on that the sensors operating on this principle are held back by the used algorithms in order to achieve high framerates in tight compute envelopes. More compute intensive approaches like e.g. machine learning-based algorithms thus seem to receive limited attention during development of these products, leaving potential untapped. The discussed publications in 2.1 tap into the possibilities of increased compute resources and modern machine learning approaches that serve as inspiration to GigaDepth. We describe how the combination of multiple of these ideas can lead to an efficient algorithm with outstanding performance. Utilizing a shallow CNN in an initial step allows us to extract a condensed feature vector describing the spatial neighborhood around each captured pixel. Parts of the pattern that retained enough intensity after traveling to the scene surface, being reflected and traveling back to the sensor are described in this condensed representation. This also applies to additional perceptions like sharp edges or shading cues that help guide the surface reconstruction in small gaps left by the pattern. Starting from these features, trees of MLPs will derive highly accurate pattern positions that consequently lead to highly accurate depth estimates. Finally we demonstrate the performance of our algorithms on two newly assembled datasets. One synthetic dataset required for training due to our methods dependence on perfectly accurate ground-truth. The second dataset captured with actual hardware and with ground-truth provided by an industrial laser-scanner. In our experiments we demonstrate superior performance compared to all baseline methods in both domains despite our approach lacking training mechanism for real-world data.

Chapter 4 describes ScalableFusion, a fusion mechanism for depth data that directly generates and operates on a triangulated and textured surface. By being able to decouple the resolution and density of color and geometry as well as scaling the density of the representation according to the provided data, it is able to provide a significant quality upgrade compared to contemporary solutions. While the use of a triangulated mesh imposes an overhead in terms of data management, the benefits vastly outweigh the downsides of increased computational effort. A series of algorithms and heuristics help keep the computation cost low and facilitate real-time operation. ScalableFusion performs the entire meshing process in the pixel space of the camera frame to eliminate the expensive neighborhood search. This method proves to be superior to colorized geometrical primitives because the camera resolution is utilized to its full potential. We also provide a sophisticated memory management scheme to enable real-time operation using a Level of Detail approach. A qualitative comparison shows that ScalableFusion creates better reconstruction for robotic applications. It creates sharper features (Figure 4.13), overcomes the issue of generating double planes (Figure 4.12), maintains the full resolution of color cameras (Figure 4.2) and overcomes memory limits of present methods (Figure 4.14).

5.2 Outlook

Although related and potentially applied in field in the same work package, the two main themes in this work are vastly different in terms of the paradigms applied. While our depth-estimation is benefiting from, as of the time of writing, almost fashionable CNNs, it is a collection of heuristics driving our 3D reconstruction. Especially the development of the latter has to be seen in the context of the time it has been created in. Scalable Fusion originated approximately 3 years prior to GigaDepth and, if developed later, might have employed a vastly different approach or replaced some of the hand-crafted heuristics with learning-based algorithms.

5.2.1 GigaDepth: Learning Depth from Structured Light with Branching Neural Networks

While other depth estimation algorithms manage to cope without explicit supervision by utilizing consistency-based loss functions, our method depends on precise ground-truth. This is because the control for traversing the decision tree is discretized and therefore forfeits the ability to back-propagate the gradients for paths not taken. Generating meaningful update steps for the classification stages by means of photometric consistency would allow for easier adoption to the real-world domain. This might also create the opportunity to find more optimal decision boundaries similar to the random forest approach of HyperDepth.

In a recent work on Switch Transformers, Fedus et al. [65] train a model that routes tokens to experts but, unlike classical transformers, does not need to evaluate for not selected experts during training. A load balancing loss ensures equal distribution to the individual experts and enables training without full supervision of the intermediate decisions while only evaluating the experts contributing to the output. Considering this and the popularity of other recent work on Mixture-of-Expert models as e.g. [65], [66] with trillions of parameters and layers being sparsely activated, it can be assumed that the training of branching architectures similar to ours will stay a research topic in the future. The method extracting invalidation maps presented in Section 3.1.4 proves that we can efficiently identify and discard most invalid disparity estimates. However, the resulting binary invalidation mask is rather pragmatic and deserves tuning. Furthermore, a confidence measure such as standard deviation with continuous values and a statistical interpretation would benefit tasks like 3d reconstructions.

Different architectures that started to replace, complement or augment CNNs in other tasks should be employed. E.g. a transformer-based architecture would better map onto current hardware than our MLP decision tree. It could revive the idea of a reference feature map of the emitted pattern: A hierarchical (cross-)attention mechanism could match features between the captured frame and a reference map of keys and values. This way, the knowledge about the reference pattern is not stored in such a hideously rigid decision tree. The current implementation for the MLP tree could receive a dramatic speedup by performing the whole traversal in one fused CUDA kernel. Similar to the works by Müller et al. [51], [67] that is used to cache raytracing results in a neural representation, this would allow to keep the intermediate results in the cache and minimize memory access.

Finally, incorporating deeper networks into our pipeline e.g. for post-processing can help fill gaps left around depth discontinuities and on challenging surfaces to create more complete depth maps.

5.2.2 Scalablefusion: High-resolution mesh-based real-time 3D reconstruction

Despite some compelling results generated by our ScalableFusion method, we have identified several shortcomings. Each of these opening multiple avenues for potential solutions and improvements.

The odometry method used, inherited from ElasticFusion, was selected to minimize the (geometric) dissent coming from different camera frames. Augmenting or even replacing the projective ICP approach with one leaning onto an RGB-based Visual Inertial Odometry system (e.g. Direct Sparse Odometry [68]) would make the integration and alignment of new geometry more challenging, but potentially handle a greater variety of edge cases.

When handling noisy or even contradicting depth-measurements, purely surface-based methods cannot easily maintain volumes of possible surfaces. Deciding which surface a specific measurement should be attributed to or when to instantiate new surfaces are central factors for reconstruction quality. Our algorithm has shown to rarely instantiate new surfaces when clearly observing an already mapped surface or keep updating an existing surface when the scene has changed. Better heuristics, acting also on color information instead of depth alone, should be employed. With enough reconstructions performed and some manual annotation, a learning-based approach could be pursued.

Our current pipeline does not feature bundle adjustment and thus does not handle loop closures. Augmenting the pipeline with loop detection found in full SLAM systems (e.g. ORB SLAM [37]) and maintaining a pose-graph, an optimization step could generate updates for the meshlets and improve consistency in challenging situations. The updating transformations for the dense geometry could be cached, accumulated and would only need to be applied when meshlets are visible.

The textures of surfaces are updated by applying data from keyframes with unobstructed view. Heuristics better controlling exposure and keyframe selection can greatly improve the appearance of the reconstruction. Learning-based methods, inspired by neural radiance fields as e.g. [49], could also be used to extract a more flexible surface parametrization such as a Bidirectional Reflectance Distribution Functions (BRDF) [69] to decouple lighting from our surface representation.

Due to issues in calibration and synchronization between Depth and RGB-sensor, textures will not perfectly overlap with the geometry. By finding/employing a geometry to texture alignment metric and a metric quantifying the alignment between textures of neighboring meshlets, may they be learning-based or not, it would be possible to adjust the textures and provide visually appealing results.

As ScalableFusion allows for multiple layers of textures to be attached to the surfaces, it is possible to extend the reconstruction with additional information such as semantic, or object labels. Going further than using such information as an overlay over existing geometry, the work of Herb et al. [70] describes a method where semantic labels guide the meshing process and enable persistently labelled mesh-reconstructions. While the geometry generated by [70] is much sparser, the ideas provide a glimpse of what can be achieved when image-level analysis via neural networks flows into the mesh-creation process.

Finally, we want to emphasize that our plans for online optimization of the geometry were more ambitious than shown in this paper. The textures used to keep account of standard deviations on the surfaces are intended to be used to detect surfaces/meshlets that we seek to densify. In a similar fashion, we planned to select planar triangles with low standard deviation to sparsify meshlets and (on occasion) combine those into bigger, less expensive blocks. Implementing these features would allow long-term management of reconstructions enabling applications in robotics and spatial computing.

Bibliography

[1] Z. Yan, S. Schreiberhuber, G. Halmetschlager-Funek, T. Ducket, M. Vincze, and N. Bllotto, “Robot Perception of Static and Dynamic Objects with an Autonomous Floor Scrubber,” in *Intelligent Service Robotics (ISR)*, 2020 (cit. on p. 2).

[2] H. Hirschmüller, “Accurate and efficient stereo processing by semi-global matching and mutual information,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2005, 807–814 vol. 2 (cit. on pp. 11, 14, 32).

[3] G. Kuschk and D. Cremers, “Fast and Accurate Large-scale Stereo Reconstruction using Variational Methods,” in *ICCV Workshop on Big Data in 3D Computer Vision*, 2013 (cit. on p. 11).

[4] V. Tankovich, C. Hane, Y. Zhang, A. Kowdle, S. Fanello, and S. Bouaziz, “HITNet: Hierarchical Iterative Tile Refinement Network for Real-time Stereo Matching,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 14 362–14 372 (cit. on p. 11).

[5] J.-R. Chang and Y.-S. Chen, “Pyramid Stereo Matching Network,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 5410–5418 (cit. on p. 11).

[6] S. Khamis, S. Fanello, C. Rhemann, A. Kowdle, J. Valentin, and S. Izadi, “StereoNet: Guided Hierarchical Refinement for Real-Time Edge-Aware Depth Prediction,” in *European Conference on Computer Vision (ECCV)*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham, 2018, pp. 596–613, ISBN: 978-3-030-01267-0 (cit. on pp. 11, 14, 26).

[7] C. Godard, O. M. Aodha, M. Firman, and G. Brostow, “Digging Into Self-Supervised Monocular Depth Estimation,” in *International Conference on Computer Vision (ICCV)*, 2019, pp. 3827–3837 (cit. on p. 11).

[8] J. Watson, O. M. Aodha, V. Prisacariu, G. Brostow, and M. Firman, “The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth,” in *Computer Vision and Pattern Recognition (CVPR)*, 2021 (cit. on p. 11).

[9] R. Ranftl, A. Bochkovskiy, and V. Koltun, “Vision Transformers for Dense Prediction,” in *International Conference on Computer Vision (ICCV)*, 2021, pp. 12 179–12 188 (cit. on p. 11).

[10] J.-W. Bian, H. Zhan, N. Wang, *et al.*, "Unsupervised Scale-consistent Depth Learning from Video," *International Journal of Computer Vision (IJCV)*, 2021 (cit. on p. 11).

[11] Y. Zhang and T. Funkhouser, "Deep Depth Completion of a Single RGB-D Image," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018 (cit. on p. 11).

[12] W. Van Gansbeke, D. Neven, B. De Brabandere, and L. Van Gool, "Sparse and Noisy LiDAR Completion with RGB Guidance and Uncertainty," in *2019 16th international conference on machine vision applications (MVA)*, IEEE, 2019, pp. 1–6 (cit. on p. 11).

[13] M. Hu, S. Wang, B. Li, S. Ning, L. Fan, and X. Gong, "Towards Precise and Efficient Image Guided Depth Completion," 2021 (cit. on p. 11).

[14] S. M. H. Miangoleh, S. Dille, L. Mai, S. Paris, and Y. Aksoy, "Boosting Monocular Depth Estimation Models to High-Resolution via Content-Adaptive Multi-Resolution Merging," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021 (cit. on pp. 11, 21).

[15] M. D. Altschuler, J. L. Posdamer, G. Frieder, B. R. Altschuler, and J. Taboada, "The Numerical Stereo Camera," in *Three-Dimensional Machine Perception*, B. R. Altschuler, Ed., International Society for Optics and Photonics, vol. 0283, SPIE, 1981, pp. 15 –24 (cit. on pp. 13, 14).

[16] B. Carrihill and R. Hummel, "Experiments with the intensity ratio depth sensor," *Computer Vision, Graphics, and Image Processing*, vol. 32, no. 3, pp. 337–358, 1985, ISSN: 0734-189X (cit. on pp. 13, 14).

[17] P. Mirdehghan, W. Chen, and K. N. Kutulakos, "Optimal Structured Light à la Carte," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018 (cit. on p. 14).

[18] W. Chen, P. Mirdehghan, S. Fidler, and K. N. Kutulakos, "Auto-Tuning Structured Light by Optical Stochastic Gradient Descent," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020 (cit. on p. 14).

[19] M. Gupta and N. Nakhate, "A Geometric Perspective on Structured Light Coding," in *European Conference on Computer Vision (ECCV)*, 2018 (cit. on p. 14).

[20] S. V. der Jeught and J. J. J. Dirckx, "Deep neural networks for single shot structured light profilometry," *Opt. Express*, vol. 27, no. 12, pp. 17 091–17 101, 2019 (cit. on p. 14).

[21] H. Nguyen, Y. Wang, and Z. Wang, "Single-Shot 3D Shape Reconstruction Using Structured Light and Deep Convolutional Neural Networks," *Sensors*, vol. 20, no. 13, 2020, ISSN: 1424-8220 (cit. on p. 14).

[22] M. Martinez and R. Stiefelhagen, "Kinect Unleashed: Getting Control over High Resolution Depth Maps," in *Proceedings of the International Conference on Machine Vision Applications*, 2013, pp. 247–240 (cit. on pp. 14, 18).

[23] S. R. Fanello, C. Rhemann, V. Tankovich, *et al.*, “HyperDepth: Learning Depth From Structured Light Without Matching,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016 (cit. on pp. 14, 18, 19, 21, 22, 26, 27, 30, 32, 35, 41).

[24] G. Riegler, Y. Liao, S. Donne, V. Koltun, and A. Geiger, “Connecting the Dots: Learning Representations for Active Monocular Depth Estimation,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019 (cit. on pp. 14, 18–21, 26, 27, 29, 30, 34, 35).

[25] M. Johari, C. Carta, and F. Fleuret, “DepthInSpace: Exploitation and Fusion of Multiple Video Frames for Structured-Light Depth Estimation,” in *International Conference on Computer Vision (ICCV)*, 2021, pp. 6019–6028 (cit. on pp. 14, 18, 20, 26, 27, 29, 30, 32, 34, 35, 37).

[26] S. Schreiberhuber, J.-B. Weibel, T. Patten, and M. Vincze, “GigaDepth: Learning Depth from Structured Light with Branching Neural Networks,” in *European Conference on Computer Vision (ECCV)*, 2022 (cit. on pp. 14, 19).

[27] Y. Zhang, S. Khamis, C. Rhemann, *et al.*, “ActiveStereoNet: End-to-End Self-Supervised Learning for Active Stereo Systems,” in *European Conference on Computer Vision (ECCV)*, 2018 (cit. on pp. 14, 20, 26, 27, 30–32).

[28] R. A. Newcombe, S. Izadi, O. Hilliges, *et al.*, “KinectFusion: Real-time dense surface mapping and tracking,” in *10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136 (cit. on pp. 15, 45).

[29] W. E. Lorensen and H. E. Cline, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH, Association for Computing Machinery, 1987, 163–169, ISBN: 0897912276 (cit. on p. 15).

[30] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, “Kintinuous: Spatially Extended KinectFusion,” in *Proceedings of Robotics: Science and Systems (Workshop on RGB-D: Advanced Reasoning with Depth Cameras)*, 2012 (cit. on pp. 15, 45).

[31] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, “Real-time 3D Reconstruction at Scale Using Voxel Hashing,” *ACM Transactions on Graphics*, vol. 32, no. 6, 169:1–169:11, 2013 (cit. on p. 15).

[32] R. A. Newcombe, D. Fox, and S. M. Seitz, “DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time,” in *Conference on Computer Vision and Pattern Recognition*, 2015, pp. 343–352 (cit. on pp. 15, 45).

[33] K. Kolev and D. Cremers, “Continuous Ratio Optimization via Convex Relaxation with Applications to Multiview 3D Reconstruction,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009 (cit. on p. 15).

[34] T. Whelan, M. Kaess, J. J. Leonard, and J. McDonald, “Deformation-based loop closure for large scale dense RGB-D SLAM,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 548–555 (cit. on p. 15).

[35] N. Fioraio, J. Taylor, A. Fitzgibbon, L. Di Stefano, and S. Izadi, "Large-scale and drift-free surface reconstruction using online subvolume registration," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4475–4483 (cit. on p. 15).

[36] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration," *ACM Transactions on Graphics*, vol. 36, p. 1, May 2017 (cit. on p. 16).

[37] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017 (cit. on pp. 16, 65).

[38] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion," in *2013 International Conference on 3D Vision - 3DV 2013*, 2013, pp. 1–8 (cit. on p. 16).

[39] T. Whelan, S. Leutenegger, R. S. Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Proceedings of Robotics: Science and Systems*, 2015 (cit. on pp. 16, 45, 48, 50).

[40] Y. Fu, Q. Yan, L. Yang, J. Liao, and C. Xiao, "Texture Mapping for 3D Reconstruction With RGB-D Sensor," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4645–4653 (cit. on pp. 16, 56, 58).

[41] M. Waechter, N. Moehrle, and M. Goesele, "Let There Be Color! Large-Scale Texturing of 3D Reconstructions," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 836–850 (cit. on p. 16).

[42] Q.-Y. Zhou and V. Koltun, "Color Map Optimization for 3D Reconstruction with Consumer Depth Cameras," *ACM Transactions on Graphics*, vol. 33, no. 4, 155:1–155:10, 2014 (cit. on p. 16).

[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014 (cit. on p. 17).

[44] A. Antoniou, A. Storkey, and H. Edwards, "Data Augmentation Generative Adversarial Networks," 2018. arXiv: `1711.04340 [stat.ML]` (cit. on p. 17).

[45] V. Sandfort, K. Yan, P. Pickhardt, and R. Summers, "Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks," *Scientific Reports*, vol. 9, Nov. 2019 (cit. on p. 17).

[46] K. Park, T. Patten, and M. Vincze, "Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation," in *International Conference on Computer Vision (ICCV)*, 2019 (cit. on p. 17).

[47] S. M. A. Eslami, D. J. Rezende, F. Besse, *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018 (cit. on p. 17).

[48] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *European Conference on Computer Vision (ECCV)*, 2020 (cit. on p. 17).

[49] M. Boss, R. Braun, V. Jampani, J. T. Barron, C. Liu, and H. P. Lensch, "NeRD: Neural Reflectance Decomposition from Image Collections," in *International Conference on Computer Vision (ICCV)*, 2021 (cit. on pp. 17, 65).

[50] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking Neural Radiance Fields for Real-Time View Synthesis," *International Conference on Computer Vision (ICCV)*, 2021 (cit. on p. 17).

[51] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, 102:1–102:15, Jul. 2022 (cit. on pp. 17, 64).

[52] D. Rückert, L. Franke, and M. Stamminger, "ADOP: Approximate Differentiable One-Pixel Point Rendering," *ACM Transactions on Graphics*, vol. 41, no. 4, 2022, ISSN: 0730-0301 (cit. on p. 17).

[53] T.-W. Hui, X. Tang, and C. C. Loy, "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8981–8989 (cit. on p. 27).

[54] A. X. Chang, T. Funkhouser, L. Guibas, *et al.*, "ShapeNet: An Information-Rich 3D Model Repository," 2015. arXiv: `1512.03012 [cs.GR]` (cit. on pp. 27, 34, 35).

[55] Q. Chen and V. Koltun, "Fast MRF Optimization with Application to Depth Reconstruction," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 3914–3921 (cit. on p. 34).

[56] O. Wasenmüller and D. Stricker, "Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision," in *Computer Vision – ACCV 2016 Workshops*, C.-S. Chen, J. Lu, and K.-K. Ma, Eds., Cham: Springer International Publishing, 2017, pp. 34–45 (cit. on p. 45).

[57] G. Halmetschlager-Funek, M. Suchi, M. Kampel, and M. Vincze, "Xtion's Gone! What's Next? An Evaluation of Ten Different Depth Sensors for Robotic Systems," *IEEE Robotics Automation Magazine*, 2018 (cit. on pp. 45, 50).

[58] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "ElasticFusion: Real-time dense SLAM and light source estimation," *International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016 (cit. on p. 45).

[59] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks," in *International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4628–4635 (cit. on p. 45).

[60] S. V. Alexandrov, J. Prankl, M. Zillich, and M. Vincze, "Towards dense SLAM with high dynamic range colors," in *Proceedings of Compute Vision Winter Workshop*, 2017 (cit. on pp. 45, 46, 58).

[61] ——, “Calibration and correction of vignetting effects with an application to 3D mapping,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016, pp. 4217–4223 (cit. on pp. 46, 58).

[62] S. Schreiberhuber, J.-B. Weibel, T. Patten, and M. Vincze, “ScalableFusion: High-resolution Mesh-based Real-time 3D Reconstruction,” in *Conference on Robotics and Automation (ICRA)*, 2019 (cit. on p. 47).

[63] A. Handa, T. Whelan, J. McDonald, and A. Davison, “A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM,” in *International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1524–1531 (cit. on pp. 56, 59).

[64] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A Benchmark for the Evaluation of RGB-D SLAM Systems,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robot Systems*, 2012, pp. 573–580 (cit. on p. 56).

[65] W. Fedus, B. Zoph, and N. Shazeer, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” *J. Mach. Learn. Res*, vol. 23, pp. 1–40, 2021 (cit. on p. 64).

[66] N. Shazeer, A. Mirhoseini, K. Maziarz, *et al.*, “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer,” Jan. 2017 (cit. on p. 64).

[67] T. Müller, F. Rousselle, J. Novák, and A. Keller, “Real-Time Neural Radiance Caching for Path Tracing,” *ACM Transactions on Graphics*, vol. 40, no. 4, 2021, ISSN: 0730-0301 (cit. on p. 64).

[68] J. Engel, V. Koltun, and D. Cremers, “Direct Sparse Odometry,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, 2018 (cit. on p. 64).

[69] B. Burley, “Physically Based Shading at Disney,” in *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH*, ACM, 2012 (cit. on p. 65).

[70] M. Herb, T. Weiherer, N. Navab, and F. Tombari, “Lightweight Semantic Mesh Mapping for Autonomous Vehicles,” in *International Conference on Robotics and Automation (ICRA)*, 2021, pp. 6732–3738 (cit. on p. 65).

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, April 2023

Simon Schreiberhuber