Imperial College London
Department of Computing

Sparse octree algorithms for scalable dense volumetric tracking and mapping

Emanuele Vespa

January 5, 2020

Supervised by: Prof. Paul H. J. Kelly, Dr. Stefan Leutenegger

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing
of Imperial College London
and the Diploma of Imperial College London

Declaration

I herewith certify that the material in this thesis that is not my own work has been properly acknowledged and referenced.

Emanuele Vespa

Abstract

This thesis is concerned with the problem of *Simultaneous Localisation and Mapping* (SLAM), the task of localising an agent within an unknown environment and at the same time building a representation of it. In particular, we tackle the fundamental scalability limitations of dense volumetric SLAM systems. We do so by proposing a highly efficient hierarchical data-structure based on octrees together with a set of algorithms to support the most compute-intensive operations in typical volumetric reconstruction pipelines.

We employ our hierarchical representation in a novel dense pipeline based on occupancy probabilities. Crucially, the complete space representation encoded by the octree enables to demonstrate a fully integrated system in which tracking, mapping and occupancy queries can be performed seamlessly on a single coherent representation. While achieving accuracy either at par or better than the current state-of-the-art, we demonstrate run-time performance of at least an order of magnitude better than currently available hierarchical data-structures.

Finally, we introduce a novel multi-scale reconstruction system that exploits our octree hierarchy. By adaptively selecting the appropriate scale to match the effective sensor resolution in both integration and rendering, we demonstrate better reconstruction results and tracking accuracy compared to single-resolution grids. Furthermore, we achieve much higher computational performance by propagating information up and down the tree in a lazy fashion, which allow us to reduce the computational load when updating distant surfaces.

We have released our software as an open-source library, named *su-*

pereight, which is freely available for the benefit of the wider community. One of the main advantages of our library is its flexibility. By carefully providing a set of algorithmic abstractions, supereight enables SLAM practitioners to freely experiment with different map representations with no intervention on the back-end library code and crucially, preserving performance. Our work has been adopted by robotics researchers in both academia and industry.

Acknowledgements

This thesis marks the end of a long and tortuous journey and I want to acknowledge all my friends and colleagues who helped me reaching this life and career milestone.

First and foremost, my biggest gratitude goes to my supervisor, Paul Kelly for giving me the opportunity to pursue this doctorate. I feel incredibly lucky to have had Paul as supervisor. Paul's enthusiasm and vision guided me throughout these years and his relentless support has been essential to survive the unavoidable dark moments that every PhD student has to endure. I am sincerely grateful for this and I will be always proud to have had you as my supervisor. Another big thanks goes to my co-supervisor, Dr. Stefan Leutenegger. Working with Stefan has been an invaluable opportunity, his high standards pushed me when I needed it and I am convinced that our collaborations were essential for the success of this work. I also want to thank my examiners, Prof. Andrew Davison and Dr. Maurice Fallon for their thoughtful remarks and feedback, which I believe helped shaping a better thesis.

I am very grateful for the time I spent at Imperial and for all the extraordinary people I interacted with throughout these years. First and foremost I want to thank my good friend Fabio Luporini, with whom I enjoyed countless hours of scientific discussion, trolling and most importantly Teeworlds. Our years at Imperial cemented even more an important friendship which I believe will always continue regardless of time and distance. I also want to thank my other good friend Andrea Nicastro, with whom not only share the same passion for computer vision and robotics but most importantly for climbing. I look forward to explore Switzerland and send some hard projects together.

I want to thank all the people from the Software Performance Optimisation and Robotic Vision groups I had the pleasure to work with, such as Luigi Nardi, Zeeshan Zia, Lawrence Mitchell, Francis Russell, TJ, Miklos Homolya, Ankur Handa and all the others. It's hard to quantify how much I learnt from you and I am very grateful for this. Thank you also to our regular visiting professor Prof. J. Ram Ramanujam for the numerous stimulating chats, coffees and pizzas, I enjoyed each one of them.

I also treasure the time spent interning at ARM and Microsoft, which considerably contributed to my professional growth. Thank you Giacomo Gabrielli, Ning Xu and Harpreet Sawhney.

Since this is (hopefully) the conclusion of my academic education, I also want to acknowledge three key figures from my high-school years. These are Prof. Maurizia Burrai, philosophy and history teacher, who instilled in me more critical thinking than any science and math teacher I had before her; Prof. Letizia Floris, Italian and Latin teacher, who first taught me how to write clearly; and finally Prof. Anna Porra, English teacher, who introduced me to English literature, which I still study and enjoy. I feel really lucky to have come accross such brilliant teachers during my education.

Finally, I want to express my gratitude to my closest ones. First and foremost Francesca, who has been by my side throughout this experience, during the good and the bad times. Her constant support has been of fundamental importance for me and I believe that without her this thesis would not exist. I am also thankful for my parents and family who have always supported me and gave me the tranquillity to work on what I love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We are fast approaching a future where autonomous robots, mixed reality devices and virtual reality will be ubiquitous in our daily life activities. Fully autonomous self-driving cars will radically change the way our cities are lived. Virtual reality headsets promise to change the way we interact with our colleagues or family members. Autonomous robots are already being employed for a variety of activities, from domestic cleaning to search and rescue missions in hazardous environments. Common to all these scenarios is the ability of the device or agent to accurately sense its surrounding environment and have an understanding of it.

In the past decade, motivated by the ever increasing and diverse applications which require real-time positional and mapping data, *Simultaneous Localisation and Mapping* (SLAM) research has evolved at an unprecedented pace. The widespread availability of commodity depth sensors fuelled a true paradigm shift from *sparse* systems, in which typically the maps consisted of sparse landmarks, to fully *dense* methods where essentially the full scene geometry can be reconstructed. The richness of the geometric information which dense methods are able to estimate is of fundamental importance for a variety of downstream applications, such as autonomous cars or unmanned aerial vehicles, which exploit knowledge about surfaces to perform safe navigation and plan obstacle-free paths.

In this thesis we focus on dense SLAM algorithms. We specifically tackle the scalability issues that dense representations incur into. As we will show, we develop an efficient octree library which supports the essential operations required by dense reconstruction pipelines and it does so in a generic way. This is justified by the need of different field representations depending on the application at hand. For example, while in surface reconstruction signed-distance function mapping is extremely popular, many path-planning or object avoidance algorithms in robotics assume an occupancy based map. Our framework naturally supports arbitrary continuous field representations and allows the application programmer to freely experiment with alternative designs *without* sacrificing performance.

To demonstrate the effectiveness of our approach, we i) implement a standard dense pipeline using our framework and compare it to the current state-of-the-art in volumetric SLAM; ii) introduce a novel, fully integrated SLAM system based on occupancy mapping, where occupancy queries *and* tracking and mapping can be performed on a single, unified representation; iii) propose a novel multi-resolution dense tracking and mapping pipeline that adaptively select the best integration and rendering scale given the effective sensor resolution.

1.1. Thesis outline and contributions

The thesis is centred around two main topics: i) the design and develop of a high performance octree library which targets volumetric SLAM applications; ii) the investigation of novel surface representations which can be used in dense SLAM pipelines. These two aspects are tightly related: we propose abstract interfaces which seamlessly allow us to experiment with alternative surface representations.

The thesis comprises seven chapters, including introduction and conclusions. In Chapter 2 we review the literature related to this work, providing its context and motivation. Furthermore, the essential mathematical tools used in the thesis are briefly discussed. Chapters 3, 4 and 5 present our main contributions, as follows:

Chapter 3 In this technical chapter we report in detail the design of *supereight*, our high-performance octree library and first contribution. First, we discuss the general design principles and how they affect what computations are possible to express using our library. Secondly, perhaps more importantly, we introduce the algorithms which allow supereight to reach its performance level. We then proceed by describing the implementation of a dense reconstruction pipeline, based on the seminal KinectFusion algorithm (Newcombe et al. [2011a]). A thorough quantitative evaluation is given, where we compare our supereight-based implementation against the current state-of-the-art in volumetric reconstruction (InfiniTAM of Kähler et al. [2015]). The main outcome is that our hierarchical data-structure offers performance levels on par with flat hash tables, but with the advantage of providing a complete spatial indexing of the scene. Furthermore, as expected the accuracy of the overall SLAM pipeline is at par or better than the state-of-the-art. When presenting our results we also discuss our benchmarking process and justify the choices taken to enable a fair comparison between differently engineered systems.

Chapter 4 In this chapter we introduce a novel dense reconstruction pipeline based on a fully probabilistic surface representation. We adopt the surface reconstruction framework introduced by Loop et al. [2016] and adapt it to the needs of real-time incremental fusion which characterise dense SLAM algorithms. The result is a complete system based on occupancy mapping which is able to precisely reconstruct surfaces with a rigorous probabilistic interpretation. Crucially, the same representation can be used to seamlessly fuse new information, to render model views for camera tracking *and* perform spatial occupancy queries, enabling fully integrated tracking, mapping and planning operations. Finally, we provide a qualitative and quantitative analysis of our pipeline. Our occupancy-based pipeline achieves tracking accuracy on par or better than TSDF-based systems, but crucially is *at least* one order of magnitude faster than the state-of-the-art in robotics occupancy mapping, enabling real-time operations which where simply not possible with the tools previously available to the community.

Chapter 5 In this chapter we propose a multi-resolution dense tracking and mapping system based on our octree structure which adaptively selects the adequate level of detail in both measurement integration and scene rendering. With the goal of reducing aliasing artefacts, this is done according to the effective pixel resolution given the distance from the observed surface. We introduce a novel algorithm to perform *lazy* propagation of information across octree levels to keep the hierarchy consistent on a per-frame basis. Our strategy reaches considerably higher computational performance compared to equivalent single-resolution grids. More importantly, we demonstrate how reducing aliasing artefacts yields much better reconstruction of scenes with high-frequency details that are observed at different scales over time.

1.2. Dissemination

The body of work presented in this thesis has been made available to the wider community through various publications and presentations. The software projects that have been developed in this thesis have been released under open-source licences and are being used to carry further research.

Publications This thesis is based on the following publications:

- Vespa et al. [2018]: E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly and S. Leutenegger, "*Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping*" in IEEE Robotics and Automation Letters, vol. 3, no. 2, pp. 1144-1151, April 2018.
- Vespa et al. [2019]: E. Vespa, N. Funk, P. H. J. Kelly, and S. Leutenegger, "*Adaptive resolution octree-based dense SLAM*". 2019 International Conference on 3D Vision (3DV), Quebec City, Canada, September 2019.

Additionally, the following papers have been co-authored:

- Zia et al. [2016]: M. Z. Zia, L. Nardi, A. Jack, E. Vespa, B. Bodin, Paul H J Kelly, A. J. Davison, "*Comparative design space exploration of dense and semi-dense SLAM*" 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, 2016, pp. 1292-1299.

- Nardi et al. [2017]: L. Nardi, B. Bodin, S. Saeedi, E. Vespa, A. J. Davison and P. H. J. Kelly, *"Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper"* 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, 2017, pp. 1434-1443.

- Saeedi et al. [2018]: Saeedi et al., *Navigating the Landscape for Real-Time Localization and Mapping for Robotics and Virtual and Augmented Reality*, Proceedings of the IEEE, November 2018.

In Vespa et al. [2018] we introduce the first two technical contributions of this thesis. We describe the algorithmic solutions used in our octree-library (Chapter 3) and a novel real-time dense SLAM pipeline based on occupancy probabilities (Chapter 4). In Vespa et al. [2019] we present the adaptive-resolution reconstruction method reported in Chapter 5.

Zia et al. [2016] and Nardi et al. [2017] are collaborative works that have been conducted as part of a large inter-university project known as the PAMELA project. Specifically, we investigate and extend the benchmarking methodologies and tools first introduced by Nardi et al. [2015], which constitute the backbone of the rigorous experimental work conducted in this thesis. All the work carried over in the PAMELA project has been subsequently summarised in the invited paper Saeedi et al. [2018].

Presentations Various presentations have been given, both in oral and poster format. A selection, in chronological order:

- *Comparative design space exploration of dense and semi-dense SLAM*, International Conference on Robotics and Automation (ICRA), Stockholm, 2016.

- *Real-time 3D scene understanding: volumetric SLAM using Octrees and Morton numbers*, ARM Research Summit, Cambridge, 2017.

- *Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping*, International Conference on Robotics and Automation (ICRA), Brisbane, 2018.

- *supereight: Real-time Dense SLAM with Signed-Distance and Occupancy Mapping*, Microsoft HoloLens, Redmond, 2018

Videos
- Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping: https://youtu.be/n-18Lcx1LTU
- Adaptive resolution octree-based dense SLAM: https://youtu.be/sXEzLGJozQQ

Software As part of the present work, we developed a generic C++ software library, named "supereight", which we have released[1] under a permissive open-source licence.

1.3. Funding

We would like to thank ARM (through an EPSRC Industrial CASE studentship), the EPSRC grants PAMELA (EP/K008730/1) and (HiPEDS EP/L016796/1), and Imperial College London for generously funding this research.

[1]https://github.com/emanuelev/supereight

Chapter 2

Background

In this chapter we discuss the research context of the work presented in this thesis. We first give an a brief historical review of the field, followed by a detailed discussion of the research literature more closely related to our work. Finally, we present the notation and the fundamental mathematical tools used throughout this thesis.

2.1. Visual Simultaneous Localisation and Mapping

The task of localising an agent as it moves through an unknown environment has being subject of research for the past thirty years. Given a stream of measurements coming from one or multiple sensors, it is possible to accurately compute the agent location. Additionally, the abundant information collected may be used to estimate a consistent map of the environment being explored. When these two tasks are carried out concurrently, the general estimation problem is referred to as *Simultaneous Localisation and Mapping*, in short SLAM.

While a variety of sensors may be employed to solve the SLAM problem, in this work we focus on systems that rely primarily on visual information. Furthermore, we are particularly interested in solutions that are able to achieve *real-time* performance. Satisfying such a constraint essentially means that both agent position and map should be updated at sensor frame-rate. The SLAM problem has strong connections to what in photogrammetry is known as *bundle adjustment* (Triggs et al. [2000]) or in computer vision communities as *structure from motion* (Szeliski [2010]).

(a) Microsoft HoloLens

(b) ANYmal

Figure 2.1.: **(a)** Microsoft's HoloLens mixed reality device (image from www.microsoft.com/en-us/hololens). Equipped with multiple cameras and depth sensors, supports fully untethered operations. **(b)** ANYbotics quadrupedal robot ANYmal (image from www.anybotics.com) . Its advanced stability and perceptions capabilities combined with its rugged design allow it to operate autonomously in harsh conditions.

While the underpinning principles are the same, the main discriminant is the online nature of SLAM estimation, as opposed to the batch optimisation nature of bundle adjustment and structure from motion.

Real-time localisation and mapping systems have a wide range of applicability and provide a foundational platform for a variety of applications. *Mobile robotics*, in which autonomous robots equipped with sensors explore and interact with the environment, has long been one of the main drivers behind SLAM research and one of the major adopters of the technology. Nowadays, dozens of industrial solutions in different markets deliver products that exploit some form of SLAM algorithms, ranging from domestic vacuum cleaners, autonomous drones or self-driving cars.

Recently, enormous steps forward have been made in the *augmented* and *virtual reality* communities. A plethora of devices and services, targeting both the consumer and the professional markets have been released, bringing to the general population entirely new experiences. Augmented reality (AR) devices, such as the Microsoft HoloLens, virtual objects are superimposed into the user's visual field of view. In order to achieve precise placements of synthetic objects, both headset's position and real world geometry must be accurately recovered. Virtual reality (VR) headsets, such as the Oculus Quest, use vision-based localisation systems to enable completely untethered operations. Finally, AR/VR applications are now widely available on mobile devices such as smartphones, with each major OS platform providing native development kits to application developers (notably, Apple's ARKit and Google's ARCore).

Despite the evident differences between the aforementioned applications, a common trait is the basic requirement for fast and precise localisation, as well as awareness of the surrounding environment, being it for enabling interaction or for safe navigation. This is precisely what SLAM algorithms provide and the main reason behind their success in various fields. Although being subject of intensive research for the past tree decades, the field is still very active. Modern advances in machine learning research begun to heavily influence and cross-pollinate SLAM research. Several works in recent years consider not only the geometry estimation problem but also the object classification task at the same time, incorporating semantic information into the maps estimated by the SLAM system. Effectively, SLAM is rapidly evolving in a technique to achieve full scene understanding in a broad sense.

2.2. Related Work

We structure the literature review presented in this section in three main parts. Section 2.2.1 gives an historical perspective on the field of Simultaneous Localisation and Mapping. Section 2.2.2 discusses in detail the body of work most closely related to our research, mainly focusing on RGB-D dense reconstruction systems. Finally, in Section 2.2.3 we give a review of hierarchical spatial data-structures.

2.2.1. A brief review of sparse and semi-dense visual SLAM

Historically, a variety of sensors have been successfully used in SLAM systems, such as sonar, radar or laser range finders. However, in the past decade systems based on visual information have gained enormous popularity in the computer vision community. Typically, the maps estimated by such systems would consists of a collection of *sparse*, distinctive image features with associated 3D position which are easy to track in frame sequences, like corners or interestingly textured colour patches. Complete models would be made of hundreds or up to thousands of points.

One of the earliest examples of a real-time system based purely on visual information is the breakthrough work MonoSLAM of Davison et al. [2007]. Using a single monocular camera, the system is able to estimate full 6DoF camera pose and the 3D positions of distinctive features in the scene. Both the state of the camera and the landmarks locations are jointly represented with a unimodal, multivariate Gaussian distribution. Probabilistic state estimation is then performed using an *Extended Kalman Filter* (EKF). Crucially for the agility of the system, covariance estimates are exploited to predict features positions in new frames, considerably reducing the search radius and hence speeding up matching.

One of the major drawbacks of EKF-based methods is the computational and space complexity associated with the covariance of the state vector. In fact, while the state grows linearly with the number of tracked features, its covariance grows quadratically, strictly bounding the size of the map which can be estimated in real-time. To overcome such limitations, alternative filtering methods have been proposed. FastSLAM (Montemerlo et al. [2002]) employs a Rao-Blackwelised particle filter in which the poses distribution is estimated independently from the landmarks positions, which are associated with an individual Kalman Filter for each particle. This approach was further developed and applied to monocular SLAM by Eade and Drummond [2006].

In the meantime, optimisation methods derived from offline bundle adjustment have started to appear. One of the key differences between batch and filter-based approaches is how the previous poses are handled. While filtering marginalises all historical poses except the current one, graph-based approaches retain the whole history and solve the whole graph from scratch every time. Since in a real-time setting this would be unfeasible as more poses are recorded, hence typically the graph is carefully sparsified and only a small subset of poses are retained.

Nister et al. [2004] implement a real-time visual odometry system for monocular and stereo cameras, where the camera trajectory is estimated in closed form and 3D points are triangulated from 2D correspondences. The lack of a mechanism to deal with re-observation of past landmarks leads inevitably to pose drift over time. Engels et al. [2006] demonstrate

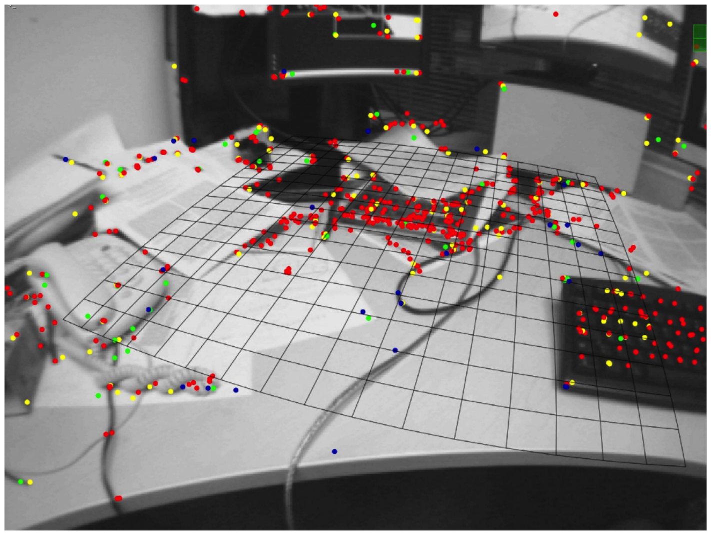


Figure 2.2.: Example of sparse maps produced by the PTAM system (image from **?**). Dots indicates successfully tracked feature points, while the grid shows an estimated planar surface which can be used to draw virtual objects on the scene.

that applying bundle adjustment in a sliding window fashion drastically improves the accuracy and robustness of the system, while still allowing for real-time operations.

An large contribution to the field has been made by **?** with their *Parallel Tracking and Mapping* (PTAM) system (Figure 2.2). Their crucial intuition was to decouple the tracking and mapping stage. In PTAM, camera tracking given a known map is performed at high frame-rates by a tracking front-end. Map estimation and pose optimisation is carried out in a concurrent thread, which can perform bundle-adjustment at a much slower frequency. Importantly, the map is made of a set of carefully selected, wide baseline keyframes, which cover the area of operation. The real-time camera pose is then robustly recovered by minimising the re-projection error of matching feature points belonging to the key-frames map projected onto live camera frames.

The trade-offs between filtering and keyframes based approaches have been thoroughly investigated by Strasdat et al. [2012]. The main conclusion is that optimisation-based systems over a large set of features but lacking joint distribution estimates support greater accuracy and robustness compared to filter-based methods over fewer map points.

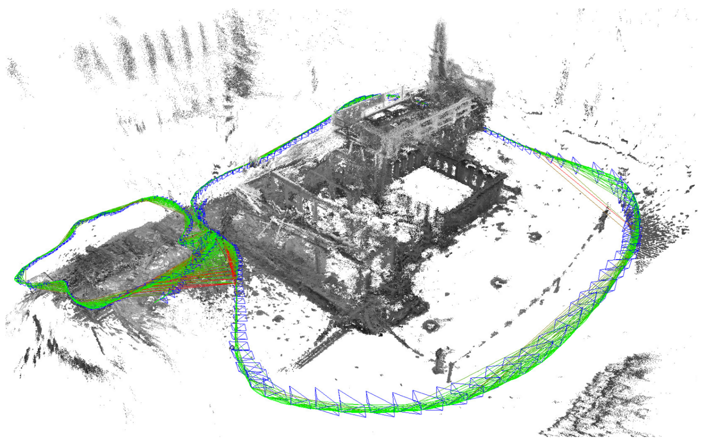


Figure 2.3.: Large scale reconstruction obtained with the LSD-SLAM system (figure from Engel et al. [2014]). Notice how compared to sparse methods the reconstruction is significantly denser and geometric structure is easily recognizable.

A number of works addressed the scaling limitations of PTAM, such as the sub-maps approach of Castle et al. [2011] and the FrameSLAM systems of Konolige and Agrawal [2008] which specifically address large-scale mapping and global loop-closures. Strasdat et al. [2010] intelligently observe that monocular exploration over long trajectories eventually also leads to scale drift. Their system perform live 6DoF camera alignment and sliding window bundle adjustment for map optimisation. However, when a loop-closure is detected, a lightweight pose-graph optimisation is performed over 7DoF, i.e. a similarity transform between keyframes is computed, compensating for scale drift. Finally, structure-only bundle adjustment is performed over the whole map. These approaches have been adopted and further refined in the ORB-SLAM system of Mur-Artal et al. [2015].

Common to the methodologies described in the above is their reliance on accurate feature detection and matching. Recently, a number of systems have been proposed that directly work with pixel intensities as an alternative to higher level descriptors. Instead of minimising a geometric re-projection error, such systems optimise a photometric cost function. Typically, methods belonging to this family are called *direct* methods, as opposed to feature-based *indirect* methods.

Engel et al. [2013] introduce a novel approach to monocular reconstruction and SLAM where depth information is recovered only for high gradient pixel, obtaining what is called a *semi-dense* reconstruction (Figure 2.3). Their initial visual odometry work has been subsequently extended to a complete keyframe-based SLAM system which supports both monocular (Engel et al. [2014]) and stereo (Engel et al. [2015]) camera set-ups.

Forster et al. [2014] develop an efficient hybrid visual odometry system which combines elements of direct tracking and mapping with indirect back-end optimisation. Depth of points and live camera poses are estimated using a direct formulation which minimises the photometric error between the reference and live frame for each point patches. In the back-end, poses and structure are optimised jointly with an indirect formulation which uses the correspondences found with the direct model.

Finally, it should be noted that several methodologies have been proposed to increase the accuracy and robustness of visual odometry and SLAM systems by incorporating Inertial Measurements Units (IMU) readings in the estimation problem. Such sensors can provide higher accuracy and robustness in challenging scenarios, such as rapid camera motions or navigation through environments which do not offer enough visual features. Furthermore, metric scale become directly observable, providing monocular systems with scale information which would be not recoverable from purely visual information. Consequently, IMU estimates have been integrated both in filter-based approaches (as in Mourikis and Roumeliotis [2007], Li and Mourikis [2013] and Bloesch et al. [2015]) and iterative optimisation frameworks (such as in Jones and Soatto [2011], Leutenegger et al. [2015], Forster et al. [2017] or Qin et al. [2018]).

Up to this point, we have considered systems which rely purely on passive visual information, being it monocular or stereo. However, the recent advances in depth sensing technologies changed the landscape of SLAM algorithms, by bringing to the forefront of research *dense* tracking and mapping. This family of algorithms is strictly related to the work presented in this thesis and will be presented in depth in the following section.

2.2.2. Dense Visual SLAM

As opposed to sparse and semi-dense methods, *dense* algorithms try to exploit *all* the visual information available. This is reflected both in how the camera tracking and pose optimisation is performed and the type of geometry that algorithms are able to produce. One of the earliest real-time reconstruction pipelines was proposed by Newcombe and Davison [2010]. Their system use PTAM to obtain local camera poses and point estimates. A dense surface is then fitted to the sparse map and subsequently refined to obtain a watertight photo-consistent surface estimate. Shortly after, Newcombe et al. [2011b] published their monocular DTAM system, where fully dense depth maps are estimated for selected keyframes. Differently from their earlier work, DTAM does not utilise an external SLAM system for pose estimation, but performs dense model-to-live frame alignment to recover the 6DoF camera motion. A different approach is taken by Pizzoli et al. [2014] with their REMODE framework, where probabilistic per-pixel depth estimation is combined with spatial regularisation, using SVO (Forster et al. [2014]) as tracking system.

The advent of compact and cheap depth sensors at the beginning of this decade brought a considerable paradigm shift in the SLAM landscape. Pioneered by Microsoft with their Kinect sensor, these new devices provide accurate depth information and, importantly, they do so at high framerate (i.e. 30Hz). Depth estimation is achieved via active vision techniques, such as structured light or time-of-flight approaches. Structured light sensors such as Kinect v1 or the Structure Sensor[1] project a pattern on the world surfaces and infer depth by observing the pattern distortion from a secondary camera. Time-of-flight sensors, such as the Azure Kinect[2], instead measure the time that is required for a light ray emitted by an illumination unit to reach the observed object and back to the sensor. We refer the reader to the excellent comparison of the two approaches by Sarbolandi et al. [2015]. It must be noted that even if able to provide dense depth maps, active vision sensors have a series of limitations which impair their ability to work in general settings. In particular, error sources

[1]https://structure.io
[2]https://azure.microsoft.com/en-in/services/kinect-dk/

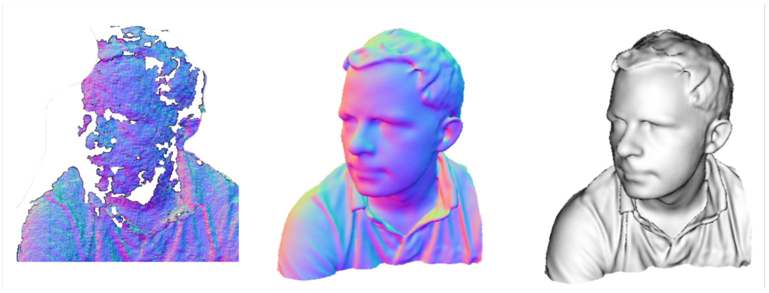


Figure 2.4.: Dense reconstruction obtained with KinectFusion (figure from Newcombe et al. [2011a]). On the left is shown the raw point cloud obtained from the Kinect sensor. In the middle is represented the surface normal map and on the right the Phong-shaded render of the surface.

such as ambient background light or multi-path effects affect their usability in outdoor scenarios, where passive stereo cameras may still be the better choice. Furthermore, most sensors offer a limited operating range, usually below 10m, that may be too limiting even in medium scale indoor scenario.

Despite the above limitations, researchers readily adopted the new technology and a vast number of systems specifically using RGB and Depth (in short RGB-D) information have been published. Henry et al. [2012] propose one of the earliest methods were feature-based RGB alignment and depth-based *Generalised Iterative Closest Point* (GICP) are fused together to incrementally build a pose-graph. This is subsequently optimised using *Sparse Bundle Adjustment* (SBA) and a global surfel map is obtained. Similarly, Endres et al. [2012] exploit visual features and depth information to grow a pose-graph, but then fuse a globally consistent point-cloud into an octree-based 3D occupancy grid. Kerl et al. [2013] use a direct alignment formulation which consists of both photometric and geometric error terms for visual odometry. The map is represented by a graph of keyframes and a globally consistent point-cloud can be obtained after loop-closure detection and map optimisation.

Newcombe et al. [2011a], with their seminal KinectFusion system, diverged from more traditional, keyframes-based SLAM solutions to achieve real-time high quality 3D reconstruction (Figure 2.4). KinectFusion adopts the volumetric representation of Curless and Levoy [1996], where the map is represented with an implicit *Truncated Signed Distance Function* (TSDF). The explored region is discretised in a voxel grid in which each point holds the distance to the closest surface. Actual surface boundaries can be recovered by extracting the zero-isosurface. Visual odometry is obtained by performing dense frame-to-model alignment via with a modified ICP algorithm. Thanks to GPGPU acceleration, KinectFusion was the first system to provide real-time tracking and mapping with fully textured, sub-centimetre resolution reconstruction.

After the release of KinectFusion, a whole body of research has appeared to address its limitations. One line of work, aimed at improving the accuracy of the pipeline by providing alternate fusion and tracking formulations. Canelhas et al. [2013] and Bylow et al. [2013] suggest to replace ICP-based tracking with direct point-cloud to TSDF alignment. A new scan is registered to the global TSDF by minimising a cost function which penalises divergence from the surface zero crossing. Slavcheva et al. [2016] further develop this approach by proposing a signed function to signed distance function alignment. After the online fusion and tracking stage, their system also performs a final offline refinement of the model.

Nguyen et al. [2012] turn their attention to accurately modelling sensor noise model and incorporate it in the reconstruction pipeline. They empirically derive a noise model for the Kinect V1 sensor and demonstrate interesting improvements in accuracy and reconstruction precision. Similarly, Fankhauser et al. [2015] derive a noise model for the Kinect V2 and use it to improve depth-based navigation in a robotics setting. Both Bylow et al. [2013] and Oleynikova et al. [2017] provide noise-aware voxel update equations in the form of dynamically sized truncation bandwidth.

Another line of research tackled one of the major limitations of KinectFusion, i.e. is its scalability. Voxel grids scale poorly as the number of voxels grows cubically as a function of resolution or area covered. In its initial implementation, KinectFusion was consequently able to operate only on small, desktop-sized areas. Roth and Vona [2012] extend the range of operation by spatially shifting and rotating a fixed size grid as the camera moves through the environment. The TSDF values for the transformed grid are computed via trilinear interpolation. The Kintinuous system of Whelan et al. [2012] takes a similar approach, but contrary to the work of Roth and Vona [2012] it allows only volume translation, constrained to voxel units. This permits to efficiently implement shifting operations with circular buffer and no interpolation is needed to recover information. In a following work (Whelan et al. [2015a]), Kintinuous is extended to support real-time triangular mesh extraction and deformation upon loop-closure. The global mesh is obtained incrementally as the volume moves through space, i.e. slices of volume that leave the current operational range are converted to an explicit mesh. Interestingly, instead of running expensive algorithms such as marching cubes (Lorensen and Cline [1987]) a point cloud is extracted via an axis aligned ray-casting and subsequently triangulated.

Zeng et al. [2013] propose a GPU-based octree structure to represent the TSDF volume which is able to map considerably larger areas compared to preallocated grids. However, their rendering algorithm does increase the chances of drift as no trilinear interpolation is used to recover the surface. Furthermore, the computational performance gains compared to a standard KinectFusion implementation are very marginal despite the considerable reduction in memory usage.

Steinbrucker et al. [2014] develop a fully multi-resolution octree structure in which contiguous bricks of voxels are stored at different height of the tree explicitly and hence explicitly encoding the signed distance function at different scales. However, contrary to Zeng et al. [2013] they employ an external RGB-D SLAM system for pose estimation and the volume is computed after all the input frames have been processed.

Chen et al. [2013] introduce a GPU-based regular hiearchical data-structure which exploits a parametric number of subdivisions per level. This can be interpreted as a squashed octree where levels are compressed together as a function of level and aggregation factor. Furthermore, they introduce an intelligent streaming mechanism which allows to download inactive voxels to the CPU when the camera moves towards the edges of the currently mapped area. Contrarily to Kintinuous, inactive voxels may also be reactivated and reintegrated in the live reconstruction, however no mechanism for global loop-closure is provided.

Nießner et al. [2013] move away from hierarchical data-structures and develop a very efficient reconstruction pipeline based on voxel hashing.

Brick of voxels are dynamically allocated and indexed via pointers stored in a GPU-friendly hash-table. Similarly to the previous work of Chen et al. [2013], blocks are streamed in and out the GPU, allowing for virtually unbounded reconstructions. Variations of this strategy has been subsequently re-implemented in a number of works (Klingensmith et al. [2015], Oleynikova et al. [2017], Millane et al. [2018]).

Voxel hashing is adopted also by the InfiniTAM reconstruction framework of Kähler et al. [2015] which demonstrate impressive state-of-the-art computational and accuracy performance. A multi-resolution version of this work is introduced in Kähler et al. [2016], where a hierarchy of hash-tables, each representing a grid at different resolution, is exploited to compress planar surfaces into a progressively coarser grid.

Recently, a number of works have appeared that tackle volumetric loop-closure in real-time. In general, one of the biggest problems when using a volumetric representation is the difficulty of updating the surface estimate upon loop-closure correction. Dai et al. [2017] propose a complex pipeline in which all the historical frames are taken into account when solving for the optimal global trajectory and model. In order to keep the problem tractable, local alignment is computed over small batches of input frames while global optimisation is performed over a selection of carefully chosen keyframes. Importantly, their TSDF fusion allows for both *integration* and *deintegration* of measurements.

Kähler et al. [2016] extend InfiniTAM with loop-closure detection, but contrarily to the work of Dai et al. [2017] they propose an approach based on sub-maps. The global map is subdivided into a number of sub-maps which are spawned according to a visibility criteria. Loop closures are detected via the keyframe-based relocalisation method of Glocker et al. [2015]. Once the constraints have been gathered, sub-maps are densely aligned and a globally consistent reconstruction is obtained.

Over the years, researchers have investigated alternative map representations to dense voxel grids. Notably, Keller et al. [2013] introduce a surfel-based reconstruction pipeline, in which input depth scans are fused into an explicit global point-cloud. Synthetic views for frame-to-model alignment are then obtained via surface splatting (Pfister et al. [2000]).

State-of-the-art reconstruction and tracking accuracy are demonstrated by Whelan et al. [2015b] with their ElasticFusion system (Figure 2.5).

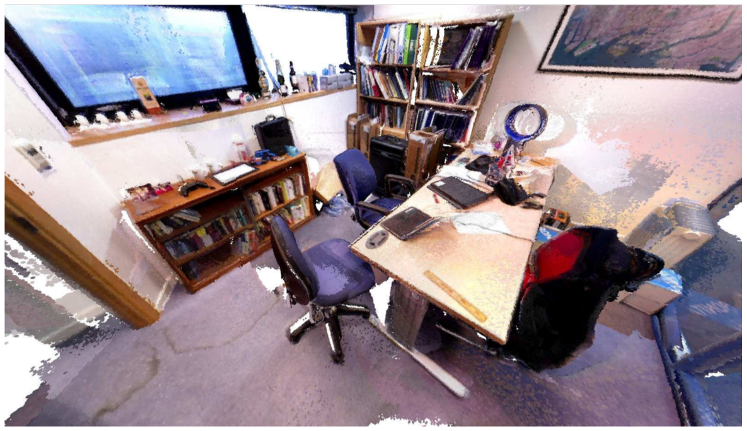


Figure 2.5.: ElasticFusion reconstruction output (figure from Whelan et al. [2015b]). The surfel density ensure a nearly holes-free reconstruction and impressive visual results.

While adopting the point-based representation of Keller et al. [2013] they significantly improve the algorithm by providing small and large scale loop-closure with non-rigid map deformations. Subsequently, a number of works adapted ElasticFusion's base pipeline to more specific scenarios, such as the extension to support inertial measurements of Laidlow et al. [2017] or the Co-Fusion system of Rünz and Agapito [2017] that explicitly deals with non-static objects. Recently, Schops et al. [2019] introduced a surfel-based dense SLAM algorithm which jointly optimises camera trajectory and surface geometry. In contrast to ElasticFusion, surfels are associated to key-frames which are selected based on a temporal criteria. To keep the problem size tractable, geometry and pose-graph optimisation are alternated until convergence. While providing very impressive visual results, the main drawbacks of surfel-based methods is their inability to capture surface connectivity or explicitly represent empty space that has been observed, which is a desirable features in many application scenarios.

Enabled by the recent advances in deep learning technologies, SLAM researchers started to evolve classic algorithms, which essentially aimed at solving a geometry estimation problem, to more complete scene understanding methodologies. Salas-Moreno et al. [2013] presents one of the first object-level prototype SLAM systems, where instead of building a fully dense 3D model of the environment, a graph of known objects (with their individual model and position) is grown and optimised over time. McCormac et al. [2017] extend ElasticFusion to include a per-surfel semantic label inferred via a Convolutional Neural Network which provides semantic annotation on a per-pixel basis. In a similar spirit, the aforementioned Co-Fusion work of Rünz and Agapito [2017] densely recognize and reconstruction objects with the aim of robustly tracking them in a dynamic scene. A further step forward towards truly semantic SLAM has been recently made by McCormac et al. [2018], with their Fusion++ system. A pose-graph of per-object volumetric models is employed as space representation. Objects models are spawned and incrementally refined as new depth data is observed and are used for tracking, loop-closure detection and object-graph optimisation.

2.2.3. Spatial indexing

Efficient data-structures for indexing spatial data have been widely researched in a variety of fields, ranging from graphics, vision or physics. The work presented in this thesis specifically adopts *octrees* as its spatial representation. An octree is a regular data-structure which partitions 3D space into eight cubes, which in turn can be recursively subdivided (see Section 3.4 for a detailed presentation). While many systems in the context of volumetric mapping and more generally SLAM have successfully applied octrees, our work is more heavily influenced by the high performance graphics community. Of particular relevance, are the tree construction algorithms of Lauterbach et al. [2009], Garanzha et al. [2011] or Zhou et al. [2011] that exploit Morton numbering (Section 3.4.1) and breadth-first traversal strategies to efficiently build the hierarchy. Their work has been later improved by Karras [2012] who provide a fully parallel method which avoids synchronisation at tree levels. Similar algorithms are given by Bédorf et al. [2012] and Burtscher and Pingali [2011], where octree decomposition is used to accelerate *n-bodies* simulations.

In robotics, octrees have also been successfully applied to accelerate occupancy mapping (Section 4.2). Fairfield et al. [2007] employ an efficient octree data-structure to implement a particle-filter based SLAM system. The current de-facto standard is the OctoMap framework of Hornung et al. [2013]. While achieving good memory compression and ease of use, its performance is completely inadequate for real-time usage. Our work aims to provide a considerably more flexible system while improving both computational and accuracy performance, which we thoroughly discuss in Chapter 4.

In the context of partial differential equations (PDE) solvers, The *adaptive mesh refinement* (AMR) framework of Burstedde et al. [2011] employs an efficient octree representation known as *linear octrees*, firstly introduced by Gargantini [1982], in which only leaves of the tree are stored and sorted according to their morton code and scale. The linear representation does not keep explicit parent-child pointers, but individual nodes can be retrieved by searching the linear buffer according to their relative sorted order. Their work is later extended by Isaac et al. [2015] to support complex queries such as mesh-polytope intersection and tree balancing operations.

Miller et al. [2011] propose an interesting approach where shallow octree hierarchies are represented as bit strings and actual data stored in separate buffers in breadth-first order. Large scale mapping is achieved by allocating and maintaining a forest of shallow trees. However, the data layout poses significant challenges when growing or refining tree cells which must be done every frame at interactive frame-rates.

In this section we have discussed the research context of the work presented in this thesis. While there is a large number of systems that provide dense localisation and mapping, most of them are unable to either provide the level of performance required for real-time operations on CPU or the flexibility to work with different surface representations. In the following chapters we will describe our methods and how we address the limitations of the current state-of-the-art frameworks. Specifically, in Chapter 3 we detail our octree-based mapping library, which shares many aspects with InfiniTAM system (Kähler et al. [2015]) but complement it in two ways: i) providing a complete space index of the reconstructed scene; ii) providing adequate facilities to work with different data-types with minimal intervention on user code. In Chapter 4 we detail our occupancy-based dense SLAM pipeline. Compared to the current standard in occupancy mapping, that is the OctoMap framework of Hornung et al. [2013], we believe our system significantly advances the state-of-the-art and can be successfully used in real-world robotics systems. Finally, in Chapter 5 we report our recent progress in multi-resolution volumetric mapping and rendering and how that can be exploited to achieve overall better accuracy in both tracking and reconstruction, while achieving much better computational performance.

2.3. Technical Background

In this section we introduce the mathematical notation and tools that will be used throughout this work.

2.3.1. Mathematical notation

Matrices and vectors are denoted with boldface letters. Homogeneous vectors are indicated with the *dot* notation:

$$\dot{\mathbf{x}} \equiv \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1}, \quad \text{where} \quad \mathbf{x} \in \mathbb{R}^n$$

We denote the rigid body transformations from a reference frame b to reference frame a with elements of the special Euclidean group $\mathbf{T}_{ab} \in \mathbb{SE}_3$, defined as

$$\mathbf{T}_{ab} = \begin{bmatrix} \mathbf{R}_{ab} & \mathbf{t}_{ab} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.1}$$

where $\mathbf{t}_{ab} \in \mathbb{R}^3$ is the translational component and $\mathbf{R}_{ab} \in \mathbb{R}^{3x3}$ is the rotation component belonging to the special orthogonal group $\mathbb{SO}_3$ of three-dimensional rotations, which satisfies $\mathbf{R}_{ab}\mathbf{R}_{ab}^T = \mathbf{I}$ and $det(\mathbf{R}_{ab}) = 1$.

An inhomogeneous vector $\mathbf{x}$ can be transformed using its homogeneous representation via direct multiplication to an element of the $\mathbb{SE}^3$ group:

$$\dot{\mathbf{x}}_a = \mathbf{T}_{ab}\dot{\mathbf{x}}_b$$

where the subscripts a and b indicates the vectors' reference frames. Equivalently, such transformation can be expressed in inhomogeneous coordinates as

$$\mathbf{x}_a = \mathbf{R}_{ab}\mathbf{x}_b + \mathbf{t}_{ab}$$

When using the homogeneous representation, transformations can be eas-

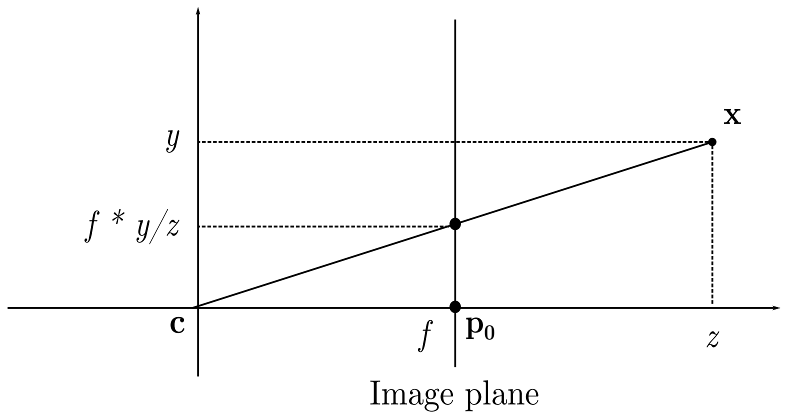


Figure 2.6.: Basic camera geometry. **c** is the camera centre, f is the focal length and **p** is the principal point. The 2D coordinates on the image plane of 3D point **x** are found at the intersection between the ray $\mathbf{x} - \mathbf{c}$ and the focal plane f.

ily chained. For example, if we have the pose of two cameras, $\mathbf{T}_{wa}$ and $\mathbf{T}_{wb}$, relative to a common world frame w, we may transform a point seen from the first one to the second passing through the common reference frame:

$$\dot{\mathbf{x}}_a = \mathbf{T}_{wa}^{-1}\mathbf{T}_{wb}\dot{\mathbf{x}}_b$$

Finally, we define an operator π which performs de-homogenisation:

$$\pi(\dot{\mathbf{x}}) = \frac{1}{x_n}\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad \text{where} \quad \dot{\mathbf{x}} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

2.3.2. Camera geometry

We assume a basic pinhole camera model where 3D world points are imaged on the camera focal plane via central projection. More formally, given a 3D point **x**, its 2D coordinates p are obtained by finding the intersection between the line that passes through the point and the centre of projection with the image plane, as shown in Figure 2.6.

This is expressed by the mapping:

$$\mathbf{p} \equiv \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} fx/z + p_{0x} \\ fy/z + p_{0y} \end{bmatrix} = f\pi(\mathbf{x}) + \mathbf{p}_0$$

where $\mathbf{p}_0$ is the principal point on the image plane. Equivalently, this can be expressed in homogeneous coordinates as:

$$\dot{\mathbf{p}} = \begin{bmatrix} f & 0 & p_{0x} \\ 0 & f & p_{0y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.2}$$

The matrix

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_{0x} \\ 0 & f & p_{0y} \\ 0 & 0 & 1 \end{bmatrix}$$

is known as the *camera calibration matrix*.

Given a 3D point in the world reference frame, its projected camera coordinates are given by:

$$\mathbf{p}_c = \pi\left(\mathbf{K}\mathbf{T}_{cw}\dot{\mathbf{x}}_w\right)$$

Conversely, the *back-projection* of an image point to 3D coordinates is obtained as:

$$\mathbf{x}_w = \mathbf{T}_{wc}\mathbf{K}^{-1}\dot{\mathbf{p}}_c$$

2.3.3. Numerical optimisation

Given a scalar field $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$, we denote its first derivative as the vector field $\nabla f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$. This is known as the *gradient* of f and it is defined as

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \frac{\partial f(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}^T \tag{2.3}$$

The second derivatives of the field f is a function $\mathbf{H} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$, known as the *Hessian* matrix, defined as

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \dots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_n} \end{bmatrix} \tag{2.4}$$

Given a vector field $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$ its first derivative is expressed by the matrix $J \in \mathbb{R}^{m \times n}$, known as the Jacobian:

$$\nabla \mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{2.5}$$

where f_i with $i = 1 \dots m$ are the individual components of the vector valued function $\mathbf{f}$. Finally, its second derivative is given by the three-dimensional Hessian matrix $\mathbf{H} \in \mathbb{R}^{n \times m \times n}$.

We are often interested in optimising a cost function in order to estimate a set of parameters, i.e. we aim at finding the minimum of f:

$$\arg\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \tag{2.6}$$

For a general cost function f there is no guarantee that the global minimum is reachable in a finite number of steps. Hence, we seek to find a local minimum around an initial parameter estimate $\mathbf{x}^0$ and iteratively refine it in its local neighbourhood, that is we try to find a displacement $\mathbf{x}^{k+1} = \mathbf{x}^k + \delta\mathbf{x}$ that reduces the cost function.

2.3.4. Second-order methods

A quadratic local approximation of non-linear cost function f can be obtained using its second-order Taylor expansion:

$$f(\mathbf{x} + \delta\mathbf{x}) \approx f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)^T \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{x}^T \mathbf{H} \delta\mathbf{x} \tag{2.7}$$

Setting the derivative with respect to the increment $\delta\mathbf{x}$ equal to zero we obtain:

$$\begin{aligned} \nabla f(\mathbf{x}^k) + \mathbf{H}\delta\mathbf{x} &= \mathbf{0} \\ \delta\mathbf{x} &= -\mathbf{H}^{-1}\nabla f(\mathbf{x}^k) \end{aligned} \tag{2.8}$$

By iteratively solving Equation 2.8 and applying the computed increment

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{H}^{-1}\nabla f(\mathbf{x}^k) \tag{2.9}$$

we perform what is known as the *Newton method*.

2.3.5. Gauss-Newton method

Let us consider a case in which the cost function is a summation of quadratic error terms, in the following form:

$$E(\mathbf{x}) = \frac{1}{2}\sum_i w_i(\mathbf{z}_i - f_i(\mathbf{x}))^2 \tag{2.10}$$

where $\mathbf{z}_i$ is the i-th *observation* and $f_i(\mathbf{x})$ is a prediction model. Denoting as $\mathbf{r}_i(\mathbf{x}) = \mathbf{z}_i - f_i(\mathbf{x})$ the i-th residual and stacking each contribution in a vector, Equation 2.10 can be rewritten in matrix form:

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{W}\mathbf{r}(\mathbf{x}) \tag{2.11}$$

Differentiating with respect to the parameter set gives the gradient and Hessian matrix in terms of the Jacobian and Hessian matrix of the prediction model:

$$\begin{aligned} \nabla E(\mathbf{x}) &= \mathbf{J}_\mathbf{r}(\mathbf{x})^T\mathbf{W}\mathbf{r}(\mathbf{x}) \\ \mathbf{H}_E &= \mathbf{J}_\mathbf{r}(\mathbf{x})^T\mathbf{W}\mathbf{J}_\mathbf{r}(\mathbf{x}) + \mathbf{H}_\mathbf{r}(\mathbf{x})\mathbf{W}\mathbf{r}(\mathbf{x}) \end{aligned} \tag{2.12}$$

The Hessian tensor $\mathbf{H}_\mathbf{r}$ of the prediction model may be too complex and costly to implement. However, if the prediction error is small or the local model is nearly linear, it is likely that $\mathbf{H}_\mathbf{r}$ will be small compared to the first order term in $\mathbf{H}_E$. Dropping it gives the *Gauss-Newton* approximation:

$$\mathbf{H}_E \approx \mathbf{J}_\mathbf{r}(\mathbf{x})^T\mathbf{W}\mathbf{J}_\mathbf{r}(\mathbf{x}) \tag{2.13}$$

Plugging Equations 2.12 and 2.13 in Equation 2.8 leads to the *Gauss-Newton* or *normal* equations:

$$\left(\mathbf{J}_{\mathbf{r}}(\mathbf{x})^T \mathbf{W} \mathbf{J}_{\mathbf{r}}(\mathbf{x})\right) \delta\mathbf{x} = -\mathbf{J}_{\mathbf{r}}(\mathbf{x})^T \mathbf{W} \mathbf{r}(\mathbf{x}) \tag{2.14}$$

The incremental update $\delta\mathbf{x}$ can then be obtained by solving the above linear system.

2.4. Summary

In this chapter we have discussed the most closely related literature to the research presented in this thesis. While more detailed bibliographical remarks will be made in each chapter when appropriate, our goal is to give the reader a thorough overview of the research field. In the remainder of the chapter we have introduced the fundamental notations and mathematical tools in order to make this document as self-contained as possible. We have given a brief overview to the numerical optimisation techniques used to solve non-linear least-square problems. This is motivated by the fact that such techniques are used and referenced in the reconstruction pipelines described in this thesis.

Chapter 3

supereight: a high performance template octree library for spatial mapping

In this chapter we introduce our template octree library, named `supereight`, and the fundamental algorithms behind our dense volumetric mapping system. We will show quantitative and qualitative results of a KinectFusion implementation using our octree volume and a thorough comparison with the current state-of-the-art dense SLAM systems.

3.1. Motivation

Volumetric SLAM systems such as KinectFusion (Newcombe et al. [2011a]), represent the mapped environment as an implicit projective *truncated signed-distance function* (TSDF), discretised in a regular voxel grid. However, this representation induces considerable overheads in both space and time, as the number of voxels to be stored and processed grows cubically with either the resolution or the space covered. It is well understood that in order to mitigate these penalties, the inherent sparsity of the world can be exploited to significantly reduce the amount of information. On top of this, the continuous TSDF itself exposes further compression opportunities. Figure 3.1 shows a typical TSDF curve in one dimension. In red is represented the line of sight from the camera to the actual surface measurement, while in blue is represented the corresponding TSDF

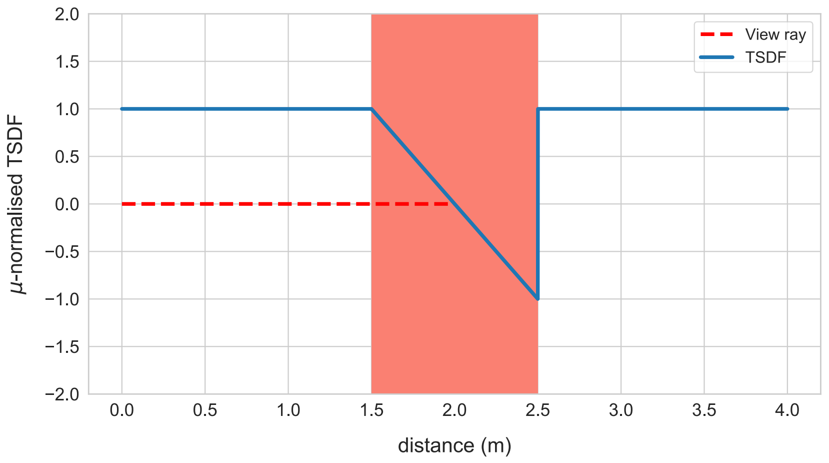


Figure 3.1.: One dimensional truncated signed-distance function (TSDF) along the line of sight (in red) from the camera centre.

curve for the given measurement. The salmon shaded area indicates the truncation region, with bandwidth 2μ. As the signal is constant outside the truncation region, it can be efficiently compressed or represented implicitly in a sparse data structure.

Different works have successfully exploited such a property and demonstrated significant speed-ups. In the literature we can distinguish two predominant approaches: hierarchical data structures, such as octrees (Zeng et al. [2013]) or N^3 trees (Chen et al. [2013]), and flat hash-tables (Nießner et al. [2013]). Although enormous speed-ups have been demonstrated with voxel hashing, the performance of tree based data structures have not been very satisfactory. However, hierarchical data structures may be desirable in a variety of scenarios as they provide a full, explicit spatial indexing of a scene. Furthermore, information can be naturally stored at different levels of detail and consequently it is possible to easily compress uniform regions of the map. Publicly available hierarchical frameworks fall either short in terms of performance or in terms of flexibility, being tied to a particular continuous map representation. With this work, we aim to provide the wider community with a high performance octree library suitable for real-time operations on a CPU, which can be easily extended to support different data-types without sacrificing performance and allow to fast and efficient prototyping. We achieve this by: i) providing a set of core data-type independent algorithms which provide the most common operations required by typical dense SLAM systems; ii) exposing a template based interface to interact with the core container for user-defined data-types.

Summarising, the contributions that we present in this chapter are:

- A set of algorithms for mapping and interpolation on structured sparse grids;

- An open-source octree library for real-time CPU-based dense simultaneous localisation and mapping;

- An implementation of a TSDF-based SLAM pipeline using our octree container and a thorough evaluation against the current state-of-the-art.

3.2. Concepts

Supereight is an octree template library specifically targeting dense volumetric simultaneous localisation and mapping. In the supereight model, we assume that the underlying data stored in the container is a three-dimensional, differentiable *scalar field*. The scalar field itself may be comprised of more than one *channel*, which can be logically thought of as having multiple fields stacked on top of each other. As described in Section 3.4, the unit of information is a *voxel*, i.e. a unit cube in integer coordinate space. Each voxel is associated with a field value which covers its whole volume. This is equivalent to a piece-wise constant representation of the encoded space. Higher order approximations may be recovered via trilinear interpolation, while gradients and higher order derivatives can be found via finite difference.

While its interface is quite generic and in principle it can handle any data type, some functionalities are restricted to types having certain characteristics. In particular, the scalar field assumption may be relaxed for other data types, such as signed or unsigned integers as long as they can be trivially converted to real data-types, as required by the interpolation and gradient functions.

3.3. Type interface

In order to provide support for arbitrary data-types, supereight provides a flexible type traits interface, as shown in Listing 3.1. There are three main elements to be defined:

- `value_type` specifies the data actually stored in the octree container. Notice that this could be different from the type for which the trait class is being specialised. In this way, the user can decouple the *external* data-type (`my_field_t` in this case) from its internal representation. This can be particularly useful if the target application can afford computation at reduced precision, for instance storing real values as 16bits integer in memory.

- the static function `value_type init_value()` specifies the default voxel value.

- the static function `value_type empty()` specifies what value should be returned in case of missing data from the container. This is used in spatial queries to signal absence of data. If information is not useful in the target application it can be simply set equal to `init_value`.

```
typedef struct {
  float x;
  float y;
} my_field_t;

template<>
struct se::voxel_traits<my_field_t> {
  typedef my_field_t value_type;
  static inline value_type init_val(){ return {1.f, 0.f}; }
  static inline value_type empty(){ return {1.f, -1.f}; }
};
```

Listing 3.1: Type traits specialisation for user-define data types.

3.4. Octree representation

An octree is a recursive tree data structure in which each node, or *octant*, of the tree has eight children. Typically, the root of the tree is associated

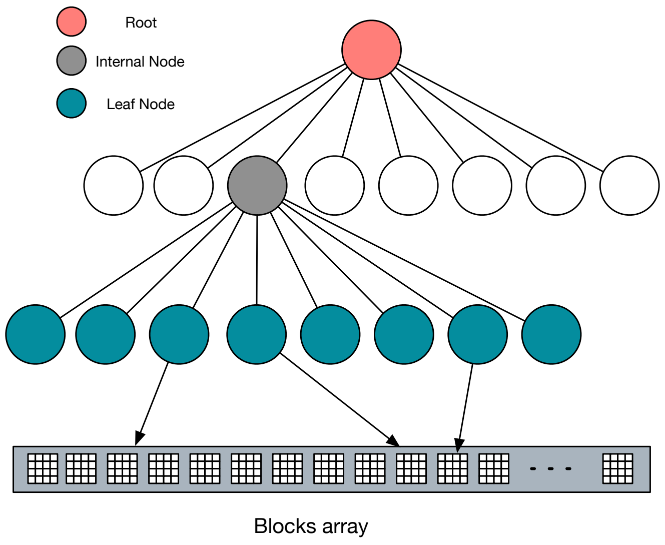


Figure 3.2.: Tree-based octree representation

with a cuboid of a certain extent. Recursively, each child defines a smaller cuboid of half the edge size of the parent till a maximum tree depth is reached. The regular decomposition scheme implies that descending of one level in the hierarchy doubles the resolution in each side.

Figure 3.2 displays a graphical representation of our tree data structure. Similarly to Nießner et al. [2013] and Kähler et al. [2015] we aggregate voxels at the finest resolution into aggregated contiguous blocks of parametric size, by default 8^3 voxels. This is in contrast to previous work on octrees by Zeng et al. [2013], where the deepest level stores individual voxels. In this perspective, the map simply becomes a collection of unordered sparsely allocated voxel blocks and the tree a spatial index of the scene that allows the correct piece of data to be retrieved given its integer coordinates.

Listing 3.2 shows the structure of the node and aggregated voxel block structure. For each node we keep an explicit list of pointers to its children plus with a corresponding bitmask, which we can be used to perform fast queries on the number of active children. Each node is uniquely identified by its linearised coordinates, stored in a single 64bit unsigned integer which we will discuss in the next section. For a voxel block instead, we also keep in memory its 3D coordinates, as the extra memory required

```
template <typename T>
class Node {
  key_t code_; // octant code
  unsigned char children_mask_; // children validity mask
  Node* children_[8]; // child pointers array
};

template <typename T>
class VoxelBlock : Node<T> {
  static constexpr unsigned side = 8;
  Vec3i coordinates; // 3D lower left corner coordinates
  T block_data_[side*side*side]; // aggregated voxel data
};
```

Listing 3.2: Octree node structure

is negligible and they are frequently used in their de-linearised form to compute the corresponding entry in the `block_data_` array given a point belonging to the block. Notice that while in this work we have opted for a pointer-based structure, there are several viable alternatives, such as pointer-less representations, as shown in Burstedde et al. [2011] and Thomas et al. [2010]. Usually such schemes are implemented via hashing, which makes individual octant queries more efficient, while requiring a more complex strategy to manage the hash-table, especially in concurrent applications. These approaches are logically equivalent and could be easily adopted in our framework with no impact on the user code.

3.4.1. Information access

Each point in an n-dimensional grid can be uniquely identified via their linearised coordinates. This is achieved via a bijective mapping from the multidimensional to the linear domain, known in the literature as space-filling curves, see Bader [2012] for an extensive treatment of the subject. In this work, we employ the Z-order curve, also known as Morton [1966] order, which is defined as it follows.

Definition 1. Given an n-dimensional vector $(x_0, x_1, \dots, x_n) \in \mathbb{N}^n$, where $x_i \in [0, 2^b]$, a Morton number is constructed by interleaving the individual bits from each component in a single number

$$m = \left[x_n^b x_{n-1}^b \dots x_0^b, \cdots, x_n^0 x_{n-1}^0 \dots x_0^0\right] \in \left[0, 2^{n*b}\right]$$

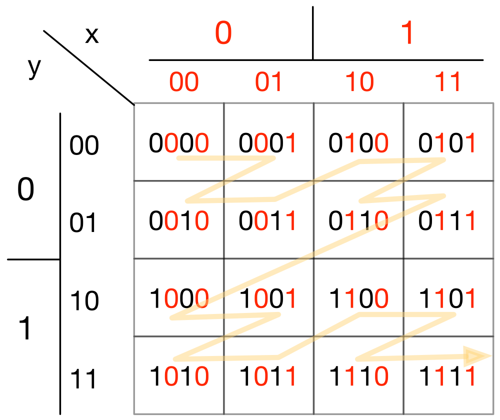


Figure 3.3.: Morton codes and traversal ordering for a 2D grid.

where the superscript indicates the *i-th* bit from the *n-th* coordinate. To perform the interleaving/deinterleaving we use a method based on *magic numbers*[1], which we detail in Appendix A. Throughout this work we assume three-dimensional coordinate vectors unless stated otherwise. Notice, however, that the algorithms shown in the following hold for any number of dimensions.

Figure 3.3 shows an illustrative example on a two-dimensional four-by-four grid. As we can see, interleaved bits from the x and y coordinate form a unique code for each cell. A crucial property of these numbers is that they not only uniquely identify voxels in a regular grid, but that the higher bits recursively represent the location of an ancestor cell in a coarser grid, effectively specifying a full traversal of the correspondent tree and implicitly defining its structure. As an example, if we consider cell $(x, y) = (2, 1)$ with its associated code 0110, starting from the root we would first descend to the top-right sub-grid (code 01) and then select the child with code 10, i.e. our target pixel with code 0110.

Given the properties discussed above, several operations on octants' coordinates can be performed directly on their linear representations. Algorithms 1 and 2 show key operations that can be performed directly on the Morton codes and that are extensively used in the library. Algorithm 1 computes the parent code of a given octant via a bitwise AND opera-

[1]See https://graphics.stanford.edu/~seander/bithacks.html for a list of alternative algorithms.

ALGORITHM 1: Compute parent's code given an octant

```
Global: prefix_mask[MAX_BITS] // precomputed array of prefixes
Function parent(morton_t code, int level) : morton_t code
{
    parent_code ← code & prefix_mask[level-1];
    return parent_code;
}
```

ALGORITHM 2: Compute the child id of a given octant

```
Function child_id(morton_t octant, int level) : int id
{
    int shift ← max_depth - level;
    octant ← octant >> shift*3;
    int idx = (octant & 0x01) | (octant & 0x02) | (octant & 0x04);
    return idx;
}
```

tion between the octant code and the corresponding upper level bitmask. Each level of the tree is associated with a bitmask in which all the most significant bits till the corresponding bit depth are set to one, while the remainder of the code is set to zero. Intuitively, in terms of integer coordinates this is equivalent to rounding a given point to its enclosing octant in the corresponding coarser grid. The pre-computed prefix mask for 64bit keys is given in Appendix A. Similarly, Algorithm 2 computes the octant position amongst its siblings by extracting the lower 3 bits of an octant code at a desired level.

3.4.2. Voxel block allocation

Our library targets real-time mapping applications, hence it assumes a continuously growing mapped space. This implies that the allocation of new voxel blocks in the hierarchy must be performed extremely fast and with the lowest overhead possible. Parallel tree allocation strategies have been widely explored in the computer graphics domain as hierarchical data structures are common accelerators for ray-tracing and collision detection algorithms (Laine and Karras [2010], Garanzha et al. [2011]). To maximise parallelism in the tree construction, we adopt a technique based on Morton numbers inspired by Garanzha et al. [2011] and Bédorf et al. [2012].

ALGORITHM 3: Top-bottom voxel list allocation

```
Function allocate(morton_t key_list[N] : int allocated
{
    key_list ← sort (key_list);
    unique_keys [M] ← sort (key_list);
    unique_keys ← compact (unique_keys);
    for level ← root.level to aggregate_level do
        unique_keys ← [ k & prefix_mask [level] for k in unique_keys ];
        unique_keys ← compact (unique_keys);
        parallel_alloc (unique_keys, level);
    end for
}
```

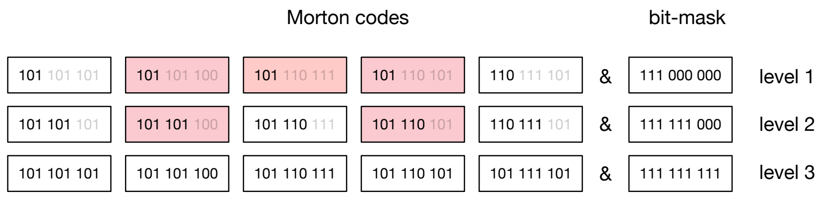


Figure 3.4.: Bit-masking the key-set at each allocation level, coloured boxes indicate duplicate codes.

We use a breadth-first top-to-bottom allocation which takes full advantage of this numbering property. Algorithm 3 describes the main algorithm. As input, it expects a list of Morton numbers, produced in a pre-processing step. The list of keys is then sorted and possible duplicates are removed via a compaction operation. For each level in the tree, we filter the key list by masking each code with the appropriate prefix mask for the current level. The bit-mask for a given level sets the lower bits corresponding to finer sub-grids to zero. This procedure will generate duplicate keys which are then filtered out, as illustrated in Figure 3.4. This, together with the fact that by construction the structure to reach a given node would have been allocated at a previous step, allows us to allocate all the nodes in parallel without requiring any synchronisation between threads. This technique still requires a lock-step execution from one level to the next, however we found its performance satisfactory. More complex algorithms that avoid the synchronisation step are found in the literature, e.g. Karras [2012], if faster allocation is needed. Notice that, in a typical reconstruction scenario, after a transient initial phase the number

```
struct update_valid
{
  template <typename DataHandlerT>
  void operator()(DataHandlerT& handler,
    const Eigen::Vector3i  vox&,
    const Eigen::Vector3f& pos,
    const Eigen::Vector2f& pixel)
    {
      // if depth sample is valid
      if(depth[pixel.x + depthSize.x * pixel.y] > 0.f)
      {
        int val = handler.get();  // get the current value
        handler.set(val++); // set the new value
      }
  }

  update_valid(const float * d, const Eigen::Vector2i framesize) :
    depth(d), depthSize(framesize) { };

  const float * depth;
  Eigen::Vector2i depthSize;
};
```

Listing 3.3: Defining a function object which performs an update operation on a single voxel.

```
void mark_as_seen(OctreeInt& map,
  float* depth,                 // depth frame
  const Eigen::Vector2i& imageSize,   // frame size
  const Eigen::Matrix4f& K,           // pinhole camera matrix
  const Eigen::Matrix4f& Tcw)            // world-to-camera SE3
      transform
{
  struct update_valid update_funct(depth, imageSize);
  se::projective_map(map, Tcw, K, imageSize, update_funct);
}
```

Listing 3.4: Applying the function defined in Listing 3.3 to an se::Octree instance using the se::projective_map functor .

of voxel blocks to be allocated per frame decreases considerably as new blocks will most likely be required at the frame border or in previously occluded regions.

3.4.3. Field update

One of the crucial operations in reconstruction pipelines is that of map update. This is usually performed by associating world data, i.e. a voxel

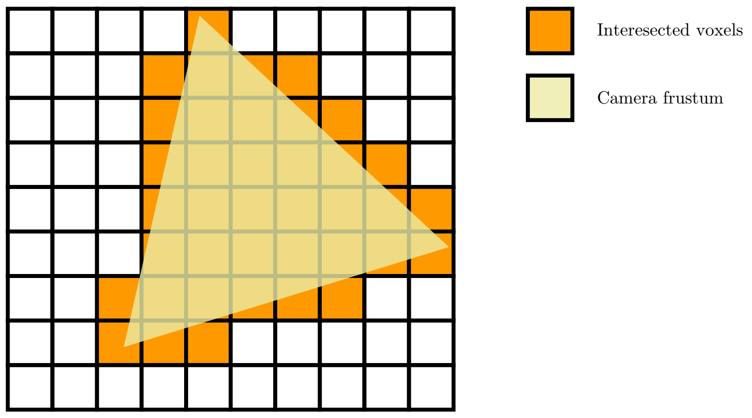


Figure 3.5.: View frustum - voxels intersection.

in global coordinates to a corresponding sensor measurement in sensor space, e.g. current depth estimate or colour information. There are two main approaches to achieve this association: *ray-casting*, as used in OctoMap (Hornung et al. [2013]) and *projection*, as used in KinectFusion (Newcombe et al. [2011a]) and Bylow et al. [2013]. In the *ray-casting* approach, for each valid range measurement a ray is marched from the camera centre through the corresponding pixel. Each voxel intersected along the ray is updated accordingly. Notice that this implies that a given voxel may be updated more than once per frame, as it might be intersected by different rays. In the *projection* method instead, voxels are projected back onto the camera frame and the corresponding set of measurements is used for the update. In practice, this is often approximated via nearest neighbour interpolation, effectively obtaining a one-to-one mapping between voxels and sensor measurements. We refer the reader to Klingensmith et al. [2015] for an interesting comparison between these two techniques.

Our library supports high level constructs to easily perform projective updates. We provide a functor, named `projective_map`, which applies a user-defined update function to all the voxels which fall inside the camera frustrum. Listings 3.3 and 3.4 provide an example of such a function. The user-defined function should expect:

1. `DataHandler`: opaque object to access the underlying voxel data. Whilst we could have simply provided a reference to the actual data, we chose to keep the data-handling opaque in such a way

to maximise flexibility in the way data is managed in the back-end. Although not currently implemented, this would allow us to experiment with different data layout storages with minimal, if any, impact on the user code.

2. $\mathbf{x}_w \in \mathbb{N}^3$: voxel coordinates, in world integer space.

3. $\mathbf{p}_c \in \mathbb{R}^3$: transformed voxel coordinates to camera reference frame, in metric units. This is computed by transforming the voxel into camera coordinates and scaling it according to the actual voxel resolution.

4. $\mathbf{u}_c \in \mathbb{R}^2$: corresponding 2D pixel location in camera frame. We provide the floating point coordinates in such a way that the user may then decide what kind of interpolation use to sample the data.

The `projective_map` functor instantiation requires the following data:

1. `Octree<T>& map`: a mutable reference to an object of the Octree template class, target of the transformation.

2. $\mathbf{T}_{cw} \in \mathbb{SE}_3$: world-to-camera coordinate transform.

3. $\mathbf{K} \in \mathbb{R}^{4\times 4}$: homogeneous pinhole camera matrix.

4. $\mathbf{\Omega} \in \mathbb{N}^2$: camera frame size.

5. `UnaryOp op`: function object which satisfies the signature specified in the above.

Notice that in our implementation we use the Eigen template library (Guennebaud et al. [2010]) for matrix and vector algebra.

The use of generic functors brings many advantages. First, it provides a flexible and compact way to express how the map should be updated. Crucially, compile-time resolution of function calls via function objects results into fully in-lined code, hence performance are preserved. Second, and perhaps more importantly, it allow us to abstract away computational details of the geometric transformations involved in the map projection update.

Volume projection to camera frame can be understood as:

$$\dot{\mathbf{p}}(\mathbf{v}) = \mathbf{K}\mathbf{T}_{cw}\alpha\mathbf{v} \quad \forall \mathbf{v} \in \mathbf{V} \tag{3.1}$$

where $\mathbf{V} \subseteq \mathbb{Z}^n$ is the set of all allocated voxels, $\mathbf{T_{cw}}$ is the $\mathbb{SE}3$ world-to-camera transform, $\mathbf{K}$ is the camera matrix and α is a scalar factor to map integer voxels to the actual world scale. In general, since voxel updates are independent of each other, the grid traversal can be performed in any order. Usually, most efficient traversal strategies perform a scan-line iteration over the volume along an axis aligned direction, which can be expressed as:

$$\dot{\mathbf{p}}(\mathbf{v}) = \mathbf{K}\mathbf{T}_{cw}\alpha \cdot (\mathbf{b} + i \cdot \mathbf{dir}) \quad \forall i \in 0 \ldots n-1 \quad \wedge \quad \mathbf{b} + i \cdot \mathbf{dir} \in \mathbf{V} \tag{3.2}$$

where $\mathbf{dir}$ is an axis aligned vector (e.g. $\mathbf{dir} = [1, 0, 0]^T$) which indicates the direction of iteration, $\mathbf{b}$ is the base voxel and i represents unit stride increments without loss of generality. Given that $\mathbf{b}$ and $\mathbf{dir}$ remain constant between iterations and given the linearity of the transform operations, they can be precomputed:

$$\begin{aligned} &\mathbf{P} = \mathbf{K}\mathbf{T}_{cw}\alpha \\ &\dot{\mathbf{p}}(\mathbf{v}) = \mathbf{P}\mathbf{b} + i \cdot \mathbf{P} \cdot \mathbf{dir} \quad \forall i \in 0 \ldots n-1 \quad \wedge \quad \mathbf{b} + i \cdot \mathbf{dir} \in \mathbf{V} \end{aligned} \tag{3.3}$$

Consequently, the final voxel coordinates can be obtained with just one scalar multiplication and one vector addiction. Furthermore, decoupling the geometry computation from the actual data update operation allow us to easily change iteration strategy without impacting user code.

3.4.4. Field interpolation

Iso-surface extraction algorithms, such as ray-casting and marching cubes, rely heavily on repeated field sampling, hence it is crucial to have efficient ways of querying the underlying representation. To this purpose, our framework provides optimised nearest neighbour and tri-linear interpolation functions. In line with the framework philosophy, we provide a simple template interface to allow the user to specify which field should be interpolated, as shown in Listing 3.5. The `interp` function expects a

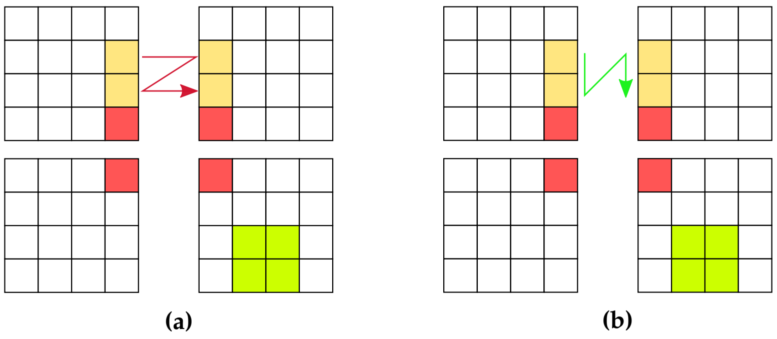


Figure 3.6.: Different access patterns for a 4-points square stencil on a 2D blocked grid. Green shaded pixels indicate the optimal case, where all the accessed data is local to a block. Red denotes the worst case, where all the pixels belong to different blocks. In 5.6a it is shown the sub-optimal gathering order for the remaining case where the base pixel lies on a block edge (in yellow). Accessing the data in row-major order (red arrow) implies jumping from one block to the other. In 3.6b instead is depicted the optimal access pattern for this case (green arrow), where block-local pixels are accessed first.

sampling location and a function object which specifies which on which channel to operate. Such function should take a reference to a field element as input and return the interpolation target channel, as done by the lambda `get_y` in the example snippet.

```
// Data type: struct Foo { float x; float y;  }
// se::Octree<Foo> tree;

Eigen::Vector3f pos(110.2f, 54.5f, 65.67f);
auto get_y = [](const Foo& val) { return val.y; }:
float res = tree.interp(pos, get_y);
```

Listing 3.5: interp calling procedure.

Tri-linear interpolations require the eight discrete voxels surrounding the sampling points to be gathered. In a sparsely allocated grid this could be expensive as several tree traversals are required to gather the desired points. We limit such performance penalty by observing that there is a finite number of access patterns that can occur. Figure 3.6 shows a graphical representation of the possible cases, on a two-dimensional grid for the sake of clarity. First, if the point to be interpolated falls in the middle of a voxel block, then all eight points will be local to that block and hence only one tree traversal is required. This scenario is indicated in green in

Figure 3.6. The other extreme case is when the point falls exactly on the corner of a voxel block, denoted in red, in which case eight tree traversals will be needed. There are six remaining configurations which correspond to the case in which the sampling point is on a voxel block edge along one or two dimension. In this case the query order is particularly important as we should ensure as much locality as possible, since a bad ordering might imply more tree traversals than actually needed. The pattern access shown in Figure 5.6a, denoted by the red arrow, requires three times more traversals compared to the optimal shown in Figure 3.6b, denoted by the green arrow.

In Kähler et al. [2015] this issue is addressed by caching the last-accessed block, but this still does not help if the gathering order jumps from one block to another invalidating the cached block. Instead, we pre-compute statically a traversal order for all the possible configurations and at run-time we simply select the optimal order for the requested sampling point position. This can be achieved by simple integer arithmetic as follows. For a given base point, we need to estimate if it lies inside, at a corner or along the edge of its containing voxel block. As shown below, this can be done by taking each individual coordinate, computing its relative position and checking if it lies at the boundary:

```
unsigned dx = (pos.x() % block_size) ==  (block_size - 1);
unsigned dy = (pos.y() % block_size) ==  (block_size - 1);
unsigned dz = (pos.z() % block_size) ==  (block_size - 1);
unsigned mask = (dx << 2) | (dy << 1) | (dz);
```

We then combine the result of each individual coordinate into a single `mask` which can be used to select the appropriate order traversal given the base point. Notice that when `mask == 0` then all the required points will be local to the block. When `mask == 7` all the points belong to different blocks. Finally, the remaining 6 cases indicate the different combinations of boundary violations:
For each case, we then alternate block fetching and local point gathering, as exemplified below in Listing 3.6, where `gather_points` is the function which populates `N` entries in the `points` array.

The method described above guarantees the optimal reuse of tree traversals without resorting to any complicated caching strategy. Furthermore

```
   mask ==  1; /* z    crosses */
   mask ==  2; /* y    crosses */
   mask ==  3; /* y, z cross   */
   mask ==  4; /* x    crosses */
   mask ==  5; /* x, z cross   */
   mask ==  6; /* x, y cross   */
```

```
Scalar points[8];
...
case 1:

  VoxelBlock<T>* block  = tree.fetch(pos);
  gather_points<4>(block, pos, points);

  // Cross the z boundary
  block  = tree.fetch(pos + Eigen::Vector3i(0, 0, 1));
  gather_points<4>(block, pos, points + 4);
  break;
...
```

Listing 3.6: Point gathering example.

it eliminates unnecessary run-time control flow which a caching strategy implies, further simplifying the computation. Notice also that our technique is quite general and can be applied to any sparse data structure with fixed-size block decomposition, such as hash tables or N^3 trees.

3.5. SDF tracking and mapping

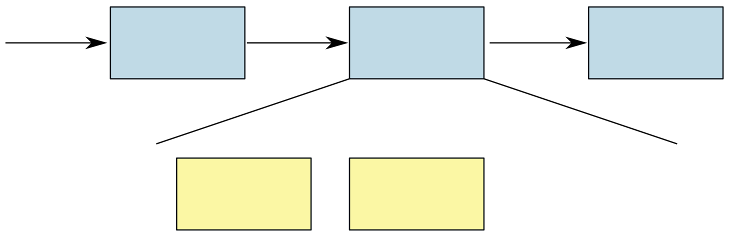


Figure 3.7.: KinectFusion algorithmic pipeline.

In this section we describe our implementation of the KinectFusion algorithm (Newcombe et al. [2011a]) with supereight as the back-end container. A high level view of the algorithm is shown in pipeline structure is shown in Figure 3.7, where each stage performs the following operations:

Preprocessing : depth data coming from the sensor is transformed into its corresponding vertex and normal map.

Tracking : camera ego-motion is estimated against a rendered view of the model synthesised at a previous time.

Integration : given the estimated camera pose, the depth data is integrated into the volumetric map.

Raycasting : render a synthetic depth and normal map to be used for tracking at the next frame.

3.5.1. Preprocessing

At each time-step t a new depth map is acquired from the sensor. Optionally, it can be de-noised with an appropriate filter. In our case we use bilateral filtering (Tomasi and Manduchi [1998]), as per Equation 3.4:

$$\begin{aligned}
\mathbf{D}_t(i,j) &= \frac{\sum_{u,v} \mathbf{D}_t(u,v) w(i,j,u,v)}{\sum_{u,v} w(i,j,u,v)} \\
w(i,j,u,v) &= \exp(w_p(i,j,u,v) + w_i(i,j,u,v)) \\
w_p(i,j,u,v) &= -\frac{(i-u)^2 + (j-v)^2}{2\sigma_p^2} \\
w_i(i,j,u,v) &= -\frac{\|\mathbf{D}_t(i,j) - \mathbf{D}_t(u,v)\|^2}{2\sigma_i^2}
\end{aligned} \tag{3.4}$$

where $\mathbf{D}_t$ is a depth frame at time-step t and $w(\cdot)$ is the weighting coefficient.

Once the depth image has been de-noised, vertex and normal maps are generated using back-projection and cross-product respectively:

$$\begin{aligned}
\mathbf{V}_t(x,y) &= \mathbf{D}_t(x,y)(\mathbf{K^{-1}}[x,y,1]^T) \\
\mathbf{N}_t(x,y) &= (\mathbf{V}_t(x+1,y) - \mathbf{V}_t(x-1,y)) \times (\mathbf{V}_t(x,y+1) - \mathbf{V}_t(x,y-1))
\end{aligned} \tag{3.5}$$

3.5.2. Tracking

Camera egomotion is estimated via a variant of the well known *iterative closest point* (ICP) algorithm (Besl and McKay [1992]). Other approaches have been proposed in the literature, notably Canelhas et al. [2013], Bylow et al. [2013] and Slavcheva et al. [2016], where the camera position is tracked directly against the volumetric *signed-distance function*. Even if such techniques show levels of accuracy on par with point-based alignment methods, in this work we opted for a more traditional ICP alignment as it relies only on point-clouds and can be easily reused between different map representations. Hence we minimise the following energy function via Gauss-Newton:

$$E_t = \sum_{\mathbf{u} \in \Omega_t} \| (\mathbf{T}_t \mathbf{V}_t(\mathbf{u}) - {}_W\hat{\mathbf{V}}_{t-1}(\hat{\mathbf{u}}))^T {}_W\hat{\mathbf{N}}_{t-1}(\hat{\mathbf{u}}) \|_2^2 \tag{3.6}$$

where Ω_k is the set of all pixels in image frame. $\hat{\mathbf{u}}$ is the corresponding pixel in the reference depth frame, obtained via projective data association:

$$\hat{\mathbf{u}} = \pi(\mathbf{T}_{t-1}^{-1} \mathbf{T}_t \mathbf{V}_t(\mathbf{u})) \tag{3.7}$$

$$\text{if} \quad \| \mathbf{V}_t(\mathbf{u}) - \hat{\mathbf{V}}_{t-1}(\hat{\mathbf{u}}) \|_2 < \phi_{dist} \tag{3.8}$$

$$\text{and} \quad \mathbf{N}_t^T(\mathbf{u}) \hat{\mathbf{N}}_{t-1}(\hat{\mathbf{u}}) < \phi_{normal} \tag{3.9}$$

where ${}_W\hat{\mathbf{V}}_{k-1}$ and ${}_W\hat{\mathbf{N}}_{k-1}$ are the vertex and normal maps represented in the world frame rendered from the previous camera pose. Equations 3.8 and 3.9 state that a data association is successful only if the corresponding vertex are sufficiently close in 3D space and the normals difference is below a certain threshold. The overall minimisation problem is solved using an iterative, coarse-to-fine scheme using three pyramid levels.

3.5.3. Integration

As briefly outlined in Section 3.1, KinectFusion represents the mapped space in a *truncated signed-distance function* (TSDF) which is discretised into a regular grid, as shown in Figure 3.8. In a TSDF, points that fall outside the truncation region do not carry any meaningful information and consequently do not need to be explicitly represented. Hence, we modify

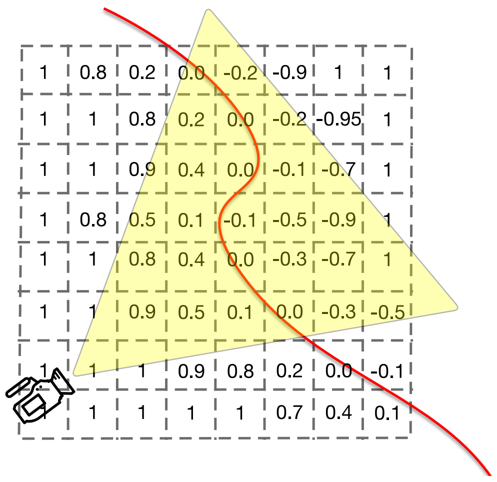


Figure 3.8.: Truncated signed-distance function encoded in a two-dimensional grid. Values different from one are assigned to pixels inside the truncation region. Positive values correspond to observed space in in front of the surface, while negative values indicate space behind the surface.

the standard KinectFusion integration pipeline to explicitly allocate space for newly seen regions. In particular, we break the integration stage into three sub tasks, as depicted in Figure 3.7: i) from the current depth frame $\mathbf{D}_t$ and the current map estimate $\mathbf{M}_t$ we infer which blocks need to be allocated; ii) once the list is constructed, we allocate the required blocks using supereight's allocation routines detailed in 3.4.2; iii) finally, we update the TSDF field using our `projective_map` operator.

3.5.3.1. Scene allocation

In the literature, different techniques have been proposed. Zeng et al. [2013] and Chen et al. [2013] sweep over the hierarchical grid projecting voxels from coarse to fine grain resolution, marking which voxels fall within truncation region of the current frame. A similar approach, proposed in Klingensmith et al. [2015], is to allocate all the blocks that fall within the camera view-frustum bounding box. However this method significantly over-allocates and requires garbage collection to deallocate voxels that fall outside the truncation region. Instead, we choose to follow the ray-casting method proposed in Nießner et al. [2013], Chen et al. [2013]. As shown in Figure 3.9, for each pixel in the image frame we ray-

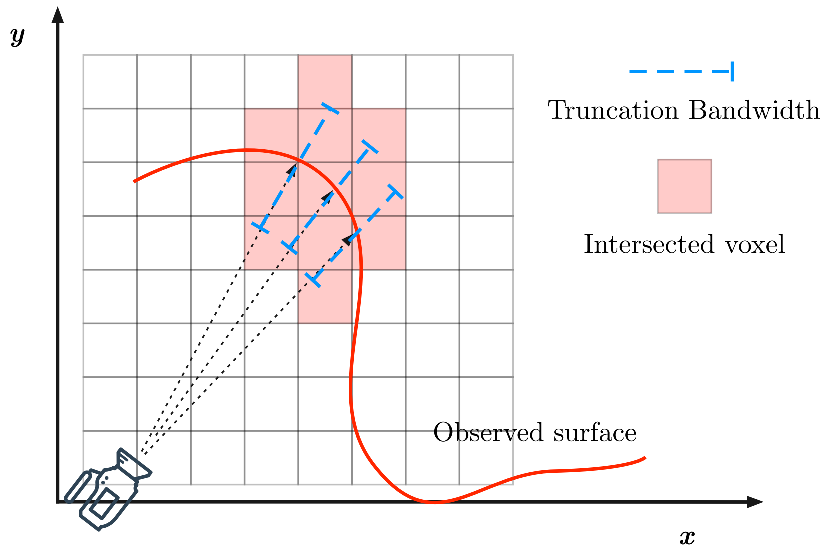


Figure 3.9.: Rays originating from the camera intersect the voxel grid. Voxels that fall close to the surface, in pink, are added to an allocation buffer.

cast along the line of sight within the user specified μ bandwidth enclosing the corresponding depth measurement and collect all the intersected voxels $[\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n]$. Mathematically:

$$\begin{aligned} &[\mathbf{x} \mid \lfloor \mathbf{T}_t \mathbf{V}_t(\mathbf{u}) + s \cdot \mathbf{d} \rfloor \quad \text{with} \quad s \in -\mu, .., +\mu] \\ &\mathbf{d} = \frac{\mathbf{R}_t \mathbf{V}_t(\mathbf{u})}{\|\mathbf{R}_t \mathbf{V}_t(\mathbf{u})\|_2} \end{aligned} \tag{3.10}$$

where $\mathbf{R}_t$ is the rotational component of $\mathbf{T}$. Furthermore, we discard the ones that are already allocated in the map. We build this list in parallel, where each thread writes in a shared buffer via atomically protected writes.

3.5.3.2. Depth fusion

Once the new parts of the scene have been allocated, the measurement integration is done in the same fashion as in Newcombe et al. [2011a]. Each voxel at position $\mathbf{x}$ is projected into the current depth image $\mathbf{D}_t$ with known pose $\mathbf{T}_t$ and its TSDF value $f_t \equiv f_t(\mathbf{x})$ from the corresponding depth measurement is computed, as

$$
\begin{aligned}
\eta(\mathbf{x}) &= \mathbf{D}_t(\pi(\mathbf{T}_t^{-1}\mathbf{x})) - x_z, \\
f_t &= \min(1, \frac{\eta(\mathbf{x})}{\mu}) \ \text{ iff } \ \eta \geq -\mu, \\
F_t &= \max\left(\min\left(\frac{y_{t-1}F_{t-1} + f_t}{y_{t-1}+1}, 1\right), -1\right), \\
y_t &= \min(y_{\max}, y_{t-1}+1).
\end{aligned} \tag{3.11}
$$

The computed TSDF sample f_t is then integrated in the global TSDF F_t by means of block averaging, where $y_t \equiv y_t(\mathbf{x})$ denotes the weight associated with voxel $\mathbf{x}$.

It should be noted that, as shown in the original work of Curless and Levoy [1996], assuming statistical independence between depth measurements and a Gaussian noise distribution, the above weighted average corresponds to a maximum-likelihood least square estimate of the true TSDF. While here we use a simple unitary weight for the new samples, other possibilities exists. Nguyen et al. [2012] empirically derive a noise model for the Kinect sensor and use it to weight SDF values proportionally to the inverse quadratic square depth of the measured surface. Oleynikova et al. [2017] adopt a similar weighting but with a sharper behind-surface drop-off to limit the influence of unobserved voxels. Notice that the update formula of Equation 3.11 implies that *all* the space between the camera and the surface boundary is updated. Intuitively, this is consistent with the fact that observing *empty* space *is* information, although not explicitly encoded in the TSDF representation. From a correctness point of view, this is required as non-null values lying between the camera and a current surface observation would be a free-space violation and hence need to be removed or smoothed out. A typical scenario in which this may occur is in presence of moving objects.

3.5.4. Surface prediction

Estimating the camera motion as described in Section 3.5.2 requires a synthetic view of the mapped scene in order to perform frame-to-model alignment. As in Newcombe et al. [2011a] a depth map is rendered via ray-casting the signed-distance field. Hierarchical data structures such as the

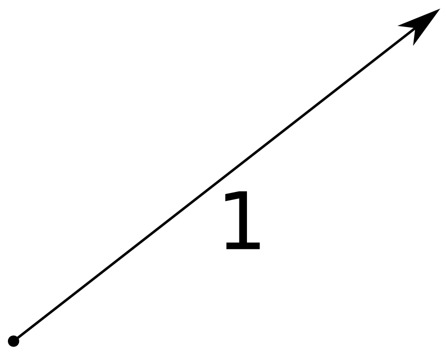


Figure 3.10.: Hierarchical ray-casting. The darker shade indicate voxels which are traversed for the example ray while the numbering correspond to the voxels traversal order.

one presented in this work are well known accelerators for a ray-casting algorithm. Implicitly the structure of the mapped space is encoded in the tree and it can be leveraged to advance each ray as close as possible to the first visible surface along the ray direction. Given the TSDF field representation discussed in the previous sections, the actual surface boundaries are found at the zero-crossings of the TSDF function. Hence, for each pixel in image frame, we cast a ray passing through it onto the volume and search for the first zero-crossing on the line of sight. We adopt a modified version of the top-down ray-casting algorithm by Laine and Karras [2010] where for a given ray, the tree is traversed depth-first, collecting all the nodes along its path. A graphical representation of the descent order is given in Figure 3.10. The darker shades indicate voxel that are traversed, while the numbering corresponds to the traversal order. We provide it as a forward iterator class:

```
se::Octree<FieldType> map(...);
Eigen::Vector3f origin(...);
Eigen::Vector3f direction(...);
float nearPlane = 0.3f;
float farPlane  = 4.f;
se::ray_iterator ray(map, origin, direction, nearPlane, farPlane);

VoxelBlock<FieldType> * block = ray_iterator.next();
if(block != nullptr) {
  ...
}
```

Listing 3.7: The ray-iterator class.

Each successive increment of the iterator position returns the next block along the ray. Given the current observed block, the zero-crossing search may proceed locally via repeated trilinear interpolation sampling. As this may not be found in the currently examined block, the search must proceed to the next. One option is to alternate hierarchical visits to the tree via the ray-iterator and local dense sampling. However, the hierarchical traversal is relatively expensive compared to the efficient tri-linear interpolation functions described in Section 3.4.4. As voxel blocks are allocated only close to the surface measurement, it is reasonable to assume that a zero-crossing can be found close to the ray entry point on the first visible block. We exploit this fact and use the ray-iterator only to *prune* the search space for a given pixel. Once the first block is retrieved, the search continue by iteratively sampling the map *without* resorting to the iterator after the first call. We have found this methodology to give the best results in term of performance.

3.6. Experimental results

In this section we will detail our experimental results. All our tests have been performed within the SLAMBench framework of Nardi et al. [2015] on a Skylake i7-6700HQ CPU at 2.60GHz, with 16GB of memory, Ubuntu 16.10 and frequency scaling disabled. The software has been compiled with GCC 5.4.1 and -O3 optimisations enabled. We compare our volumetric SLAM pipeline implementation against the current state-of-the-art InfiniTAM by Kähler et al. [2015].

3.6.1. Algorithmic configuration

In Listing 3.1 are reported the default values for the main algorithmic parameters. The values have been chosen in way that the pipeline performs well across different scenes and data-sets. In general, algorithmic, compilation and architectural parameters have high influence on the overall performance of the reconstruction algorithm. Furthermore, their relationship is highly complex and it is not clear at all how to pick sensible values that maximise the desired performance. Selecting the key performance indicators themselves is an intricate task and suitable choices are usually dependent on the final use case of the SLAM system. In general, is desirable to optimise for multiple metrics at the same time, e.g. tracking accuracy and speed. In Zia et al. [2016] and in Nardi et al. [2017] we investigated more complex strategies to navigate this complex configuration space. We consider multi-objective optimisation goals to which a single solution does not exists. Instead, the focus is shifted to *Pareto-optimal* configuration points, i.e. points where improvements in any objective would imply deterioration on others. The parameter space is explored by using an active learning strategy in conjunction with random forest decision trees.

Parameter	Value
input resolution	320x240
voxel resolution	1cm
icp threshold	1e-5
pyramid levels	3
truncation bandwidth	10cm

Table 3.1.: TSDF-fusion algorithmic parameters.

Even if the benefits of the parameter search strategy described above are significant, in this evaluation we chose to not use the HyperMapper optimisation framework and instead use a reasonable default configuration for all the algorithms tested in this section. The reason is twofold: first, different implementations of similar algorithms may behave very differently and hence tuning the respective pipelines could lead to unfair comparisons. Secondly, we want to test default configurations which port reasonably well to a variety of datasets or input sensors. This is particularly important in case of algorithmic parameters such as truncation bandwidth and ICP convergence threshold as they are effectively defined by the sensor noise and motion characteristics.

3.6.2. Tracking accuracy

System	ATE (m)			
	LR_0	LR_1	LR_2	LR_3
Ours	**0.0113**	**0.0117**	**0.0040**	0.7582
InfiniTAM	0.3052	0.0214	0.1725	**0.4858**

Table 3.2.: Absolute trajectory error (ATE) comparison between our TSDF fusion, and InfiniTAM on ICL-NUIM dataset.

System	ATE (m)			
	fr1_xyz	fr1_desk	fr2_desk	fr3_office
Ours	0.0295	0.1030	0.0641	**0.0686**
InfiniTAM	**0.0273**	**0.0647**	**0.0598**	0.0996

Table 3.3.: Absolute trajectory error (ATE) comparison between our TSDF fusion and InfiniTAM on the TUM dataset.

We evaluate the accuracy of our pipeline across two widely used datasets, i.e. TUM RGB-D (Sturm et al. [2012]) and the ICL-NUIM (Handa et al. [2014]). The former provides RGB-D sequences with trajectory ground truth, estimated via a high frequency motion capture system. Likewise, the latter provides synthetic RGB and depth data, together with ground truth poses. Furthermore, the ICL-NUIM dataset provides also the ground truth synthetic model that we use for surface reconstruction evaluation in Sections 4.6 and 5.7. The metric chosen is the mean absolute trajectory error (ATE), which estimates the absolute distance between the ground truth and computed trajectories (Sturm et al. [2012]), defined as:

$$ATE = \frac{1}{N}\sum_{t=0}^{N} \|\mathbf{g}_t - \mathbf{t}_t\|_2$$

were $\mathbf{g}_t$ and $\mathbf{t}_t$ are respectively the ground-truth and estimated position at time t. As detailed above, for a fair comparison, we use the same parameters throughout. Tables 3.2 and 3.3 report our experimental results, where boldface values indicate the best result per sequence. We compare the volumetric pipeline described in Section 3.5 to the state-of-the-art volumetric pipeline InfiniTAM (Kähler et al. [2015]). This has been tested using the default tracker with depth only tracking to ensure a fair comparison with our solution. As we can see, our system obtains accuracy levels on par with the state-of-the-art. Interestingly, on some sequences, all systems lose track completely (as in fr3_floor and fr3_plant, not shown in Tables for the sake of compactness), and on others, at least some systems fail partly. Note that the extension to use combined geometric and photometric tracking would be straightforward, and we consider this as future work.

3.6.3. Runtime performance

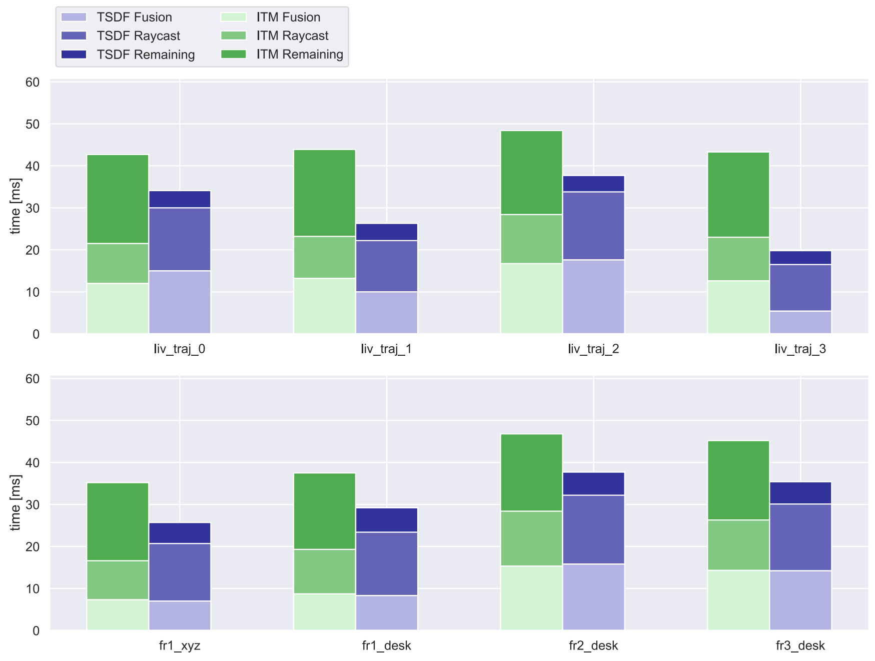


Figure 3.11.: Per-frame performance evaluation of InfiniTAM (ITM), and supereight-based TSDF fusion.

Figure 3.11 reports the runtime performance of each pipeline benchmarked in the previous section. For each implementation, we provide timings for the depth fusion and ray-casting stage, plus an aggregated time for the rest of the pipeline which accounts for preprocessing and tracking. It is worth stressing that we are comparing fully engineered pipelines which have very different code-bases, hence part of the differences in runtime performance are attributable to different implementation choices.

First, we want to highlight how our supereight-based TSDF mapping offers performance comparable or even superior to the state-of-the-art InfiniTAM's voxel hashing implementation. Note that apart from the voxel allocation and retrieval, the two pipelines are in fact very similar in principle. Clearly, the traversal and interpolation strategies described in Sections 3.4.4 and 3.5.4 allow us to amortise the overall cost of querying the tree. Furthermore, despite being asymptotically slower compared to the hash based algorithms presented by Kähler et al. [2015], the allocation routines introduced in Section 3.4.2 have a small footprint on the overall computation.

3.6.4. Memory consumption

Dataset	TSDF	Dataset	TSDF
LR_0	7.67%	fr1_xyz	1.95%
LR_1	8.45%	fr1_desk	7.70%
LR_2	13.77%	fr2_desk	10.15%
LR_3	13.33%	fr3_office	12.50%

Table 3.4.: Relative memory consumption compared to a pre-allocated grid covering the same area at the same resolution.

In Table 3.4 we provide the relative memory consumption compared to a pre-allocated grid. To make a fair comparison, we ran the KinectFusion implementation provided by Nardi et al. [2015], and for a fixed resolution (i.e. 1cm) we allocated the minimum volume required to cover the entire sequence. As expected, memory savings are significant and this is reflected in both memory usage and computational performance. In this analysis we have not included the InfiniTAM framework since the open-source implementation pre-allocates a large pool of memory, as also noted in Bodin et al. [2018], making it difficult to isolate the effective memory consumption for voxel data.

3.7. Conclusions

In this chapter we have presented our octree library *supereight*. Our contribution is twofold. First, our hierarchical algorithms offer competitive performance compared to state-of-the-art hashing methods while providing a full spatial index of the scene. Secondly, we move away from software frameworks tightly coupled with the underlying map representation. Instead, we provide a generic interface which does not compromise performance while allowing the end-user to freely experiment with alternative data-types.

3.7.1. Limitations

One of the assumptions of the octree container is that meaningful information is stored exclusively at the deepest level of the tree, i.e. at full resolution. The octree is consequently used as a space index of the mapped environment, but no data information can be recovered where space is not allocated at full resolution. While this is acceptable in a TSDF map setting, it may not work in other scenarios, e.g. occupancy mapping. We will discuss how to lift this assumption in the next chapter.

Currently, the field update operations supported by our `projective_map` operators are unary, meaning that for a given voxel, the next value may depend only on its previous and has no visibility of its neighbourhood. This means that stencil operations, e.g. convolutions on the tree, are not supported.

3.7.2. Future work

There are several directions in which we want to take this work forward. Performing arbitrary stencil updates on the tree is a topic of high interest which we plan to tackle in future work as we believe it will gain more importance in SLAM pipelines with the advent of semantic and deep learning-based methods. The challenge will be to find a proper computational abstraction in such a way to generate efficient code. Approaches based on domain-specific languages have recently gained significant attention in neighbouring fields such as image processing, notably Halide (Ragan-Kelley et al. [2013]). We believe that similar models may be applied to dynamic, tree-like data structures such as the one presented in this chapter.

Connected to the above, we are interested in extending the interpolation algorithms presented in this chapter to more general access patterns, like the ones found in computation of higher order derivatives. We believe a set-theoretic approach may help formalising and consequently generalising the optimal traversal scheme discussed in Section 3.4.4

Chapter 4

Dense probabilistic tracking and mapping

In this chapter we present a dense probabilistic SLAM pipeline based on occupancy maps. Furthermore, we detail the algorithmic extensions to our supereight library necessary to accommodate full occupancy mapping.

4.1. Motivations

Many vision applications require a complete description of the mapped space. This is especially true in the robotics context, where space maps may be used for path planning and collision avoidance purposes. Occupancy maps (Elfes [1987], Moravec [1988]) are a popular representations that meet the above requirement. Occupancy maps partition the space (either 2D or 3D) into a regular grid of voxels. Each voxel is then assigned with a probability of being occupied or free space according to rigorous Bayesian inference rules. However, one of the limitations of such an approach is its inability to express precise, sub-voxel surface boundaries. On the other hand, TSDF-based maps such as the one described in the previous chapter, do not have a strong probabilistic interpretation and it cannot properly capture information about empty space. Loop et al. [2016] bridge this gap by introducing a new occupancy map framework in which the surface geometry is well defined by construction and hence it could be used for precise 3D reconstruction, while retaining all the semantic properties of an occupancy grid. This implies that precise surface contours can be extracted via ray-casting, and hence such representation can be easily incorporated into the dense SLAM pipeline described in Chapter 3.

The adoption of an occupancy-based representation brings important advantages for a variety of applications. First, all the mapped space is precisely labelled. Free space carries as much information as occupied space and this is expressed explicitly in the map. Being able to reason about free space is of paramount importance for a variety of navigation tasks, as the longevity of occupancy-based solutions in the robotics community testify. Second, a fully probabilistic map allows to naturally take into account the inherent non-determinism in the mapping process, either in the form of sensor noise or in terms of noisy pose estimation. In TSDF-based maps this is achieved with an additional layer of logic, usually via non-uniform and sensor dependent weighting, as in Nguyen et al. [2012] or Oleynikova et al. [2017]. Finally, being able to perform tracking and mapping on an occupancy grid also enables path planning on a single, tightly integrated pipeline, without the need for external tracking systems, as usually assumed by occupancy mapping systems.

In the following sections, we will first give an overview of the classical occupancy grid methods and how Loop et al. [2016] overcome some of their limitations. We will then describe our extensions to their Bayesian fusion framework to adapt it to a online SLAM setting, together with our extensions to supereight. Summarising, our contributions are:

- An extension of the Bayesian fusion framework of Loop et al. [2016] to allow incremental mapping in a SLAM context;
- An updated version of our supereight library (Chapter 3) which offer state-of-the-art performance in occupancy mapping;
- A full dense SLAM pipeline based on 3D occupancy mapping together with a thorough evaluation and comparison against the current state-of-the-art TSDF-based SLAM systems.

4.2. Occupancy Mapping

Occupancy maps aim to fully describe the explored space given agent positions and sensor data. In other words, they compute the posterior probability over possible maps given the data:

$$p(\mathbb{S} \mid \mathbf{z}_{1:t}, \mathbf{T}_{1:t}) \tag{4.1}$$

where $\mathbb{S}$ is the map, $\mathbf{z}_{1:t}$ is the set of sensor measurements taken from known poses $\mathbf{T}_{1:t}$. The map is discretised into a regular grid, hence it consists of a set of voxels:

$$\mathbb{S} = \{S_{\mathbf{x}}\} \tag{4.2}$$

where the subscript $\mathbf{x}$ indicates the 3D position of cell $S_{\mathbf{x}}$. Each voxel is associated with an occupancy probability which specifies whether the voxel is occupied or not, respectively indicated by 1 and 0. Such probability is expressed as $p(S_{\mathbf{x}} = 1)$. The dimensionality of the problem expressed in Equation 4.1 makes it impractical to be solved even for modest size maps. For instance, a room-sized map of 5^3 meters with voxel size of 10cm per side would imply 125,000 voxels, leading to $2^{125,000}$ possible map configurations. From early approaches, such as in Elfes and Matthies [1987], the estimation problem has been simplified by assuming independence between cells and dividing it into smaller, separate problems

$$p(S_{\mathbf{x}} \mid \mathbf{z}_{1:t}, \mathbf{T}_{1:t}) \tag{4.3}$$

and then computing the full posterior as

$$p(\mathbb{S} \mid \mathbf{z}_{1:t}, \mathbf{t}_{1:t}) = \prod_{x \in \mathbb{S}} p(S_{\mathbf{x}} \mid \mathbf{z}_{1:t}, \mathbf{T}_{1:t}) \tag{4.4}$$

Map estimation is then reduced to a binary estimation problem, which can be solved individually for each cell using a binary filter over the two possible states. We refer to Thrun et al. [2005] for full details on the methodology.

While easy to implement, the above factorisation does not properly model any dependency between adjacent cells. In particular, given a surface from which a corresponding sensor measurement is obtained, no other surface can exist between the sensor and such surface. This imposes a visibility constraint on the intersected voxels along the ray from the sensor. Furthermore, voxel dependencies become more and more complex as multiple frames are integrated together. A number of methods that explicitly deal with such dependencies have appeared. In the context of image-based reconstruction, Liu and Cooper [2014] and Ulusoy et al. [2015] formulate the reconstruction problem as inference on the Markov Random Field implied by the visibility constraints between multiple observations. However, their algorithm considers all the input images at once, meaning that the whole inference process should be repeated for each new image in an online setting. The earlier method of Woodford and Vogiatzis [2012], which assumes RGB-D input data, sports a similar formulation to that of Ulusoy et al. [2015], but propose a simpler inference process and furthermore provide an incremental update rule for online reconstruction. However, their strategy requires a per-frame re-sampling of the estimated state, leading to significantly poorer performance compared to standard TSDF grids.

4.3. Bayesian fusion equations

For completeness, in this section we will briefly report the main ideas behind the Bayesian fusion framework by Loop et al. [2016], which we employ as space representation for the occupancy-based SLAM system presented in this chapter. In the following, continuous and discrete probability distributions will be denoted respectively with lower-case p and upper-case P. As usual, subscript t indicates current time stamp, while boldface $\mathbf{x} \in \mathbb{Z}^3$ denotes voxel coordinates.

We start by considering the probability of a single voxel being occupied given a set of measurements $\mathbb{Z}$, that is $P(S_{\mathbf{x}=1} \mid \mathbb{Z})$. Using Bayes' Theorem, this can be written as:

$$P(S_{\mathbf{x}} = 1 \mid \mathbb{Z}) = \frac{p(\mathbb{Z} \mid S_{\mathbf{x}} = 1) P(S_{\mathbf{x}} = 1)}{\sum_{s=0}^{1} p(\mathbb{Z} \mid S_{\mathbf{x}} = s) P(S_{\mathbf{x}} = s)} \tag{4.5}$$

The surface prior $P(S_{\mathbf{x}} = s)$ and the conditional probability $p(\mathbb{Z} \mid S_{\mathbf{x}} = s)$ now need to be defined. The former is set to a minimum informative value:

$$P(S_{\mathbf{x}} = s) = \frac{1}{2} \tag{4.6}$$

For the latter, Loop et al. [2016] make the following key observations. First, since $\{S_{\mathbf{x}}\}$ are independent and identically distributed, the set of

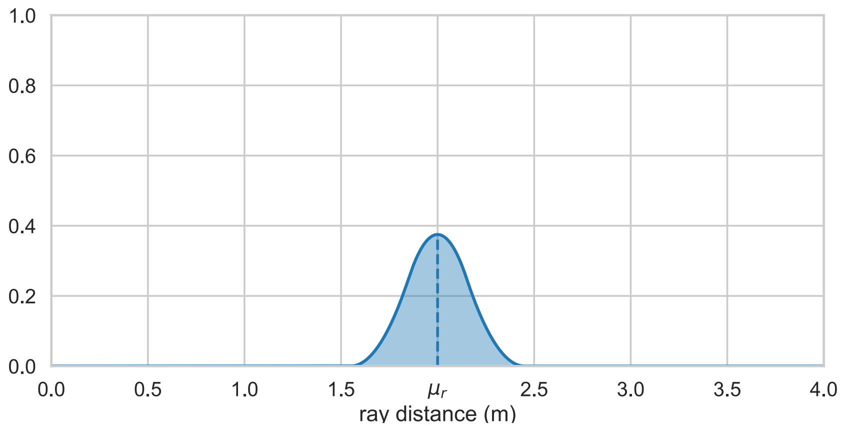


Figure 4.1.: B-spline noise distribution. The dotted line at μ_r corresponds to the location of the surface along the ray.

rays $\mathbb{Z}_{\mathbf{x}} \doteq \{z_r \mid z_r \in \mathbf{x}\}$ that pass through $\mathbf{x}$ are conditionally independent given $S_{\mathbf{x}}$ as $\mathbf{x}$ is the only point that they have in common. Furthermore, the set of rays that do not pass through $\mathbf{x}$, $\mathbb{Z}_{\overline{\mathbf{x}}} \doteq \mathbb{Z} \setminus \mathbb{Z}_{\mathbf{x}}$, is primarily dependent on $\mathbb{Z}_{\mathbf{x}}$. Hence:

$$\begin{aligned} p(\mathbb{Z} \mid S_{\mathbf{x}} = s) &= p(\mathbb{Z}_{\overline{\mathbf{x}}} \mid \mathbb{Z}_{\mathbf{x}}) p(\mathbb{Z}_{\mathbf{x}} \mid S_{\mathbf{x}} = s) \\ &= p(\mathbb{Z}_{\overline{\mathbf{x}}} \mid \mathbb{Z}_{\mathbf{x}}) \prod_{z_r \in \mathbb{Z}_{\mathbf{x}}} p(z_r \mid S_{\mathbf{x}} = s) \end{aligned} \tag{4.7}$$

Plugging 4.6 and 4.7 into 4.5 and simplifying gives:

$$P(S_{\mathbf{x}} = 1 \mid \mathbb{Z}) = \frac{\prod_{z_r \in \mathbb{Z}_{\mathbf{x}}} p(z_r \mid S_{\mathbf{x}} = 1)}{\sum_{s=0}^{1} \prod_{z_r \in \mathbb{Z}_{\mathbf{x}}} p(z_r \mid S_{\mathbf{x}} = s)} \tag{4.8}$$

Finally, again using Bayes' theorem

$$p(z_r \mid S_{\mathbf{x}} = s) = \frac{P(S_{\mathbf{x}} = s \mid z_r) p(z_r)}{P(S_{\mathbf{x}} = s)}$$

we obtain:

$$P(S_{\mathbf{x}} = 1 \mid \mathbb{Z}) = \frac{\prod_{z_r \in \mathbb{Z}_{\mathbf{x}}} P(S_{\mathbf{x}} = 1 \mid z_r)}{\sum_{s=0}^{1} \prod_{z_r \in \mathbb{Z}_{\mathbf{x}}} P(S_{\mathbf{x}} = s \mid z_r)} \tag{4.9}$$

The single ray behaviour is described in terms of the occupancy dis-

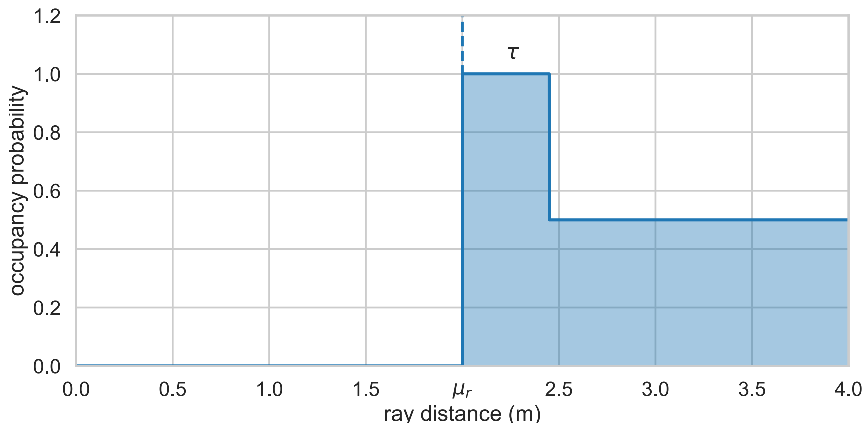


Figure 4.2.: Occupancy distribution given true depth. The dotted line at μ_r corresponds to the location of the surface along the ray and τ denotes the surface thickness.

tribution given the true surface and the true surface distribution given a noisy measurement, mathematically:

$$P(S_{\mathbf{x}} = 1 \mid z_r) = \int_0^\infty P(S_{\mathbf{x}} = 1 \mid \mu_r) p(\mu_r \mid z_r) d\mu_r \tag{4.10}$$

The true surface distribution $p(\mu_r \mid z_r)$ is modelled with a quadratic b-spline, graphically shown in Figure 4.1, defined as:

$$p(\mu_r \mid z_r) = q(s) = \begin{cases} \frac{1}{16}(3+s)^2 & \text{if } -3 \leq s \leq -1, \\ \frac{1}{8}(3-s^2) & \text{if } -1 \leq s \leq 1, \\ \frac{1}{16}(3-s)^2 & \text{if } 1 \leq s \leq 3, \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

where

$$s := \frac{(\mu_r - z_r)}{\sigma_r} \tag{4.12}$$

denotes the distance from the camera centred around the true distance and normalised with the standard deviation σ_r of the measurement. Importantly, we can now set σ_r to be proportional to z_r^2 corresponding to a more realistic triangulation-based depth camera noise model than assuming it constant (Gallup et al. [2008]). The occupancy probability given the true surface, $P(S_{\mathbf{x}} = 1 \mid \mu_r)$, is modelled as zero in front of the true surface, 1 from the true surface to a certain minimum thickness τ, and then followed by 0.5, as shown in Figure 4.2. Conveniently, the convolution integral of Equation 4.10 has an analytic solution

$$P\left(S_{\mathbf{x}} = 1 \mid z_r\right) = h(s) = q_{\text{cdf}}(s) - \frac{1}{2}q_{\text{cdf}}(s-3), \tag{4.13}$$

with

$$q_{\text{cdf}}(s) = \begin{cases} 0 & \text{if } s < -3, \\ \frac{1}{48}(3+s)^3 & \text{if} -3 \leq s \leq -1, \\ \frac{1}{2} + \frac{1}{24}s(3+s)(3-s) & \text{if } -1 < s < 1, \\ 1 - \frac{1}{48}(3-s)^3 & \text{if } 1 \leq s \leq 3, \\ 1 & \text{if } 3 < s. \end{cases} \tag{4.14}$$

We visualise an example of this per-ray occupancy "measurement" function in Figure 4.3. Setting the surface thickness τ equal to three times the standard deviation σ_r gives an occupancy probability of exactly $\frac{1}{2}$ in correspondence of the unit-pulse transition from empty to occupied space, which defines the precise location of the surface. It is worth noting that the adoption of a finite support b-spline noise model, as opposed to a more common Gaussian distribution, is justified by the fact that the latter results in inconsistent occupancy estimates, even when Equation 4.10 is evaluated using exact surface measurements (see Section 4.2.3 of Loop et al. [2016] for a detailed discussion).

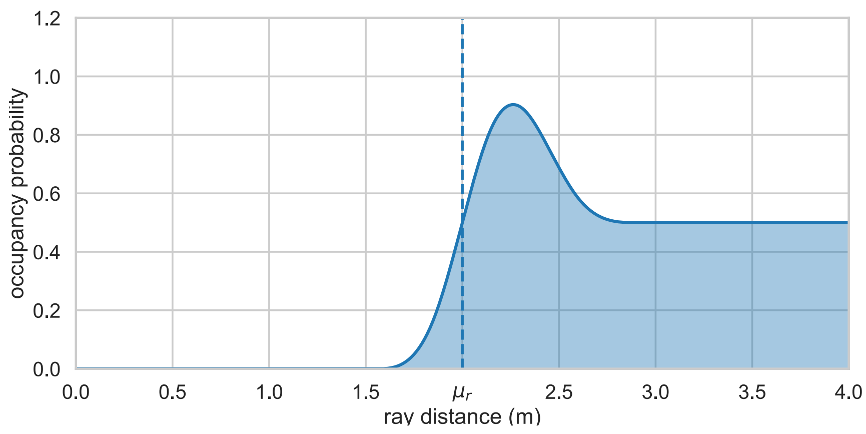


Figure 4.3.: Example occupancy probabilities along a ray resulting from Equation 4.10

Occupancy measurements $o_t(\mathbf{x})$ associated with the above values $h(s)$ along every ray observing depth can now be used for fusion into our octree-based map volume storing occupancy values $O_t(\mathbf{x})$ at each position $\mathbf{x}$. However, instead of the multiplicative update following direct application of Bayes' rule (as used in Loop et al. [2016]), we adopt the *log-odd* space which is mathematically equivalent. The incremental update accordingly is:

$$l_t(\mathbf{x}) = \log \frac{o_t(\mathbf{x})}{1 - o_t(\mathbf{x})}, \tag{4.15}$$

$$L_t(\mathbf{x}) = L_{t-1}(\mathbf{x}) + l_t(\mathbf{x}), \tag{4.16}$$

where $L_0 = \log \frac{0.5}{1-0.5} = 0$.

The $h(s)$ function, although it correctly encodes the visibility constraints discussed in Section 4.2, has a mayor drawback, i.e. the finite support of the b-spline model admits zero-probability values. This breaks incremental mapping in two ways. First, zero-values make it impossible to recover from related outliers or to deal with more realistic scenes in which dynamic elements are present. Second, zero-values are not admissible in the log-odds formulation, hence we clamp h to the interval $[P_{\min}, P_{\max}]$. In our experiments, we choose the admissible interval as $[0.03, 0.97]$. In contrast to typical occupancy mapping formulations, the crucial feature of the above model is that the surface boundary is well defined by construction, and it can be found where the occupancy probability transitions from $O_t(\mathbf{x}) < \frac{1}{2}$ to $O_t(\mathbf{x}) > \frac{1}{2}$, or, equivalently in log-odd space, from $L_t(\mathbf{x}) < 0$ to $L_t(\mathbf{x}) > 0$. However, whilst the surface location does not change its position as more accurate estimates are combined, the gradient of the occupancy curve quickly increases, and the occupancy estimate becomes overconfident. For this reason, we choose to extend the Bayesian fusion model with a windowed update step which introduces uncertainty proportionally to the time difference between subsequent updates, in order to accommodate for otherwise unmodelled effects, most importantly dynamic scene content and uncertainty of the tracking. Specifically, we apply a moving average before each measurement is fused into the map.

Thus, our final update rule is defined as:

$$L^{+}_{k-1} = L_{k-1}(\mathbf{p}) \frac{1}{1 + \frac{\Delta t}{\tau}}, \tag{4.17}$$

$$L_k(\mathbf{p}) = L^{+}_{k-1}(\mathbf{p}) + l_k(\mathbf{p}), \tag{4.18}$$

where Δt is the time difference since the last update for the current cell and τ is a time constant. In our experiments we chose $\tau = 5$sec. Note that this also acts as a forgetting feature: when $\Delta t \to \infty$, $O^{+}_{t-1}(\mathbf{p}) \to 0.5$. In other words, we assume that if we have not updated a specific cell for a long period of time, we don't know its occupancy state.

4.4. Multi-scale octant operations

The occupancy field defined in Section 4.3 fully describes the mapped space, both empty and occupied. Hence, it is desirable to have a finer control over which parts of the map are allocated and at which resolution. One of the assumptions made in Section 3.4 is that supereight's octree structure is used for indexing a sparsely allocated pool of fine resolution blocks, but the nodes belonging to the tree structure itself do not contain any data. While this is reasonable in case of TSDF mapping, where meaningful data can be found only close to the surface boundaries, in this chapter we need to lift this assumption. Specifically, occupancy values for free space are not constant and hence need to be represented explicitly. In the next sections we will cover in detail how we approach this problem in supereight.

4.4.1. Node representation

As defined in Section 3.4, internal octree nodes have structural significance, but they do not hold any information regarding the underlying represented field. We extend the basic node structure to also store field information, as shown in Listing 4.1. Instead of assigning one value per node, we associate a data value per child octant, *regardless* of whether the child is actually allocated or not. In other words, children data values are completely described by their parents. In this way, given a node at an arbitrary tree height, its partition at the children resolution will be uni-

```
template <typename T>
class Node {
  key_t code_; // octant descriptor
  unsigned char children_mask_; // children validity mask
  Node* children_[8]; // child pointers array
  T data_[8]; // field data
};
```

Listing 4.1: Octree node structure with associated data.

(a) **(b)**

Figure 4.4.: (a) Partial and (b) full subdivision of a quadrant. Dots represent the quadrant anchors and sampling points.

form, even if a subset of its children is not recursively refined. Figure 4.4 depicts the sample point distribution according to the data association scheme implemented in Listing 4.1. Notice how the refinement of Figure 4.4a induces an asymmetrical segmentation of the covered area. Associating a data point to each *potential* child solves this problem, as Figure 4.4b shows. This is semantically equivalent to a *full* octree, i.e. where each node has either zero or eight children, but it achieves the same while sparing the memory overhead induced by the node structure. Octants which share the same physical location but have different sizes (e.g. parent and first child) will be encoded with the same Morton number. In order to fully identify a node, we append the height of the tree at which it resides, in a similar fashion to Sundar et al. [2008].

As described in Section 3.4 the finest resolution voxel blocks are stored contiguously in memory in blocks of 8^3 voxels, hence only one positional key is required to identify 512 elements. This is reflected on the generated morton codes: being $max\_depth - 3$ the deepest level of the tree, the lowest 9 bits are always zero. We use the trailing bits to store the node level,

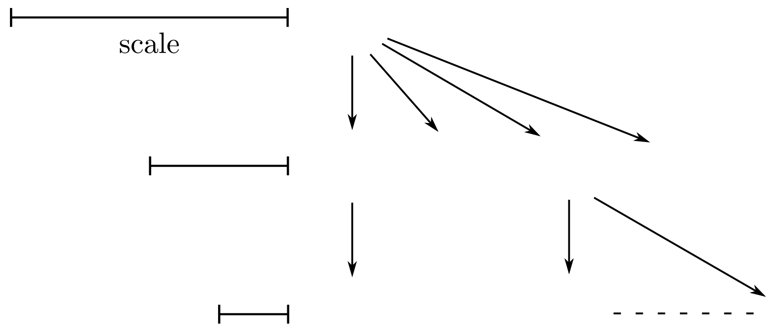


Figure 4.5.: Logical layering of nodes at different height of the tree. Nodes with the same color correspond to the same physical location but have different extension.

starting from the root which has zero depth. A pleasant property of such an augmented key is that it preserves ordering between octants, implying that an octant will come before its descendants in sorted order. We exploit this property to perform duplicate removal and parallel allocation.

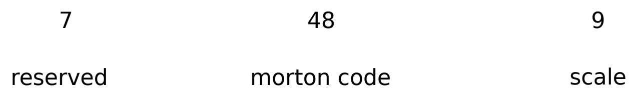


Figure 4.6.: 64 bit positional code of an octree node.

Figure 4.6 displays the final structure of our 64 bit node descriptor. As discussed above, the lowest 9 bits specifies the node level. The following 48 bits section stores the Morton number, allowing for a 16 level-deep octree. Finally we keep the last 7 bits reserved. These bits can be used for various purposes, such as node-specific flags or further extending the tree maximum height.

4.4.2. Node encoding

According to the encoding scheme presented in the previous section, coarser octree nodes can be uniquely identified and consequently should be possible to selectively allocate them. To do so, supereight provides encoding and decoding functions, shown in Listing 4.2. Octants, regardless of their scale, are addressed in terms of their coordinates at the finest resolution. Exact location is not required, as given a tuple `(x, y, z)` and

```
1   key_t encode(const int x, const int y, const int z,
2       const int level, const int max_depth) {
3     const int offset = MAX_BITS - max_depth + level - 1;
4     return (compute_morton(x, y, z) & MASK[offset]) | level;
5   }
6
7   Eigen::Vector3i decode(const se::key_t key) {
8     return unpack_morton(key & ~SCALE_MASK);
9   }
```

Listing 4.2: Octant encoding and decoding.

a level l, coordinates are rounded to the left-most bottom corner of their enclosing octant at level l. Practically, this is achieved by selecting the appropriate prefix mask (line 3) and applying it to the corresponding full Morton code (line 4). Finally, the complete node descriptor is obtained by appending the level to the computed key.

4.4.3. Node allocation

In contrast with the allocation strategy presented in Section 3.4.2 where internal nodes are implicitly allocated, we provide allocation functions which permit explicit allocation of nodes at any level of the tree. Hence, allocation requests may arise for:

1. A voxel at a level deeper than the leaf aggregation level. In this case the allocation stops at the aggregation level and the whole voxel block containing the requested voxel is allocated.

2. A voxel at a level higher than the aggregation level. The allocation procedure stops at the specified level.

In both cases, *any* ancestors of a node will be implicitly allocated. We modify Algorithm 3 in order to take into account the multi-scale node allocation in two ways. First, from the allocation list redundant nodes have to be removed. A node is considered redundant if any octant in the allocation buffer implies its allocation, i.e. any of its descendants are present in the allocation list. Algorithm 5 is used to check if two octants are in an ancestor-descendant relationship. This is done by checking the condition that the octant has a smaller scale and it shares the prefix with the ancestor up to the ancestor's level.

ALGORITHM 4: Top-bottom multiscale voxel list allocation

```
1  Function allocate(key_t key_list[N] : int allocated
2  {
3      key_list ← sort (key_list);
4      unique_keys ← filter_ancestors (key_list);
5      for level ← root.level to aggregate_level do
6          unique_keys ← [ k & prefix_mask [level] for k in unique_keys ];
7          unique_keys ← compact (unique_keys, level);
8          parallel_alloc (unique_keys, level);
9      end for
10 }
```

ALGORITHM 5: Test ancestry relationship between two octants. Returns true if *octant* is a descendant of *ancestor*

```
1  Function descendant(key_t ancestor, key_t octant) : bool
2  {
3      int level_a ← ancestor.level();
4      int level_o ← octant.level();
5      octant ← octant.code() & MASK[level_a];
6      ancestor ← ancestor.code();
7      return level_o ≥ level_a && !(ancestor ⊕ octant);
8  }
```

Algorithm 6 filters ancestors from a *sorted* input list of descriptors. When an ancestor-descendant relationship is detected, the ancestor is simply overwritten with its descendant. Finally, the compact operation at

ALGORITHM 6: Filter ancestors from octant list

```
1  Function filter_ancestors(key_t key_list[N]) : int new_size)
2  {
3      last ← 0;
4      for i ← 0 to key_list.size() do
5          if descendant (key_list [last], key_list [i]) then
6              key_list [last] ← key_list [i];
7          else
8              key_list [++last] ← key_list [i];
9          end if
10     end for
11     return last + 1;
12 }
```

line 7 of Algorithm 4 eliminate duplicates *and* filters out nodes which are coarser than the currently allocated level. This is required to guarantee

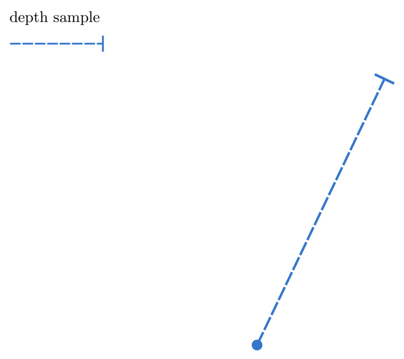


Figure 4.7.: Raycast-based allocation from depth sample. The salmon triangle represents the camera frustum, pale yellow squares 2D orthogonal projections of octree octants (in 2D for visualisation purposes) and black dots sampling points.

that octants are allocated only up to their maximum resolution, specified via the scale field in the node descriptor.

4.5. Reconstruction pipeline

The extended Bayesian fusion equations described in Section 4.3 provide us with a probabilistic field representation that can be used for various tasks, such us mapping, tracking and planning. Indeed, one of the main objective of this work is to demonstrate a fully integrated pipeline for autonomous robotic navigation, in which path planning and SLAM occur on the same tightly integrated loop and not as a collection of different subsystems. In order to achieve this, we introduce a modified version of the reconstruction pipeline of Section 3.5.

4.5.1. Scene allocation

As usual, the system bootstraps with an empty scene which need to be populated given the sensor data. Similarly to our TSDF-based pipeline, we find the missing nodes via ray-casting. However, contrary to a TSDF field the empty space representation is not constant. There are several different ways to approach this problem. Naively, it could be recognised that empty-seen space effectively is valuable yet non-uniform information and so it needs to be represented explicitly. However, this would lead to an unreasonable memory footprint for scenes bigger than modestly sized environments. OctoMap, the de-facto standard for occupancy mapping by Hornung et al. [2013], in a first pass allocates and processes the whole volume implied by the view frustum. In a subsequent pass, empty space which is considered uniform below a certain occupancy threshold is compressed and the tree pruned. While this certainly helps in terms of memory footprint, computationally is not very efficient as keeping the tree sparse requires repeated pruning operations as new frames are integrated. We take a different approach: we allocate space close to the surface boundary at maximum resolution and then progressively allocate coarser voxels in correspondence of empty space. There are two main advantages with our approach: first, it does not require an explicit pruning step to coarsen blocks covering empty space. Second, fusing new depth measurements does not invalidate the allocations done at the previous step, i.e. if the environment is static coarse blocks will stay coarse.

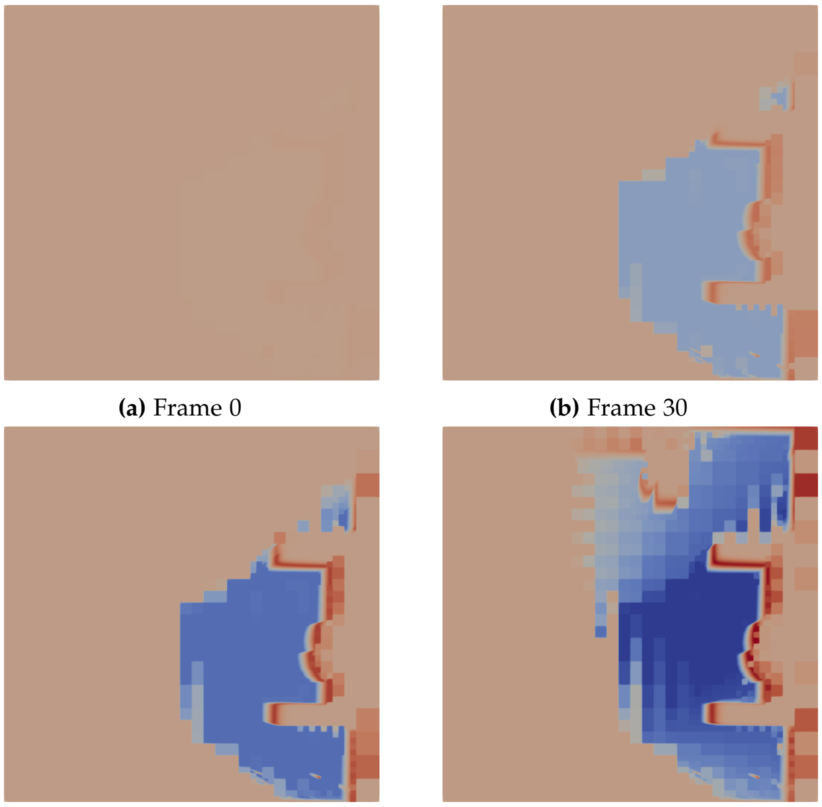
Algorithm 7 implements the logic described above. As usual, in parallel each pixel is back-projected to its 3D position in world coordinates by applying the inverse camera matrix function π^{-1} and the camera to world transform T_{wc}. Then a ray is marched from the camera centre to the point collecting all the intersected octants along the ray. Crucially, the step size is not uniform, but rather adjusted dynamically depending on the distance from the measured surface. Functions `step_function` and `step_to_level` respectively adjust the step size (line 16) and select the appropriate octant scale, which is then used to synthetise the corresponding node descriptor for the visited octant (line 14). If the currently visited node is not found in the map, it is atomically enqueued in an allocation buffer. Figure 4.7 shows a graphical depiction of the resulting allocation list for a single ray. Notice how octants become more refined as the ray gets closer to the depth sample.

ALGORITHM 7: Variable step raycast for scene allocation.

```
Function compute_octants(Image depthmap, Octree map, Transform T_wc,
 Function step_function, Function step_to_level) : key_t [n]
{
    key_t keylist[n] ← 0;
    int2 pix ← (0, 0);
    float3 origin ← T_wc.translation();
    foreach pix ∈ depthmap do
        float3 dir ← normalise (T_wc * $\pi^{-1}$ (pix) - origin);
        float dist ← $\left\|\text{T\_wc} * \pi^{-1}(\text{pix}) - \text{origin}\right\|$;
        float3 pos ← origin;
        float t ← 0;
        float step ← step_function (t, dist);
        while $t \leq dist$ do
            if (pos + t * dir) ∉ map then
                keylist ← map.encode(pos + t * dir, step_to_level (step));
            end if
            step ← step_function (t, dist);
            t ← t + step;
        end while
    end foreach
    return keylist;
}
```

4.5.2. Field update and interpolation

The algorithms presented in Section 3.4.3 and 3.4.4 are basically unchanged. Given the sample point voxel model discussed in the previous chapter we can employ the same projective functors operators on intermediate nodes with minimal changes in the library back-end, chiefly to take into account the wider spacing between data points in intermediate nodes. This is easily achieved as node descriptors have all the information required to compute the size of an octant and its children. Figure 4.8 shows an example of an occupancy grid map after 180 frames have been fused. Blue shades indicate empty space, salmon unseen space and white to red gradients occupied space. Notice how empty and occupied space grows in confidence as more data is fused into the map. Furthermore, the gradient between empty and occupied space becomes steeper but still smooth thanks to our windowed update step. Another key aspect that can be observed from the occupancy renderings is how we handle rays that lie on the frustum boundaries. While in theory the boundary between seen


(a) Frame 0 **(b)** Frame 30

(c) Frame 60 **(d)** Frame 180

Figure 4.8.: 2D slice of the occupancy map along the x-z plane after 0, 30, 60 and 180 fused frames from the ICL-NUIM Living Room 2 trajectory. Blue shades indicate empty space, salmon unseen space and white to red gradients occupied space.

and unseen space can be treated as a surface and resolved to an arbitrary resolution, we do not handle them differently than any other ray in camera frame. This is key for performance, since as the camera moves around and new space is discovered, empty space at frustum boundaries would be overly detailed and it would accumulate in time, requiring explicit pruning to sparsify the map. On the other hand, the system never loses accuracy in presence of real surfaces, hence we found this simplification to work very well in practice.

As stated in Section 4.4.2, voxels can be retrieved using their coordinates at the finest level of detail, but we also provide routines that accept an extra scale argument to stop the recursive search at a given height of the tree. Tri-linear interpolation is instead limited to voxels which are allocated at the maximum resolution. If such information is not available, the system falls back to nearest neighbour approximation. We found this an acceptable compromise as the applications which supereight targets allocate voxels at the maximum resolution where trilinear interpolation is effectively needed, while they can tolerate less accuracy anywhere else. Full trilinear interpolation on non-uniformly spaced grids will be addressed in Chapter 5, where we lift this constraint.

Conveniently, as in TSDF-based maps, surface boundaries can be found at the zero-crossing of the log-odd field (Equation 4.16). However, the ray-casting operation differs in that once a ray-block intersection is found, the ray should be marched advancing in unitary steps. This is different from TSDF maps where steps proportional to the interpolated distance value can be taken. This indeed implies slower ray-casting performance as shown in our quantitative experiments.

4.6. Experimental results

In this section we provide quantitative results for our occupancy-based SLAM pipeline. Using the same experimental set-up of Section 3.6 we compare our solution against the TSDF-based pipeline, InfiniTAM (Kähler et al. [2015]) and OctoMap (Hornung et al. [2013]).

The TSDF and Occupancy Mapping pipelines are set up using the same values for shared parameters, such as voxel resolution or ICP convergence thresholds. These, together with the occupancy fusion (in short OFusion) specific parameters are summarised in Table 4.1.

4.6.1. Tracking and reconstruction accuracy

Tables 4.2 and 4.3 reports the absolute trajectory errors (ATE) for the three pipelines on sequences from the ICL-NUIM and TUM RGB-D datasets. According to our experiments, tracking and mapping based on fused occupancy probabilities matches and often exceeds the tracking accuracy of TSDF based pipeline, both in ours or InfiniTAM's implementation, scoring the best accuracy on challenging trajectories such as LR_3, fr1_desk and fr3_office. Notice that in this experiment set our TSDF based pipeline

Parameter	Value
input resolution	320x240
voxel resolution	1cm
icp threshold	1e-5
pyramid levels	3
stddev scaling (k)	0.01
sliding window (τ)	5sec
Pmin	0.03
Pmax	0.97

Table 4.1.: Default algorithmic parameters for the occupancy fusion reconstruction pipeline.

loses track on the LR_3 sequence, which is particularly challenging due to the extensive close-up views with very little geometric structure. We also evaluated the reconstruction precision of our pipelines against the ground-truth mesh provided with ICL-NUIM and using the companion evaluation tool SurfReg[1] which computes the Root Mean Square Error (RMSE) distance of the reconstructed mesh from the ground truth model. The output reconstruction is extracted from the TSDF and occupancy fields via marching cubes (Curless and Levoy [1996]). Table 4.4 reports our quantitative evaluation. Note that SurfReg performs a pre-alignment of the compared meshes before evaluating the surface distance, hence off-

[1]https://github.com/mp3guy/SurfReg

System	ATE (m)			
	LR_0	LR_1	LR_2	LR_3
TSDF	**0.0113**	**0.0117**	**0.0040**	0.7582
OFusion	0.0236	0.0185	0.0046	**0.1095**
InfiniTAM	0.0850	0.0214	0.1584	0.1354

Table 4.2.: Absolute trajectory error (ATE) comparison between our TSDF fusion, and Occupancy Fusion and InfiniTAM on ICL-NUIM dataset.

System	ATE (m)			
	fr1_xyz	fr1_desk	fr2_desk	fr3_office
TSDF	0.0295	0.1030	0.0641	0.0686
OFusion	**0.0149**	**0.0524**	0.1080	**0.0266**
InfiniTAM	0.0273	0.0647	**0.0598**	0.0996

Table 4.3.: Absolute trajectory error (ATE) comparison between our TSDF fusion occupancy fusion and InfiniTAM on the TUM dataset.

setting part of the error due to pose drift. This is reflected by the more uniform results across the different systems in terms of reconstruction accuracy.

System	RMSE (m)			
	LR_0	LR_1	LR_2	LR_3
TSDF	**0.0054**	0.0066	**0.0051**	0.0541
OFusion	0.0061	0.0060	**0.0051**	**0.0440**
InfiniTAM	0.0060	**0.0057**	0.0504	0.0585

Table 4.4.: Root Mean Square Error (RMSE) of the distance of the reconstruction from the ground-truth mesh.

4.6.2. Run-time performance

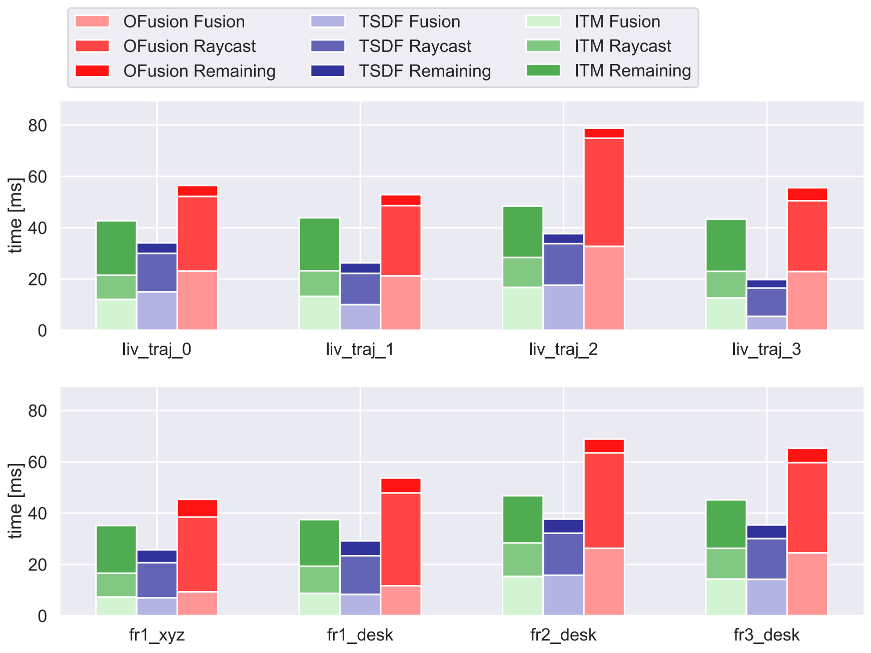


Figure 4.9.: Per-frame performance evaluation of InfiniTAM (ITM), supereight-based TSDF and Occupancy fusion.

Figure 4.9 shows the runtime performance of each pipeline. As usual, we provide timings for the fusion and rendering stages, plus an aggregated time for the rest of the pipeline. Evidently, the occupancy grid mapping formulation is more expensive. This is inherent to the method itself as the b-spline sampling and the log-odd update is more costly than the simple weighted average performed by the TSDF method. Furthermore, occupancy mapping has to process a larger amount of information as empty-seen space is explicitly stored and updated. Additionally, finding the zero-crossing in ray-casting is more efficient in a TSDF field compared to occupancy mapping, since the distance encoding allows to skip empty space with larger steps in a safe way (that is without skipping valid intersections). In contrast, occupancy mapping requires the rays to be marched one voxel at the time once the first intersected voxel block is reached. A possible way to mitigate this penalty would be to implement a mapping from occupancy probability to distance.

We also benchmarked OctoMap on the test sequences used in this experiment set, configured with 5cm voxel size, but we omitted these results from Figure 4.9 for visualisation purposes. Mapping times per frame range between 338ms (fr1_desk) to over 1500ms (fr2_desk). The large performance gap compared to our pipeline is attributable to the slower algorithm OctoMap employs, i.e. ray-casting based measurement integration (discussed in Section 3.4.3), and a lack of a proper parallelisation strategy. Oleynikova et al. [2017], in their VoxBlox mapping framework, propose various optimisations for ray-cast based map update and demonstrate interesting speed-ups, but still requiring at least 60ms per scan at 5cm voxel resolution. Notice that for use cases in which a coarser map is suitable, other approaches are possible. Saarinen et al. [2013] demonstrate how using *normal distribution transform occupancy maps* (NDT-OM) they are able to achieve comparable results to OctoMap while using an 8 times coarser grid, achieving 20 times faster measurement integration.

4.6.3. Memory consumption

Dataset	TSDF	OFusion	Dataset	TSDF	OFusion
LR_0	7.67%	11.15%	fr1_xyz	1.95%	3.01%
LR_1	8.45%	13.68%	fr1_desk	7.70%	8.81%
LR_2	13.77%	22.52%	fr2_desk	10.15%	17.70%
LR_3	13.33%	17.68%	fr3_office	12.50%	17.95%

Table 4.5.: Relative memory consumption compared to a pre-allocated grid covering the same area at the same resolution.

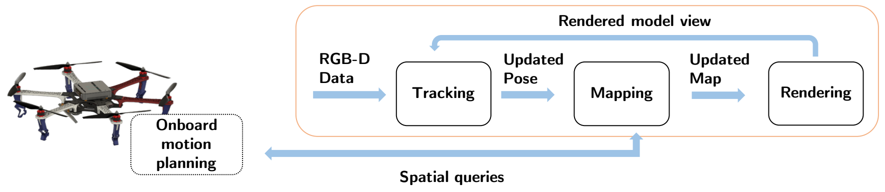


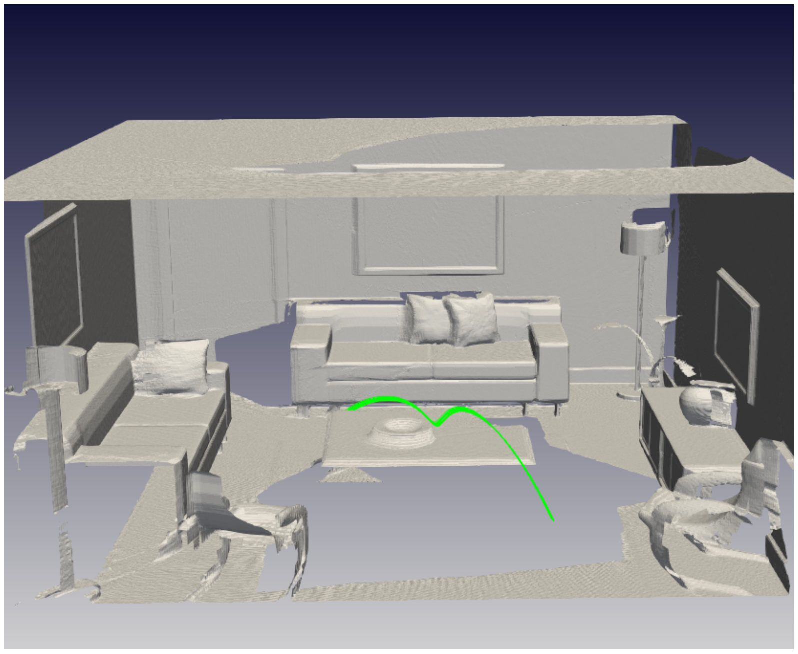
Figure 4.10.: Schematic representation of our integrated dense SLAM pipeline with support for online spatial queries.

Table 4.5 shows a side by side comparison of memory consumption between our TSDF and Occupancy Mapping based pipelines, relative to a common baseline of a pre-allocated, tightly fitting voxel grid at the same resolution. As expected, the memory footprint of the occupancy-based system is larger compared to the TSDF solution. This clearly is due to the allocation of voxels in correspondence of seen-empty space. Although this results in more memory used, the adaptive allocation scheme described in Section 4.5.1 mitigates the requirement, without impacting the overall accuracy of the SLAM pipeline.

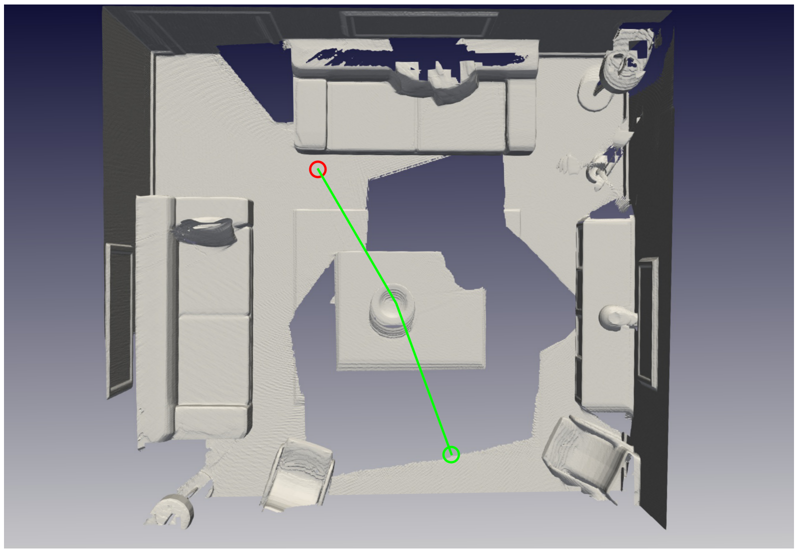
4.6.4. An application to path planning

In Vespa et al. [2018] we also demonstrated and analysed our occupancy-based mapping pipeline in a path planning application for a multicopter Micro Aerial Vehicle (MAV). The full algorithmic pipeline is depicted in Figure 4.10. We used the Open Motion Planning Library (OMPL) of Şucan et al. [2012] to generate collision-free paths in our occupancy-based environment, with the MAV modelled as an axis aligned bounding box (AABB). We used Informed RRT* Gammell et al. [2014] for the straight-line segment planning. Furthermore, we compared our supereight implementation against an OctoMap-based map. The times needed to find the first feasible path for an obstructed 2.83 m start-goal distance can be seen in Table 4.6. They were obtained on an Intel Core i7-6600U CPU at 2.60GHz, compiled on GCC version 5.4.0, averaged over 10,000 executions.

Furthermore, we calculated a smooth trajectory from the initial RRT* plans based on polynomial planning as described in Richter et al. [2016]. We fixed the start and goal position in both mapping implementations and recorded the time needed to linearly optimize a collision-free poly-



(a)



(b)

Figure 4.11.: 3D (4.11a) and top (4.11b) views of an example trajectory (green) for a multicopter Micro Aerial Vehicle (MAV) computed with the Informed RRT* planning algorithm (with smoothing) depicted on a fine-grained occupancy map obtained by our SLAM system. Starting and target positions are represented respectively by the red and green circles.

nomial trajectory. The timings averaged over 1,000 executions are listed in Table 4.7. It can be seen in the recorded timings in Table 4.6 and Table 4.7 that the straight-line planning and the linear polynomial trajectory optimisation is faster with our implemented method. For illustration, we have plotted an example trajectory into a map rendering in Figure 4.11. These results confirm that our octree-based occupancy map is at least as fast at handling spatial queries as OctoMap, the *de-facto* standard used in research for robotic planning.

4.7. Conclusions

In this chapter we have presented an efficient octree-based dense SLAM system based on occupancy mapping. We contribute with an extension to a fully probabilistic fine-grained occupancy mapping formulation. The resulting representation is not only used for surface reconstruction and camera tracking, but enables real-time, in-the-loop path planning on the very same representation. We experimentally evaluate our formulation in a variety of sequences, demonstrating state-of-the-art accuracy and performance results, including a comparison. We furthermore demonstrated the capabilities for planning using a probabilistic planner and trajectory smoothing. Importantly, efficient spatial queries as needed for planning are intrinsically not supported by flat hashing architectures as employed by competing SLAM systems. We thus believe this work will help to further bridge the gap between SLAM and down-stream operations and increase related efficiency by sharing a map representation of wider use-

	time	std dev	min	max
OFusion	12.6ms	14.6ms	4.29ms	109.6ms
OctoMap	17.7ms	11.4ms	5.16ms	113.2ms

Table 4.6.: Timings for straight-line planning for a start-goal distance of 2.83 metres averaged over 10,000 executions.

	time	std dev	min	max
OFusion	1.57ms	0.59ms	0.43ms	3.47ms
OctoMap	2.06ms	0.78ms	0.32ms	4.17ms

Table 4.7.: Timings for the linear trajectory optimisation averaged over 1,000 executions.

fulness. Concluding, we see this as an interesting alternative to the most common approach of implementing tracking and mapping as two separate sub-systems that do not share common information. However, we acknowledge that, especially in resource constrained systems where dense rendering and alignment may be prohibitive, a hybrid approach based on sparse tracking could still be a more suitable choice.

4.7.1. Limitations and further work

Occupancy probabilities are computed and updated independently for each node (Section 4.5.2). However, no mechanism to keep the hierarchy consistent is used. In the next chapter, we lay the foundations to achieve this in a real-time setting. This will eventually allow to perform occupancy queries at any scale with the guarantee that all the available information is being consistently used.

Another issue with the occupancy mapping formulation is its lower computational performance compared to TSDF mapping. This is particularly noticeable in ray-casting, as spatial information encoded by the TSDF can be used to skip larger portions of empty space. To achieve this with an occupancy map, an interesting approach would be to relate occupancy values to distances at interpolation time, in a way reversing the ray model described by Equation 4.10.

Chapter 5

Adaptive resolution octree-based dense SLAM

The effective resolution of visual information may dramatically vary on a per-pixel basis. Surfaces close to the camera are resolved at higher resolution compared to distant surfaces. Consequently, as the sensor browses a scene, the same objects may be observed at different scales throughout the mapping process. This effect should be taken into account in any reconstruction system. In this chapter, we propose an adaptive-resolution dense SLAM pipeline that leverages the multi-scale representation provided by our octree structure to perform dynamic resolution fusion and rendering of RGB-D data.

5.1. Motivations

Reconstructing high-fidelity models of the environment brings many challenges. First, dense methods are notoriously expensive, both in terms of computational cost and memory footprint. Second, faithfully capturing the fine details of the scene is challenging, as maintaining a uniformly high resolution map is neither feasible, nor necessary. Any depth estimation system, whether variable baseline multi-view stereo, stereo rigs or active depth cameras, provide information at different scales. Close-up views capture fine details that are not preserved when observed at farther distances. Hence, fusing such data at uniform scale is wrong for two reasons: i) aliasing artefacts arising from a resolution mismatch between the map and sensor data may result in degraded map quality; ii) fine details recovered from close-up views may be lost when the scene is observed from a greater distance. While most recent research has addressed the scalability limitations of dense SLAM systems, very few have explicitly tackled these issues.

In this chapter we introduce a novel multi-scale dense SLAM pipeline that supports variable resolution integration and rendering of depth data. As in the systems described in Chapters 3 and 4, our algorithm alternates between fusion of new information coming from the sensor and pose estimation against synthetic views of the scene being reconstructed. Crucially, in both integration and rendering the appropriate voxel resolution is chosen dynamically. Although we use an RGB-D camera as input, our system is sensor agnostic as long as depth is provided. In this work we employ a signed distance function as the implicit surface representation (Section 3.5.3). To represent the volume with variable level of detail (LOD), we further extend supereight and build upon the multi-resolution facilities presented in Chapter 4. Specifically, we propose a new method for selecting the appropriate fusion and rendering scale adaptively, together with a set of algorithmic solutions to guarantee a true multi-scale representation at any level of the octree. As we discuss in the following sections, the main challenge is to keep the hierarchy consistent when fusing at variable scales.

Our pipeline achieves much higher performance compared to single-resolution grids but more importantly it obtains a better reconstruction quality of cluttered scenes with thin structure. In addition for better recovery of map details, the proposed approach substantially improves runtime thanks to more efficient map updates at larger distance. The hierarchical map nature allows for integration of new information up to the selected resolution, while only propagating updates further down when needed, effectively implementing a form of lazy information propagation up and down the octree.

Summarising, our contributions in this chapter are the following:

1. An extension to supereight to support multi-resolution queries at any level of detail;

2. An efficient fusion algorithm that adaptively selects the appropriate

resolution and maintains the surface representation consistent across the hierarchy;

3. A qualitative and quantitative evaluation on synthetic and real datasets, which reveals that we substantially improve map accuracy, as well as runtime.

5.2. Related work

A common trait among the real-time volumetric systems discussed in Section 2.2 is that the underlying grid discretisation has uniform resolution. Even hierarchical approaches, such those of Zeng et al. [2013] and Chen et al. [2013], integrate sensor data at the finest level of the detail and use the tree representation for acceleration purposes. Instead, we want to harness the inherently multi-scale nature of visual information. Off-line algorithms that explicitly take into account the effective pixel resolution have been recently proposed. Fuhrmann and Goesele [2011] introduce an incremental fusion algorithm where each depth map is first triangulated and each triangle is associated with a scale according to its corresponding pixel footprint. The resulting mesh is then used to build a signed-distance function stored in an octree. Multi-scale integration is achieved by fusing triangles only at the corresponding level of the tree given their associated scale. In a subsequent work, Fuhrmann and Goesele [2014] propose a reconstruction algorithm which employs an implicit signed function as surface representation, built from oriented surface samples. The final iso-surface is extracted from discrete sampling positions corresponding to the corners of octree cells. Ummenhofer and Brox [2017] present a global reconstruction approach which uses a finite element scheme to infer a regularised signed distance function and its gradient from a global point cloud. Also in this case, an octree is used to process input points at the appropriate volumetric scale. While achieving impressive reconstruction results, these methods are computationally very expensive and hence unsuitable to be employed in real-time settings, such as augmented/virtual reality applications or mobile robotics.

In the context of real-time methods, our work shares many aspects with the multi-scale reconstruction framework of Steinbrucker et al. [2014], where an octree data-structure is used to store a signed-distance function representation at multiple level of detail. However, no facilities for real-time multi-scale rendering are provided and consequently it relies on an external SLAM system for pose estimation. Kähler et al. [2016] propose a hierarchical hashing system in which the scene is represented with multiple hash-tables each with a different resolution. Interestingly, the level of refinement is chosen according to surface curvature rather than distance from the sensor. Their method is complementary to ours and we plan to unify both approaches in future work. Stückler and Behnke [2014] develop a complete tracking and mapping pipeline using adaptive resolution surfel maps organised in an octree. Recently, Zienkiewicz et al. [2016] propose a reconstruction framework based on 2.5D height-maps with adaptive mesh refinement. While achieving sub-millimetre reconstructions in real-time, their method cannot handle generic 3D shapes, which is limiting it to a small number of application scenarios. However, their incremental coarse-to-fine surface refinement is certainly related to the down-propagation strategies we exploit in our work.

5.3. Overview

From a high-level perspective, we adopt the same dense tracking and mapping pipeline presented in Chapter 3. The computation is structured as a closed loop divided in three main stages:

1. A *tracking* stage, where the camera egomotion is estimated by aligning the most recent depth map $\mathbf{D}_t$ coming from the sensor against a synthetic view of the model $\mathbf{D}_{t-1}$ obtained at the previous time step. This is achieved using a variant of the well-known *Iterative Closest Point* (ICP) algorithm using point-to-plane distance and projective data-association Besl and McKay [1992]. We stress that our reconstruction method is agnostic to the tracking algorithm employed and alternatives exist. A popular choice is to use a sparse SLAM system as the tracking front-end while performing dense fusion, as in Steinbrucker et al. [2014] or Zienkiewicz et al. [2016].

2. An *integration* stage, where sensor data is fused into the volume. Crucially, and in contrast to previous volumetric approaches, we

do this by dynamically selecting the level of detail at which data is fused. Furthermore, we maintain the hierarchy of voxels consistent across octree levels in a subsequent step. We report our method in detail in Section 5.5.

3. A *rendering* stage, where the volume is ray-cast from the most recent camera position and surface points are extracted at the zero-crossing of the TSDF function. The produced renderings are then used for dense ICP tracking and visualisation. Also in this case, we exploit multi-resolution information by selecting the appropriate interpolation scale depending on distance from the camera. We detail our approach in Section 5.6.

In the next sections we present the details of our method. After a brief description of the underlying data-structure and extensions to supereight, we describe our reconstruction pipeline. Finally, we show qualitative and quantitative experimental results on both custom sequences and standard evaluation data-sets.

5.4. Data representation

Fundamentally, as a means to avoid aliasing we store the TSDF values at a selectable resolution – where we have to keep the values consistent. The octree regular subdivision exposes a simple relationship between refinement levels – octant's TSDF values can be expressed as a function (normally the mean) of their children.

As discussed in previous chapters, there is a wide spectrum of possible octree layouts. The simplest solution is to keep the full tree structure explicit, as in Zeng et al. [2013]. While this allows fine grained control over which voxels are allocated, it is subject to significant overheads when it comes to depth integration and ray-casting. To mitigate this, a common solution is to aggregate the last levels of the tree into contiguous blocks of voxels, usually of size 8^3, (Nießner et al. [2013], Kähler et al. [2015], Chen et al. [2013], Vespa et al. [2018]). However, as a consequence the multi-scale representation of the bottom levels is lost. Steinbrucker et al. [2014] address this issue by allocating voxel blocks of the same aggregation factor at *each* level of the tree, effectively resulting in a hierarchy of stacked

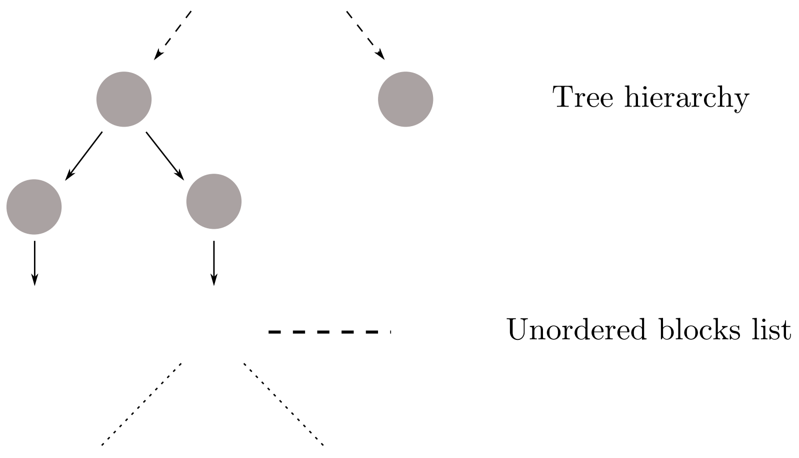


Figure 5.1.: High-level representation of our octree structure. Coarser nodes are connected via pointers up to a maximum depth at which voxels are aggregated in continuous blocks. In this bi-dimensional representation the focused block shows its internal structure, were progressively coarser grids are logically overlaid on top of each other (red, green and blue dots).

bricks. For efficiency reasons, we choose an intermediate approach. We keep an 8^3 aggregation factor at the finest level of the tree, but we also store its mipmapped representation contiguously, i.e. blocks of 4^3 and 2^3 voxels. This is in fact equivalent to instantiating a full octree of three levels for each voxel block, but without the pointers' connectivity overhead and with guaranteed spatial locality. Figure 5.1 depicts the architecture described in the above. An unordered set of voxel blocks is stored in a contiguous, yet dynamically grown array and indexed via a pointer-based tree structure. Voxel blocks are allocated at the deepest level (leaves) of the tree, which also corresponds to the maximum resolution attainable. The magnified voxel block shows its internal structure. Logically, in this two-dimensional 4x4 grid example, a coarser grid (green dots) is overlaid on top of its underlying finer resolution grid (red dots). In practice, we store the grids contiguously one after the other. Compared to single resolution blocks, this implies only a 14% increase in memory footprint, while significantly simplifying block management operations.

5.4.1. Data indexing and retrieval

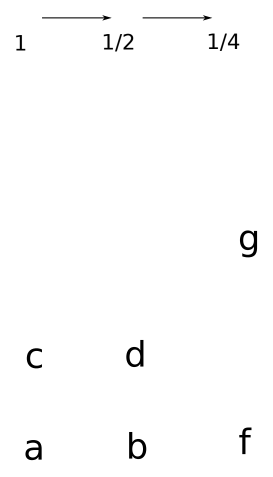


Figure 5.2.: Example of quad-tree with cell-centred sampling locations.

The layout described in the previous section enables voxel retrieval in their corresponding coarser or finer grids efficiently and in a simple manner. While we retain the indexing scheme of Section 4.4, we add the possibility to address voxels in the mip-mapped blocks. Voxels at any scale are denoted by their integer coordinates (3D indices) $\mathbf{x}_w^l$ at the *finest* resolution possible. A scale parameter determines the factor to be applied to retrieve the corresponding entry in coarser grids. If d is the maximum depth of the tree and l the desired voxel depth, the scaling factor is given by $s = 2^{d-l}$, with $d - \log_2(8) \leq l \leq d$. Hence, index coordinates to access the multi-resolution grid at any scale l can be obtained by simple arithmetic as: $\mathbf{x}_w^l = s \left\lfloor \mathbf{x}_w^d / s \right\rfloor$. Also, we distinguish the world frame index coordinate representation $\mathbf{x}_w$ from block index coordinates $\mathbf{x}_b$, where we store the block offset $\mathbf{b}_w$.

Finally, to access a value from memory relative to the dynamically allocated block, we linearise the block coordinate index as

$$i = \text{offset}(s) + \frac{\mathbf{x}_{bx}^{l}}{s} + \frac{\mathbf{x}_{by}^{l}}{s} v^{l} + \frac{\mathbf{x}_{bz}^{l}}{s} {v^{l}}^{2}, \tag{5.1}$$

where offset$(\cdot)$ is a precomputed function that returns the starting position of the local sub-grid and v^l is the number of voxels per side at scale l.

Another significant difference compared to Section 3.4.1 is the adoption of cell-centred voxels: we place the sampling location at the centre of its enclosing octant. While this can be trivially achieved by linear shifting, it has important consequences in terms of local consistency. As shown in Figure 5.2, in a cell-centred grid parent voxels are exactly the linear combination of its children. This is crucial in that it allows node-local exchange of data between tree levels. We exploit this property extensively in the algorithms described in Section 5.5. Notice that if no information is exchanged between octree levels, anchored voxels as used in Chapter 3 allow for easier computation of interpolation points and scale independent cell projection, as sampling locations coincide with voxel coordinates. In our implementation we keep the choice between the two modes parametric, so that the best can be selected for the particular use case.

5.5. Multi-resolution mapping

We structure the field update into two distinct phases. First, the last acquired depth map is fused into the map. In contrast to previous volumetric approaches, this is done by selecting the appropriate scale proportional to the distance from the camera. If the currently selected scale is finer than the scale at which integration was last performed, we *first* do a coarse-to-fine propagation of the available information from parent octants (Section 5.5.3). In a second step, we propagate fine-to-coarse the newly fused information to the coarser nodes (Section 5.5.2). This is done in a *lazy* way: we perform upward and downward propagation only up to the tree levels which are needed for the fusion and rendering operations at the current time step. In the following, we report our method in detail.

5.5.1. Single level update

Each new depth measurement will observe regions of space that possibly have not yet been allocated on the tree hierarchy. Hence, as detailed in Section 4.5.1, we allocate the scene via ray-casting and collect all the intersected voxels around the truncation region $\pm\mu$. However, we select the allocation scale dynamically, such that voxels far away from the camera that do not reach the aggregation layer of the tree will not be allocated as contiguous blocks. Processing the depth frame at time t, the TSDF sample f_t for each allocated voxel at scale l, at position $\mathbf{p}^l_w$, is computed by projecting it into camera frame and taking the signed distance to the corresponding depth measurement, as shown in Equation (5.2), where λ is a factor that transforms a distance along the z-axis to a range distance.

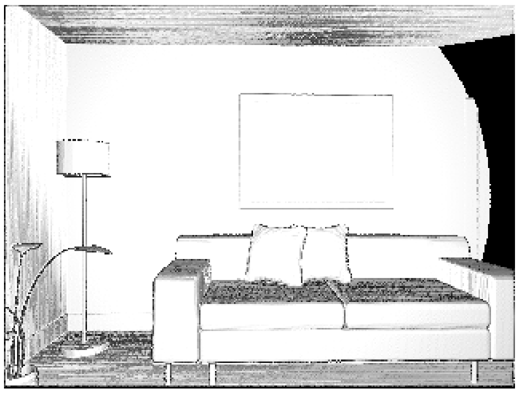
$$\begin{aligned} d &= \lambda(\mathbf{D}_t(\pi(\mathbf{T}_{cw}\mathbf{p}^l_w)) - \mathbf{p}^l_{cz}), \\ f_t(\mathbf{p}^l_c) &= \min\left(1, \frac{d}{\mu}\right), \end{aligned} \tag{5.2}$$

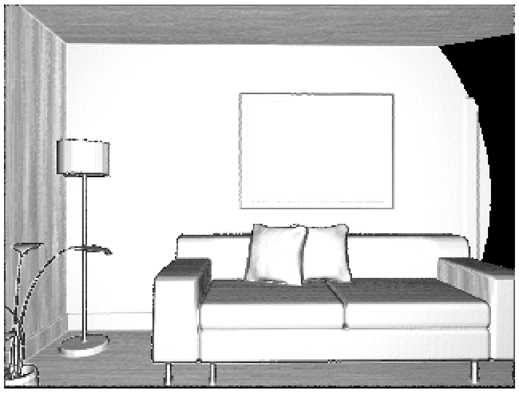
where $\pi(\cdot)$ denotes the projection from 3D coordinates in camera frame to pixels. The current sample is then integrated in the global map F^l_t in an incremental fashion:

$$\begin{aligned} F^l_t &= \max\left(\min\left(\frac{y^l_{t-1}F^l_{t-1} + f^l_t}{y^l_{t-1} + 1}, 1\right), -1\right), \\ y^l_t &= \min(y_{\max}, y^l_{t-1} + 1), \\ \Delta y^l_t &= \Delta y^l_{t-1} + 1, \end{aligned} \tag{5.3}$$

where y^l denotes the weighting and is clamped to a maximum weight of $y_{\max}$ (we use $y_{\max} = 100$) and F^l_{t-1} is either the function value at the previous time step or the down-propagated value in case of integration scale change. Note that we also keep track of the increment on y^l in the form of Δy^l_t, which will be needed later for down propagation. Notice that for compactness we omit the voxel parameter and denote $f_t(\mathbf{p}^l_c) \equiv f^l_t$ when clear from the context.

We are left with the issue of selecting the fusion scale l. We aim to keep the ratio between the back-projected pixel size and the corresponding voxel as close as possible to one. Hence, we select the appropriate resolution proportional to the distance from the camera. In principle, each voxel can be updated at a different scale as long as it is consistent among its siblings. However, this would imply unnecessary overhead when updating the densely allocated voxel blocks and complicate the subsequent up-propagation operations. Instead, we simplify the problem by assigning each *block* a uniform scale. This works very well in practice and it has the main advantage of enabling uniform iteration over the appropriate mipmapped grid (see Section 5.5.2).


(a) 2mm


(b) 4mm

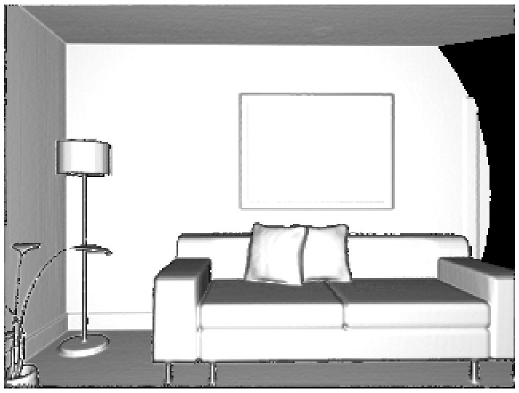

(c) 8mm

Figure 5.3.: Synthetic rendering at run-time using progressively coarser voxel resolutions.

5.5.2. Fine-to-coarse propagation

Once a depth map has been integrated, we propagate the updated information along the hierarchy in the upward direction. As we report in Section 5.6, this is required to guarantee scale consistency amongst neighbouring blocks and octants. Furthermore, we stop the up-propagation at the coarsest scale observed during the current frame, bounding the number of octants to be updated. The cell-centred data-model ensures that each octant is fully described by its children by means of simple linear interpolation:

$$\begin{aligned} F^l &= \frac{\sum_{i=1}^{8} F_i^{l+1}}{8}, \\ y^l &= \frac{\sum_{i=1}^{8} y_i^{l+1}}{8}, \end{aligned} \tag{5.4}$$

where the subscript i denotes the i-th child at scale $l+1$. In the rare event that not all children are initialised, we compute the mean of the initialised ones. Figure 5.3 demonstrates the soundness of our approach. In an artificial experiment, we fuse information at the finest resolution possible and up-propagate. We then render the same frame at progressively coarser voxel resolution via ray-casting, obtaining consistent results. Notice how the reconstruction appears smoother as the voxel size used for interpolation and gradients grows.

5.5.3. Coarse-to-fine propagation

We perform downward propagation of information when the camera moves closer to the surface and requires a finer resolution for depth integration. However, coarse-to-fine propagation is significantly more challenging compared to upward propagation. One possible approach is to tri-linearly interpolate field values from the coarser grid, in a similar fashion to multi-grid methods (Golub and Van Loan [1996]), but this would smooth already reconstructed details which are then revisited. Instead, we take an approach similar to that of Zienkiewicz et al. [2016] in the context of height map reconstruction. In order to preserve details, we propagate to the children octant a fixed delta which represents the difference between the last updated values at the parent scale. In order to enable lazy propagation we need to keep track of all the updates performed in the

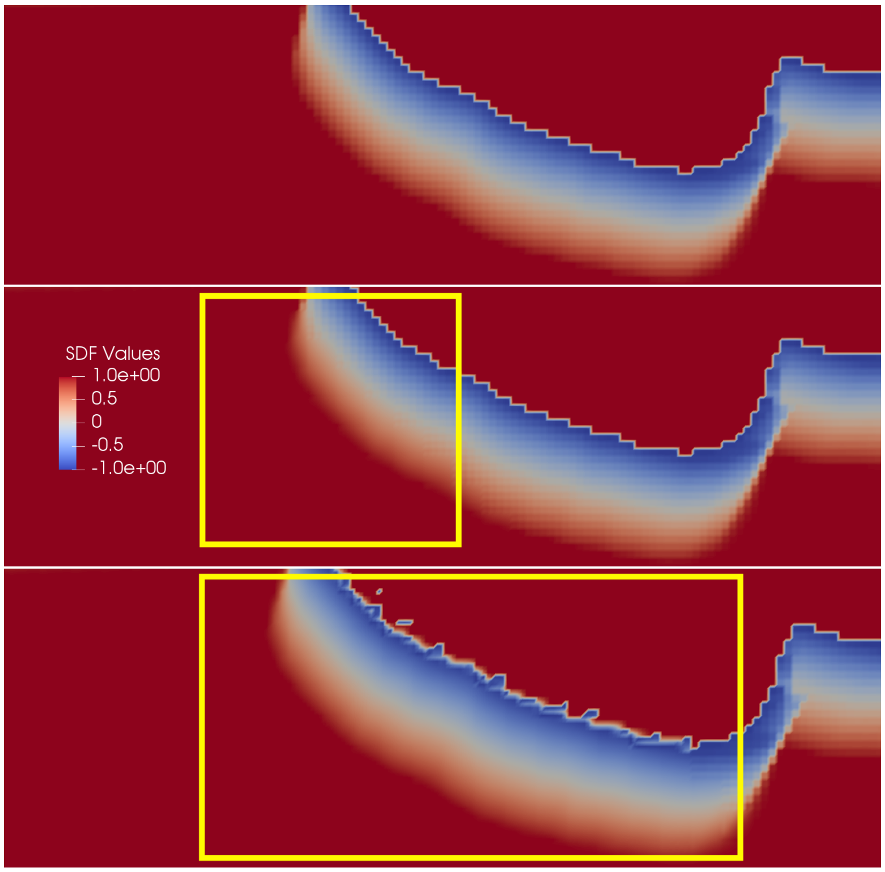


Figure 5.4.: Progressive initialisation of fine resolution voxels from parent octants. Images display a horizontal slice of the TSDF volume ordered top-to-bottom from the older to the newer. Red shades indicate empty and unseen space, salmon-to-white shades denote space in front of the surface and blue shades correspond to back surfaces. Regions of space that appear blocky have not yet been refined at the rendering resolution. Notice how the field is progressively smoothed in proportion to the scale refinement caused by the camera moving closer towards to the surface (highlighted by the yellow boxes).

time interval $[t, \dots, t+n]$. A key observation is that this would be equal to the difference between the children's means $\bar{F}_t^l$ at time t and the last parent value F_{t+n}^l at time $t+n$. Since $\bar{F}_t^l$ is unchanged between consecutive frames in which children's values have not been up-propagated, it holds that $\bar{F}_{t+n}^l = \bar{F}_t^l$. Using similar reasoning for the weight values, we can formulate down-propagation in compact form as:

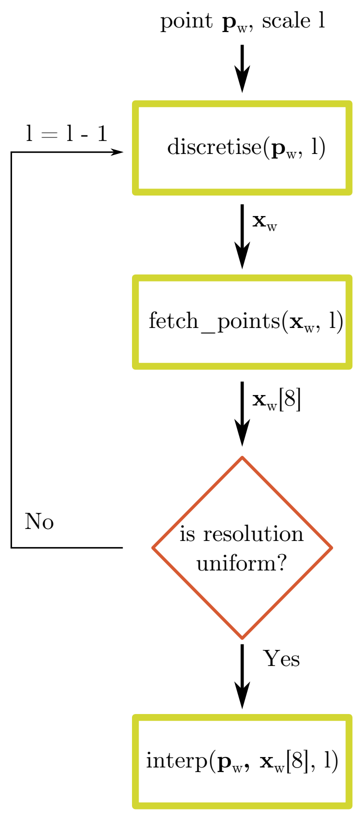
$$
\begin{aligned}
\Delta F_t^l &= F_{t-1}^l - \bar{F}_{t-1}^l, \\
F_{t,i}^{l+1} &= F_{t-1,i}^{l+1} + \Delta F_t^l, \\
y_{t,i}^{l+1} &= \min(y_{\max}, y_{t-1,i}^{l+1} + \Delta y_t^l), \\
\Delta y_t^{l+1} &= \Delta y_{t-1}^l + \Delta y_{t-1}^{l+1}, \\
\Delta y_{t+1}^l &= 0.
\end{aligned}
\tag{5.5}
$$

Finally, to achieve smoother propagation, we enforce scale changes in unitary steps, i.e. the current integration scale can be at most double or half the resolution of the previous used.

One subtle issue that remains is how to deal with voxels at finer scales that have yet to be initialised, since the delta propagation described in this section clearly would not work. Instead, on initialisation we interpolate the values from the parent grids. To keep computation as local as possible, we resort to extrapolation in order to compute values for voxels at the boundaries of contiguous blocks. This avoids complex synchronisation issues to guarantee that the neighbouring blocks are up-to-date and it ensures a smooth initialisation, as we show in Figure 5.4.

5.6. Adaptive-resolution volume rendering

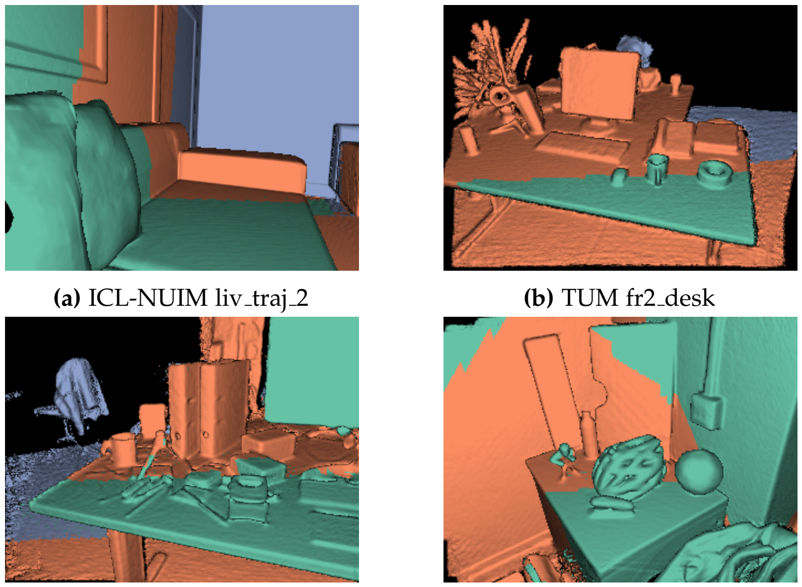
Volume rendering is implemented via the hierarchical traversal algorithm described in Section 3.5.4. Once in the proximity of the surface the ray is marched in steps proportional to the distance from the zero-crossing. At each step, the SDF field is sampled via tri-linear interpolation. Once the ray transitions from positive to negative space the accurate 3D position of the surface is computed as in KinectFusion (Newcombe et al. [2011a]). Our fusion method and delayed propagation impacts the way that interpolation and gradient calculations have to be performed. The key issue is that interpolating points at the boundaries of voxel blocks requires access to neighbouring blocks. However, there is no guarantee that the lastly integrated data has the same resolution. This would mean interpolating between points with variable spacing (i.e. akin to an unstructured grid) and consequently simple tri-linear interpolation rules would not work.



point $\mathbf{p}_w$, scale l
l = l - 1
discretise($\mathbf{p}_w$, l)
$\mathbf{x}_w$
fetch_points($\mathbf{x}_w$, l)
$\mathbf{x}_w[8]$
No
is resolution uniform?
Yes
interp($\mathbf{p}_w$, $\mathbf{x}_w[8]$, l)

Figure 5.5.: Schematic representation of the interpolation point-search algorithm. Given a point $\mathbf{p}_w$ and a scale l, the point is converted to voxel coordinates $\mathbf{x}_w^l$ and the corresponding 8-neighbour is fetched (*fetch* function). If the resolution is uniform among the neighbouring points, interpolation is performed at scale l via the *interp* function, otherwise the search is repeated at a coarser scale.

There are several ways to solve this problem. Kähler et al. [2016] build a linear system and solve it to derive the interpolation coefficients. However, this is computationally expensive. While the occurrence of interpolation across blocks of different resolution should be relatively rare and restricted to the depth regions at which the switch occurs, we opted for a simpler strategy, schematically shown in Figure 5.5. Given a 3D sampling point, we look for its corresponding voxel in the hierarchy at the finest, *most recently* updated resolution. Once we have found the base point in voxel space, we search for the neighbouring points at the *same* resolution. If any of the required points violates the resolution constraint, i.e. its last



(a) ICL-NUIM liv_traj_2 **(b)** TUM fr2_desk

(c) Desk dataset **(d)** Helmet dataset

Figure 5.6.: Example of multi-resolution ray-cast rendering on different scenes. The colour coding indicates the variable resolution used (green finest, orange intermediate and purple coarsest).

updated scale is greater than the current one, we repeat the search process of fetching points at the coarser scale. In other words, we perform the interpolation on the common finest grid between neighbouring voxel blocks. Figure 5.6 shows an example of renderings obtained with the ray-casting strategy detailed in the above. The colour-coding denotes progressively coarser interpolation scales proportionally to the surface distance. Notice how surfaces at the interface of scale changes are smoothly rendered.

5.7. Experimental evaluation

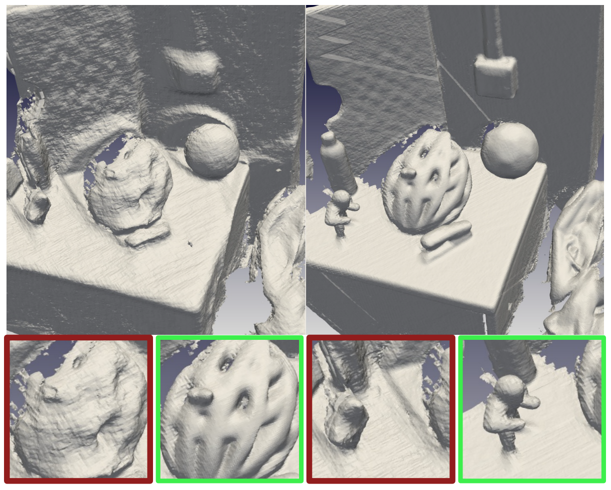
In this section, we report our experimental results. All our tests have been performed on a Intel Haswell i7-4770 CPU at 3.40GHz with 16GB of memory, Ubuntu 16.10 and frequency scaling disabled. The software has been compiled with GCC 8.0 with OpenMP acceleration and -O3 optimisations. We evaluate our dense SLAM system, which we refer to as *multires*, against the single resolution pipeline of Chapter 3, denoted as *baseline*. We have configured both systems with 2mm maximum voxel resolution and 5cm truncation bandwidth. ICP tracking uses a three level image pyramid with 10^{-5} convergence threshold. As the finest resolution, we use QVGA, 320x240 pixels, i.e. downsampling the VGA resolution by a factor of two. In our experiments we have not found any gain in terms of tracking accuracy using full resolution images, while ray-casting one forth of the pixels provides significant speed-ups. We stress that the same set of parameters has been used for *all* the datasets.

5.7.1. Qualitative evaluation

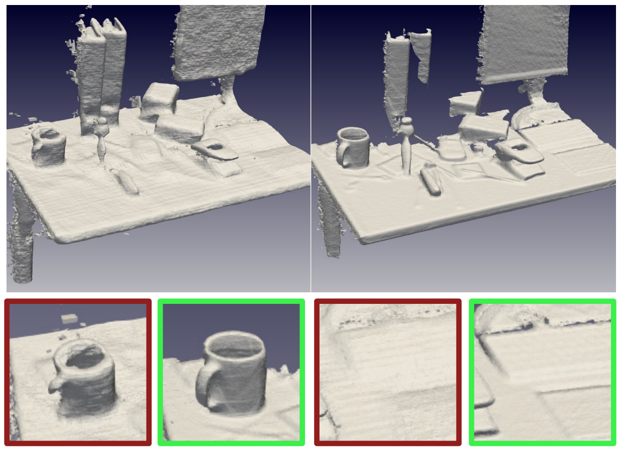
To demonstrate the effectiveness of our multi-scale fusion approach we have recorded a series of sequences with fine structure components, such as scissors, cables, handles and objects with more complex geometry such as a helmet or a drone. The sensor used for this data collection was an ASUS Xtion Pro Live. In all the recorded sequences we have simulated a realistic scanning scenario, where first the scene is observed closely and then the camera slowly moves away to scan other parts of the environment. We believe this kind of scenario is particularly relevant for instance in case of augmented reality (AR) applications. It is common that in a bootstrap phase, the user is required to scan the surrounding environment. Then virtual characters and objects may be placed on desktops or cluttered scenes and have to navigate and interact with the map in an accurate way. Similarly, the scenario is very relevant in robotic exploration, e.g. using a drone, with the aim of accurately reconstructing an indoor space. Figures 5.7a and 5.7b compare the output of our novel reconstruction pipeline against a traditional single-resolution system. Both meshes are extracted via marching cubes (Curless and Levoy [1996]) at the maximum available resolution. Our method is able to preserve the fine structure details and furthermore provides a smoother estimate of planar surfaces. We attribute this to the reduction of aliasing and better smoothing of sensor noise at coarser resolutions.

5.7.2. Tracking and reconstruction accuracy

We evaluated the tracking accuracy of our pipeline on standard synthetic (ICL-NUIM Handa et al. [2014]) and real (TUM RGB-D Sturm et al. [2012])



(a) Helmet sequence



(b) Desk sequence

Figure 5.7.: Reconstruction comparison between a single resolution pipeline (left and in red) and our multi-resolution approach (on the right and in green) on the helmet and desk sequences.

data-sets using the SLAMBench framework (Nardi et al. [2015]). In this quantitative evaluation we also show accuracy and run-time performance results of InfiniTAM (Kähler et al. [2015]), the current state-of-the-art voxel hashing implementation. For the sake of a fair comparison, we have configured InfiniTAM with the same algorithmic parameters of our two pipelines, including tracking parameters. Table 5.1 shows the absolute

	ATE (m)		
Dataset	Baseline	Multires	InfiniTAM
ICL_LR_0	0.1315	**0.0031**	0.1384
ICL_LR_1	**0.0024**	0.0061	0.0044
ICL_LR_2	0.0083	**0.0052**	0.0065
ICL_LR_3	0.1213	**0.0539**	0.1071
TUM_fr1_xyz	**0.0137**	0.0140	0.0273
TUM_fr1_desk	0.0633	0.0622	**0.0492**
TUM_fr2_desk	**0.0838**	0.0853	0.0887
TUM_fr3_office	**0.0226**	0.0227	0.1022

Table 5.1.: Absolute trajectory error (ATE) comparison between our multi-resolution algorithm (multires), a single resolution pipeline (baseline) and voxel hashing (InfiniTAM).

	RMSE (m)		
Dataset	Baseline	Multires	InfiniTAM
ICL_LR_0	0.0532	**0.0054**	0.0541
ICL_LR_1	**0.0048**	0.0080	0.0057
ICL_LR_2	0.0051	**0.0048**	0.0049
ICL_LR_3	0.0568	**0.0110**	0.0547

Table 5.2.: Reconstruction accuracy comparison between our multi-resolution algorithm (multires), a single resolution pipeline (baseline) and voxel hashing (InfiniTAM) in terms of the root mean squared distance from the ground truth mesh.

trajectory error (ATE) across different sequences from each data-set. With regard to accuracy, our multi-resolution system achieves same or better results than the single resolution grids. This is particularly noticeable on synthetic sequences from the ICL-NUIM dataset. We attribute this to the fact that being noise-free, discretisation artefacts stemming from aliasing are a major source of inaccuracy, while on real data sensor noise becomes predominant. Also, as expected, voxel hashing accuracy is in line with our single resolution pipeline.

We validated our reconstruction approach against the synthetic ground truth of the ICL-NUIM dataset (Table 5.2). As shown in Table 5.2 our multi-resolution beats the baseline in virtually all cases. It also achieves the same accuracy levels as voxel hashing, however it should be noted that the results reported in Kähler et al. [2015] are slightly superior to what we report, but were obtained at VGA resolution and at coarser voxel resolution, while for fairness we used the same voxel and input resolu-

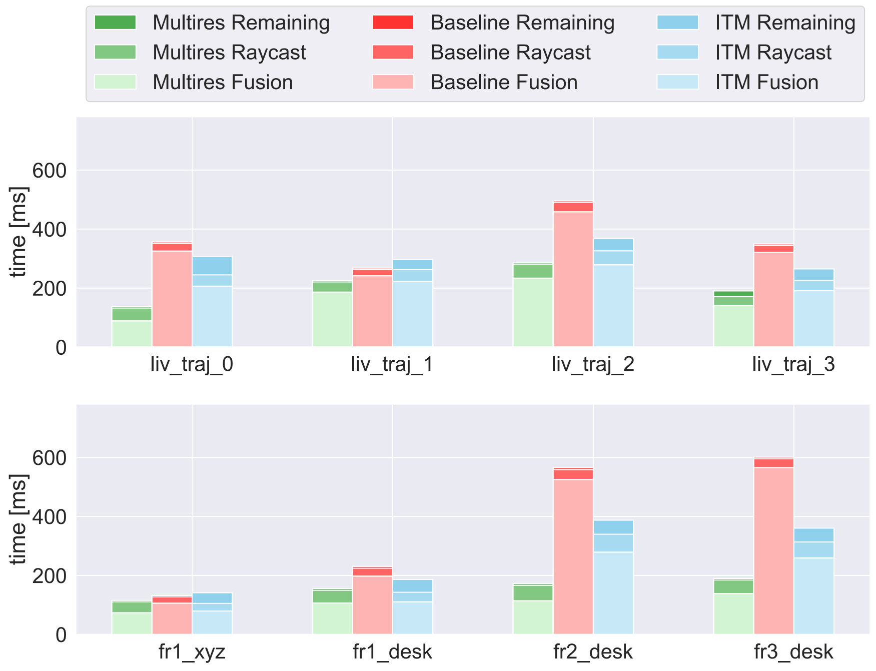


Figure 5.8.: Runtime comparison of our multi-resolution reconstruction pipeline (multires), single-resolution grid (baseline) and voxel hashing (ITM). The top row indicates timings for the ICL-NUIM synthetic benchmark while the bottom row for the TUM RGB-D real dataset.

tions (2mm, QVGA) across all experiments. Also note that we manually tuned various InfiniTAM parameters such as the truncation distance and found that best results were obtained when using the same parameters as our approach. Finally, we highlight that these sequences are not the best to showcase the merits of a multi-resolution reconstruction approach, as most surfaces are observed at uniform scale and the scene lacks thin structure components. Consequently, given the same tracking accuracy, it is expected to have a reconstruction precision very close to single resolution grids.

5.7.3. Runtime performance

The timings shown in Figure 5.8 demonstrate how our multi-resolution approach brings reductions in computational cost, achieving up to 3x higher frame rate compared to the baseline and reaching consistently better performance compared to voxel hashing. This is due to the combination of our dynamic resolution fusion, as it reduces the number of voxels updated per frame according to the distance from the camera, and the delayed propagation of newly fused information. The timings difference between our single resolution pipeline and InfiniTAM are attributable to the deeper hierarchy required to reach the fine resolution used in this experiments. It is also worth observing that the difference is negligible in sequences where short range views are predominant, such as liv_traj_1, fr1_xyz and fr1_desk. Notice that rendering is faster when using a single resolution grid. In our current implementation, the extra control logic required by the interpolation scheme described in Section 5.6 induces a non-negligible execution time penalty compared to the simpler interpolation algorithm usable in case of uniform scale. Overall, the system presented in this chapter achieves frame rates between 5hz and 10hz on a commodity CPU, while providing very fine scale reconstructions. This is a strong indication that the method is scalable and if ported to GPU accelerators could reach levels of performance far exceeding the camera's frame rate of 30fps even at the full VGA resolution.

5.8. Conclusions

We have presented a method for the online volumetric integration of depth images at adaptive levels of detail. Our system dynamically selects the best fusion and rendering scale to match the sensor resolution and propagates up and down the octree hierarchy as required in a lazy fashion, guaranteeing reconstruction consistency and significantly improved runtime performance compared to equivalent single-resolution grids.

There are several directions in which we plan to extend our work. The surface refinement criteria based on local curvature proposed by Kähler et al. [2016] allows for significant reduction of memory usage in correspondence of flat surfaces. We believe that this can be combined with our distance-based criteria and yield significantly better performance while preserving reconstructed details. Another interesting direction could be to further increase the computational and accuracy performance by avoiding to update confident cells when observed from larger distances. However, discarding information could be undesirable in the case of movement of previously static objects, hence more complex change detection mechanisms would be required. Finally, we also plan to port our approach to occupancy mapping, specifically extending the formulations introduced in Chapter 4. This would allow us to better take into account sensor noise and information exchange between octree levels in a probabilistically sound framework.

Chapter 6

Conclusions

In this chapter, we review the main contributions of this thesis. We then highlight its principal limitations and discuss future research directions.

6.1. Summary

The contributions of this work are centred around two main themes: i) the design and implementation of a high-performance sparse octree library for volumetric tracking and mapping; ii) the research of novel hierarchical algorithms and representations for volumetric SLAM pipelines. We have developed these themes in each chapter of this thesis, which we highlight in the following:

- In Chapter 3, we introduce our novel octree library, named *supereight*. The main principles behind its design are flexibility and performance. We achieve the former via generic interfaces which allow the end-users to experiment freely and with minimum intervention with different underlying surface representations. Performance is guaranteed via a careful choice of parallel algorithms and novel optimisations that particularly target the operations common to volumetric SLAM pipelines. Importantly, we quantitatively demonstrate the effectiveness of our approach against the current state-of-the-art, showing how our hierarchical representation achieves performance on-par with voxel hashing while providing a complete spatial index of the scene.

- In Chapter 4 we introduce a novel dense SLAM pipeline based on occupancy probabilities. We extend the Bayesian fusion framework of Loop et al. [2016] to work under prolonged mapping sessions and employ it as an underlying surface representation. We then introduce a complete SLAM pipeline which employs supereight as a volumetric data-structure, and demonstrate a fully integrated system on which tracking, mapping *and* spatial occupancy queries can be performed seamlessly on the same representation, a desirable feature for several down-stream applications such as robotic path planning. Finally, we show how our system is considerably faster than the current de-facto standard in occupancy mapping and offers comparable performance to TSDF-mapping.

- In Chapter 5 we leverage the multi-scale representation of our octree framework to perform resolution-aware tracking and mapping. We introduce a novel mapping algorithm based on a multi-scale TSDF representation which adaptively select the appropriate fusion scale according to the effective measurement resolution. Crucially, and in contrast to previous work, we provide a novel algorithm to preserve the consistency across octree levels, which propagates information in a lazy fashion. This allow us to achieve much higher mapping performance, but perhaps more importantly can preserve fine structure details when observing a scene from farther distances. We also use the multi-scale information when rendering the synthetic depth maps for frame-to-model alignment, effectively reducing aliasing by dynamically matching the sensor resolution. We demonstrate experimentally how this is beneficial in terms of tracking accuracy and reconstruction quality.

6.2. Limitations

While particular limitations have been discussed in each chapter, here we focus on what perhaps are the common limitations among all the reconstruction methods described in this thesis. At the present time, supereight does not support loop-closure detection and correction, a feature which is of fundamental importance when mapping over long trajectories and large scale environments. On one hand, this is an intentional choice, as the focus of this thesis is on the building blocks of volumetric mapping, both in the form of efficient data-structures and novel field representations. In this sense, loop-closure can be tackled as a separate issue and added as a separate stage to the pipelines presented in this work. On the other hand, volumetric loop-closure correction is not a trivial task, and would require specific research effort. More specifically, we believe it would be particularly interesting to investigate correction mechanisms that preserve the multi-scale representation presented in Chapter 5.

Another aspect which is not treated in this thesis, but that would be straightforward to implement, is the usage of additional sensors in the odometry estimation, such as colour images for photometric alignment and inertial measurement unit (IMU) measurements. In particular, IMU readings may greatly increase the robustness and precision of the ICP alignment, as shown for example in the dense surfel-based system of Laidlow et al. [2017]. This is mainly due to the additional information that can be exploited when depth is non-informative, for instance when observing flat surfaces with little structure. In terms of colour integration, we are interested in evaluating if our multi-scale fusion approach is beneficial in reducing colour artefacts stemming from aliasing, and hence improving frame-to-model RGB-D alignment.

From an algorithmic standpoint, the set of operations efficiently supported by our library is somewhat limited. While parallel projective functors and trilinear interpolation routines are sufficient to express most dense volumetric SLAM pipelines (notice, with either octrees or voxel hashing as concrete data-structure), more complex operations such as arbitrary stencils are not supported yet. We believe that supporting such access patterns efficiently will be of great importance. For example, three-dimensional convolutional networks such as OctNet (Riegler et al. [2017]) or O-CNN (Wang et al. [2017]) exploit the tree structure to perform efficient convolution operations only in region of space which contain useful data. We believe that the access pattern optimisations presented in Section 3.4.4 can be generalised and benefit this kind of computations.

6.3. Further work

The work presented in this thesis leaves many doors open for future research directions, which we discuss in the following.

Loop-closure for volumetric reconstruction There are various approaches that can be taken in order to incorporate loop-closure corrections in volumetric grids. In the literature, two main approaches have been proposed. Dai et al. [2017] propose a complex global optimisation strategy that considers the *entire* history of input frames. Consecutive frames are partitioned into fixed size chunks which are locally optimised. A sparse inter-chunk alignment is then performed to recover globally consistent poses for all the input frames. Similarly to Nießner et al. [2013], a TSDF-based model is maintained and continuously updated. Crucially, in order to enforce global consistency depth data is both integrated and *de-integrated* upon pose correction. This is possible as all the input frames and poses are retained. While able to achieve impressive results, the method is computationally very expensive, requiring two high-end GPUs to reach real-time performance.

A different approach is taken by Kähler et al. [2016]. Again, a TSDF-based map is constructed from RGB-D data. The global map is partitioned into sub-maps which are spawn according to a visibility criteria. Upon loop-closure detection, sub-maps are globally aligned via pose-graph optimisation. Contrarily to the method of Dai et al. [2017], voxel data is *not* re-sampled, but rather reconstructed on-the-fly by combining TSDF values from different sub-maps which are simultaneously visible. The system is able to achieve real-time performance exploiting high-end GPU hardware.

It is clear that the current state-of-the-art is too computationally demanding for a variety of scenarios where volumetric loop closure is desired. Furthermore, is not clear if the sub-mapping approach is able to handle local loop-closures in a way that methods such as the one of Whelan et al. [2015b] are able to correct. We believe that an interesting research direction would be to investigate non-rigid surface corrections in volumetric maps. The idea would be to actually warp the signed-distance field instead of rigidly transforming the sub-maps. Doing this efficiently is still an open research problem. One of the main issues is to identify which areas are affected by the map correction and apply the warp consequently. For instance, in Kintinuous (Whelan et al. [2015a]) the TSDF is first converted to a compact mesh representation and then non-rigidly deformed, but no correction is propagated back to the volumetric representation, meaning that upon revisiting the TSDF information is not consistent with the global mesh. BundleFusion (Dai et al. [2017]) works around this issue maintaining the whole history of integrated frames, such that upon loop closure information can be de-integrated and re-integrated from the newly optimised camera poses. However this approach is clearly not scalable. Hence, we aim at investigating novel approaches in future research.

Domain-specific languages for dense SLAM As often highlighted in this thesis, real-time performance is a strong requirement of modern SLAM systems. In practice, given the pipelines' complexity and the many variables that influence both computational and algorithmic performance, this is often *very* hard to achieve. Obtaining maximum accuracy and target frame-rates force SLAM developers to carefully select algorithmic parameters and low-level design choices. This comes at huge engineering cost. First, the resulting implementation may be very rigid and hard to change or adapt to different algorithmic scenarios. For instance, a reconstruction pipeline as the one described in Chapter 3 may also fuse into the map other types of information, such as colour or semantics. Even if superficially this could seem a simple addition, the final, optimised code might differ drastically from one version to another. This issues are exacerbated by the diversity of the possible target execution platforms: a GPU powered mobile device may have very different architectural constraints from a self-driving car or a flying drone. Ideally, one should be able to express algorithmic ideas at a higher level of abstraction, closer to the actual meaning of the computation, and let the tool-chain generate optimised code for a given execution platform. The need for better programming abstractions has driven the scientific community into developing *domain-specific languages* (DSLs) and active libraries. DSLs provide constructs closer to the application domain, considerably reducing development time but still achieving competitive performance by intelligently generating target-specific code in the back-end. As an example, the Eigen library may be seen as a linear-algebra DSL embedded in C++. Behind the C++ interface, Eigen's back-end constructs a chain of expression templates which are then used to generate at compile time optimised code for a variety of CPU platforms.

In our work, we moved the first steps towards defining higher-level abstractions. The fusion and interpolation operators defined in Sections 3.4.3 and 3.4.4 were designed with this goal in mind: to allow the SLAM developer to freely experiment with different field representations without having to change the underlying algorithms implementation. Expressiveness is not the only desirable feature of a domain-specific language or library. Performance has to be competitive with hard-coded, hand-tuned implementations. While our operators exploit Eigen's vectorisation capabilities, much can be done to significantly improve the library's performance. In terms of low-level optimisations, we highlight the following key opportunities:

1. Memory layout transformations. Our framework offers a simple interface to the user in order to specify the desired data-type. Unfortunately, the resulting memory organisation is quite rigid. Since the C++ language does not offer any facility for static reflection out of the box, we store the user-specified data-types as packed data-structures, a layout that is commonly known as *Array of Structures* (AoS). However, this may be sub-optimal in a variety of cases. First, SIMD parallelisation greatly benefits from data stored unpacked and contiguously in memory, known as *Structure of Array* (SoA) layout (see Listing 6.1 for an example). Contiguous vector loads and saves offer much better performance as opposed to *gather/scatter* operations.

   To better understand the implications of the memory layout, let us consider an example. The fusion rules detailed in Equation 3.11 require reading both the SDF and weight values for each voxel. For simplicity, let's assume that a vector architecture supports 256 bit registers, each with 32 bit lanes. We can than load 8 SDF values and 8 weights in two different registers and perform the block averaging in parallel. However, if the data is stored in an AoS layout, the load (and symmetrically the save) operations are not so trivial. A single

```cpp
struct AoS {
  float x;
  float y;
};
AoS data_aos[N];

struct SoA {
  float x[N];
  float y[N];
};
SoA data_soa;
```

Listing 6.1: AoS vs SoA memory layout

load instruction will fill a vector register with interleaved data that ultimately need to be unpacked and copied to the target register. Alternatively, data can be loaded from memory with a non-unitary stride access, but it would be sub-optimal compared to contiguous memory accesses. If instead a SoA layout is used, contiguous voxel data can be simply loaded into registers with two contiguous loads, without any extra instruction or memory access overhead.

Independently from the data-structure layout, we can also consider the order in which 2D or 3D arrays are stored. Common choices are either *row-major* or *column-major* orders, but other possibilities exist, such as *blocked* or *Morton* order (see Thiyagalingam et al. [2006] for a thorough discussion). The latter in particular may be beneficial for the interpolation procedures described in Section 3.4.4, as adjacent voxels in 3D space would be stored contiguously in memory.

2. Whole pipeline vectorisation. At the moment, supereight exploits explicit vectorisation only in the geometric back-end. User-defined kernels, such as the one provided to the `projective-map` operators (Section 3.4.3) are not explicitly vectorised. Even with the use of automatic vectorisation tools such as OpenMP's `#pragma omp simd` the success is not granted, as typical kernels are not trivial to auto-vectorise. Rendering is another obvious area in which vectorisation may be beneficial, but the complexity of hierarchical ray-casting makes it very challenging. Regardless of these difficulties, we know that whole-pipeline vectorisation brings significant performance increases, as we have demonstrated in Nica et al. [2018] for a standard

KinectFusion pipeline.

3. Loop transformations. An important class of compiler optimisations is that of loop transformations. For instance, given a series of nested loops, the order of two or more loops in the loop nest can be *exchanged* in order to expose parallelism or increase data-locality. Consecutive loop-nests may be *fused* together to achieve better data reuse and minimise looping overheads. There exist a plethora of possible transformations, we refer the reader to the survey of Bacon et al. [1994] for an in depth discussion. For a concrete example of scenario where these may be beneficial, we can consider the color fusion algorithm of Whelan et al. [2015a]. In their system, a TSDF map is built together with colour information. To reduce artefacts, each voxel is projected into the corresponding colour image and a colour sample is obtained by averaging the 7x7 neighbourhood of the pixel the voxel projects to. It is clear as in this scenario a proper reordering of the iteration over voxels may increase reuse of image data, as spatially close voxels are likely to project to the same image area. Hence, this kind of computation may take advantages of techniques such as *loop tiling*. Similarly, when ray-casting the TSDF volume (Section 3.5.4), tiling the pixel iteration space is advantageous, as we have shown in Nica et al. [2018]. Regardless which particular transformation is appropriate for a given loop-nest or sequence, we believe that this space should be explored automatically and facilitated as much as possible.

Once identified key optimisations, one has to face the question of how to deliver them. One obvious choice is to commit to one set of optimisations and code those by hand. However, we argue that is neither scalable nor desirable. First the process of producing hand-optimised code is tedious and most importantly error prone. Exploring the whole design-space is simply not feasible, hence the developer has to rely on his experience and intuition. In contrast, auto-tuning tools have been demonstrated to be very effective in finding design points which meet or exceed performance targets. In Zia et al. [2016] and Nardi et al. [2017] we have demonstrated the effectiveness of said approach on different pipelines, where we explored the design-space of algorithmic parameters, compiler optimisations and execution platforms. However, we believe that even more interesting performance can be reached by exploring alternative algorithm *implementations*. The effects of combining data-layout transformations, loop reordering and parallelisation are not trivial to predict and capture in a cost model.

We believe domain-specific languages are very attractive tools that ease most of the difficulties highlighted in the above. Automatic code generation relieves the application developer from writing low-level code, and at the same time exploration of the design space is hugely facilitated by it. While we do not want to provide a thorough survey on DSLs in this section, we believe it is worthwhile highlighting few success stories and how these are related to the work done in this thesis. Image processing DLSs have been extensively researched in the past and have been demonstrated particularly effective. Cornwall et al. [2009] proposed an active library targeting task-graph computations on images. The main idea behind this work is to enable powerful optimisations by means of metadata describing parallelism and data access patterns, effectively eliminating the need of complex data-dependence analysis by the general purpose compiler. This enable the run-time code generator to implement aggressive inter-component optimisations, such as loop fusion and array contractions based on execution-dependent conditions.

Ragan-Kelley et al. [2013] introduce Halide, a functional language to express image processing pipelines embedded in C++. Image manipulation is usually implemented as a series of interconnected passes, which may consist of regular stencil operations or complex reductions applied to image data. Similarly to Cornwall et al. [2009], stencils are expressed as pure functions on 2D coordinates domains. In this perspective, a multi-stage pipeline is nothing but a chain of functions, effectively hiding the data manipulation at the loop level. Optimised code is then generated at run-time, and various architectural targets are supported (such as SIMD CPUs, GPGPUs or OpenGL). Interestingly, Halide comes with an auto-tuner which permits to explore the performance of different optimisation combinations. They demonstrate that the inferred configurations can run up to 5x faster than hand-optimised code by domain experts. It is evident that DSLs like these may be very well suited to express part of the SLAM pipelines discussed in this thesis, such as image de-noising or pyramid creation (Section 3.5.2).

Image manipulation is only one aspect of virtually every SLAM algorithm. Optimisation obviously plays a central role as ego-motion and structure computation are formulated as non-linear optimisation problems. DeVito et al. [2017] develop a domain-specific language for expressing and solving non-linear least squares minimisation in the graphics domain, appropriately named the Opt language. Interestingly, they demonstrate its applicability to the kind of problems often encountered in real-time SLAM, such as pose-graph optimisation or surface deformation. In a nutshell, their framework allow the specification of high-level energy functions, in a formalism very close to the actual mathematical formulation, and automatically generate the code for the non-linear solver. Crucially, they demonstrate how a generative approach offers much better performance compared to state-of-the-art solver libraries.

There are few examples of DSLs for volume processing, such as Diderot (Chiw et al. [2012]) or Vivaldi (Choi et al. [2014]). However, none of them supports sparse volumes, making their adoption in volumetric SLAM unfeasible. We believe that there are significant opportunities for further research in this space. One thing that is clear to us is that any approach that tries to find a single, unifying abstraction to express SLAM algorithms as a whole is very likely destined to fail. The computational structure is too diverse and sub-stages operate on very heterogeneous data. Instead, we advocate the use of multiple DSLs that address different aspects of the pipelines. In our opinion, a combination of an optimisation-oriented high-level language such as Opt (targeting numerical optimisation stages) and a more flexible, stencil-based language for mapping and rendering would offer sufficient expressiveness. Specifically, we believe that a DSL such as Halide, combined with the abstractions defined in Chapter 3, would provide an attractive programming model and competitive performance for a variety of volumetric algorithms. However, as demonstrated in this thesis, efficient support for sparse data processing is of paramount importance. To this end, we have confidence that supereight would be an excellent candidate as a back-end data-structure.

Appendices

Appendix A

Morton encoding and decoding functions

In our implementation, we compute the Morton number of three-dimensional integer coordinates using the magic numbers method. In this appendix we review how this works. As explained in Section 3.4.1, the goal is to interleave the bits from each coordinate component into a single number. To do so, each component has to be *dilated* or *expanded* in order to make space for the bits belonging to the others. Listing A.1 shows the dilation operation.

```
1 inline uint64_t expand(const uint64_t value) {
2   uint64_t x = value & 0x1fffff;
3   x = (x | x << 32) & 0x1f00000000ffff;
4   x = (x | x << 16) & 0x1f0000ff0000ff;
5   x = (x | x << 8)  & 0x100f00f00f00f00f;
6   x = (x | x << 4)  & 0x10c30c30c30c30c3;
7   x = (x | x << 2)  & 0x1249249249249249;
8   return x;
9 }
```

Listing A.1: Expands a 21bit integer into a 64bit integer

It is quite interesting to see in detail how these masking operations affect the bit pattern of a given input. Listing A.2 shows the effect of each operation of the expand function on the 20bit number 0xFFFFF. The first 21bits of the input number are extracted at line 1. This is to ensure that if the number exceed the 21bits limit, there will not be leftover bits in the dilated integer. The bit groups are then iteratively split in half by

```
1 inline uint64_t expand(const uint64_t value = 0xFFFFF) {
2 ____________________________________________11111111111111111111
3 ____________1111________________________________1111111111111111
4 ____________1111________________11111111________________11111111
5 ____________1111________1111________1111________1111________1111
6 ________11____11____11____11____11____11____11____11____11____11
7 ______1__1__1__1__1__1__1__1__1__1__1__1__1__1__1__1__1__1__1__1
8 }
```

Listing A.2: Expansion by magic numbers of integer 0xFFFFF, visualised. Underscores correspond to zeros in the bit strings and each line shows the effect of the corresponding operation in Listing A.1

a left shift and a bitwise OR operation. Duplicate bits are removed by masking the resulting value against the corresponding magic number. In total, five iterations are required to dilate the bits such that each one is 2 bits apart from the next, as shown in Line 7 of Listing A.2. Consequently, a Morton number can be constructed by computing the dilated version of each coordinate and combining them via bitwise OR, in the following way:

Listing A.3: Construct a Morton number given three integer coordinates.

```
inline uint64_t compute_morton(uint64_t x, uint64_t y, uint64_t z)
    {
  uint64_t code = 0;
  x = expand(x);
  y = expand(y) << 1;
  z = expand(z) << 2;
  code = x | y | z;
  return code;
}
```

Notice how, by respectively left-shifting of one and two positions the bits from the *y* and *z* coordinates, they end up in the free slots left by the dilate operation.

Conversely, a Morton number can be de-linearised by applying the inverse transformations, as shown by Listings A.4 and A.5.

```
inline uint64_t compact(const uint64_t value) {
  uint64_t x = value & 0x1249249249249249;
  x = (x | x >> 2)   & 0x10c30c30c30c30c3;
  x = (x | x >> 4)   & 0x100f00f00f00f00f;
  x = (x | x >> 8)   & 0x1f0000ff0000ff;
  x = (x | x >> 16)  & 0x1f00000000ffff;
  x = (x | x >> 32)  & 0x1fffff;
  return x;
}
```

Listing A.4: Compact a 64bit integer into a 20bit integer.

Listing A.5: Extract individual 3D coordinates from a Morton number.

```
inline auto unpack_morton(uint64_t code){
  return std::make_tuples(compact(code),
                          compact(code >> 1),
                          compact(code >> 2));
}
```

Bibliography

D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Comput. Surv.*, 26(4):345–420, Dec. 1994. ISSN 0360-0300. doi: 10.1145/197405.197406. URL `http://doi.acm.org/10.1145/197405.197406`.

M. Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Publishing Company, Incorporated, 2012. ISBN 3642310451, 9783642310454.

J. Bédorf, E. Gaburov, and S. Portegies Zwart. A sparse octree gravitational n-body code that runs entirely on the gpu processor. *J. Comput. Phys.*, 231(7):2825–2839, Apr. 2012. ISSN 0021-9991. doi: 10.1016/j.jcp.2011.12.024. URL `http://dx.doi.org/10.1016/j.jcp.2011.12.024`.

P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, Feb. 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL `http://dx.doi.org/10.1109/34.121791`.

M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, Sep. 2015. doi: 10.1109/IROS.2015.7353389.

B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle. Slambench2: Multi-objective head-to-head benchmarking for visual slam. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, May 2018. doi: 10.1109/ICRA.2018.8460558.

C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011. doi: 10.1137/100791634.

M. Burtscher and K. Pingali. An efficient CUDA implementation of the tree-based barnes hut n-body algorithm. In *GPU Computing Gems Emerald Edition*, pages 75–92. Morgan Kaufmann, 2011. ISBN 978-0-12-384988-5. URL `http://iss.ices.utexas.edu/Publications/Papers/burtscher11.pdf`.

E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3D reconstruction using signed distance functions. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013. doi: 10.15607/RSS.2013.IX.035.

D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal. Sdf tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3671–3676, Nov 2013. doi: 10.1109/IROS.2013.6696880.

R. O. Castle, G. Klein, and D. W. Murray. Wide-area augmented reality using camera tracking and mapping in multiple regions. *Computer Vision and Image Understanding*, 115(6):854 – 867, 2011. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2011.02.007. URL `http://www.sciencedirect.com/science/article/pii/S1077314211000701`.

J. Chen, D. Bautembach, and S. Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4):113:1–113:16, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2461940. URL `http://doi.acm.org/10.1145/2461912.2461940`.

C. Chiw, G. Kindlmann, J. Reppy, L. Samuels, and N. Seltzer. Diderot: A parallel dsl for image analysis and visualization. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 111–120, New York, NY, USA, 2012.

ACM. ISBN 978-1-4503-1205-9. doi: 10.1145/2254064.2254079. URL `http://doi.acm.org/10.1145/2254064.2254079`.

H. Choi, W. Choi, T. M. Quan, D. G. C. Hildebrand, H. Pfister, and W. Jeong. Vivaldi: A domain-specific language for volume processing and visualization on distributed heterogeneous systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2407–2416, Dec 2014. ISSN 1077-2626. doi: 10.1109/TVCG.2014.2346322.

J. L. Cornwall, L. Howes, P. H. Kelly, P. Parsonage, and B. Nicoletti. High-performance simt code generation in an active visual effects library. In *Proceedings of the 6th ACM Conference on Computing Frontiers*, CF '09, pages 175–184, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-413-3. doi: 10.1145/1531743.1531772. URL `http://doi.acm.org/10.1145/1531743.1531772`.

B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. *SIGGRAPH 96 Conference Proceedings*, pages 303–312, 1996. ISSN 00978930. doi: 10.1145/237170.237269.

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 Trans. Graph.*, 36(4), May 2017. ISSN 0730-0301. doi: 10.1145/3072959.3054739. URL `http://doi.acm.org/10.1145/3072959.3054739`.

A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6): 1052–1067, June 2007. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1049. URL `http://dx.doi.org/10.1109/TPAMI.2007.1049`.

Z. DeVito, M. Mara, M. Zollöfer, G. Bernstein, C. Theobalt, P. Hanrahan, M. Fisher, and M. Nießner. Opt: A domain specific language for non-linear least squares optimization in graphics and imaging. *ACM Transactions on Graphics 2017 (TOG)*, 2017.

E. Eade and T. Drummond. Scalable monocular slam. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 469–476, June 2006. doi: 10.1109/CVPR.2006.263.

A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, June 1987. ISSN 2374-8710. doi: 10.1109/JRA.1987.1087096.

A. Elfes and L. Matthies. Sensor integration for robot navigation: Combining sonar and stereo range data in a grid-based representataion. In *26th IEEE Conference on Decision and Control*, volume 26, pages 1802–1807, Dec 1987. doi: 10.1109/CDC.1987.272800.

F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *2012 IEEE International Conference on Robotics and Automation*, pages 1691–1696, May 2012. doi: 10.1109/ICRA.2012.6225199.

J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 1449–1456, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4799-2840-8. doi: 10.1109/ICCV.2013.183. URL `https://doi.org/10.1109/ICCV.2013.183`.

J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, volume 8690 of *Lecture Notes in Computer Science*, pages 834–849. Springer International Publishing, 2014. ISBN 978-3-319-10604-5. doi: 10.1007/978-3-319-10605-2_54. URL `http://dx.doi.org/10.1007/978-3-319-10605-2_54`.

J. Engel, J. Stückler, and D. Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942, Sep. 2015. doi: 10.1109/IROS.2015.7353631.

C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. In *In Photogrammetric Computer Vision*, 2006.

N. Fairfield, G. Kantor, and D. Wettergreen. Real-time slam with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 24(1-2):03–21, 2007. doi: 10.1002/rob.20165. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20165`.

P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 388–394, July 2015. doi: 10.1109/ICAR.2015.7251485.

C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, May 2014. doi: 10.1109/ICRA.2014.6906584.

C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, Feb 2017. ISSN 1941-0468. doi: 10.1109/tro.2016.2597321. URL `http://dx.doi.org/10.1109/TRO.2016.2597321`.

S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. *ACM Trans. Graph.*, 30(6):148:1–148:8, Dec. 2011. ISSN 0730-0301. doi: 10.1145/2070781.2024182. URL `http://doi.acm.org/10.1145/2070781.2024182`.

S. Fuhrmann and M. Goesele. Floating scale surface reconstruction. *ACM Trans. Graph.*, 33(4):46:1–46:11, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601163. URL `http://doi.acm.org/10.1145/2601097.2601163`.

D. Gallup, J. Frahm, P. Mordohai, and M. Pollefeys. Variable baseline/resolution stereo. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. doi: 10.1109/CVPR.2008.4587671.

J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, Sep. 2014. doi: 10.1109/IROS.2014.6942976.

K. Garanzha, J. Pantaleoni, and D. McAllister. Simpler and faster HLBVH with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 59–64, New York, NY, USA, 2011. ACM.

I. Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, 1982. doi: 10.1145/358728.358741. URL `https://doi.org/10.1145/358728.358741`.

B. Glocker, J. Shotton, A. Criminisi, and S. Izadi. Real-time RGB-D camera relocalization via randomized ferns for keyframe encoding. *IEEE Transactions on Visualization and Computer Graphics*, 21(5):571–583, May 2015. ISSN 2160-9306. doi: 10.1109/TVCG.2014.2360403.

G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

A. Handa, T. Whelan, J. McDonald, and A. J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1524–1531, May 2014. doi: 10.1109/ICRA.2014.6907054.

P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012. doi: 10.1177/0278364911434148. URL `https://doi.org/10.1177/0278364911434148`.

A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. doi: 10.1007/s10514-012-9321-0. URL `http://octomap.github.com`. Software available at `http://octomap.github.com`.

T. Isaac, C. Burstedde, L. Wilcox, and O. Ghattas. Recursive algorithms for distributed forests of octrees. *SIAM Journal on Scientific Computing*, 37 (5):C497–C531, 2015. doi: 10.1137/140970963. URL `https://doi.org/10.1137/140970963`.

E. S. Jones and S. Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of*

Robotics Research, 30(4):407–430, 2011. doi: 10.1177/0278364910388963. URL `https://doi.org/10.1177/0278364910388963`.

O. Kähler, V. A. Prisacariu, and D. W. Murray. Real-time large-scale dense 3D reconstruction with loop closure. In *Computer Vision – ECCV 2016*, pages 500–516, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46484-8.

T. Karras. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, pages 33–37. Eurographics Association, 2012. ISBN 978-3-905674-41-5.

M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. *3Dv*, pages 1–8, 2013. doi: 10.1109/3DV.2013.9. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6599048`.

C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. *IEEE International Conference on Intelligent Robots and Systems*, pages 2100–2106, 2013. ISSN 21530858. doi: 10.1109/IROS.2013.6696650.

M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015. doi: 10.15607/RSS.2015.XI.040.

K. Konolige and M. Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *Trans. Rob.*, 24(5):1066–1077, Oct. 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.2004832. URL `https://doi.org/10.1109/TRO.2008.2004832`.

O. Kähler, V. Prisacariu, C. Ren, X. Sun, P. Torr, and D. Murray. Very high frame rate volumetric integration of depth images on mobile devices. *Visualization and Computer Graphics, IEEE Transactions on*, PP(99): 1–1, 2015. ISSN 1077-2626. doi: 10.1109/TVCG.2015.2459891.

O. Kähler, V. Prisacariu, J. Valentin, and D. Murray. Hierarchical voxel block hashing for efficient integration of depth images. *IEEE Robotics* *and Automation Letters*, 1(1):192–197, Jan 2016. ISSN 2377-3766. doi: 10.1109/LRA.2015.2512958.

T. Laidlow, M. Bloesch, W. Li, and S. Leutenegger. Dense rgb-d-inertial slam with map deformations. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6741–6748, 2017.

S. Laine and T. Karras. Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 55–63, New York, NY, USA, 2010. ACM.

C. Lauterbach, M. Garland, S. Sengupta, D. P. Luebke, and D. Manocha. Fast BVH construction on GPUs. *Comput. Graph. Forum*, 28:375–384, 2009.

S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015. doi: 10.1177/0278364914554813. URL `https://doi.org/10.1177/0278364914554813`.

M. Li and A. I. Mourikis. High-precision, consistent ekf-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6): 690–711, 2013. doi: 10.1177/0278364913481251. URL `https://doi.org/10.1177/0278364913481251`.

S. Liu and D. B. Cooper. Statistical inverse ray tracing for image-based 3d modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):2074–2088, Oct 2014. ISSN 0162-8828. doi: 10.1109/TPAMI.2014.2315820.

C. Loop, Q. Cai, S. Orts-Escolano, and P. A. Chou. A closed-form bayesian fusion equation using occupancy probabilities. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 380–388, Oct 2016. doi: 10.1109/3DV.2016.47.

W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Aug. 1987. ISSN 0097-8930. doi: 10.1145/37402.37422. URL `http://doi.acm.org/10.1145/37402.37422`.

J. McCormac, A. Handa, A. Davison, and S. Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4628–4635, May 2017. doi: 10.1109/ICRA.2017.7989538.

J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger. Fusion++: Volumetric object-level slam. In *2018 International Conference on 3D Vision (3DV)*, pages 32–41, Sep. 2018. doi: 10.1109/3DV.2018.00015.

A. Millane, Z. Taylor, H. Oleynikova, J. I. Nieto, R. Siegwart, and C. Cadena. C-blox: A scalable and consistent TSDF-based dense mapping approach. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 995–1002, 2018. doi: 10.1109/IROS.2018.8593427. URL `https://doi.org/10.1109/IROS.2018.8593427`.

A. Miller, V. Jain, and J. L. Mundy. Real-time rendering and dynamic updating of 3-D volumetric data. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-4, pages 8:1–8:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0569-3. doi: 10.1145/1964179.1964190. URL `http://doi.acm.org/10.1145/1964179.1964190`.

M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Eighteenth National Conference on Artificial Intelligence*, pages 593–598, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0. URL `http://dl.acm.org/citation.cfm?id=777092.777184`.

H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Mag.*, 9(2):61–74, July 1988. ISSN 0738-4602. URL `http://dl.acm.org/citation.cfm?id=46184.46187`.

G. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, Canada, 1966.

A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE Inter-*

national Conference on Robotics and Automation, pages 3565–3572, April 2007. doi: 10.1109/ROBOT.2007.364024.

R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015. ISSN 1941-0468. doi: 10.1109/TRO.2015.2463671.

L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2015. arXiv:1410.2167.

L. Nardi, B. Bodin, S. Saeedi, E. Vespa, A. J. Davison, and P. H. J. Kelly. Algorithmic performance-accuracy trade-off in 3d vision applications using hypermapper. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1434–1443, May 2017. doi: 10.1109/IPDPSW.2017.107.

R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1498–1505, June 2010. doi: 10.1109/CVPR.2010.5539794.

R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 127–136, Oct 2011a. doi: 10.1109/ISMAR.2011.6092378.

R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2320–2327, Washington, DC, USA, 2011b. IEEE Computer Society. ISBN 978-1-4577-1101-5.

C. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 524–530, Oct 2012. doi: 10.1109/3DIMPVT.2012.84.

A. Nica, E. Vespa, P. G. de Aledo, and P. H. J. Kelly. Investigating automatic vectorization for real-time 3d scene understanding. In *Proceedings of the 2018 4th Workshop on Programming Models for SIMD/Vector Processing*, WPMVP'18, pages 5:1–5:8, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5646-6. doi: 10.1145/3178433.3178438. URL `http://doi.acm.org/10.1145/3178433.3178438`.

M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6): 169:1–169:11, Nov. 2013. ISSN 0730-0301. doi: 10.1145/2508363.2508374. URL `http://doi.acm.org/10.1145/2508363.2508374`.

D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, June 2004. doi: 10.1109/CVPR.2004.1315094.

H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: 10.1145/344779.344936. URL `http://dx.doi.org/10.1145/344779.344936`.

M. Pizzoli, C. Forster, and D. Scaramuzza. REMODE: Probabilistic, monocular dense reconstruction in real time. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2609–2616, May 2014. doi: 10.1109/ICRA.2014.6907233.

T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, Aug 2018. ISSN 1941-0468. doi: 10.1109/TRO.2018.2853729.

J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2014-6. doi: 10.1145/2491956.2462176. URL `http://doi.acm.org/10.1145/2491956.2462176`.

C. Richter, A. Bry, and N. Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Springer International Publishing, 2016.

G. Riegler, A. O. Ulusoy, and A. Geiger. OctNet: Learning deep 3D representations at high resolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6620–6629, July 2017. doi: 10.1109/CVPR.2017.701.

H. Roth and M. Vona. Moving volume KinectFusion. In *Proceedings of the British Machine Vision Conference*, pages 112.1–112.11. BMVA Press, 2012. ISBN 1-901725-46-4. doi: http://dx.doi.org/10.5244/C.26.112.

M. Rünz and L. Agapito. Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4471–4478, May 2017. doi: 10.1109/ICRA.2017.7989518.

J. P. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal. 3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments. *The International Journal of Robotics Research*, 32(14):1627–1644, 2013. doi: 10.1177/0278364913499415. URL `https://doi.org/10.1177/0278364913499415`.

S. Saeedi, B. Bodin, H. Wagstaff, A. Nisbet, L. Nardi, J. Mawer, N. Melot, O. Palomar, E. Vespa, T. Spink, C. Gorgovan, A. Webb, J. Clarkson, E. Tomusk, T. Debrunner, K. Kaszyk, P. Gonzalez-De-Aledo, A. Rodchenko, G. Riley, C. Kotselidis, B. Franke, M. F. P. O'Boyle, A. J. Davison, P. H. J. Kelly, M. Luján, and S. Furber. Navigating the landscape for real-time localization and mapping for robotics and virtual and augmented reality. *Proceedings of the IEEE*, 106(11):2020–2039, Nov 2018. ISSN 0018-9219. doi: 10.1109/JPROC.2018.2856739.

R. F. Salas-Moreno, R. a. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. SLAM++: Simultaneous localisation and mapping at the level of objects. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013. ISSN 10636919. doi: 10.1109/CVPR.2013.178.

H. Sarbolandi, D. Lefloch, and A. Kolb. Kinect range sensing. *Comput. Vis. Image Underst.*, 139(C):1–20, Oct. 2015. ISSN 1077-3142. doi: 10.1016/j.cviu.2015.05.006. URL `http://dx.doi.org/10.1016/j.cviu.2015.05.006`.

T. Schops, T. Sattler, and M. Pollefeys. BAD SLAM: Bundle adjusted direct rgb-d slam. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

M. Slavcheva, W. Kehl, N. Navab, and S. Ilic. SDF-2-SDF: Highly accurate 3D object reconstruction. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 680–696, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46448-0.

F. Steinbrucker, J. Sturm, and D. Cremers. Volumetric 3D mapping in real-time on a CPU. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2021–2028, May 2014. doi: 10.1109/ICRA.2014.6907127.

H. Strasdat, J. M. M. Montiel, and A. J. Davison. Scale Drift-Aware Large Scale Monocular SLAM. *Robotics: Science and Systems*, 2:5, 2010. doi: 10.1.1.165.7975. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.165.7975&amp;rep=rep1&amp;type=pdf`.

H. Strasdat, J. M. M. Montiel, and A. J. Davison. Visual SLAM: Why filter? *Image Vision Comput.*, 30(2):65–77, Feb. 2012. ISSN 0262-8856. doi: 10.1016/j.imavis.2012.02.009. URL `http://dx.doi.org/10.1016/j.imavis.2012.02.009`.

J. Stückler and S. Behnke. Multi-resolution surfel maps for efficient dense 3D modeling and tracking. *J. Vis. Comun. Image Represent.*, 25(1):137–147, Jan. 2014. ISSN 1047-3203. doi: 10.1016/j.jvcir.2013.02.008. URL `http://dx.doi.org/10.1016/j.jvcir.2013.02.008`.

J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, Oct 2012. doi: 10.1109/IROS.2012.6385773.

I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `http://ompl.kavrakilab.org`.

H. Sundar, R. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008. doi: 10.1137/070681727.

R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 1848829345, 9781848829343.

J. Thiyagalingam, O. Beckmann, and P. H. J. Kelly. Is morton layout competitive for large two-dimensional arrays yet&quest;: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(11):1509–1539, Sept. 2006. ISSN 1532-0626. doi: 10.1002/cpe.v18:11. URL `http://dx.doi.org/10.1002/cpe.v18:11`.

L. Thomas, M. Vinícius, P. Adelailson, P. Sinésio, and L. Hélio. Fast generation of pointerless octree duals. *Computer Graphics Forum*, 29(5):1661–1669, 2010. doi: 10.1111/j.1467-8659.2010.01775.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2010.01775.x`.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846, Jan 1998. doi: 10.1109/ICCV.1998.710815.

B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 298–372, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 3-540-67973-1. URL http://dl.acm.org/citation.cfm?id=646271.685629.

A. O. Ulusoy, A. Geiger, and M. J. Black. Towards probabilistic volumetric reconstruction using ray potentials. In *2015 International Conference on 3D Vision*, pages 10–18, Oct 2015. doi: 10.1109/3DV.2015.9.

B. Ummenhofer and T. Brox. Global, dense multiscale reconstruction for a billion points. *Int. J. Comput. Vision*, 125(1-3):82–94, Dec. 2017. ISSN 0920-5691. doi: 10.1007/s11263-017-1017-7. URL https://doi.org/10.1007/s11263-017-1017-7.

E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly, and S. Leutenegger. Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping. *IEEE Robotics and Automation Letters*, 3(2): 1144–1151, April 2018. ISSN 2377-3766. doi: 10.1109/LRA.2018.2792537.

E. Vespa, N. Funk, P. H. J. Kelly, and S. Leutenegger. Adaptive-resolution octree-based volumetric slam. In *2019 International Conference on 3D Vision (3DV)*, pages 654–662, Sep. 2019. doi: 10.1109/3DV.2019.00077.

P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Trans. Graph.*, 36(4):72:1–72:11, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073608. URL http://doi.acm.org/10.1145/3072959.3073608.

T. Whelan, M. Kaess, and M. Fallon. Kintinuous: Spatially extended KinectFusion. *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012. URL http://18.7.29.232/handle/1721.1/71756.

T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. Mcdonald. Real-time large-scale dense RGB-D SLAM with volumetric fusion. *Int. J. Rob. Res.*, 34(4-5):598–626, Apr. 2015a. ISSN 0278-3649. doi: 10.1177/0278364914551008. URL http://dx.doi.org/10.1177/0278364914551008.

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, Rome, Italy, July 2015b. doi: 10.15607/RSS.2015.XI.001.

O. J. Woodford and G. Vogiatzis. A generative model for online depth fusion. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *Computer Vision – ECCV 2012*, pages 144–157, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33715-4.

M. Zeng, F. Zhao, J. Zheng, and X. Liu. Octree-based fusion for realtime 3D reconstruction. *Graph. Models*, 75(3):126–136, May 2013. ISSN 1524-0703. doi: 10.1016/j.gmod.2012.09.002. URL `http://dx.doi.org/10.1016/j.gmod.2012.09.002`.

K. Zhou, M. Gong, X. Huang, and B. Guo. Data-parallel octrees for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 17(5):669–681, May 2011. ISSN 1077-2626. doi: 10.1109/TVCG.2010.75.

M. Z. Zia, L. Nardi, A. Jack, E. Vespa, B. Bodin, P. H. J. Kelly, and A. J. Davison. Comparative design space exploration of dense and semi-dense SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1292–1299, May 2016. doi: 10.1109/ICRA.2016.7487261.

J. Zienkiewicz, A. Tsiotsios, A. Davison, and S. Leutenegger. Monocular, real-time surface reconstruction using dynamic level of detail. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 37–46, Oct 2016. doi: 10.1109/3DV.2016.82.