Large Scale Dense 3D Reconstruction via Sparse Representations

Wei Dong

CMU-RI-TR-23-29

May 18, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Michael Kaess, *chair*
Ji Zhang,
Shubham Tulsiani,
Vladlen Koltun, *Apple*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

To my family (humans, cats, and toys)

Abstract

Dense 3D scene reconstruction is in high demand today for view synthesis, navigation, and autonomous driving. A practical reconstruction system inputs multi-view scans of the target using RGB-D cameras, LiDARs, or monocular cameras, computes sensor poses, and outputs scene reconstructions. These algorithms are computationally expensive and memory-intensive due to the presence of 3D data. Thus, it is essential to exploit sparsity adequately to reduce memory footprint, increase efficiency, and improve accuracy.

In this thesis, I will develop practical systems for fast and high-quality scene reconstruction. First, I will introduce a highly efficient hierarchical reconstruction system that serves as a foundational pipeline for integrating diverse pose estimation and scene reconstruction modules. Next, I will focus on the global registration of point clouds by learning deep features and their matches. Equipped with sparse convolutional networks, these studies define the state-of-the-art at the scene scale in both supervised and self-supervised setups. They are applied to reconstruction systems to produce globally consistent poses.

I will then shift to the topic of scene representation and reconstruction, introducing a modern engine, ASH, for parallel spatial hashing in the era of tensor and auto-differentiation. I will elaborate on the details of building this efficient and user-friendly engine from the ground up and discuss a series of downstream applications. These applications include real-time dense RGB-D SLAM, large-scale surface reconstruction from LiDAR scans, and fast scene reconstruction from monocular data. While achieving comparable or better accuracy than state-of-the-art methods, we demonstrate 2-10 times speed improvements with less development effort.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Michael Kaess. He has been the best academic advisor I have ever met in my research career. He shaped my research goals, gave astute academic suggestions, provided compassionate emotional supports, and ensured the freedom of my research directions. We shared excitement from research progress, persevered challenges from the pandemic, and discussed the future of the lab. Altogether, he not only trained me to be a better researcher, but also converted me into a more considerate person.

I want to also convey the same magnitude of appreciation to Vladlen Koltun, my mentor during two internships and a long-term collaboration. He made me re-think my research interest, directed me towards system research that I am really good at, and connected me with numerous talented peer researchers and collaborators. In many ways, he has been my co-advisor in the industry.

My heartfelt thanks go to my thesis committee members, Ji Zhang and Shubham Tulsiani, whose insightful suggestions and feedback have sparked new ideas and improved my thesis study.

I am deeply grateful for the contributions of my external collaborators, including Chris Choy from NVIDIA (for my first ever CVPR paper!); Jaesik Park from POSTECH, Heng Yang from MIT, Kai Zhang from Cornell, Oz Capunaman from Penn State; Yixing Lao, German Ros, Benjamin Ummenhofer, Sameer Sheorey, Rishabh Singh, and Qianyi Zhou from the Open3D team. I also wish to acknowledge my life-long labmates and friends Xin Wang, Zike Yan, Rukun Qiao, Qiuyuan Wang, Fei Xue, Chengxiao Yu, and Jieqi Shi, as well as my former advisor Hongbin Zha who introduced me to 3D reconstruction. Special thanks to Mingcai Zhou, Vagia Tsiminaki, Martin R. Oswald, and Marc Pollefeys, whose recommendations made it possible for me to start my PhD journey.

I extend my gratitude to my friends and labmates from the RPL lab: Akshay Hinduja, with whom I shared a long-lasting stay at 2204; Akash Sharma, for our mutual challenges and growth; Jack Yi Yang, for our ongoing discussions about research and life; Ruoyang Xu, Lihong Jin, and Jui-te Huang, for enjoyable chats and memorable moments. I also appreciate the camaraderie and support of Allie Chang, Montiel Abello, Dan McGann, Easton Potokar, Mohamad Qadri, Suddhu Suresh, Talha Faiz, Samiran Gode, Tianxiang Lin, Taylor Pool, Yehonathan Litman, Eric Dexheimer, Joshua Jaekel, Paloma Sodhi, Ming Hsiao, Tian Liu, and Eric Westman.

I am grateful to my first roommate and best friend in the US, Donghan Yu, and his wife Jessica Lee. My deepest appreciation goes to Renkai Xiang and Jing Cao for their warmth and hospitality during my internship in California; Qiyu Chen and Jiaxin Yan for their support in State College amidst the pandemic; and Camila Gutiérrez and Sebastián Franchini for introducing delightful kittens into our lives.

Lastly, I am eternally indebted to my family. My beloved teddy bear Lemon (and his toy friends), my cats Kiki, Kafka, and Artorias (even if only for a brief fostering period) have brought me immeasurable happiness. My father, Jianxiong Dong, my mother, Xiaojie Sun, and my wife, Chunyu Shen, have provided me with boundless love, care, and unwavering support, empowering me to start my journey, complete my PhD, and continue moving forward.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The recent decade has witnessed significant progress in 3D computer vision and computer graphics applied to reconstruction. 3D models are being constructed with real-world data, from the object scale for Mixed/Virtual Reality (MR/VR) [215, 231], to the building scale for navigation [34, 262], and to the city scale for autonomous driving [42]. These models are processed and viewed on PCs, laptops, and cellphones. To some extent, automatic 3D reconstruction systems have become indispensable technology in our daily lives.

Typical 3D reconstruction workflows take in captured raw data from various viewpoints, transform them into a shared coordinate system through pose estimation (*localization*), and reproduce the target in a digital representation via scene reconstruction (*mapping*). While pose estimation and sparse landmark reconstruction have been well-studied in the feature-based simultaneous localization and mapping (SLAM) context [72, 127, 165], researchers in the recent decade have also started to focus on *dense* reconstruction [58, 159, 169, 171, 231] due to increasingly powerful computational resources and advances in neural representations.

Dense reconstruction emphasizes the realistic reproduction of the digitally captured world, with applications including high-quality novel view synthesis through rendering and 3D model export via surface extraction. To achieve this, a large amount of data and optimizable parameters are usually required, necessitating heavy computations. In light of this, exploiting *sparsity* for *dense* reconstruction is crucial for retrieving the essential 3D data distribution, improving both the accuracy and speed of algorithms, and building practical systems that run fast on consumer-level machines.

In this thesis, I will examine the sparsity of data in the tasks of registration and scene reconstruction for RGB-D, LiDAR, and monocular images. I will develop ground-up systems that run fast and accurately in the real world, many of which have been incorporated into the widely used open-source library Open3D [272].

1.2 Problem Definition

Given a sequence of scene scans $\{\mathcal{S}_t\}$ from a sensor (monocular camera, RGB-D camera, or LiDAR), we want to estimate sensor poses $\{\mathbf{T}_t \in \mathrm{SE}(3)\}$ for each scan and reconstruct dense surfaces represented by a function f_θ parameterized by θ. Mathematically, it can be written as a joint optimization problem, where we minimize the loss $\mathcal{L}$ between an evaluating function g (e.g., rendering, geometric transformation) and an observed property y (e.g., pixel intensity, point location) at spatial samples $\mathbf{x}_i \in \mathbb{R}^3$:

$$\underset{\theta, \{\mathbf{T}_t\}}{\arg\min}\, \mathcal{L}\Big(g(f_\theta, \mathbf{x}_i, \mathbf{T}_t); y\Big). \tag{1.1}$$

The problem is usually decoupled and solved in two steps, namely pose estimation ($\mathbf{T}_t$) [72, 123, 203, 206, 262] and dense scene reconstruction (f_θ) [67, 171, 231, 257], although incremental or batch joint optimization is possible in a SLAM configuration [118]. In this thesis, I will take a decoupled viewpoint of the two stages since it is convenient for self-inclusive discussions per topic.

In pose estimation, I will study pairwise point cloud registration [22, 51, 180] and robust pose graph optimization [46] (Chapter 2-4):

$$\underset{\{\mathbf{T}_t\}}{\arg\min} \sum_{i,j,\mathbf{p},\mathbf{q}} \mathcal{L}\Big(\mathbf{T}_j^{-1}\mathbf{T}_i\mathbf{p}, \mathbf{q}\Big), \tag{1.2}$$

where $\mathbf{p} \in \mathbb{R}^3$ is taken from scan $\mathcal{S}_i$, and $\mathbf{q} \in \mathbb{R}^3$ is its correspondence in $\mathcal{S}_j$.

In dense reconstruction, I will stick to Signed Distance Function (SDF) [56] f_θ as the implicit surface representation:

$$f_\theta : \mathbb{R}^3 \to \mathbb{R}, \tag{1.3}$$

$$\mathbf{x} \mapsto \mathrm{sgn}_\Omega(\mathbf{x}) \inf_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\|, \tag{1.4}$$

$$\mathrm{sgn}_\Omega(\mathbf{x}) = \begin{cases} +1 & x \in \Omega^+, \\ 0 & x \in \partial\Omega, \\ -1 & x \in \Omega^-. \end{cases} \tag{1.5}$$

where the 3D space is divided into positive (Ω^+) and negative (Ω^-) subspaces separated by the surface ($\partial\Omega$) to be reconstructed. It is worth mentioning that *implicit* here denotes that the surface $\partial\Omega$ is *mathematically* defined by the set $\{\mathbf{x} \mid f_\theta(\mathbf{x}) = 0\}$, regardless of whether θ is hidden in a neural network or explicitly stored at spatial grid points.

f_θ can be approximated at a single point $\mathbf{x}_i$ by projective signed distance:

$$\arg\min_\theta \mathcal{L}\Big(f_\theta(\mathbf{x}_i); D(\mathbf{x}_i; \mathcal{S}_t, \mathbf{T}_t) \Big), \tag{1.6}$$

where the projected depth at viewpoint t can be directly computed via a geometric transform function D (see Chapter 5-6). Alternatively, f_θ can be estimated at multiple points $\mathbf{x}_i^r$ along rays r by differentiable volume rendering:

$$\arg\min_\theta \mathcal{L}\Big(\mathrm{VolumeRender}(y(\mathbf{x}_i^r), f_\theta(\mathbf{x}_i^r)); Y(\mathbf{x}_i; \mathcal{S}_t, \mathbf{T}_t) \Big), \tag{1.7}$$

where y and Y denote functions of additional properties (*e.g*. color, depth, normal) from scene representations and measurements according to camera models and rigid transformations (see Chapter 7).

All the aforementioned functions $\mathcal{L}$ are differentiable. For simple compositional functions equipped with square losses (Eq. 1.2 and 1.6), we can derive analytical Jacobians w.r.t. parameter w, and apply iterative second-order optimization (Gauss-

Newton or Levenberg-Marquardt):

$$w_{k+1} = w_k - \lambda_k \Big(\mathbf{J}^\top \mathbf{J}(w_k)\Big)^{-1} \mathbf{J}^\top \mathcal{L}(w_k), \tag{1.8}$$

where $\mathbf{J}$ is the Jacobian and $\mathcal{L}$ is the residual, k is the number of iterations, and λ_k is the step size. $\mathbf{J}^\top\mathbf{J}$ and $\mathbf{J}^\top\mathcal{L}$ can usually be analytically written as a compact function of w_k. For non-trivial chained functions or non-square losses (Eq. 1.7), first-order optimization (Stochastic Gradient Descent and variations) can be performed through auto differentiation [181]:

$$w_{k+1} = w_k - \gamma_k \nabla \mathcal{L}(w_k), \tag{1.9}$$

where γ_k is the step size. A general guidance of the choice of optimizers are provided by Dellaert and Kaess [59]. Representations, derivatives, and on-manifold optimization of $\mathrm{SE}(3)$ parameters are discussed in-depth by Strasdat [212] and Sola *et al.* [208].

Apart from optimization itself, I will also demonstrate that an effective choice of $\mathbf{x}_i$ results in fast convergence. This is achieved by accurately capturing surfaces' sparse distribution in the 3D space with collision-free parallel spatial hash maps with modern interfaces.

1.3 Overview

The thesis can be roughly divided into three parts.

- In Chapter 2, I will first define a hierarchical reconstruction system (IROS 2019 [65]). The system takes in unposed scans and outputs poses and surfaces with a fragment-to-scene pipeline. While it is curated for RGB-D data, its modularized design allows generalization to other data sources and encourages improvement of separate components, leading to the following studies in the thesis.
- In Chapter 3-4, I will attempt to address the global registration problem, *i.e.* , aligning point clouds without initial relative transformations. I will use supervised (CVPR 2020 [51]) and self-supervised (CVPR 2021 [247]) approaches

to learn point-wise features and their correspondences through sparse convolutional networks. This efficiently captures the geometric properties for 3D data from RGB-D and LiDAR sensors, defining state-of-the-art accuracy for global registration at the scene scale.

- In Chapter 5-7, I will develop a modern, performant, and easy-to-use spatial hashing engine for scene representation. I will demonstrate its usage in dense RGB-D SLAM (PAMI [67]), surface reconstruction for large-scale LiDAR data (arXiv [66]), and fast scene reconstruction from monocular images (CVPR [68]). While achieving similar or better accuracy to state-of-the-art methods, these applications run 2-10$\times$ faster and require less development effort.

I will conclude the thesis with ongoing and future works aiming at faster, more robust, and more accurate reconstruction systems.

Part I

Hierarchical Scene Reconstruction System

Chapter 2

GPU Accelerated Robust Scene Reconstruction

2.1 Introduction

Dense 3D reconstruction of scenes is a fundamental component in localization and navigation tasks for intelligent systems such as robots, drones, and surveillance systems. Accurate 3D models of real-world scenes is a key element for mixed and virtual reality applications, because it is directly related to realistic content creation or telepresence. In recent years, research in scene reconstruction using RGB-D frames has flourished with the presence of affordable, high-fidelity consumer-level RGB-D sensors, and a number of off-the-shelf reconstruction systems have been introduced so far.

Reconstruction systems can be generally classified into online and offline systems. Systems including VoxelHashing [171], InfiniTAM [119], and ElasticFusion [237] are based on dense SLAM algorithms. These algorithms suffer from *pose drift*, which usually causes corrupted models in large scenes. BundleFusion [58], as an online system, is more similar to offline structure-from-motion systems. It keeps track of all the RGB-D keyframes, hence is computationally expensive and requires two high-end graphics cards to run. Offline systems such as Open3D [272] and Robust Reconstruction Pipeline [46], on the other hand, include hierarchical global constraints such that the camera poses are globally consistent and accurate even in large scenes. However, the expected running times of such systems span from several hours to days.

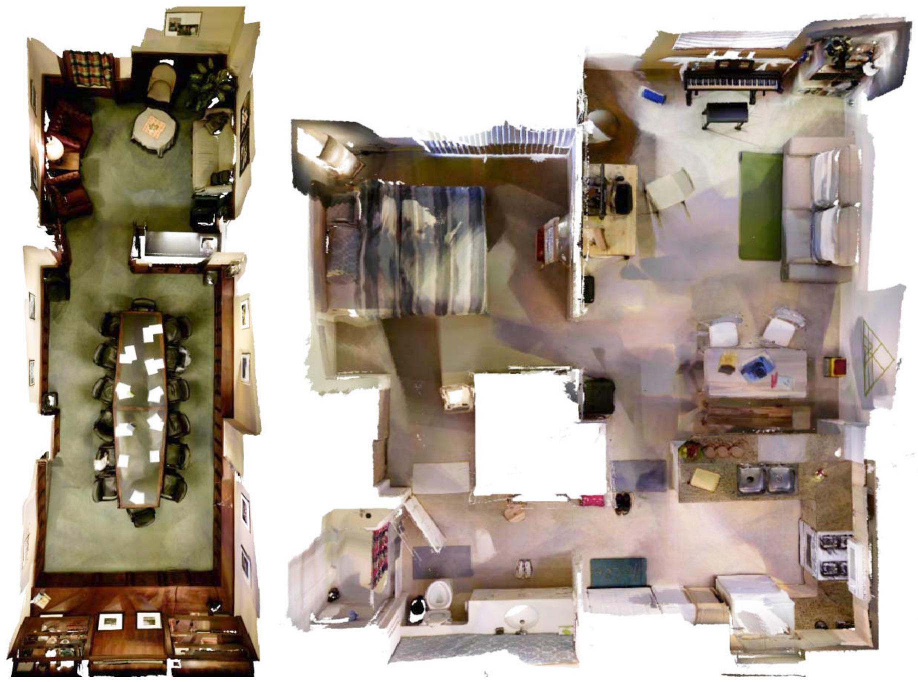


Figure 2.1: Reconstructed large indoor scenes, *boardroom* and *apartment*. The largest scene *apartment* (> 30K RGB-D frames) is reconstructed by our system within 1.5 hours, 6Hz on average. These scenes typically require more than 10 hours to run on the offline system [272], and will fail on online systems [119, 237]

In this paper, we present an accelerated offline RGB-D reconstruction system. The basic algorithm is based on offline systems such as Open3D [46, 272]. On top of this, the major contributions in this work include:

- GPU acceleration of core modules in an offline reconstruction system such as RGB-D odometry, Iterative Closest Point (ICP), global registration, volumetric integration, and Marching Cubes.
- Designing new GPU data containers to boost basic operations.
- 10x faster on average than baseline offline systems, comparable to online systems in terms of speed.
- While being fast as an offline system, the reconstruction accuracy is main-

tained on most datasets compared to the state-of-the-art offline system [46].

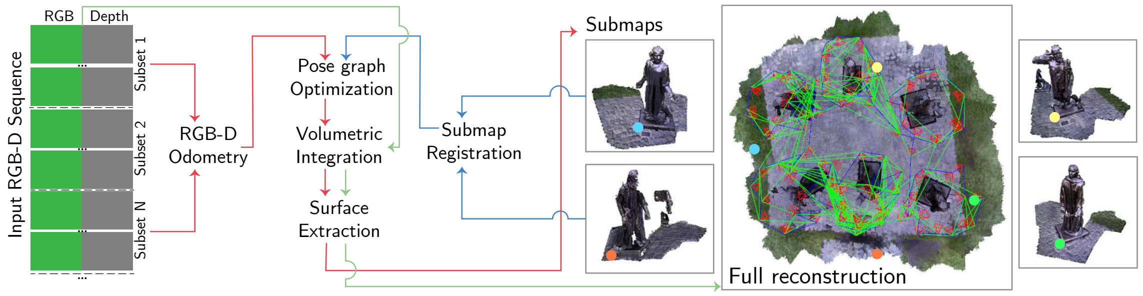


Figure 2.2: System overview. Left shows workflows of the 3 major stages, displayed in red (*make submaps*), blue (*register submaps* and *refine registration*), and green (*integrate scene*) lines respectively. Right shows local and global reconstruction results, along with the pose graph of submaps. Best viewed in color.

2.2 Related Work

State-of-the-art online dense 3D reconstruction systems [58, 119, 171, 237] track the pose of the current frame using visual odometry (VO), fuse data into the dense map, and search for loop closures to reduce drift. While these methods are fast enough to track input sequences, inevitable drift can cause incorrect dense maps.

ElasticFusion [237] proposes to correct drift in a surfel-based [122] map by introducing graph deformation. However, the deformation is sensitive to user parameters and may easily fail on various benchmarks. In addition, as surfels accumulate quickly, the system may not scale to larger scenes. InfiniTAM [119] and VoxelHashing [171] apply frame-to-model tracking [168] to overcome the drift issue. RGB-D data is fused into a Truncated Signed Distance Field (TSDF) [56] with estimated poses, and the system extracts the model from the TSDF for tracking. However, it may break down when incorrect camera poses corrupt the dense map, even only locally. InfiniTAM [119] introduces a relocalizer to recover the system from tracking failures, but there is no strategy to handle the corruptions in the TSDF map.

BundleFusion [58] performs exhaustive feature-based bundle adjustment on the keyframes in the entire sequence to correct camera poses, and uses brute-force re-integration [74] to correct the TSDF map. For this reason, the system requires two high-end graphics cards[1] to ensure real-time, which is beyond consumer level. Moreover, it keeps track of all the keyframes and performs global optimization frequently, hence it is only applicable to scenes with ≤ 25000 RGB-D frames. It suggests that the computational cost may be reduced by splitting large scenes and processing hierarchical optimization.

On the other hand, offline reconstruction systems [46, 272] address the drift issues in the online system by adopting a hierarchical framework, as mentioned in [58]. The system splits input sequences of varying lengths into subsets and generates submaps using methods similar to online SLAM systems. Afterwards, these submaps are registered pairwise to form a pose graph [93], which is optimized to provide accurate poses for submaps. Finally, all RGB-D frames with accurate poses are integrated into a global TSDF volume to extract a complete 3D model. This approach can easily modularize the overall procedure into small tasks such as VO, dense mapping, and surface registration. Each decoupled component can benefit from available state-of-the-art algorithms.

The offline systems use direct RGB-D odometry to estimate the pose between subsequent frames. They can fully utilize available depth and color information by *directly* minimizing joint photometric and geometric error [124, 180, 209], as well as overcome challenging textureless scenes where feature-based method [164] may fail. Assuming that pose drift is small over the course of each submap, the systems fuse raw RGB-D data into a TSDF volume [168, 171] and extract the mesh or point cloud as a submap.

At the level of submaps, point cloud registration is required to estimate relative poses between corresponding point sets. With proper initialization, classical ICP [22] iteratively computes data association and registration. The state-of-the-art Colored ICP [180] achieves high accuracy between dense colored point clouds. In cases where the pose initialization is not reliable, globally optimal methods are preferred. [249] searches the $\mathrm{SE}(3)$ space with bound-and-branch strategy for potential initializations and then performs ICP. A faster yet accurate algorithm Fast Global Registration (FGR) [271] relies on Fast Point Feature Histograms (FPFH) [198] to generate possible matches, and filters incorrect matches with fast and robust line

[1] An NVIDIA TITAN X and a TITAN black card were used for the real-time demonstration.

processes [83] defined in [271].

The obvious limitation of aforementioned offline systems is that they take a very long time to reconstruct large scale scenes. In this paper, we propose a GPU-accelerated offline system to reach the level of online system performance while not sacrificing accuracy.

2.3 Reconstruction System

The basic principle of the proposed system is similar to state-of-the-art offline reconstruction systems [46, 272]. The overall procedure of the system consists of four major stages, as shown in Fig. 4.1:

1. *Make submaps*. Frame-to-frame camera pose is estimated for evenly divided subsets of the input sequence. Afterwards, TSDF volume integration is performed for each subset of the sequence, and a *submap* is extracted in the form of a triangle mesh.
2. *Register submaps*. Pairwise submap registration is applied. The submap pairs that are temporally adjacent are registered using *Colored ICP* [180] with reliable initial relative poses from RGB-D odometry. Temporally distant submaps are registered using geometric feature-based *FGR* [271] to detect loop closures. A pose graph is constructed and optimized to determine the poses of submaps in the *global* coordinate system.
3. *Refine registrations*. The relative poses between each registered submap pair, including both the adjacent and the distant submap pairs, are further refined using multi-scale Colored ICP. A subsequent pose graph optimization determines the final global poses for every submap.
4. *Integrate scene*. Given the optimized poses of the submaps and the poses of every frame, TSDF integration fuses the entire RGB-D sequence, and produces the final 3D reconstruction.

2.3.1 Basic Data Containers for GPU

As the basic infrastructure, we designed several GPU data structures for general usage in our system.

- *1D arrays*. We implement atomic *push_back* operation to support multi-thread writing, mimicking a vector on CPU.
- *2D arrays*. They are stored in *row* major order. In a typical parallel execution on a large 2D array, each thread is designed to iterate over one *column*. Since the threads are approximately accessing the same *row* simultaneously, they are more likely to share the cache that loads the same section of global memory, which increases accessing efficiency.
- *Linked list and hash table*. A generic templated (key, value, and hashing function) hash table is implemented for GPU. Each hashed key corresponds to a fixed size bucket array plus a dynamic size bucket linked list to address conflicts. The hash table is essential for spatial hashing used in 3D volume storage.

All the data structures support memory transfer between GPU and CPU. Since global memory allocation is expensive on GPU, we pre-allocate enough memory, and rely on a dynamic memory manager to manually allocate and free memory on GPU on demand. Reference counting is implemented for efficient GPU memory sharing.

2.3.2 RGB-D Odometry

In RGB-D odometry, we seek to optimize an error function of relative pose between two consecutive RGB-D frames $\langle \mathcal{F}_s, \mathbf{T}_s \rangle$ and $\langle \mathcal{F}_t, \mathbf{T}_t \rangle$.[2] The error function is constructed as in [180], including both the photometric error r_I and the difference of

[2]We define the terms as follows. $\langle \mathcal{F}, \mathbf{T} \rangle$ denotes an RGB-D frame. Here, $\mathcal{F} = \langle \mathcal{I}, \mathcal{D} \rangle$ consists of color and depth images, and $\mathbf{T} \in \mathrm{SE}(3)$ represents the 6-DOF pose vector in the local submap's coordinate system.

depth r_D:

$$E(\mathbf{T}_s^t) = \sum_{\mathbf{p}} (1-\sigma)\ r_I^2(\mathbf{T}_s^t, \mathbf{p}) + \sigma\ r_D^2(\mathbf{T}_s^t, \mathbf{p}), \tag{2.1}$$

$$r_I = \mathcal{I}_s[\mathbf{p}] - \mathcal{I}_t[\mathcal{W}(\mathbf{T}_s^t, \mathbf{p}, \mathcal{D}_s[\mathbf{p}])], \tag{2.2}$$

$$r_D = \mathcal{T}(\mathbf{T}_s^t, \mathbf{p}, D_s[\mathbf{p}]).z - \mathcal{D}_t[\mathcal{W}(\mathbf{T}_s^t, \mathbf{p}, \mathcal{D}_s[\mathbf{p}])], \tag{2.3}$$

where $\mathbf{T}_s^t \in \mathrm{SE}(3)$ is the relative pose from $\mathcal{F}_s$ to $\mathcal{F}_t$, $\mathbf{p} \in \mathbb{R}^2$ is the pixel in the RGB-D image $\mathcal{F}_s$, and $\sigma \in [0, 1]$ is a hyper parameter to balance the appearance and geometry terms. We use $\mathcal{T}(\mathbf{T}, \mathbf{p}, d)$ and $\mathcal{W}(\mathbf{T}, \mathbf{p}, d)$ to denote rigid transformation and warping (transformation + projection) of a 3D point with its pixel coordinate $\mathbf{p}$ and depth value d respectively. For simplicity, the intrinsic matrix is not displayed in these functions.

To minimize the non-linear error function, Gauss-Newton optimization is applied. By computing the first-order linearization, we have

$$r_I(\mathbf{T}_s^t + \Delta\mathbf{T}, \mathbf{p}) \approx r_I(\mathbf{T}_s^t) + J_I(\mathbf{T}_s^t, \mathbf{p})\Delta\mathbf{T}, \tag{2.4}$$

$$r_D(\mathbf{T}_s^t + \Delta\mathbf{T}, \mathbf{p}) \approx r_D(\mathbf{T}_s^t) + J_D(\mathbf{T}_s^t, \mathbf{p})\Delta\mathbf{T}, \tag{2.5}$$

where J_I and J_D are Jacobian matrices for each term per pixel. Finally, the problem reduces to solving $\Delta\mathbf{T}$ in the least squares system and updating $\mathbf{T}_s^t$ iteratively:

$$\sum_{p} A(\mathbf{p})\Delta\mathbf{T} = -\sum_{\mathbf{p}} b(\mathbf{p}), \tag{2.6}$$

$$\mathbf{T}_s^t = \Delta\mathbf{T} \oplus \mathbf{T}_s^t, \tag{2.7}$$

where the system is built up with

$$A(\mathbf{p}) = (1-\sigma)\ J_I^T J_I(\mathbf{T}_s^t, \mathbf{p}) + \sigma\ J_D^T J_D(\mathbf{T}_s^t, \mathbf{p}), \tag{2.8}$$

$$b(\mathbf{p}) = (1-\sigma)\ J_I^T r_I(\mathbf{T}_s^t, \mathbf{p}) + \sigma\ J_D^T r_D(\mathbf{T}_s^t, \mathbf{p}). \tag{2.9}$$

To ensure faster convergence, we implement coarse-to-fine RGB-D odometry using a image pyramid.

It is straightforward to parallelize building the Jacobian matrix, because each pixel contributes to the Jacobian independently. However, proper handling of thread conflicts in the summation operation is critical here, because the number of GPU threads is often larger than a few thousand, and thread conflicts degrade performance. While an `Atomic-Add` operation is supported on most GPUs to avoid conflicts, the more advanced technique `Reduction` [102] can accelerate the process further by fully utilizing *shared* memory instead of *global* memory. There are two variations of Reduction, the original one [102] and `Warp Shuffle` used in [237]. Based on our experiments, the original version runs two times faster than built-in atomic-add, and outperforms warp shuffle.

As $A(\mathbf{p})$ is a symmetric matrix, only the upper-right 21 elements are summed, while the other elements are duplicated on CPU. Plus 6 elements in $b(\mathbf{p})$, we need to sum 27 elements separately. To accelerate, we allocate 3 arrays of shared memory and sum 3 elements in the linear system at one time. With reasonable shared memory consumption, the cost of frequent thread synchronization is reduced: reduction takes around $2{\sim}3$ ms to sum elements in the linear system for a 640×480 image. This general reduction module is also used in Sec. 2.3.5, where similar linear systems are defined.

2.3.3 Integration

With the known pose $\mathbf{T}$ of each frame $\mathcal{F}$, we can use TSDF integration to fuse raw RGB-D data into the volumetric scalar field. To improve the storage efficiency, we use spatial hashing as described in [119, 171], with the modified hash table structure described in Sec. 2.3.1. The space is coarsely divided into voxel blocks for spatial hashing; each block contains an array of $8 \times 8 \times 8$ voxels for fast and direct parallel memory access by GPU threads.

The integration consists of three passes. In the first pass, a point cloud is generated from frame $\mathcal{F}$ and transformed using its pose $\mathbf{T}$. The blocks that can cover the generated points will be created if they do not exist yet. In the second pass, all generated blocks in the $\mathcal{F}$'s viewing frustum will be collected into a buffer. Finally, parallel integration will be performed on collected blocks in the buffer. Each voxel in the blocks will be projected onto $\mathcal{F}$ to find the projective closest pixel.

After that, the voxels update their stored TSDF value using a weighted average:

$$\begin{aligned} d &= \phi(\mathcal{D}[\mathcal{W}(\mathbf{T}, \mathbf{x})] - \mathbf{x}.z), && (2.10)\\ \mathrm{TSDF}[\mathbf{x}] &= \frac{\mathrm{TSDF}[\mathbf{x}] + d}{\mathrm{Weight}[\mathbf{x}] + 1}, && (2.11)\\ \mathrm{Weight}[\mathbf{x}] &= \mathrm{Weight}[\mathbf{x}] + 1, && (2.12) \end{aligned}$$

where ϕ is the truncation function in [168], $\mathbf{x} \in \mathbb{R}^3$ is the voxel coordinate, and $\mathcal{W}(\mathbf{T}, \mathbf{x})$ projects x to the frame after transforming x with $\mathbf{T}$. Weighted average generally also applies to colors.

2.3.4 Mesh Extraction

After the TSDF has been generated, Marching Cubes (MC) [146] is applied to extract surfaces from TSDF volumes. We improve the mesh extraction framework in [63] both in terms of memory and speed.

As pointed out in [63], the vertex on the shared edge between two voxels can be computed only once. By setting up a one-to-one correspondence between edges and voxels, we can compute every unique vertex once. Directly maintaining the edge-vertex correspondences in voxels along with the TSDF creates a large overhead in memory. In the improved version, we detach the structure, so that it can be allocated and attached only to the active parts of the TSDF volume that require meshing.

One of the most computationally expensive operations in MC is normal estimation from the TSDF. Given a vertex $\mathbf{x}_v = (x_v, y_v, z_v)$, the normal is computed by normalized gradient $\nabla\mathrm{TSDF}(\mathbf{x}_v)/||\nabla\mathrm{TSDF}(\mathbf{x}_v)||$, where

$$\nabla\mathrm{TSDF}(\mathbf{x}_v) = \begin{bmatrix} \mathrm{TSDF}(x_v + 1, y_v, z_v) - \mathrm{TSDF}(x_v - 1, y_v, z_v) \\ \mathrm{TSDF}(x_v, y_v + 1, z_v) - \mathrm{TSDF}(x_v, y_v - 1, z_v) \\ \mathrm{TSDF}(x_v, y_v, z_v + 1) - \mathrm{TSDF}(x_v, y_v, z_v - 1) \end{bmatrix}. \tag{2.13}$$

Since the 6 query points are not grid points, 8 spatial queries are required to get a tri-linear interpolated TSDF value per query point. 48 spatial queries and 48 interpolations for one single point is expensive. We found that the final computed normal is *identical* to a simple interpolation of normals at the two adjacent grid points, while the normal computation at a grid point only requires 6 queries, as shown in Fig. 2.3. Therefore, we need only 12 spatial queries and 1 interpolation.

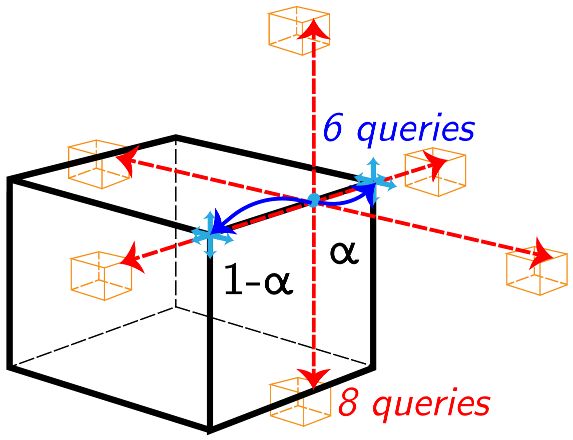


Figure 2.3: Illustration of normal extraction. Red lines show the redundant queries (tri-linear interpolation for non-grid points) to extract normal. Blue lines show the optimal interpolation of normals at adjacent grid points. α is the interpolation ratio of normals, which can be directly computed in MC.

To further speed up MC in spatial hashed voxel blocks, we specifically implement MC for

1. Voxels inside a voxel block. Modified MC in [63] is directly performed on such voxels.
2. Voxels at the boundary of a voxel block, where frequent hash table queries are required. Despite the fact that hash tables are *supposed* to be $O(1)$ look up, the overhead is considerable. To speed up, for each block, we first cache pointers of all 26 neighbor blocks. Then each voxel can access neighbor voxels from the cached block instead of looking them up in the hash table.

2.3.5 Registration

Given extracted surfaces as submaps $\mathcal{S}$, we perform a multiway registration defined in [46] to get their poses ζ.

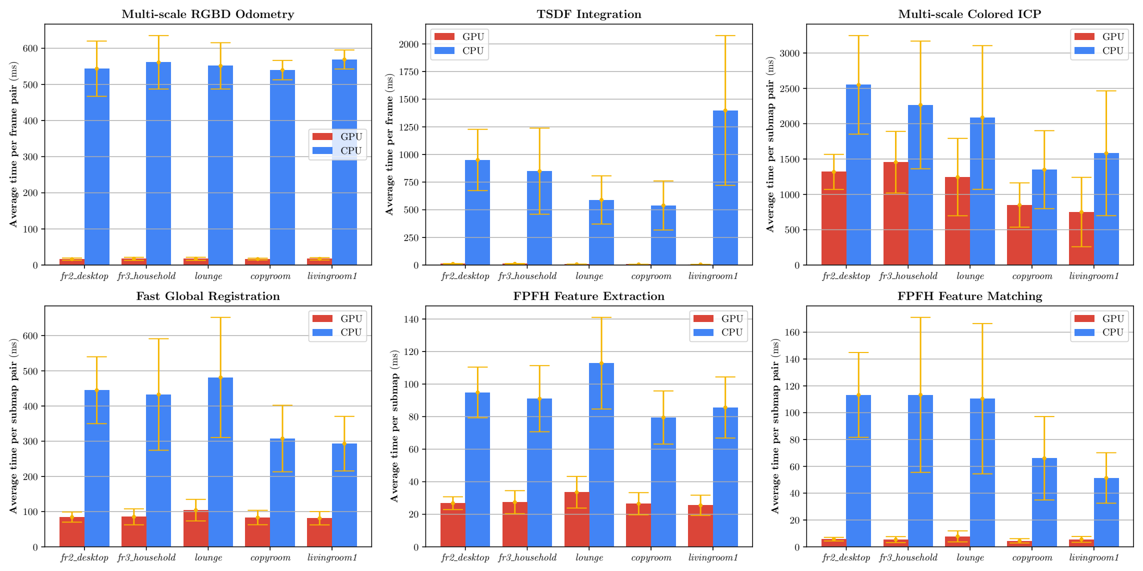


Figure 2.4: Average runtime comparison of our system (GPU) and Open3D [272] (CPU).

Colored ICP

Point cloud pairs with good initial pose guesses are registered using Colored ICP [180]. Typically, we can obtain reasonable initialization between two consecutive submaps by aligning the poses of the last RGB-D frame in the first submap and the first RGB-D frame in the second submap.

Colored ICP builds up a linear system similar to Eq. 2.2. While RGB-D images are dense in 2D, point clouds are sparse in 3D. Therefore, for point cloud registration, projective data association has to be replaced by nearest neighbor search. Since there are no satisfying alternatives on GPU, we use Fast Library for Approximate Nearest Neighbors (FLANN) [162] on CPU to find 3D nearest neighbors, as a legacy of [272]. In addition, the gradient operation that is natural on 2D images is not well-defined on point clouds, causing problems in computing Jacobians. It is easy to replace gradient of the depth image with each point's normal; with the correspondences from FLANN, we can compute approximated color gradient per

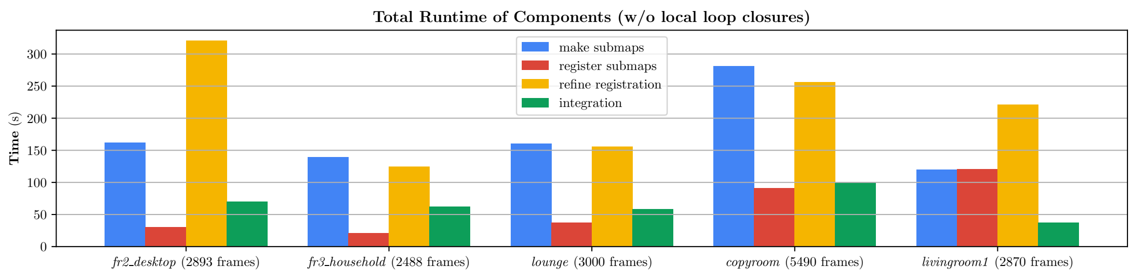


Figure 2.5: Total system runtime on datasets. $\sim$ 8Hz on average is achieved on most datasets.

point, as defined in [180]:

$$\begin{aligned} &\left(\sum_{\mathbf{p}' \in \mathcal{N}(\mathbf{p})} A(\mathbf{p}')^T A(\mathbf{p}') + \mathbf{n}(\mathbf{p})\mathbf{n}(\mathbf{p})^T \right) \mathbf{d}(\mathbf{p}) \\ &= \sum_{\mathbf{p}' \in \mathcal{N}(\mathbf{p})} A(\mathbf{p}')^T b(\mathbf{p}'). \end{aligned} \tag{2.14}$$

Here $\mathbf{p} \in \mathbb{R}^3$ denotes the point we are processing, $\mathbf{n}(\mathbf{p})$ is its normal, $\mathbf{p}' \in \mathcal{N}(\mathbf{p})$ represents its neighbors, and $\mathbf{d}(\mathbf{p})$ is the target color gradient at $\mathbf{p}$. The linear system depends on the projective distances and color differences between $\mathbf{p}$ and its neighbors:

$$A(\mathbf{p}') = \mathbf{p}' - \mathbf{n}(\mathbf{p})^T(\mathbf{p}' - \mathbf{p})\ \mathbf{n}(\mathbf{p}), \tag{2.15}$$

$$b(\mathbf{p}') = \mathcal{C}(\mathbf{p}') - \mathcal{C}(\mathbf{p}), \tag{2.16}$$

where $\mathcal{C}(\mathbf{p})$ is the intensity of a point. To solve these small linear systems in parallel, we implement a complete module for LDL decomposition and forward substitution on GPU.

After preprocessing, our implementation iterates between finding data correspondences on CPU and building the linear system on GPU until convergence. Apart from Colored ICP, classical point-to-point and point-to-plane ICP are also supported within the same architecture.

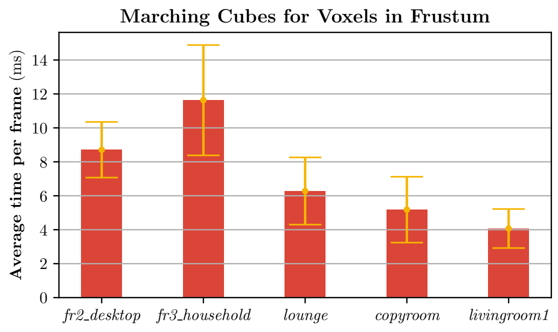


Figure 2.6: Marching Cubes runtime. Note in this experiment, ground truth trajectories are used to ensure reasonable viewing frustums.

Fast Global Registration

We compute pairwise registrations between non-adjacent submaps to detect potential loop closures. As no initialization is provided, we have to rely on 3D features to get correspondences. FPFH [198] is used in our case. FPFH only relies on local neighbors, therefore feature extraction is possible to run in parallel, provided nearest neighbors are found by FLANN.

As a high dimensional (33 dim) feature, FPFH is less efficient to match using FLANN. We turn to brute-force parallel NN matching by simplifying KNN-CUDA [81] to the 1-NN case. Brute force NN consists of 2 passes. In the first pass, a dense distance matrix is calculated in parallel after the matrix is divided into small submatrices. Here, the cache friendly column-wise iterations we designed in Sec. 2.3.1 help to accelerate the step by a factor of 2. In the second pass, each query point will search for their nearest neighbor by `Reduction` with the `min` operator, which resembles the `Reduction` with `add` we used. The drawback of brute-force matching is the memory cost. The dense distance matrix will consume all the GPU memory when the sizes of point clouds are large (> 20K). Under such circumstances, downsampling is required.

With known matches, we reuse our framework to build a linear system for the FGR algorithm [271]. As there are point-wise line processes involved, the error function is much different from Eq. 2.2. We refer readers to the original paper for more details. Before applying Gauss-Newton, feature matching tests are performed in parallel, where each thread handles a random test set. All the data

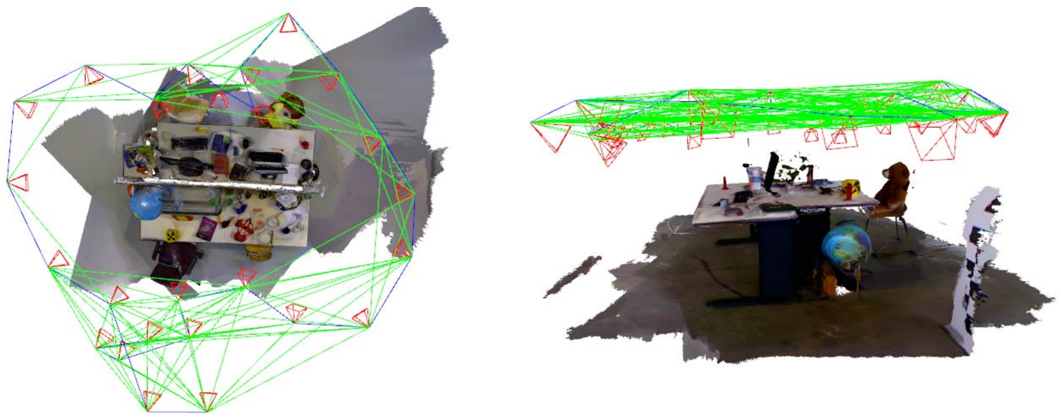


Figure 2.7: Pose graph of submaps on *TUM household* and *desk* datasets. Valid loop closures are detected between submaps.

stay on GPU after feature extraction, including parallel random number generation. No expensive CPU operation is required after FLANN has finished its job, hence FGR can run very fast and stable.

After registration, estimated poses ζ are inserted in a pose graph as nodes, with edges between adjacent submaps and potential loop closures. The information matrices on the edges are computed similar to the building process of linear systems with minor changes in residuals and Jacobians. We then perform robust multiway registration [46] to eliminate false loop closures and obtain optimized poses.

2.4 Experimentals

The proposed system can reproduce the results of the state-of-the-art offline reconstruction system implemented in Open3D [46, 272] with significantly reduced time budget. In this section, we compare runtime with the baseline offline reconstruction system Open3D [272], and show qualitative and quantitative results with state-of-the-art online reconstruction systems. The tested datasets include the TUM RGB-D dataset [213], the Stanford and Redwood simulated dataset [46], and the large Indoor LIDAR-RGBD Scan dataset [180]. The voxel size is set to 6mm in real-world scenes and 8mm in Redwood simulated scenes.

The GPU part is implemented in CUDA, where CPU utility functions are built upon Open3D [272]. The following experiments were run on a laptop with an Intel

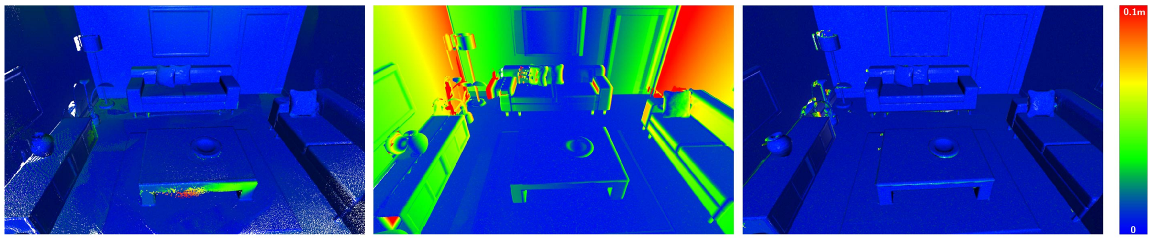


Figure 2.8: Distance heatmap from reconstructed model to ground truth on *livingroom1*, generated from CloudCompare. From left to right, *ElasticFusion*, *InfiniTAM* (hybrid color and depth tracker fails, shift occurs in ICP tracker), ours.

i7-6700 CPU and a NVIDIA 1070 graphics card with 8G GPU memory.

2.4.1 Runtime Results

In Fig. 2.4, we first show the acceleration of the components in our system.

- Multi-scale RGB-D Odometry runs with $\{20, 10, 5\}$ iterations from coarse to fine scale. It is accelerated to around ∼16ms per frame, around 40 times faster than the baseline. As it already achieves real-time, RGB-D SLAM systems may include this module.
- TSDF integration is accelerated to ~ 10ms per frame on a volume with high resolution, approximately $50 \sim 120$ times faster. This component can also serve as a part of a real-time dense SLAM / mapping system.
- Multi-scale Colored ICP runs with $\{50, 30, 14\}$ iterations from coarse to fine scale. It is accelerated only with a factor of 1.5 to 2. The major reason is that we still rely on FLANN for data correspondences, which takes a large amount of time. Note the runtime excludes the slow point cloud downsampling, which takes place in both CPU and GPU methods.
- FGR is accelerated to around 10Hz, $4 \sim 5$ times faster than CPU version. GPU based feature extraction and matching significantly reduces the runtime. These modules may also be separately used in other tasks such as naive 3D object recognition and matching.

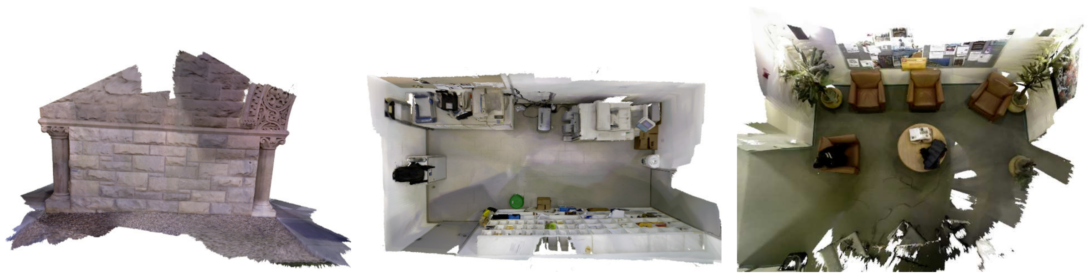
We also separately demonstrate the performance of parallel MC in Fig. 2.6. As it runs very fast for visualizing surfaces in the viewing frustum, it can serve as a visualizer for dense SLAM systems. A screenshot is shown in Fig. 2.11.

Fig. 2.5 provides a general runtime overview of the system on real-world datasets, where around 8Hz is achieved on most datasets. Note in the *make submaps* stage we only account for pure odometry. Optional feature-based loop closure detection may increase the local trajectory accuracy, with the cost of $2{\sim}3$ times runtime. A major time consuming operation in *refine registration* is point cloud downsampling on CPU, which can be accelerated on GPU.

2.4.2 Reconstruction Results

We now show the qualitative reconstruction results on the TUM dataset in Fig. 2.7. The pose graph of the submaps is also illustrated, from which we can observe correct loop closures are found with pairwise FGR. Additional experimental results on *lounge*, *copyroom*, and *stonewall* can be viewed in Fig. 2.9. We can easily observe that the details are well preserved in the scenes without drift or apparent artifacts. Fig. 6.2 shows reconstruction results on more challenging scenes *boardroom* and *apartment* with more than 20K frames. Our system can still successfully close the loops and output accurate models, while other online SLAM systems fail.

Due to difficulties in configuration, we do not run BundleFusion [58] on our machine. Instead, we compare against their results on the BundleFusion dataset qualitatively. In Fig. 2.10 we can see that while producing similar results, our pipeline can better align surfaces.



(a) stonewall (b) copyroom (c) lounge

Figure 2.9: Qualitative results on *stonewall*, *lounge*, and *copyroom*.

As for quantitative results, we show the heatmap of cloud-to-cloud distances between estimated models and ground truth on the *livingroom* from the Redwood

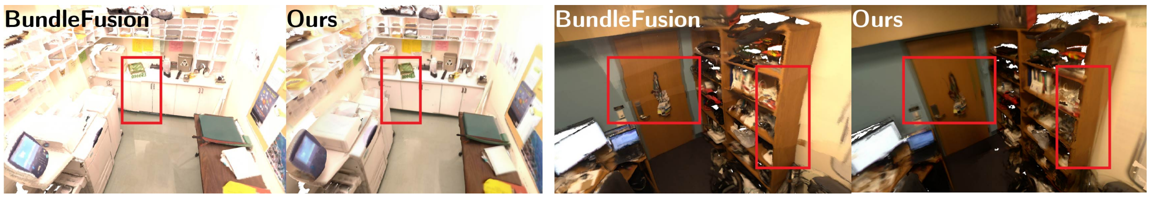


Figure 2.10: Qualitative comparison to BundleFusion on *copyroom* and *office3* in *bundlefusion* dataset. Surfaces are better aligned in our reconstructed models.

Table 2.1: Reconstruction Accuracy (meter) on simulated and real-world datasets with ground truth models.

SYSTEM	*livingroom1*		*livingroom2*		*office1*		*office2*		*apartment*		*boardroom*	
	MEAN	STD	MEAN	STD	MEAN	STD	MEAN	STD	MEAN	STD	MEAN	STD
InfiniTAM [119]	0.0391	0.0451	**0.0045**	0.0018	0.0661	0.0504	**0.0052**	0.0028	fail	-	fail	-
ElasticFusion [237]	0.0060	0.0036	0.0100	0.0051	0.0258	0.0306	0.0074	0.0038	fail	-	fail	-
Ours	**0.0049**	0.0043	0.0068	0.0049	**0.0051**	0.0040	0.0054	0.0043	**0.0533**	0.0521	**0.0607**	0.0567

simulated dataset. The ground truth model is adapted from [100]. The heatmaps are computed by *CloudCompare* software. In Fig. 2.8, we can see there are only minor discrepancies between our output model and the ground truth, while a shift is observable in the online systems. More results on the Redwood simulated and Indoor LIDAR-RGBD datasets are listed in Table 2.1. Our system has consistently higher reconstruction accuracy, and works on large scenes where other online systems fail to recover the whole scene.

2.4.3 Additional Result using Mobile Device

Our system not only works on laptops and PCs, but also on mobile devices supporting CUDA. Given real-world data collected from an Intel RealSense, our system is able to reconstruct the road in minutes on a NVIDIA Jetson TX2, see Fig. 2.12.

2.5 Conclusions

We implemented a GPU-accelerated dense RGB-D offline reconstruction system which runs fast on real-world datasets while maintaining state-of-the-art reconstruction results. The system is open source, and the major components can be used separately in various applications, such as real-time SLAM and object recognition.

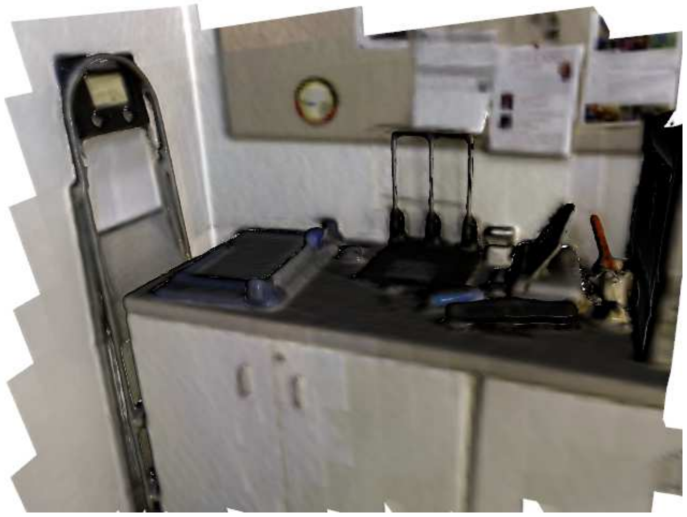


Figure 2.11: A screenshot of real-time Marching Cubes for voxels in frustum.

In the future, we intend to introduce advanced relocalizers to replace pair-wise submap matching for higher speed and robustness in loop closure detection. Equipped with an efficient relocalizer, we may bring up an online system that accepts streaming input frames. Another research direction is a fast 3D nearest neighbor search. Advance techniques such as parallel cuckoo hashing in [4] may be considered.

Acknowledgement

We thank Ben Guidarelli for the outdoor road reconstruction experiments on NVIDIA Jetson TX2.

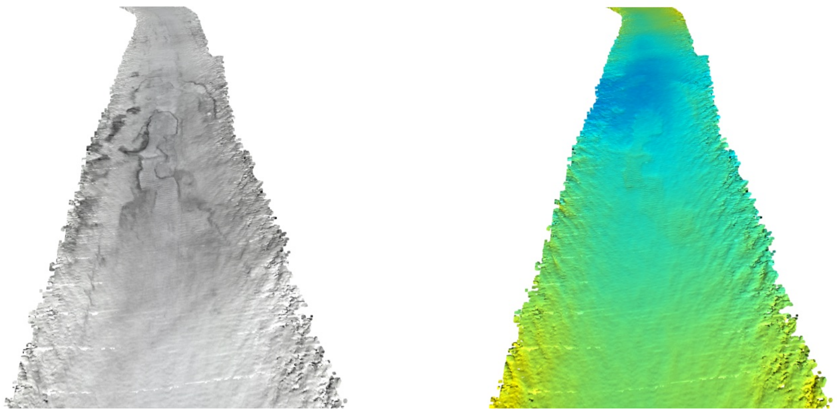


Figure 2.12: Real-world road reconstrction and elevation visualization. Results are from Ben Guidarelli.

Part II

Sparsity in Pose Estimation

Chapter 3

Deep Global Registration

3.1 Introduction

A variety of applications, including 3D reconstruction, tracking, pose estimation, and object detection, invoke 3D registration as part of their operation [32, 186, 198]. To maximize the accuracy and speed of 3D registration, researchers have developed geometric feature descriptors [50, 60, 125, 233], pose optimization algorithms [148, 201, 249, 271], and end-to-end feature learning and registration pipelines [6, 233].

In particular, recent end-to-end registration networks have proven to be effective in relation to classical pipelines. However, these end-to-end approaches have some drawbacks that limit their accuracy and applicability. For example, PointNetLK [6] uses globally pooled features to encode the entire geometry of a point cloud, which decreases spatial acuity and registration accuracy. Deep closest point [233] makes strong assumptions on the distribution of points and correspondences, which do not hold for partially overlapping 3D scans.

In this work, we propose three modules for robust and accurate registration that resolve these drawbacks: a 6-dimensional convolutional network for correspondence confidence estimation, a differentiable Weighted Procrustes method for scalable registration, and a robust $\mathrm{SE}(3)$ optimizer for fine-tuning the final alignment.

The first component is a 6-dimensional convolutional network that analyzes the geometry of 3D correspondences and estimates their accuracy. Our approach

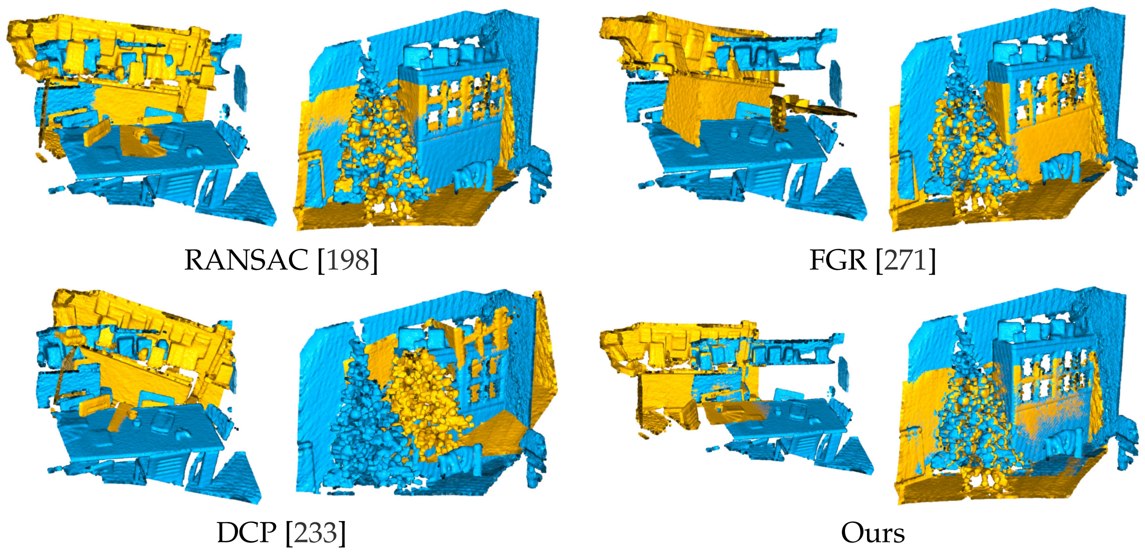


Figure 3.1: Pairwise registration results on the 3DMatch dataset [260]. Our method successfully aligns a challenging 3D pair (*left*), while RANSAC [198], FGR [271], and DCP [233] fail. On an easier pair (*right*), our method achieves finer alignment.

is inspired by a number of learning-based methods for estimating the validity of correspondences in 2D [187, 256] and 3D [178]. These methods stack coordinates of correspondence pairs, forming a vector $[\mathbf{x}; \mathbf{y}] \in \mathbb{R}^{2\times D}$ for each correspondence $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$. Prior methods treat these $2 \times D$-dimensional vectors as a set, and apply global set processing models for analysis. Such models largely disregard local geometric structure. Yet the correspondences are embedded in a metric space ($\mathbb{R}^{2\times D}$) that induces distances and neighborhood relationships. In particular, 3D correspondences form a geometric structure in 6-dimensional space [52] and we use a high-dimensional convolutional network to analyze the 6D structure formed by correspondences and estimate the likelihood that a given correspondence is correct (*i.e.*, an inlier).

The second component we develop is a differentiable Weighted Procrustes solver. The Procrustes method [90] provides a closed-form solution for rigid registration in $\mathrm{SE}(3)$. A differentiable version of the Procrustes method by Wang *et al.* [233] has been used for end-to-end registration. However, the differentiable Procrustes method passes gradients through coordinates, which requires $O(N^2)$ time and memory for N keypoints, limiting the number of keypoints that can be processed by the network. We use the inlier probabilities predicted by our first module (the 6D convolutional network) to guide the Procrustes method, thus forming a differentiable Weighted Procrustes method. This method passes gradients through the weights associated with correspondences rather than correspondence coordinates. The computational complexity of the Weighted Procrustes method is linear in the number of correspondences, allowing the registration pipeline to use dense correspondence sets rather than sparse keypoints. This substantially increases registration accuracy.

Our third component is a robust optimization module that fine-tunes the alignment produced by the Weighted Procrustes solver. This optimization module minimizes a differentiable loss via gradient descent on the continuous $\mathrm{SE}(3)$ representation space [274]. The optimization is fast since it does not require neighbor search in the inner loop [265].

Experimentally, we validate the presented modules on a real-world pairwise registration benchmark [260] and large-scale scene reconstruction datasets [46, 100, 180]. We show that our modules are robust, accurate, and fast in comparison to both classical global registration algorithms [7, 198, 249, 271] and recent end-to-end approaches [6, 178, 233]. All training and experiment scripts are available online[1].

3.2 Related Work

We divide the related work into three categories following the stages of standard registration pipelines that deal with real-world 3D scans: feature-based correspondence matching, outlier filtering, and pose optimization.

Feature-based correspondence matching. The first step in many 3D registration pipelines is feature extraction. Local and global geometric structure in 3D is analyzed to produce high-dimensional feature descriptors, which can then be used to establish correspondences.

Traditional hand-crafted features commonly summarize pairwise or higher-order relationships in histograms [116, 197, 198, 199, 226]. Recent work has shifted to learning features via deep networks [125, 260]. A number of recent methods

[1]https://github.com/chrischoy/DeepGlobalRegistration

are based on global pooling models [60, 266], while others use convolutional networks [50, 86].

Our work is agnostic to the feature extraction mechanism. Our modules primarily address subsequent stages of the registration pipeline and are compatible with a wide variety of feature descriptors.

Outlier filtering. Correspondences produced by matching features are commonly heavily contaminated by outliers. These outliers need to be filtered out for robust alignment. A widely used family of techniques for robust model fitting is based on RANdom SAmple Consensus (RANSAC) [3, 106, 158, 198, 201], which iteratively samples small sets of correspondences in the hope of sampling a subset that is free from outliers. Other algorithms are based on branch-and-bound [249], semi-definite programming [149, 154], and maximal clique selection [35]. These methods are accurate, but commonly require longer iterative sampling or more expensive computation as the signal-to-noise ratio decreases. One exception is TEASER [35], which remains effective even with high outlier rates. Other methods use robust loss functions to reject outliers during optimization [25, 271].

Our work uses a convolutional network to identify inliers and outliers. The network needs only one feed-forward pass at test time and does not require iterative optimization.

Pose optimization. Pose optimization is the final stage that minimizes an alignment objective on filtered correspondences. Iterative Closest Points (ICP) [22] and Fast Global Registration (FGR) [271] use second-order optimization to optimize poses. Makadia *et al.* [151] propose an iterative procedure to minimize correlation scores. Maken *et al.* [152] propose to accelerate this process by stochastic gradient descent.

Recent end-to-end frameworks combine feature learning and pose optimization. Aoki *et al.* [6] combine PointNet global features with an iterative pose optimization method [148]. Wang *et al.* [233, 234] train graph neural network features by backpropagating through pose optimization.

We further advance this line of work. In particular, our Weighted Procrustes method reduces the complexity of optimization from quadratic to linear and enables the use of dense correspondences for highly accurate registration of real-world scans.

3.3 Deep Global Registration

3D reconstruction systems typically take a sequence of partial 3D scans as input and recover a complete 3D model of the scene. These partial scans are scene fragments, as shown in Fig. 6.2. In order to reconstruct the scene, reconstruction systems often begin by aligning pairs of fragments [46]. This stage is known as pairwise registration. The accuracy and robustness of pairwise registration are critical and often determine the accuracy of the final reconstruction.

Our pairwise registration pipeline begins by extracting pointwise features. These are matched to form a set of putative correspondences. We then use a high-dimensional convolutional network (ConvNet) to estimate the veracity of each correspondence. Lastly, we use a Weighted Procrustes method to align 3D scans given correspondences with associated likelihood weights, and refine the result by optimizing a robust objective.

The following notation will be used throughout the paper. We consider two point clouds, $X = [\mathbf{x}_1, ..., \mathbf{x}_{N_x}] \in \mathbb{R}^{3\times N_x}$ and $Y = [\mathbf{y}_1, ..., \mathbf{y}_{N_y}] \in \mathbb{R}^{3\times N_y}$, with N_x and N_y points respectively, where $\mathbf{x}_i, \mathbf{y}_j \in \mathbb{R}^3$. A correspondence between $\mathbf{x}_i$ and $\mathbf{y}_j$ is denoted as $\mathbf{x}_i \leftrightarrow \mathbf{y}_j$ or (i, j).

3.3.1 Feature Extraction

To prepare for registration, we extract pointwise features that summarize geometric context in the form of vectors in metric feature space. Our pipeline is compatible with many feature descriptors. We use Fully Convolutional Geometric Features (FCGF) [50], which have recently been shown to be both discriminative and fast. FCGF are also compact, with dimensionality as low as 16 to 32, which supports rapid neighbor search in feature space.

3.3.2 Correspondence Confidence Prediction

Given the features $\mathcal{F}_x = \{\mathbf{f}_{\mathbf{x}_1}, ..., \mathbf{f}_{\mathbf{x}_{N_x}}\}$ and $\mathcal{F}_y = \{\mathbf{f}_{\mathbf{y}_1}, ..., \mathbf{f}_{\mathbf{y}_{N_x}}\}$ of two 3D scans, we use the nearest neighbor in the feature space to generate a set of putative correspondences or matches $\mathcal{M} = \{(i, \arg\min_j \|\mathbf{f}_{\mathbf{x}_i} - \mathbf{f}_{\mathbf{y}_j}\|) | i \in [1, ..., N_x]\}$. This procedure is deterministic and can be hand-crafted to filter out noisy correspondences with ratio or reciprocity tests [271]. However, we propose to learn this heuristic filtering process through a convolutional network that learns to analyze the underlying geometric structure of the correspondence set.

We first provide a 1-dimensional analogy to explain the geometry of correspondences. Let A be a set of 1-dimensional points $A = \{0, 1, 2, 3, 4\}$ and B be another such set $B = \{10, 11, 12, 13, 14\}$. Here B is a translation of A: $B = \{a_i + 10 | a_i \in A\}$. If an algorithm returns a set of possible correspondences $\{(0, 10), (1, 11), (2, 12), (3, 13), (4, 14), (0, 14), (4, 10)\}$, then the set of correct correspondences (inliers) will form a line (first 5 pairs), whereas incorrect correspondences (outliers) will form random noise outside the line (last 2 pairs). If we extend this to 3D scans and pointclouds, we can also represent a 3D correspondence $\mathbf{x}_i \leftrightarrow \mathbf{y}_j$ as a point in 6-dimensional space $[\mathbf{x}_i^T, \mathbf{y}_j^T]^T \in \mathbb{R}^6$. The inlier correspondences will be distributed on a lower-dimensional surface in this 6D space, determined by the geometry of the 3D input. We denote $\mathcal{P} = \{(i, j) | \, \|T^*(\mathbf{x}_i) - \mathbf{y}_j\| < \tau, (i, j) \in \mathcal{M}\}$ as a set of inliers or a set of correspondences (i, j) that align accurately up to the threshold τ under the ground truth transformation T^*. Meanwhile, the outliers $\mathcal{N} = \mathcal{P}^C \cap \mathcal{M}$ will be scattered outside the surface $\mathcal{P}$. To identify the inliers, we use a convolutional network. Such networks have been proven effective in related dense prediction tasks, such as 3D point cloud segmentation [49, 91]. The convolutional network in our setting is in 6-dimensional space [52]. The network predicts a likelihood for each correspondence, which is a point in 6D space $[\mathbf{x}_i^T, \mathbf{y}_j^T]^T$. The prediction is interpreted as the likelihood that the correspondence is true: an inlier.

Note that the convolution operator is translation invariant, thus our 6D ConvNet will generate the same output regardless of the absolute position of inputs in 3D. We use a similar network architecture to Choy et al. [50] to create a 6D convolutional network with skip connections within the spatial resolution across the network. The architecture of the 6D ConvNet is shown in Fig. 3.2. During training, we use the binary cross-entropy loss between the likelihood prediction that a correspondence (i, j) is an inlier, $p_{(i,j)} \in [0, 1]$, and the ground-truth correspondences $\mathcal{P}$ to optimize the network parameters:

$$L_{\text{bce}}(\mathcal{M}, T^*) = \frac{1}{|\mathcal{M}|} \left(\sum_{(i,j) \in \mathcal{P}} \log p_{(i,j)} + \sum_{(i,j) \in \mathcal{N}} \log p_{(i,j)}^C \right), \tag{3.1}$$

where $p^C = 1 - p$ and $|\mathcal{M}|$ is the cardinality of the set of putative correspondences.

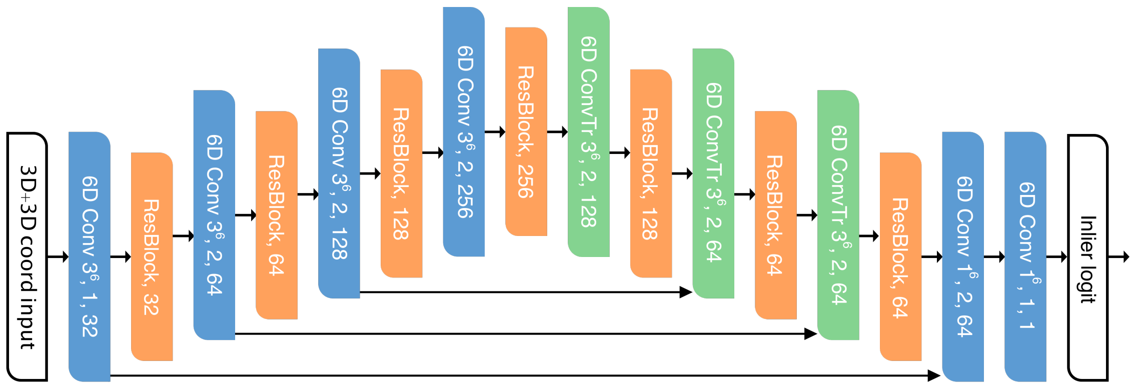


Figure 3.2: 6-dimensional convolutional network architecture for inlier likelihood prediction (Sec. 3.3.2). The network has a U-net structure with residual blocks between strided convolutions. Best viewed on the screen.

3.3.3 Weighted Procrustes for $\mathrm{SE}(3)$

The inlier likelihood estimated by the 6D ConvNet provides a weight for each correspondence. The original Procrustes method [90] minimizes the mean squared error between corresponding points $\frac{1}{N}\sum_{(i,j)\in\mathcal{M}}\|\mathbf{x}_i - \mathbf{y}_j\|^2$ and thus gives equal weight to all correspondences. In contrast, we minimize a weighted mean squared error $\sum_{(i,j)\in\mathcal{M}} w_{(i,j)}\|\mathbf{x}_i - \mathbf{y}_j\|^2$. This change allows us to pass gradients through the weights, rather than through the position [233], and enables the optimization to scale to dense correspondence sets.

Formally, Weighted Procrustes analysis minimizes:

$$\begin{aligned} e^2 &= e^2(R, \mathbf{t}; \mathbf{w}, X, Y) && (3.2)\\ &= \sum_{(i,j)\in\mathcal{M}} \tilde{w}_{(i,j)}(\mathbf{y}_j - (R\mathbf{x}_i + \mathbf{t}))^2 && (3.3)\\ &= \mathrm{Tr}\left((Y - RX - \mathbf{t}\mathbf{1}^T)W(Y - RX - \mathbf{t}\mathbf{1}^T)^T\right), && (3.4)\end{aligned}$$

where $\mathbf{1} = (1, ..., 1)^T$, $X = [\mathbf{x}_1, ..., \mathbf{x}_{|\mathcal{M}|}]$, and $Y = [\mathbf{y}_{J_1}, ..., \mathbf{y}_{J_{|\mathcal{M}|}}]$. J is a list of indices that defines the correspondences $\mathbf{x}_i \leftrightarrow \mathbf{y}_{J_i}$. $\mathbf{w} = [w_1, \cdots, w_{|\mathcal{M}|}]$ is the weight vector and $\tilde{\mathbf{w}} = [\tilde{w}_1, \cdots, \tilde{w}_{|\mathcal{M}|}] \triangleq \frac{\phi(\mathbf{w})}{\|\phi(\mathbf{w})\|_1}$ denotes the normalized weight after a nonlinear transformation ϕ that applies heuristic prefiltering. $W = \mathrm{diag}(\tilde{\mathbf{w}})$ forms the diagonal weight matrix.

Theorem 1. *: The R and $\boldsymbol{t}$ that minimize the squared error $e^2(R, \boldsymbol{t}) = \sum_{(i,j)} w_{(i,j)}(\boldsymbol{y}_j - R\boldsymbol{x}_i - \boldsymbol{t})^2$ are $\hat{\boldsymbol{t}} = (Y - RX)W\mathbf{1}$ and $\hat{R} = USV^T$ where $U\Sigma V^T = SVD(\Sigma_{xy})$, $\Sigma_{xy} = YKWKX^T$, $K = I - \sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T$, and $S = diag\,(1, \cdots, 1, det(U)det(V))$.*

Proof. The weighed mean squared error is defined as follows

$$e^2 = \sum_{(i,j)\in\mathcal{M}} \tilde{w}_{(i,j)}(\mathbf{y}_j - (R\mathbf{x}_i + \mathbf{t}))^2 \tag{3.5}$$

$$= \mathrm{Tr}\left((Y - RX - \mathbf{t}\mathbf{1}^T)W(Y - RX - \mathbf{t}\mathbf{1}^T)^T\right). \tag{3.6}$$

Theorem 2. *: The R and $\boldsymbol{t}$ that minimize the squared error $e^2(R, \boldsymbol{t}) = \sum_{(i,j)\in\mathcal{M}} w_{(i,j)}(\boldsymbol{y}_j - R\boldsymbol{x}_i - \boldsymbol{t})^2$ are $\hat{\boldsymbol{t}} = (Y - RX)W\mathbf{1}$ and $\hat{R} = USV^T$ where $U\Sigma V^T = SVD(\Sigma_{xy})$, $\Sigma_{xy} = YKWKX^T$, $K = I - \sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T$, and $S = diag\,(1, \cdots, 1, det(U)det(V))$.*

Proof. First, we differentiate e^2 w.r.t. $\mathbf{t}$ and equates the partial derivative to 0:

$$\frac{\partial}{\partial \mathbf{t}} e^2 = \frac{\partial}{\partial \mathbf{t}} \sum_{(i,j)\in\mathcal{M}} \tilde{\mathbf{w}}_{(i,j)}(\mathbf{y}_j - R\mathbf{x}_i - \mathbf{t}) \tag{3.7}$$

$$= -2\left(\sum_{(i,j)} \tilde{\mathbf{w}}_{(i,j)}\mathbf{y}_j - \sum_{(i,j)} \tilde{\mathbf{w}}_{(i,j)}R\mathbf{x}_i - \sum_{(i,j)} \tilde{\mathbf{w}}_{(i,j)}\mathbf{t}\right) = 0. \tag{3.8}$$

Thus, $\hat{\mathbf{t}} = (Y - RX)W\mathbf{1}$. Next, we substitute $X = KX + X\sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T$ on Eq. 3.6 and do the same for Y:

$$e^2 = \mathrm{Tr}\left((Y - RX - \mathbf{t}\mathbf{1}^T)W(Y - RX - \mathbf{t}\mathbf{1}^T)^T\right) \tag{3.9}$$

$$= \mathrm{Tr}\Big((YK + Y\sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T - RXK - RX\sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T - \mathbf{t}\mathbf{1}^T) \\ W(YK + Y\sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T - RXK - RX\sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T - \mathbf{t}\mathbf{1}^T)^T\Big)$$

$$= \mathrm{Tr}((YK - RXK)W(YK - RXK)^T) \tag{3.10}$$

$$= \mathrm{Tr}(YKWK^TY^T) + \mathrm{Tr}(RXKWK^TX^TR^R) - 2\mathrm{Tr}(YKWK^TX^TR^T), \tag{3.11}$$

where we use the fact that $W\mathbf{1}\mathbf{1}^T = \sqrt{\tilde{\mathbf{w}}}\sqrt{\tilde{\mathbf{w}}}^T$. The minimum occurs when we maximize the last negative term:

$$\max_R \text{Tr}(YKWK^TX^TR^T) = \sum_k \sigma_k(YKWK^TX^T), \tag{3.12}$$

where $\sigma_k(A)$ is the k-th largest singular value of the matrix A. Thus, the maximum of Eq. 3.12 occurs when $R = USV^T$ where $U\Sigma V^T = \text{SVD}(\Sigma_{xy})$, $\Sigma_{xy} = YKWKX^T$ and $S = \text{diag}\,(1, \cdots, 1, \text{det}(U)\text{det}(V))$. The last $\text{det}(U)\text{det}(V)$ is either +1 or -1 depending on the direction of the orthonomal basis. □

We can easily extend the above theorem to incorporate a scaling factor $c \in \mathbb{R}^+$, or anisotropic scaling for tasks such as scan-to-CAD registration, but in this paper we assume that partial scans of the same scene have the same scale.

The Weighted Procrustes method generates rotation $\hat{R}$ and translation $\hat{\mathbf{t}}$ as outputs that depend on the weight vector $\mathbf{w}$. In our current implementation, $\hat{R}$ and $\hat{\mathbf{t}}$ are directly sent to the robust registration module in Section 3.4 as an initial pose. However, we briefly demonstrate that they can also be embedded in an end-to-end registration pipeline, since Weighted Procrustes is differentiable. From a top-level loss function L of $\hat{R}$ and $\hat{\mathbf{t}}$, we can pass the gradient through the closed-form solver, and update parameters in downstream modules:

$$\frac{\partial}{\partial \mathbf{w}} L(\hat{R}, \hat{\mathbf{t}}) = \frac{\partial L(\hat{R}, \hat{\mathbf{t}})}{\partial \hat{R}} \frac{\partial \hat{R}}{\partial \hat{\mathbf{w}}} + \frac{\partial L(\hat{R}, \hat{\mathbf{t}})}{\partial \hat{\mathbf{t}}} \frac{\partial \hat{\mathbf{t}}(\hat{R}, \hat{\mathbf{w}})}{\partial \hat{\mathbf{w}}}, \tag{3.13}$$

where $L(\hat{R}, \hat{\mathbf{t}})$ can be defined as the combination of differentiable rotation error (RE) and translation error (TE) between predictions $\hat{R}, \hat{\mathbf{t}}$ and ground-truth $R^*, \mathbf{t}^*$:

$$L_{\text{rot}}(\hat{R}) = \arccos \frac{\text{Tr}(\hat{R}^T R^*) - 1}{2}, \tag{3.14}$$

$$L_{\text{trans}}(\hat{\mathbf{t}}) = ||\hat{\mathbf{t}} - \mathbf{t}^*||_2^2, \tag{3.15}$$

or the Forbenius norm of relative transformation matrices defined in [6, 233]. The final loss is the weighted sum of L_{rot}, L_{trans}, and L_{bce}.

3.4 Robust Registration

In this section, we propose a fine-tuning module that minimizes a robust loss function of choice to improve the registration accuracy. We use a gradient-based method to refine poses, where a continuous representation [274] for rotations is adopted to remove discontinuities and construct a smooth optimization space. This module initializes the pose from the prediction of the Weighted Procrustes method. During iterative optimization, unlike Maken *et al.* [152], who find the nearest neighbor per point at each gradient step, we rely on the correspondence likelihoods from the 6D ConvNet, which is estimated only once per initialization.

In addition, our framework naturally offers a failure detection mechanism. In practice, Weighted Procrustes may generate numerically unstable solutions when the number of valid correspondences is insufficient due to small overlaps or noisy correspondences between input scans. By computing the ratio of the sum of the filtered weights to the total number of correspondences, *i.e.*$\sum_i \phi(w_i)/|\mathcal{M}|$, we can easily approximate the fraction of valid correspondences and predict whether an alignment may be unstable. When this fraction is low, we resort to a more time-consuming but accurate registration algorithm such as RANSAC [3, 198, 201] or a branch-and-bound method [249] to find a numerically stable solution. In other words, we can detect when our system might fail before it returns a result and fall back to a more accurate but time-consuming algorithm, unlike previous end-to-end methods that use globally pooled latent features [6] or a singly stochastic matrix [233] – such latent representations are more difficult to interpret.

3.4.1 $\mathrm{SE}(3)$ Representation and Initialization

We use the 6D representation of 3D rotation proposed by Zhou *et al.* [274], rather than Euler angles or quaternions. The new representation uses 6 parameters $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{R}^3$ and can be transformed into a 3×3 orthogonal matrix by

$$f\left(\begin{bmatrix} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{bmatrix}\right) = \begin{bmatrix} | & | & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 \\ | & | & | \end{bmatrix}, \tag{3.16}$$

where $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \in \mathbb{R}^3$ are $\mathbf{b}_1 = N(\mathbf{a}_1)$, $\mathbf{b}_2 = N(\mathbf{a}_2 - (\mathbf{b}_1 \cdot \mathbf{a}_2)\mathbf{b}_1)$, and $\mathbf{b}_3 = \mathbf{b}_1 \times \mathbf{b}_2$, and $N(\cdot)$ denotes L2 normalization. Thus, the final representation that we use is $\mathbf{a}_1, \mathbf{a}_2, \mathbf{t}$ which are equivalent to $R, \mathbf{t}$ using Eq. 3.16.

To initialize $\mathbf{a}_1, \mathbf{a}_2$, we simply use the first two columns of the rotation matrix R, i.e., $\mathbf{b}_1$, $\mathbf{b}_2$. For convenience, we define f^{-1} as $f^{-1}(f(R)) = R$ though this inverse function is not unique as there are infinitely many choices of $\mathbf{a}_1, \mathbf{a}_2$ that map to the same R.

3.4.2 Energy Minimization

We use a robust loss function to fine-tune the registration between predicted inlier correspondences. The general form of the energy function is

$$E(R, \mathbf{t}) = \sum_{i=1}^{n} \phi(w_{(i,J_i)}) L(\mathbf{y}_{J_i}, R\mathbf{x}_i + \mathbf{t}), \tag{3.17}$$

where $\tilde{w}_i$ and J_i are defined as in Eq. 3.5 and $\phi(\cdot)$ is a prefiltering function. In the experiments, we use $\phi(w) = I[w > \tau]w$, which clips weights below τ elementwise as neural network outputs bounded logit scores. $L(\mathbf{x}, \mathbf{y})$ is a pointwise loss function between $\mathbf{x}$ and $\mathbf{y}$; we use the Huber loss in our implementation. The energy function is parameterized by R and $\mathbf{t}$ which in turn are represented as $\mathbf{a}_1, \mathbf{a}_2, \mathbf{t}$. We can apply first-order optimization algorithms such as SGD, Adam, etc. to minimize the energy function, but higher-order optimizers are also applicable since the number of parameters is small. The complete algorithm is described in Alg. 1.

3.5 Experiments

We analyze the proposed model in two registration scenarios: pairwise registration where we estimate an $\mathrm{SE}(3)$ transformation between two 3D scans or fragments, and multi-way registration which generates a final reconstruction and camera poses for all fragments that are globally consistent. Here, pairwise registration serves as a critical module in multi-way registration.

For pairwise registration, we use the 3DMatch benchmark [260] which consists of 3D point cloud pairs from various real-world scenes with ground truth trans-

Algorithm 1: Deep Global Registration

```
Input: $X \in \mathbb{R}^{n\times 3}, Y \in \mathbb{R}^{m\times 3}$
Output: $R \in \mathbb{R}^{3\times 3}, \mathbf{t} \in \mathbb{R}^{3\times 1}$
1  $\mathcal{F}_x \leftarrow \text{Feature}(X)$                                        // § 3.3.1
2  $\mathcal{F}_y \leftarrow \text{Feature}(Y)$
3  $J_{x\rightarrow y} \leftarrow \text{NearestNeighbor}(\mathcal{F}_x, \mathcal{F}_y)$                  // § 3.3.2
4  $\mathcal{M} \leftarrow \{(i, J_{x\rightarrow y,i}) \mid i \in [1, ..., n]\}$
5  $\mathbf{w} \leftarrow \text{InlierProbability}(\mathcal{M})$
6  if $\mathbb{E}_i \phi(w_i) < \tau_s$ then
7      return SafeGuardRegistration$(X, Y)$                             // §3.4
8  else
9      $\hat{R}, \hat{\mathbf{t}} \leftarrow \arg\min_{R,\mathbf{t}} e^2(R, \mathbf{t}; \mathbf{w}, X, Y)$                     // § 3.3.3
10     $\mathbf{a} \leftarrow f^{-1}(\hat{R}), \mathbf{t} \leftarrow \hat{\mathbf{t}}$                                  // § 3.4.1
11     while *not converging* do
12         $\ell \leftarrow \sum_{(i,j)\in\mathcal{M}} \phi(w_{(i,j)}) L(Y_j, f(\mathbf{a})X_i + \mathbf{t})$
13         $\mathbf{a} \leftarrow \text{Update}(\mathbf{a}, \frac{\partial}{\partial \mathbf{a}} \ell(\mathbf{a}, \mathbf{t}))$
14         $\mathbf{t} \leftarrow \text{Update}(\mathbf{t}, \frac{\partial}{\partial \mathbf{t}} \ell(\mathbf{a}, \mathbf{t}))$
15     return $f(\mathbf{a}), \mathbf{t}$
```

formations estimated from RGB-D reconstruction pipelines [58, 97]. We follow the train/test split and the standard procedure to generate pairs with at least 30% overlap for training and testing [50, 60]. For multi-way registration, we use the simulated Augmented ICL-NUIM dataset [46, 100] for quantitative trajectory results, and Indoor LiDAR RGB-D dataset [180] and Stanford RGB-D dataset [46] for qualitative registration visualizations. Note in this experiment we use networks trained on the 3DMatch training set and do not fine-tune on the other datasets. This illustrates the generalization abilities of our models. Lastly, we use KITTI LIDAR scans [82] for outdoor pairwise registration. As the official registration splits do not have labels for pairwise registration, we follow Choy *et al.* [50] to create pairwise registration train/val/test splits.

For all indoor experiments, we use 5cm voxel downsampling [198, 272], which randomly subsamples a single point within each 5cm voxel to generate point clouds with uniform density. For safeguard registration, we use RANSAC and the safeguard threshold $\tau_s = 0.05$, which translates to 5% of the correspondences should be valid. We train learning-based state-of-the-art models and our network on the training split of the 3DMatch benchmark. During training, we augment data by applying random rotations varying from -180 to 180 degrees around a random axis. Ground-truth pointwise correspondences are found using nearest neighbor search in 3D space. We train the 6-dimensional ConvNet on a single Titan XP with batch size 4. SGD is used with an initial learning rate 10^{-1} and an exponential learning rate decay factor 0.99.

3.5.1 Pairwise Registration

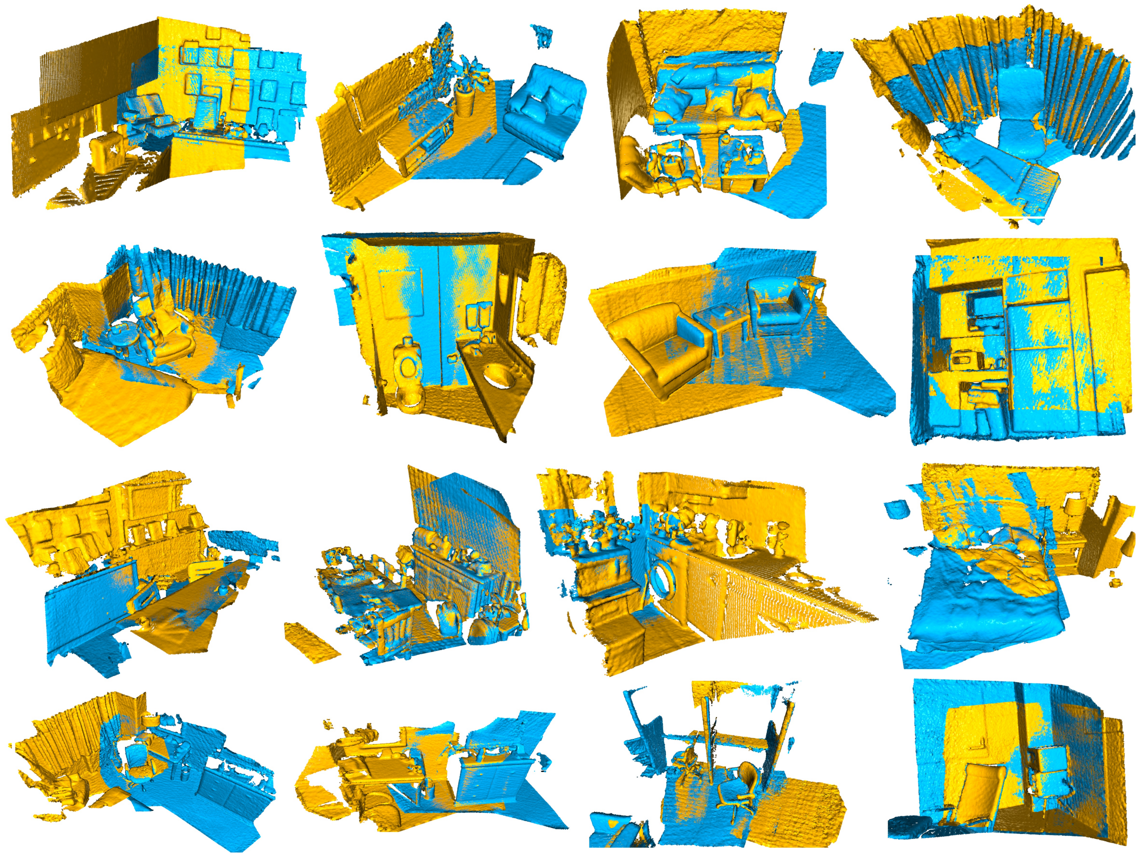


Figure 3.3: Global registration results of our method on all 8 different test scenes in 3DMatch [260]. Best viewed in color.

In this section, we report the registration results on the test set of the 3DMatch benchmark [260], which contains 8 different scenes as depicted in Fig. 3.3. We

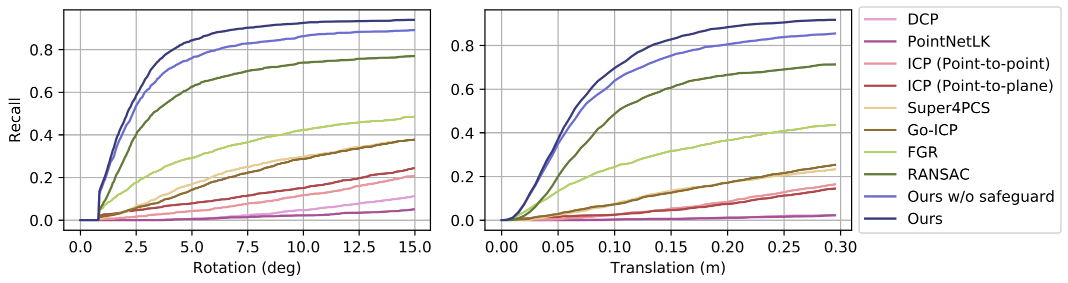


Figure 3.4: Overall pairwise registration recall (y-axis) on the 3DMatch benchmark with varying rotation (left image) and translation (right image) error thresholds (x-axis). Our approach outperforms baseline methods for all thresholds while being $6.5\times$ faster than the most accurate baseline.

measure translation error (*TE*) defined in Eq. 3.14, rotation error (*RE*) defined in Eq. 3.15, and *recall*. Recall is the ratio of successful pairwise registrations and we define a registration to be successful if its rotation error and translation error are smaller than predefined thresholds. Average TE and RE are computed only on these successfully registered pairs since failed registrations return poses that can be drastically different from the ground truth, making the error metrics unreliable.

We compare our methods with various classical methods [198, 249, 271] and state-of-the-art learning based methods [6, 178, 233, 234]. All the experiments are evaluated on an Intel i7-7700 CPU and a GTX 1080Ti graphics card except for Go-ICP [249] tested on an Intel i7-5820K CPU. In Table 3.1, we measure recall with the TE threshold 30cm which is typical for indoor scene relocalization [165], and RE threshold 15 degrees which is practical for partially overlapping scans from our experiments. In Fig. 3.4, we plot the sensitivity of recall on both thresholds by changing one threshold and setting the other to infinity. Fig. 3.5 includes detailed statistics on separate test scenes. Our system outperforms all the baselines on recall by a large margin and achieves the lowest translation and rotation error consistently on most scenes.

Classical methods. To compare with classical methods, we evaluate point-to-point ICP, Point-to-plane ICP, RANSAC [198], and FGR [271], all implemented in Open3D [272]. In addition, we test the open-source Python bindings of Go-ICP [249] and Super4PCS [158]. For RANSAC and FGR, we extract FPFH from

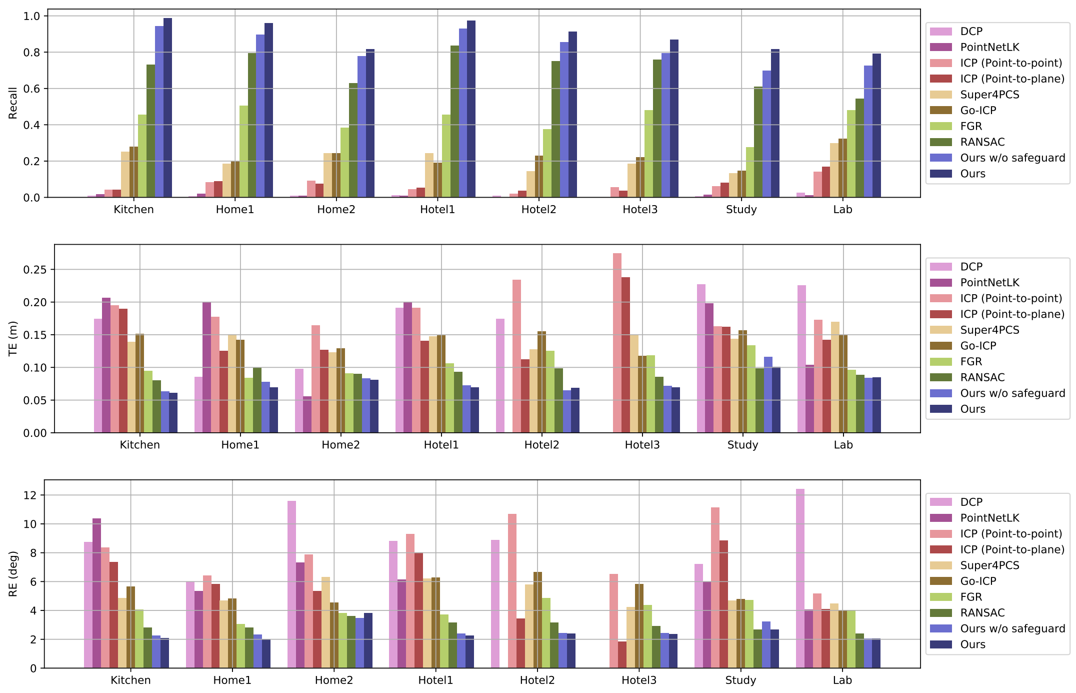


Figure 3.5: Analysis of 3DMatch registration results per scene. *Row 1*: recall rate (higher is better). *Row 2-3*: TE and RE measured on successfully registered pairs (lower is better). Our method is consistently better on all scenes, which were not seen during training. Note: a missing bar corresponds to zero successful alignments in a scene.

voxel-downsampled point clouds. The results are shown in Table 3.1.

ICP variants mostly fail as the dataset contains challenging 3D scan sequences with small overlap and large camera viewpoint change. Super4PCS, a sampling-based algorithm, performs similarly to Go-ICP, an ICP variant with branch-and-bound search.

Feature-based methods, FGR and RANSAC, perform better. When aligning 5cm-voxel-downsampled point clouds, RANSAC achieves recall as high as 70%, while FGR reaches 40%. Table 3.1 also shows that increasing the number of RANSAC iterations by a factor of 2 only improves performance marginally. Note that our method is about twice as fast as RANSAC with 2M iterations while achieving higher recall and registration accuracy.

Learning-based methods. We use 3DRegNet [178], Deep Closest Point (DCP) [233],

Table 3.1: *Row 1-6*: registration results of our method and classical global registration methods on point clouds voxelized with 5cm voxel size. Our method outperforms RANSAC and FGR while being as fast as FGR. *Row 7-10*: results of ICP variants. *Row 11 - 12*: results of learning-based methods. The learning-based methods generally fail on real-world scans. Time includes feature extraction.

	Recall	TE (cm)	RE (deg)	Time (s)
Ours w/o safeguard	85.2%	7.73	2.58	0.70
Ours	**91.3**%	**7.34**	**2.43**	1.21
FGR [271]	42.7%	10.6	4.08	**0.31**
RANSAC-2M [198]	66.1%	8.85	3.00	1.39
RANSAC-4M	70.7%	9.16	2.95	2.32
RANSAC-8M	74.9%	8.96	2.92	4.55
Go-ICP [249]	22.9%	14.7	5.38	771.0
Super4PCS [158]	21.6%	14.1	5.25	4.55
ICP (P2Point) [272]	6.04%	18.1	8.25	0.25
ICP (P2Plane) [272]	6.59%	15.2	6.61	0.27
DCP [233]	3.22%	21.4	8.42	0.07
PointNetLK [6]	1.61%	21.3	8.04	0.12

Table 3.2: ATE (cm) error on the Augmented ICL-NUIM dataset with simulated depth noise. For InfiniTAM, the loop closure module is disabled since it fails in all scenes. For BAD-SLAM, the loop closure module only succeeds in 'Living room 2'.

	ElasticFusion [237]	InfiniTAM [120]	BAD-SLAM [204]	Multi-way + FGR [271]	Multi-way + RANSAC [272]	Multi-way + Ours
Living room 1	66.61	46.07	fail	78.97	110.9	**21.06**
Living room 2	24.33	73.64	40.41	24.91	**19.33**	21.88
Office 1	**13.04**	113.8	18.53	14.96	14.42	15.76
Office 2	35.02	105.2	26.34	21.05	17.31	**11.56**
Avg. Rank	3	5	5	3.5	2.5	**2**

PRNet [234], and PointNetLK [6] as our baselines. We train all the baselines on 3DMatch with the same setup and data augmentation as ours for all experiments.

For 3DRegNet, we follow the setup outlined in [178], except that we do not manually filter outliers with ground truth, and train and test with the standard realistic setup. We find that the registration loss of 3DRegNet does not converge during training and the rotation and translation errors are consistently above 30 degrees and 1m during test.

We train Deep Closest Point (DCP) with 1024 randomly sampled points for each point cloud for 150 epochs [233]. We initialize the network with the pretrained weights provided by the authors. Although the training loss converges, DCP fails to achieve reasonable performance for point clouds with partial overlap. DCP uses a singly stochastic matrix to find correspondences, but this formulation assumes that all points in point cloud X have at least one corresponding point in the convex hull of point cloud Y. This assumption fails when some points in X have no corresponding points in Y, as is the case for partially overlapping fragments. We also tried to train PRNet [234] on our setup, but failed to get reasonable results due to random crashes and high-variance training losses.

Lastly, we fine-tune PointNetLK [6] on 3DMatch for 400 epochs, starting from the pretrained weights provided by the authors. PointNetLK uses a single feature that is globally pooled for each point cloud and regresses the relative pose between objects, and we suspect that a globally pooled feature fails to capture complex scenes such as 3DMatch.

In conclusion, while working well on object-centric synthetic datasets, current end-to-end registration approaches fail on real-world data. Unlike synthetic data, real 3D point cloud pairs contain multiple objects, partial scans, self-occlusion, substantial noise, and may have only a small degree of overlap between scans.

3.5.2 Multi-way Registration

Multi-way registration for RGB-D scans proceeds via multiple stages. First, the pipeline estimates the camera pose via off-the-shelf odometry and integrates multiple 3D scans to reduce noise and generate accurate 3D fragments of a scene. Next, a pairwise registration algorithm roughly aligns all fragments, followed by multi-way registration [46] which optimizes fragment poses with robust pose graph optimization [133].

We use a popular open-source implementation of this registration pipeline [272] and replace the pairwise registration stage in the pipeline with our proposed modules. Note that we use the networks trained on the 3DMatch training set and test on the multi-way registration datasets [46, 100, 180]; this demonstrates cross-dataset generalization.

We test the modified pipeline on the Augmented ICL-NUIM dataset [46, 100] for quantitative trajectory results, and Indoor LiDAR RGB-D dataset [180] and Stanford RGB-D dataset [46] for qualitative registration visualizations. We measure the absolute trajectory error (ATE) on the Augmented ICL-NUIM dataset with

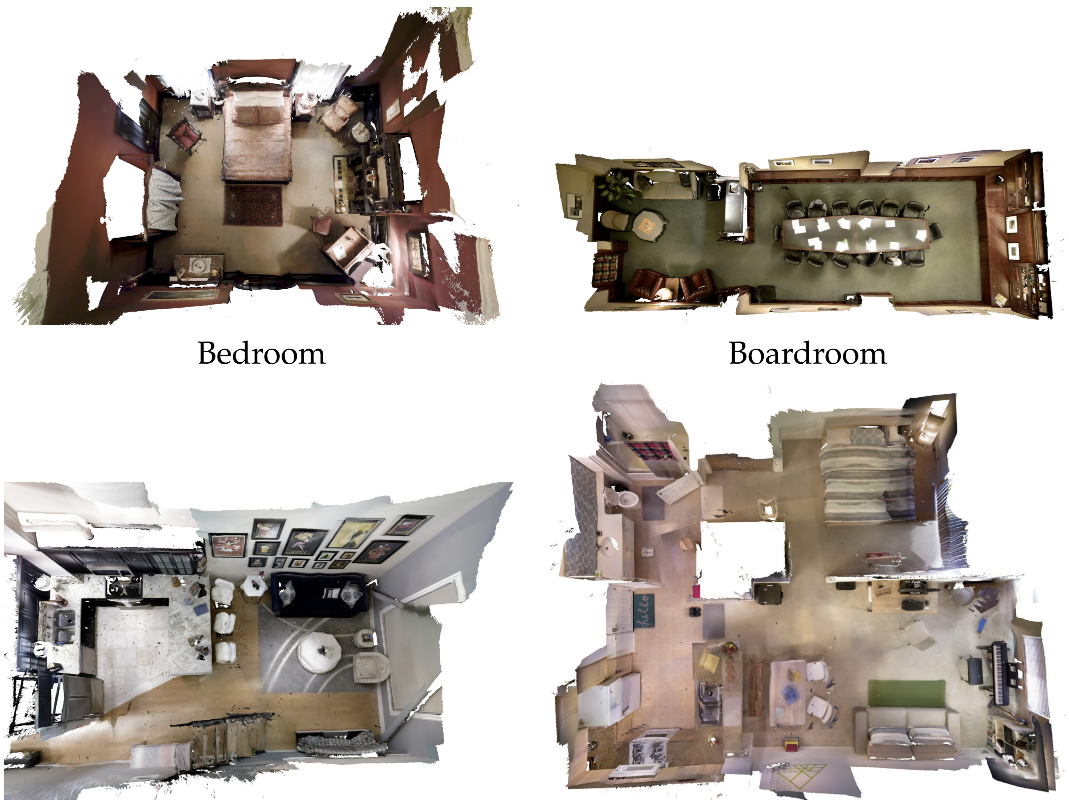


Figure 3.6: Full scene reconstructions from a modified multi-way registration pipeline with deep global registration for pairwise registration.

simulated depth noise. As shown in Table 3.2, compared to state-of-the-art online SLAM [120, 204, 237] and offline reconstruction methods [271], our approach yields consistently low error across scenes.

For qualitative results, we compare pairwise fragment registration on these scenes against FGR and RANSAC in Fig. 3.8-3.9. Full scene reconstruction results are shown in Fig. 3.6.

3.5.3 Outdoor LIDAR Registration

We use outdoor LIDAR scans from the KITTI dataset [82] for registration, following [50]. The registration split of Choy *et al.* [50] uses GPS-IMU to create pairs that are at least 10m apart and generated ground-truth transformation using GPS followed by ICP to fix errors in GPU readings. We use FCGF features [50] trained on the training set of the registration split to find the correspondences and trained the 6D ConvNet for inlier confidence prediction similar to how we trained the system for indoor registration. We use voxel size 30cm for downsampling point clouds for all experiments. Registration results are reported in Tab. 3.3 and visualized in Fig. 3.7.

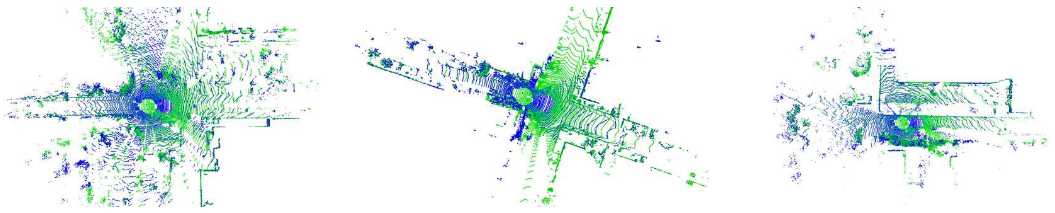


Figure 3.7: KITTI registration results. Pairs of scans are at least 10m apart.

Table 3.3: Registration on the KITTI test split [50, 82]. We use thresholds of 0.6m and 5 degrees. 'Ours + ICP' refers to our method followed by ICP for fine-grained pose adjustment. The runtime includes feature extraction.

	Recall	TE (cm)	RE (deg)	Time (s)
FGR [271]	0.2%	40.7	1.02	1.42
RANSAC [198]	34.2%	25.9	1.39	1.37
FCGF [50]	**98.2%**	10.2	0.33	6.38
Ours	96.9%	21.7	0.34	**2.29**
Ours + ICP	98.0	**3.46**	**0.14**	2.51

3.6 Conclusions

We presented Deep Global Registration, a learning-based framework that robustly and accurately aligns real-world 3D scans. To achieve this, we used a 6D convolutional network for inlier detection, a differentiable Weighted Procrustes algorithm for scalable registration, and a gradient-based optimizer for pose refinement. Experiments show that our approach outperforms both classical and learning-based registration methods, and can serve as a ready-to-use plugin to replace alternative registration methods in off-the-shelf scene reconstruction pipelines.

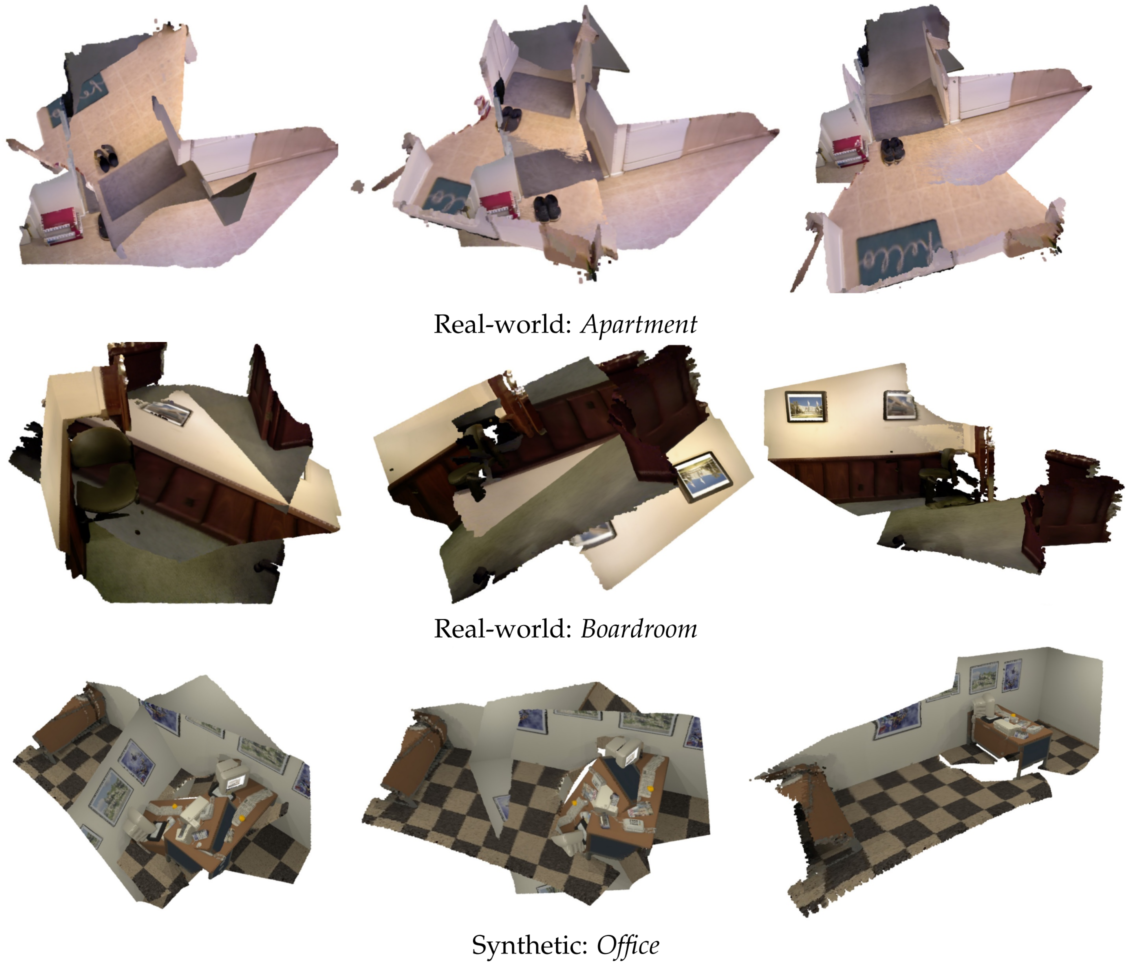


Figure 3.8: Fragment registrations on [46, 180]. From left to right: FGR [271], RANSAC [198], Ours. Our method succeeds on scenes with small overlaps or ambiguous geometry structures while other methods fail.

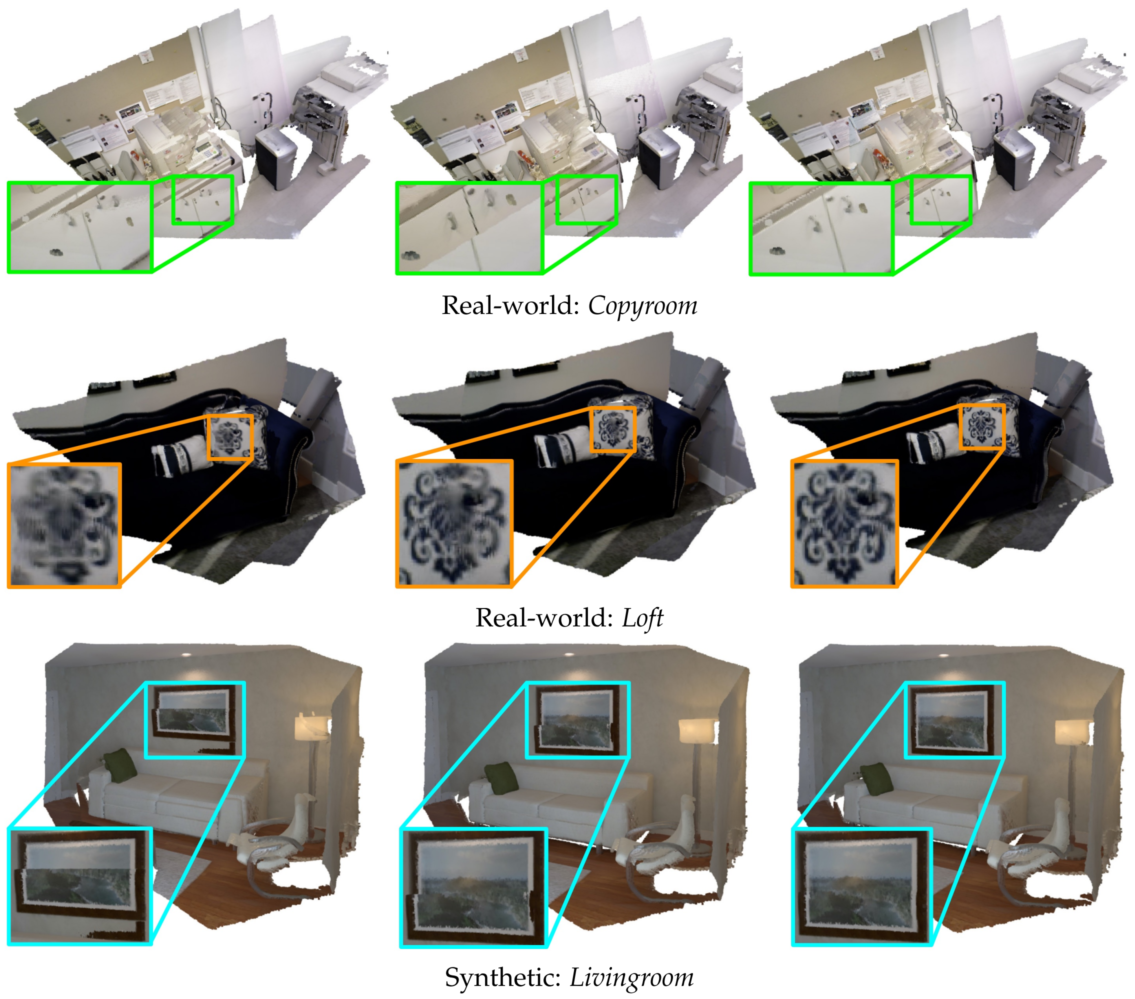


Figure 3.9: By combining Weighted Procrustes and gradient-based refinement, our method outputs more accurate registrations in one pass, leading to better aligned details.

Chapter 4

Self-supervised Geometric Perception

4.1 Introduction

Geometric perception is the task of estimating geometric models (*e.g.*, camera poses, rigid transformations, and 3D structures) from visual measurements (*e.g.*, images or point clouds). It is a fundamental class of problems in computer vision that has extensive applications in object detection and pose estimation [35, 259], motion estimation and 3D reconstruction [46, 65], simultaneous localization and mapping (SLAM) [30], structure from motion (SfM) [202], and virtual and augmented reality [127], to name a few.

Modern geometric perception typically consists of a *front-end* that detects, represents, and associates (sparse or dense) keypoints to establish *putative correspondences*, and a *back-end* that performs estimation of the geometric models while being robust to *outliers* (*i.e.*, incorrect correspondences). Traditionally, hand-crafted keypoint detectors and feature descriptors, such as SIFT [147] and FPFH [198], have been used for feature matching in 2D images and 3D point clouds. Despite being general and efficient to compute, hand-crafted features typically lead to an overwhelming number of outliers so that robust estimation algorithms struggle to return accurate estimates of the geometric models. For example, it is not uncommon to have over 95% of the correspondences estimated from FPFH be outliers in point cloud registration [28, 246]. As a result, learning feature descriptors from data, particularly using deep neural networks, has become increasingly popular. Learned feature descriptors have been shown to consistently and significantly outperform their hand-crafted counterparts across applications such as relative camera pose estimation [200, 232], 3D point cloud registration [50, 86], and object detection and pose estimation [183, 222, 238, 259].

However, existing feature learning approaches have several major shortcomings. First, a large number of *ground-truth* geometric model labels are required for training. For example, ground-truth relative camera poses are needed for training image keypoint descriptors [71, 157, 232], pairwise rigid transformations are required for training point cloud descriptors [50, 86, 233, 240, 258], and object poses are used to train image keypoint predictors [183, 259]. Second, although obtaining ground-truth geometric labels is trivial in some controlled settings such as robotic manipulation [76], in general the labels come from full 3D reconstruction pipelines (*e.g.*, COLMAP [202], Open3D [272]) that require delicate parameter tuning, partial human supervision, and extra sensory information such as IMU and GPS. As a result, the success of feature learning is limited to a handful of datasets with ground-truth annotations [26, 57, 143, 238, 260].

In this paper, we ask the key question: *Can we design a general framework for feature learning that requires no ground-truth geometric labels or sophisticated reconstruction pipelines?* Our answer is affirmative.

Contributions. We formulate geometric perception as an optimization problem that jointly searches for the best feature descriptor (for correspondence matching) and the best geometric models given a large corpus of visual measurements. This formulation incorporates robust model fitting and deep feature learning as two *subproblems*: (i) *robust estimation* only searches for the geometric models, while consuming putative correspondences established from a given feature descriptor; (ii) *feature learning* searches purely for the feature descriptor, while relying on full supervision from the ground-truth geometric models. This generalization naturally endows geometric perception with an iterative algorithm that solves the joint optimization based on alternating minimization, which we name as *self-supervised geometric perception* (SGP). At each iteration, SGP alternates two meta-algorithms: a *teacher*, that generates geometric pseudo-labels using correspondences established from the learned features, and a *student*, that refines the learned features under the *noisy* supervision from the updated geometric models. SGP is initialized by generating geometric pseudo-labels using a bootstrap descriptor, *e.g.*, a descriptor that is hand-crafted or is trained using synthetic data. We apply SGP to solve two perception problems – relative camera pose estimation and 3D point cloud registration – and demonstrate that (i) SGP achieves on-par or superior performance compared to the supervised oracles; (ii) SGP sets the new state of the art on the MegaDepth [143] and 3DMatch [260] benchmarks.

4.2 Related Work

Deep feature learning. With the recent advance of deep learning, a plethora of deep features have been developed to replace classical hand-crafted feature descriptors such as SIFT [147] and FPFH [198] for correspondence matching, and boost the performance of geometric perception tasks. For 2D features, Choy *et al.* [48] develop Universal Correspondence Network (UCN) for visual correspondence estimation with metric contrastive learning. Tian *et al.* [224] introduce L2-Net to extract patch descriptors for keypoints. While these methods require direct correspondence supervision, Wang *et al.* [232] only use 2D-2D camera poses to supervise the learning of feature descriptors. The success of 2D feature learning extends to 3D. Khoury *et al.* [125] created Compact Geometric Features (CGF) by optimizing deep networks that map high-dimensional histograms into low-dimensional Euclidean spaces. Gojcic *et al.* [86] propose 3DSmoothNet for 3D keypoint descriptor generation with its network structure based on L2-Net. Choy *et al.* [50] developed fully convolutional geometric features (FCGF) based on sparse convolutions. Bai *et al.* [11] build D3Feat on kernel point convolution (KPConv) [223] and emphasize 3D keypoint detection. Since ground-truth 3D correspondences are non-trivial to obtain, nearest neighbor search using known 3D transformations is the standard supervision signal.

Robust estimation. Robust estimation ensures reliable geometric model estimation in the presence of outlier correspondences. *Consensus maximization* [44] and *M-estimation* [24] are the two popular formulations. Algorithms for solving both formulations can be divided into *fast heuristics*, *global solvers*, and *certifiable* *algorithms*. Fast heuristics, such as RANSAC [12, 13, 75] and GNC [5, 245, 271], are efficient but offer few performance guarantees. Global solvers, typically based on branch-and-bound [17, 18, 28, 111, 140, 248] or exhaustive search [9, 33, 45, 73], are globally optimal but often run in exponential time. Recently proposed certifiable algorithms [242, 243, 244, 246] combine fast heuristics with scalable optimality certification. Outlier-pruning methods [28, 35, 207] can significantly boost the robustness and efficiency of estimation algorithms. In this paper, we use robust estimation to *teach* feature learning.

Self-supervision. Self-supervision has been widely adopted in visual learning [115] to avoid massive human annotation. In such tasks, labels can be automatically generated by standard image operations [135, 263], classical vision algorithms [114, 142], or simulation [69, 191]. In real-world setups, geometric vision has actively employed self-supervision in optical flow [145], depth prediction [85, 230], visual odometry [250, 273], and registration [11, 50, 255]. These tasks rely on the supervision from camera poses or relative rigid transformations for image warping and correspondence generation, and thus benefit from well-established SLAM [164], 3D reconstruction [272], and SfM [202] pipelines. Although these systems are off-the-shelf, they usually require long execution times, delicate parameter tuning, and human supervision to safeguard their correctness. In this paper, we show how to perform self-supervised feature learning without 3D reconstruction pipelines and ground-truth geometric labels.

Self-training. Self-training [92, 254], as a special case of semi-supervised learning, has gained popularity in visual learning due to its potential to adapt to large-scale unlabeled data. Self-training first trains a model on a labeled dataset, then applies it on a larger unlabeled dataset to obtain *pseudo-labels* [136] for further training. Although pseudo-labels can be noisy, recent studies have shown that SOTA performance can be achieved on image classification [239, 278], and initial theoretical analyses have been proposed [235]. Our work uses robust estimation to generate pseudo-labels without initial supervised training, the first work to showcase the effectiveness of pseudo-labels in training feature descriptors for geometric perception.

4.3 The SGP Formulation

In this section, we first formulate geometric perception as a problem that *jointly* optimizes a correspondence matching function (*i.e.*, learning a descriptor) and the geometric models given a corpus of visual data (Section 4.3.1). Then we show that two of the most important research lines in computer vision, namely *robust estimation* and *feature learning*, correspond to fixing one part of the joint problem while optimizing the other part (Sections 4.3.2 and 4.3.3).

4.3.1 Joint Feature Learning and Model Estimation

We focus on geometric perception with pairwise correspondences between visual measurements.

Problem 1 (Geometric Perception)**.** *Consider a corpus of M* pairwise *visual measurements $\{\boldsymbol{a}_i, \boldsymbol{b}_i\}_{i=1}^{M}$, such as images or point clouds, and assume $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are related through a* geometric model *with unknown parameters $\boldsymbol{x}_i \in \mathcal{X}$, where $\mathcal{X}$ is the domain of the geometric models such as 3D poses. Suppose there is a preprocessing module ϕ that can extract a sparse or dense set of keypoint locations for each measurement, i.e.,*

$$\boldsymbol{p}_i^a = \phi(\boldsymbol{a}_i) \in \mathbb{R}^{d_a \times N_{a_i}}, \quad \boldsymbol{p}_i^b = \phi(\boldsymbol{b}_i) \in \mathbb{R}^{d_b \times N_{b_i}}, \tag{4.1}$$

for all $i = 1, \dots, M$, where d_a, d_b are the dimensions of the keypoint locations (e.g., 2 for images keypoints and 3 for point cloud keypoints), and N_{a_i}, N_{b_i} are the number of keypoints in $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ (w.l.o.g., assume $N_{a_i} \leq N_{b_i}$), then the problem of geometric perception seeks to jointly learn a correspondence function $\mathcal{C}$ and estimate the unknown geometric models $\boldsymbol{x}_i$ by solving the following optimization:

$$\min_{\mathcal{C}, \{\boldsymbol{x}_i\}_{i=1}^{M} \in \mathcal{X}^M} \quad \sum_{i=1}^{M} \sum_{k=1}^{N_{a_i}} \rho\left(r\left(\boldsymbol{x}_i, \boldsymbol{p}_{i,k}^a, \boldsymbol{q}_{i,k}^b\right)\right) \tag{4.2}$$

$$s.t. \qquad \boldsymbol{q}_{i,k}^b = \mathcal{C}(\boldsymbol{p}_{i,k}^a, \boldsymbol{a}_i, \boldsymbol{p}_i^b, \boldsymbol{b}_i), \tag{4.3}$$

where $\boldsymbol{p}_{i,k}^a \in \mathbb{R}^{d_a}$ denotes the location of the k-th keypoint in $\boldsymbol{a}_i$, $\boldsymbol{q}_{i,k}^b \in \mathbb{R}^{d_b}$ denotes the location of the corresponding *keypoint in $\boldsymbol{b}_i$, $r(\cdot)$ is the residual function that quantifies the mismatch between the two keypoints $\boldsymbol{p}_{i,k}^a$ and $\boldsymbol{q}_{i,k}^b$ under the geometric model $\boldsymbol{x}_i$, $\rho(\cdot)$*

is a robust cost function that penalizes the residuals, and $\mathcal{C}(\cdot)$ is a function that takes each keypoint in $\boldsymbol{a}_i$ as input and predicts the corresponding keypoint in $\boldsymbol{b}_i$, by learning features from the visual data.

To the best of our knowledge, Problem 1 is the first formulation that considers *joint* feature learning and model estimation in geometric perception. The correspondence function $\mathcal{C}$ typically contains a *learnable* feature descriptor (*e.g.*, parametrized by a deep neural network) and a matching function (*e.g.*, soft or hard nearest neighbor search) that generates correspondences using the learned descriptor. We now give two examples of Problem 1.

Example 1 (Relative Pose Estimation)**.** *Consider a corpus of image pairs $\{\boldsymbol{a}_i, \boldsymbol{b}_i\}_{i=1}^{M}$ with known camera intrinsics, where $\boldsymbol{a}_i, \boldsymbol{b}_i$ are RGB images, let $\phi(\cdot)$ be a keypoint detector, e.g.,* SIFT *[147],* SuperPoint *[62], or a dense random pixel location sampler [232], such that $\boldsymbol{p}_i^a = \phi(\boldsymbol{a}_i) \in \mathbb{R}^{2\times N_{a_i}}$ and $\boldsymbol{p}_i^b = \phi(\boldsymbol{b}_i) \in \mathbb{R}^{2\times N_{b_i}}$ are two sets of 2D keypoint locations. Relative pose estimation seeks to jointly learn a correspondence prediction function $\mathcal{C}$ and estimate the relative poses $\boldsymbol{x}_i = (\boldsymbol{R}_i, \boldsymbol{t}_i) \in \mathrm{SO}(3) \times \mathbb{S}^2$ between images.*[1] *In particular, following [232], let $\mathcal{C}$ be a composition of a deep feature descriptor $\mathcal{F}(\cdot)$, a* softmax *function [88], and a weighted average:*

$$\boldsymbol{q}_{i,k}^b = \sum_{j=1}^{N_{b_i}} \boldsymbol{p}_{i,j}^b \frac{\exp\left(\mathcal{F}(\boldsymbol{p}_{i,k}^a, \boldsymbol{a}_i)^\mathsf{T} \mathcal{F}(\boldsymbol{p}_{i,j}^b, \boldsymbol{b}_i)\right)}{\sum_{j=1}^{N_{b_i}} \exp\left(\mathcal{F}(\boldsymbol{p}_{i,k}^a, \boldsymbol{a}_i)^\mathsf{T} \mathcal{F}(\boldsymbol{p}_{i,j}^b, \boldsymbol{b}_i)\right)}, \tag{4.4}$$

where the descriptor $\mathcal{F}$ takes the image and the keypoint location as input and outputs a high-dimensional feature vector for each keypoint, i.e., $\mathcal{F}(\boldsymbol{p}_{i,k}^a, \boldsymbol{a}_i) \in \mathbb{R}^{d_\mathcal{F}}$, where $d_\mathcal{F}$ denotes the dimension of the descriptor, the softmax *function computes the probability of $\boldsymbol{p}_{i,j}^b$ being a match to $\boldsymbol{p}_{i,k}^a$ according to their inner product in the descriptor space, and the weighted average function returns the keypoint location as a weighted sum of all keypoint locations discounted by their matching probabilities.*

Example 2 (Point Cloud Registration)**.** *Consider a corpus of point cloud pairs $\{\boldsymbol{a}_i, \boldsymbol{b}_i\}_{i=1}^{M}$, where $\boldsymbol{a}_i, \boldsymbol{b}_i$ are 3D point clouds, let $\phi(\cdot)$ be a 3D keypoint detector, e.g.,* ISS3D *[267],* USIP *[141], or a dense uniform voxel downsampler [50], such that $\boldsymbol{p}_i^a = \phi(\boldsymbol{a}_i) \in \mathbb{R}^{3\times N_{a_i}}$, and $\phi(\boldsymbol{b}_i) \in \mathbb{R}^{3\times N_{b_i}}$ are two sets of 3D keypoints. Point cloud registration seeks to jointly learn a correspondence function $\mathcal{C}$ and estimate the rigid transformation $\boldsymbol{x}_i = (\boldsymbol{R}_i, \boldsymbol{t}_i) \in \mathrm{SO}(3) \times \mathbb{R}^3$*

[1]The translation $\boldsymbol{t} \in \mathbb{S}^2 \doteq \{\boldsymbol{t} \in \mathbb{R}^3 \mid \|\boldsymbol{t}\| = 1\}$ is up to scale.

between point clouds. In particular, following [50, 86], let $\mathcal{C}$ be a composition of a deep feature descriptor $\mathcal{F}(\cdot)$ and nearest neighbor search:

$$\boldsymbol{q}^b_{i,k} = \underset{\boldsymbol{p}^b_{i,j} \in \boldsymbol{p}^b_i}{\arg\min} \left\| \mathcal{F}(\boldsymbol{p}^b_{i,j}, \boldsymbol{b}_i) - \mathcal{F}(\boldsymbol{p}^a_{i,k}, \boldsymbol{a}_i) \right\|, \tag{4.5}$$

where the descriptor $\mathcal{F}$ takes the point cloud and the keypoint location as input and outputs a high-dimensional feature vector for each keypoint, i.e., $\mathcal{F}(\boldsymbol{p}^a_{i,k}, \boldsymbol{a}_i) \in \mathbb{R}^{d_\mathcal{F}}$, with $d_\mathcal{F}$ denoting the descriptor dimension, and condition (4.5) asks that the corresponding keypoint $\boldsymbol{q}^b_{i,k}$ is the keypoint among $\boldsymbol{p}^b_i$ that achieves the shortest distance to $\boldsymbol{p}^a_{i,k}$ in descriptor space.[2]

Examples 1-2 represent two key problems in vision that concern pose estimation from 2D-2D and 3D-3D measurements, all of which involve the coupling of correspondence matching (*a.k.a.* data association) and geometric model estimation. Interestingly, although little is known about how to solve Problem 1 directly, significant efforts have been made to solve its two subproblems.

4.3.2 Robust Estimation

Problem 2 (Robust Estimation)**.** *In Problem 1, assuming the correspondence matching function $\mathcal{C}$ is known, robust estimation seeks to estimate the unknown parameters of the geometric models given putative correspondences (corrupted by* outliers*), by optimizing the following objective:*

$$\min_{\{\boldsymbol{x}_i\}_{i=1}^M \in \mathcal{X}^M} \quad \sum_{i=1}^{M} \sum_{k=1}^{N_{a_i}} \rho\left(r\left(\boldsymbol{x}_i, \boldsymbol{p}^a_{i,k}, \boldsymbol{q}^b_{i,k}\right)\right). \tag{4.6}$$

Problem 2 shows that robust estimation is a subproblem of Problem 1 with a known and fixed correspondence function. Despite the nonconvexity of problem (4.6) (*e.g.*, due to a nonconvex $\mathcal{X}$ or a nonconvex ρ), research in robust estimation has focused on improving the robustness [28, 246], efficiency [13] and theoretical guarantees [244] of estimation algorithms to mitigate the adversarial

[2]Alternatively, one can establish correspondences through *cross check* [271] or *ratio test* [147]. In addition to (4.5), cross check asks $\boldsymbol{p}^a_{i,k}$ is also the closest keypoint to $\boldsymbol{q}^b_{i,k}$ among $\boldsymbol{p}^a_i$, while ratio test asks the ratio $\|\mathcal{F}(\boldsymbol{p}^a_{i,k}, \boldsymbol{a}_i) - \mathcal{F}(\boldsymbol{q}^b_{i,k}, \boldsymbol{b}_i)\| / \|\mathcal{F}(\boldsymbol{p}^a_{i,k}, \boldsymbol{a}_i) - \mathcal{F}(\boldsymbol{p}^b_{i,j}, \boldsymbol{b}_i)\|$ is below a predefined threshold $\zeta < 1$ for all $\boldsymbol{p}^b_{i,j} \neq \boldsymbol{q}^b_{i,k}$.

effects of outliers on the estimated geometric models.

4.3.3 Supervised Feature Learning

Problem 3 (Supervised Feature Learning)**.** *In Problem 1, assuming the parameters of the geometric models are known and denoting them as $\boldsymbol{x}_i^\circ, i = 1, \ldots, M$, feature learning seeks to find the best correspondence matching function $\mathcal{C}_{\boldsymbol{\theta}}$ by solving the following optimization problem:*

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{N_C}} \sum_{i=1}^{M} \sum_{k=1}^{N_{a_i}} \rho(r(\boldsymbol{x}_i^\circ, \boldsymbol{p}_{i,k}^a, \boldsymbol{q}_{i,k}^b)) s.t. \quad \boldsymbol{q}_{i,k}^b = \mathcal{C}_{\boldsymbol{\theta}}(\boldsymbol{p}_{i,k}^a, \boldsymbol{a}_i, \boldsymbol{p}_i^b, \boldsymbol{b}_i), \tag{4.7}$$

where the correspondence function is parametrized by the weights $\boldsymbol{\theta} \in \mathbb{R}^{N_C}$ of a deep (descriptor) neural network and N_C is the number of weight parameters in the network.

At first glance, the optimization (4.7) is different from the loss functions designed in the supervised feature learning literature [50, 232, 259]. However, the next proposition states that, if we take $\rho(\cdot)$ to be the truncated least squares (TLS) cost function, then common loss functions can be designed using the *Augmented Lagrangian Method* (ALM) [21].

Proposition 1 (Feature Learning as ALM)**.** *Let $\rho(r) = \min\{r^2, \bar{c}^2\}$ be the* TLS *cost function [244], where $\bar{c} > 0$ sets the maximum allowed inlier residual, supervised feature learning [50, 232] in Examples 1-2 can solve the optimization* (4.7)*. In particular, the loss functions in [50, 232] can be interpreted as the Augmented Lagrangian of problem* (4.7)*.*

Proof. See the Appendix of [247]. □

Proposition 1 states that, just as robust estimation algorithms optimize geometric models given a fixed correspondence matching function, supervised feature learning methods optimize the feature descriptor given known geometric models. In the next section, we show that this framework naturally allows us to solve Problem 1 by alternating the execution of robust estimation and feature learning.

4.4 The SGP Algorithm

We first give an overview of the SGP algorithm (Section 5.3), then discuss its applications (Section 5.6.4).

4.4.1 Overview

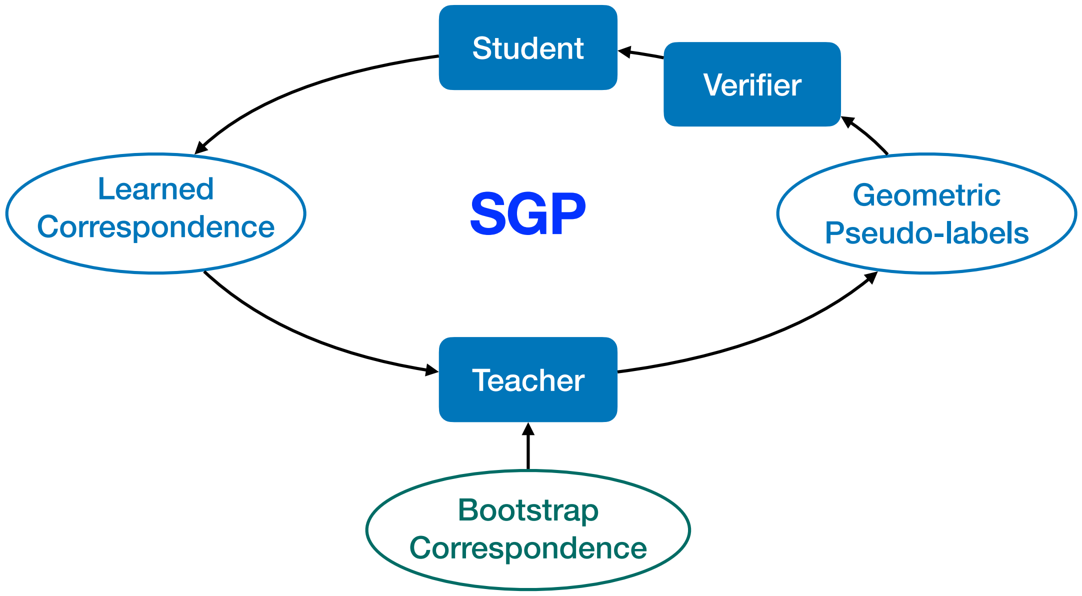


Figure 4.1: Algorithmic overview of SGP.

An overview of SGP is shown in Fig. 4.1, and details of SGP are summarized in Algorithm 2. SGP does not have access to the ground-truth geometric models and internally creates *geometric pseudo-labels*. SGP contains three key components: a *teacher*, a *student* and (optionally) a *verifier*.

Definition 1 (Teacher)**.** *An algorithm that estimates geometric pseudo-labels given a correspondence matcher.*

Definition 2 (Student)**.** *An algorithm that estimates the parameters of a correspondence matching function under the supervision of geometric models.*

Definition 3 (Verifier)**.** *An algorithm that verifies if a geometric model estimated by the teacher is correct.*

From the definitions above, one can see that a teacher is a solver for the robust estimation problem (4.6), while a student is a solver for the supervised feature learning problem (4.7). Because problems (4.6) and (4.7) are the two subproblems of the joint geometric perception problem (4.2), the SGP algorithm 2 alternates in executing the teacher and the student (*cf.* line 21-23), referred to as the *teacher-student loop*, to perform alternating minimization for the joint problem (4.2).

In particular, at the τ-th iteration of the teacher-student loop, the student initializes the network parameters at $\boldsymbol{\theta}$, and updates the parameters to $\boldsymbol{\theta}^{(\tau)}$, by minimizing problem (4.7) (using stochastic gradient descent) under the noisy "supervision" of the geometric pseudo-labels estimated from iteration $\tau - 1$ (line 21). The student either initializes $\boldsymbol{\theta}$ at random (line 17, referred to as retrain), or initializes $\boldsymbol{\theta}$ from the weights of the last iteration $\boldsymbol{\theta}^{(\tau-1)}$ (line 19, referred to as finetune). Then, using the correspondence function with updated parameters, denoted by $\mathcal{C}_{\boldsymbol{\theta}^{(\tau)}}$, the teacher solves robust estimation (4.6) to update the models (line 23).

Throughout the teach-student loop, neither the correspondence matcher nor the teacher are perfect, leading to a significant fraction of the geometric pseudo-labels being incorrect, which can potentially bias the student. Therefore, SGP optionally uses a verifier to generate a verified set of pseudo-labels, denoted by $\mathcal{S}$, that are more likely to be correct (line 11). If the flag verifyLabel is False, then $\mathcal{S} = [M]$ is the full set of pseudo-labels (line 13). The verifier design is application dependent, as discussed in Section 5.6.4.

An *initialization* is required to start the iterative updates in alternating minimization. To do so, we initialize the geometric models by performing model estimation using a *bootstrap matcher* $\mathcal{B}$ (line 6). Based on the specific application, the bootstrap matcher can be designed from a hand-crafted feature descriptor that requires no learning, or a descriptor that is trained with a small amount of data, or a descriptor that is trained on synthetic datasets. On the other hand, since we typically do not have prior information about the weights of $\mathcal{C}$, $\boldsymbol{\theta}^{(0)}$ is initialized at random.

Remark 1 (Implementation Considerations)**.** *(i)* Convergence*: In the current* SGP *implementation, we execute the teacher-student loops for a fixed number of iterations T. However, one can stop* SGP *if the difference between $\boldsymbol{x}_i^{(\tau)}$ and $\boldsymbol{x}_i^{(\tau-1)}$, or between $\boldsymbol{\theta}^{(\tau)}$ and $\boldsymbol{\theta}^{(\tau-1)}$ is below some threshold. One can also choose the best $\mathcal{C}$ from* SGP *by using a validation dataset if available. (ii)* Speedup*: When running the teacher to generate* *pseudo-labels (line 23) at each iteration, one can skip the updates for some labels that are already "stable". For example, if a label $\boldsymbol{x}_i$ remains unchanged for consecutively 3 iterations, or the robust solver achieves high confidence about $\boldsymbol{x}_i$ (e.g.,* RANSAC *has inlier rate over 80%), then the teacher can skip the update for $\boldsymbol{x}_i$.*

4.4.2 Applications

We now discuss the application of SGP to Examples 1-2.

SGP **for Example 1**. The teacher performs robust relative pose estimation [103]. Therefore, a good candidate for a teacher is RANSAC [75] (with Nister's 5-point method [173]) and its variants, such as GCRANSAC [12] and MAGSAC [13]. The student performs descriptor learning using relative camera pose supervision. Recent work CAPS [232] is able to learn a descriptor under the supervision of fundamental matrices, which can be computed from relative pose and camera intrinsics [103]. Therefore, CAPS is the student network. The verifier can be designed based on the *inlier rate* estimated by RANSAC, *i.e.*, the number of inlier matches divided by the total number of putative matches. Intuitively, the higher the inlier rate is, the more likely it is that RANSAC has found a correct solution. To initialize SGP, we use the hand-crafted SIFT descriptor (with ratio test) [147].

SGP **for Example 2**. The teacher performs robust registration. Many robust registration algorithms can serve as the teacher: RANSAC (with Horn's 3-point method [107]) and its variants, FGR [271], and TEASER++ [246]. As for the student, methods such as FCGF [50], 3DSmoothNet [86], and D3Feat [11] can learn point cloud descriptors under the supervision of rigid transformations. The verifier can be designed based on the *overlap ratio* computed from the estimated pose, *i.e.*, the number of point pairs that are close to each other after transformation, divided by the total number of points in the point cloud. One can also use the certifier in TEASER++ [246]. To initialize SGP, we can use the hand-crafted FPFH descriptor (with cross check) [198].

Remark 2 (Novelty)**.** *Hand-crafted descriptors, robust estimation and feature learning are mature areas in computer vision. In this paper, instead of creating new techniques in each area, we show that combining existing techniques from each field in the* SGP *framework can tackle self-supervised geometric perception in full generality.*

Remark 3 (Generality). *Although we only provide experimental results for relative pose estimation and point cloud registration, the joint optimization formulation in Problem 1 is general and the* SGP *algorithm 2 can be applied in any perception problem where a robust solver and a supervised feature learning method is available. For example, we also present the formulation for* object detection and pose estimation *[41, 183, 222, 259], and discuss the application of* SGP *in the Appendix.*

4.5 Experiments

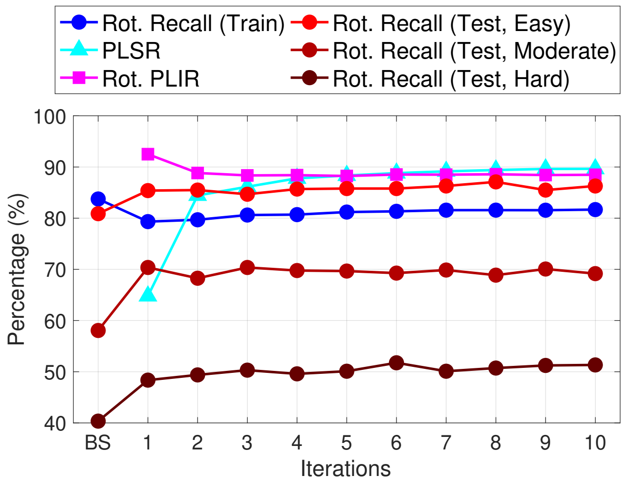
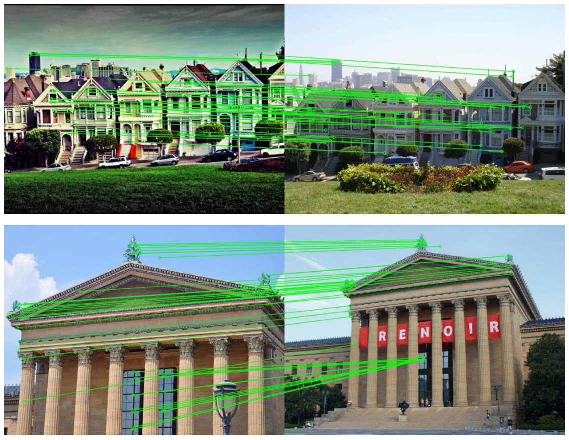


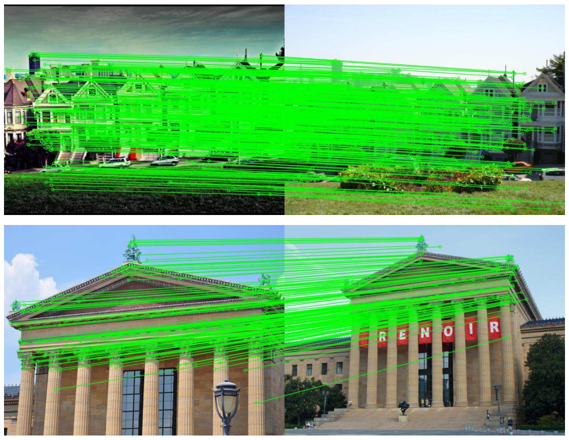
Figure 4.2: Dynamics of SGP on MegaDepth [143]. PLSR: *Pseudo-Label Survival Rate*. PLIR: *Pseudo-Label Inlier Rate*. BS: Boostrap.

We first provide results demonstrating successful applications of SGP to relative pose estimation (Section 4.5.1) and point cloud registration (Section 4.5.2), then report ablation studies on point cloud registration where we vary the algorithmic settings of SGP (Section 4.5.3). *Detailed experimental data are tabulated in the Appendix*.

4.5.1 Relative Pose Estimation

Setup. We first showcase SGP for Example 1 on the MegaDepth [143] benchmark containing a large collection of Internet images for the task of relative pose estimation. We adopted RANSAC10K (*i.e.*, RANSAC with maximum $10,000$ iterations) with


(a) Success (top) and failure (bottom) by SIFT.


(b) Successes by S-CAPS^T.

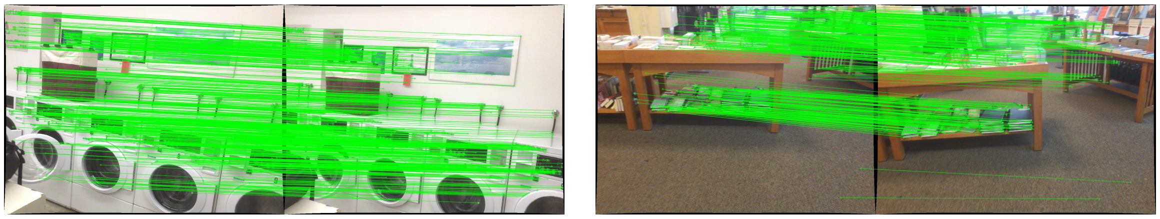

(c) Cross-dataset generalization of S-CAPS^T.

Figure 4.3: Qualitative results showing the improved performance of (b) S-CAPS^T over the bootstrap descriptor (a) SIFT for relative pose estimation on MegaDepth [143], and (c) cross-dataset generalization of S-CAPS^T for relative pose estimation on the ScanNet dataset [57]. Green lines are inlier correspondences estimated by RANSAC10K. S-CAPS^T outputs reliable and dense matches. [Best viewed digitally]

99.9% confidence and 0.001 inlier threshold as the teacher. We used the recently proposed CAPS [232] feature learning framework as the student.[3] To bootstrap SGP, we performed RANSAC10K with SIFT detector, SIFT descriptor, and 0.75 ratio test to initialize the geometric pseudo-labels (*i.e.*, relative poses).

To speed up the training of SGP, we sampled 10% of the original MegaDepth training set used in [232] uniformly at random, resulting in $78,836$ pairs of images *without* relative pose labels. To train CAPS, we modified the publicly available CAPS implementation[4], adopted a smaller batch size 5, and kept the Adam optimizer with initial learning rate 10^{-4}. We used finetune (*cf.* line 19) for the teacher-student

[3]We assumed known camera intrinsics so the fundamental matrix can be computed from the essential matrix to supervise CAPS.

[4]https://github.com/qianqianwang68/caps

loop, and in every iteration, we trained CAPS for $40,000$ steps. We trained SGP for a fixed number of $T = 10$ iterations.

In the teacher-student loop, we designed a verifier that prunes pseudo-labels according to the results of RANSAC10K – we only pass to the student pairs whose number of putative matches (either from SIFT with ratio test or CAPS with cross check) is above 100 and whose RANSAC estimated inlier rate is over 10%. Intuitively, pseudo-labels satisfying these two conditions are more likely to be correct.

We name the CAPS descriptor learned from SGP without ground-truth supervision as S-CAPS. We evaluated the performance of S-CAPS on (i) the MegaDepth test set, provided in [232], including $3,000$ image pairs equally divided into *easy*, *moderate*, and *hard* categories; (ii) the ScanNet [57] dataset to test cross-dataset generalization.

Results. Fig. 4.2 plots the dynamics of SGP on MegaDepth. PLSR stands for *Pseudo-Label Survival Rate* and is computed as $|\mathcal{S}|/M \times 100\%$, *i.e.*, the percentage of pseudo-labels that survived the verifier (*cf.* line 11). PLIR stands for *Pseudo-Label Inlier Rate* and denotes the percentage of correct labels in $\mathcal{S}$, a number that is not used by SGP but computed *a posteriori* using the ground-truth labels to show that SGP is robust to partially incorrect labels. Besides PLSR and PLIR, Fig. 4.2 plots the rotation recalls on both the training and the test sets (the translation recalls exhibit a similar trend and are shown in the Appendix).[5] The BS (bootstrap) iteration plots the training and test recalls using SIFT. We make the following observations from Fig. 4.2: (i) PLSR gradually increases and approaches 90% w.r.t. iterations, indicating that the S-CAPS descriptor establishes dense correspondences with high inlier ratio, encouraged by the verifier; (ii) PLIR remains close to 90%, and is always higher than the recall, indicating that the verifier is effective in removing wrong labels; (iii) S-CAPS gradually improves itself on both the training and the test sets. (iv) While SIFT works better than S-CAPS on the training set, S-CAPS significantly outperforms SIFT on the test set.

Table 4.1 compares the performance of two versions of S-CAPS to other SOTA methods. S-CAPS^T is the S-CAPS descriptor at the last iteration, while S-CAPS* is the S-CAPS descriptor that performs best on the MegaDepth test set. We see that both

[5]Recall is defined as the percentage of correctly estimated models divided by the total number of pairs. Following [232], we say a rotation or a translation is estimated correctly if it has angular error less than 10° w.r.t. to the groundtruth (note that translation is estimated up to scale).

versions of S-CAPS outperform the strong baseline using SIFT with ratio test and RANSAC10K, as well as the two SOTA results from the original CAPS [232] using both SIFT detector and SuperPoint detector [62].[6] Moreover, we report the performance of RANSAC10K plus the *supervised oracle*, CAPS°, that is trained using full ground-truth supervision, on the test set. One can see that S-CAPS, trained using only 10% of the unlabeled training set, performs on par compared with the supervised oracle.

Fig. 4.3 provides qualitative examples of correspondence matching results on both MegaDepth and ScanNet. More examples are provided in the Appendix.

4.5.2 Point Cloud Registration

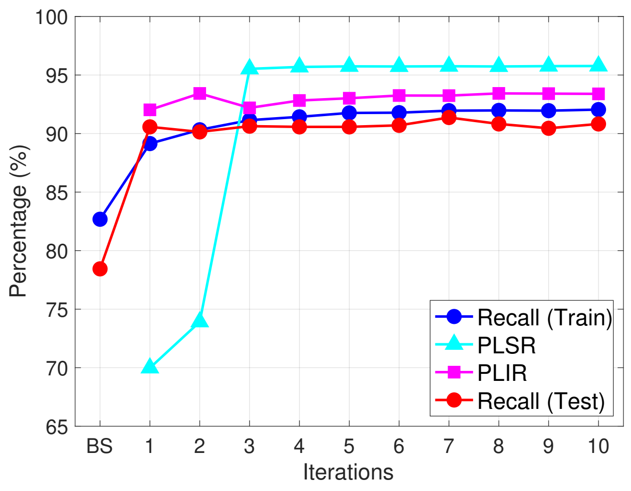


Figure 4.4: Dynamics of SGP on 3DMatch [260]. PLSR: *Pseudo-Label Survival Rate*. PLIR: *Pseudo-Label Inlier Rate*. BS: Boostrap.

Setup. To demonstrate SGP for Example 2, we conducted experiments on 3DMatch [260], a benchmark containing point clouds of real-world indoor scenes. We used RANSAC10K (with 7cm inlier threshold) plus ICP [22] as the teacher, FCGF [50] as the student, and FPFH [198] as the bootstrap descriptor to initialize transformation labels.

SGP was trained on the training set provided by DGR [51] containing $9,856$ pairs of scans, *without* ground-truth transformation labels. Input point clouds were all

[6]We suspect the RANSAC in [232] is not carefully tuned.

voxelized with 5cm resolution before feature extraction (both FPFH and FCGF) and registration. To train FCGF, we followed the configuration of the original FCGF and used SGD with initial learning rate 0.1.[8] In the teacher-student loop, we used finetune, where we train FCGF for 100 epochs at iteration 1 and 50 epochs for the rest of the iterations. We designed a verifier based on estimated overlap ratio, *i.e.*, only pairs with estimated overlap ratio over η are passed to FCGF. We set $\eta = 30\%$ for the first two iterations and $\eta = 10\%$ for the rest. SGP is trained for $T = 10$ iterations.

We name the FCGF descriptor learned from SGP without ground-truth supervision as S-FCGF. We evaluated the performance of S-FCGF on (i) the 3DMatch test set including $1,623$ pairs; and (ii) the unseen Stanford RGBD dataset [46] for multi-way registration [272].

Results. Fig. 4.4 plots the dynamics of SGP on 3DMatch. We observe that: (i) PLSR increases and approaches 96%, indicating that more pairs enter the noisy student training; (ii) PLIR remains close to 93%, and is always higher than the recall, showing the effect of the verifier; (iii) S-FCGF gradually improves itself on both training and test sets.

Table 4.2 compares the performance of S-FCGF^{T}and S-FCGF* to other SOTA methods.[9] We see that S-FCGF* outperforms the baseline FPFH, FCGF [50], and the recently proposed DGR (even with RANSAC80K) [51]. We also provide results using RANSAC10K plus the *supervised oracle*, FCGF$^\circ$, that is trained using full ground-truth supervision. S-FCGF* outperforms the supervised oracle, while S-FCGF^T achieves similar performance.

Fig. 4.5 shows qualitative results using S-FCGF for pairwise registration on 3DMatch and for multi-way registration on Stanford RGBD. More qualitative results are shown in the Appendix.

4.5.3 Ablation Study

We first study the effect of using retrain vs finetune in SGP for point cloud registration. We used the same setup as in Section 4.5.2, except that we changed from finetune to retrain, where in each iteration, we initialized the weights of FCGF at random

[8]`https://github.com/chrischoy/FCGF`

[9]Following [51], we say a registration is successful if rotation error is below 15° and translation error is below 30cm.

and trained it for 100 epochs. We also set the verifier overlap ratio $\eta = 10\%$ for all iterations. Fig. 4.6(a) plots the corresponding dynamics, which overall looks similar to Fig. 4.4. The finetune train recall is slightly higher and more stable than the retrain train recall, due to the "continuous" weight update nature of finetune. SGP with retrain achieves similar performance on the test set: S-FCGF* has overall recall 91.2% and S-FCGF^T has overall recall 90.9%.

We then study the effect of the verifier by running SGP on 3DMatch without verification, *i.e.*, we set $\eta = 0$. As shown in Fig. 4.6(b), PLSR is always 100%. Despite higher noise in the pseudo-labels, the performance of SGP remains unaffected on the test set: S-FCGF* has overall recall 91.4% and S-FCGF^T has overall recall 90.6%.

In the Appendix, we provide two more ablation studies on 3DMatch: (i) we trained SGP on the small test set and tested S-FCGF on the large training set, to show better generalization of a large training set; (ii) we replaced RANSAC10K with a non-robust registration solver as the teacher to show the importance of a robust solver.

4.6 Conclusions

We proposed SGP, the first general framework for feature learning in geometric perception without any supervision from ground-truth geometric labels. SGP iteratively performs robust estimation of the geometric models to generate pseudo-labels, and feature learning under the supervision of the noisy pseudo-labels. We applied SGP to camera pose estimation and point cloud registration, demonstrating performance that is on par or even superior to supervised oracles in large-scale real datasets.

Future research includes (i) increasing the training recall towards 100%; (ii) differentiating the robust estimation layer [89]; (iii) designing an optimality-based [244] and learnable verifier based on *cycle consistency* [87, 110, 153]; (iv) speeding up the teacher-student loop; (iv) forming image and point cloud pairs using *image retrieval* [55, 225].

4.7 Appendix

4.7.1 Application of SGP on Object Detection and Pose Estimation

Example 4 (Object Detection and Pose Estimation)**.** *Given a collection of 3D models $\{\boldsymbol{a}_i\}_{i=1}^{O}$, where each model $\boldsymbol{a}_i \in \mathbb{R}^{3\times N_{a_i}}$ consists of a set of 3D keypoints, let $\boldsymbol{a} \in \mathbb{R}^{3\times N_a}$, $N_a = \sum_{i=1}^{O} N_{a_i}$, be the concatenation of all 3D keypoints. In addition, given a corpus of 2D images $\{\boldsymbol{b}_i\}_{i=1}^{M}$, where each $\boldsymbol{b}_i$ is an RGB image that contains the (partial, occluded) projections of the 3D models plus some background. Object detection and pose estimation seeks to jointly learn a keypoint prediction function $\mathcal{C}$ and estimate the poses of the 3D models $\boldsymbol{x}_i = \{(\boldsymbol{R}_{i,j}, \boldsymbol{t}_{i,j})\}_{j\in\mathcal{S}\subset[O]} \in (\mathrm{SO}(3)\times\mathbb{R}^3)^{|\mathcal{S}|}$, where $\mathcal{S} \subset [O]$ is the subset of 3D models observed by the i-th 2D image ($|\mathcal{S}|$ denotes the cardinality of the set). In particular, following [259], let $\mathcal{C}$ be a combination of UVW mapping and semantic ID masking, i.e., for each pixel in $\boldsymbol{b}_i$, $\mathcal{C}$ predicts which 3D model it belongs to (from 1 to O, and 0 for background), and what is the corresponding 3D coordinates in the specific model, thus deciding which point in $\boldsymbol{a}$ is the corresponding 3D point.*[10]

SGP **for Example 4**. The teacher performs robust absolute pose estimation, *a.k.a. perspective-n-point* (PnP) [103, 129]. A good candidate for the teacher is RANSAC and its variants (*e.g.*, using P3P [78]). The student trains a 2D keypoint predictor under the supervision of camera poses. Recent works such as YOLO6D [222], PVNet [183], and DPOD [259] can all serve as the student network, despite using different methodologies. As for the verifier, similar to Example 1, it can be designed based on the estimated inlier rate by RANSAC. Alternatively, one can project the 3D models onto the 2D image using the estimated absolute poses and compute the overlap ratio (in terms of pixels) between the 2D projection and the estimated semantic ID mask. To initialize SGP, we can train a bootstrap predictor using synthetic datasets, *i.e.*, by rendering synthetic projections of the 3D models under different simulated poses, which is common in [41, 183, 222, 259].

[10]There are many different ways to establish 2D-3D correspondences, see PVNet [183], YOLO6D [222] and references therein.

4.7.2 Detailed Experimental Data

Relative Pose Estimation

In Section 4.5.1, Fig. 4.2 plots the rotation statistics for running SGP on the MegaDepth [143] dataset for relative pose estimation. Here in Fig. 4.7(a), we plot the translation statistics. In addition, the full statistics of SGP are tabulated in Table 4.3. Fig. 4.8 visualizes 9 qualitative examples of relative pose estimation using S-CAPS^T on the MegaDepth test set.

Point Cloud Registration

In Section 4.5.2, Fig. 4.4 plots the dynamics of runing SGP on the 3DMatch [260] dataset. Here we provide the full statistics in Table 4.4.

For qualitative results, in Fig. 4.9 we showcase multiway registration results on various RGB-D datasets [46, 47, 180, 213] in addition to Fig. 4.5. With S-FCGF, rich loop closures can be detected (in green lines), ensuring high-fidelity camera poses for dense reconstruction. It is worth noting that global registration with trained S-FCGF+RANSAC10K, unlike DGR, can easily run in parallel on a single graphics card due to its inexpensive memory cost. This results in at least $4\times$ speedup comparing to DGR in practice when multi-thread loop closure detection is enabled [272].

Ablation Study

In Section 4.5.3, Fig. 4.6 plots the dynamics of running SGP on 3DMatch with two different algorithmic settings: (a) set retrain =True and use retrain instead of finetune; (b) set verifyLabel = False and turn off the verifier. Here we provide the full statistics for (a) and (b) in Table 4.5 and Table 4.6, respectively.

Additionally, we show results for two extra ablation experiments on the 3DMatch dataset for point cloud registration.

Exchange the training and test sets. Because SGP requires no ground-truth pose labels, there is no fundamental difference between the training and test set, except that the training set ($9,856$ pairs) is much larger than the test set ($1,623$ pairs). Therefore, we ask the question: *Can* SGP *learn an equally good feature representation from the much smaller test set?* Our answer is: *it depends on the purpose*. We performed an experiment where we trained SGP on the test set, and tested the learned S-FCGF representation on the much larger training set. For SGP we used retrain = False and verifyLabel = False. Fig. 4.7(b) plots the dynamics and Table 4.7 provides the full statistics. Two observations can be made: (i) Exchanging the training and test set has almost no effect on the recall of S-FCGF on the test set (*cf.* Table 4.7 vs Table 4.4-4.6). This means that, if one only cares about the performance of the learned representation on the test set, then running SGP directly on the target test set is sufficient. (ii) Although exchanging the training and test set does not hurt the recall on the test set, it indeed decreases the recall on the training set by more than 10%. This suggests that a small training set has the shortcoming of overfitting and the learned representation fails to generalize to a larger dataset. Therefore, if one cares generalization of the learned representation, then a larger training set is still preferred. Nevertheless, this ablation study demonstrates the power of the alternating minimization nature of SGP, that is, SGP is able to find a sufficiently good local minimum.

Use a non-robust solver as the teacher. All the experiments so far showed successes of the teacher-student loop, and the robustness of the SGP algorithm to imperfections of both the student and the teacher (noisy geometric pseudo-labels). However, we ask another question: *Can we, intentionally, make* SGP *fail?* Our answer is: yes if we try badly. We performed an experiment running SGP on 3DMatch, this time replacing RANSAC10K with the non-robust Horn's method [107]. We remark that Horn's method is a subroutine of RANSAC and in practice nobody would use Horn's method alone in the presence of outlier correspondences. Nevertheless, for the purpose of ablation study, we adopted this pessimistic choice. Again, for SGP we used retrain = False, verifyLabel = True with a constant overlap ratio threshold $\eta = 10\%$. Fig. 4.7(c) shows the dynamics. We see that the PLIR is always below 20%, meaning that 8 out of 10 geometric labels passed to FCGF training are wrong. In this case, the learned S-FCGF representation keeps getting worse, as shown by the decreasing recalls on both the training and test set. Note that for testing, we actually used RANSAC10K as the registration solver to be consistent with other experiments we performed on 3DMatch. However, even with RANSAC10K, the test recall drops to below 30%. Therefore, this ablation study shows the necessity of a robust teacher for SGP to work. Fortunately, we have plenty of robust solvers, as discussed in the main text. So we think this is a strength of SGP, rather than a weakness.

Algorithm 2: SGP

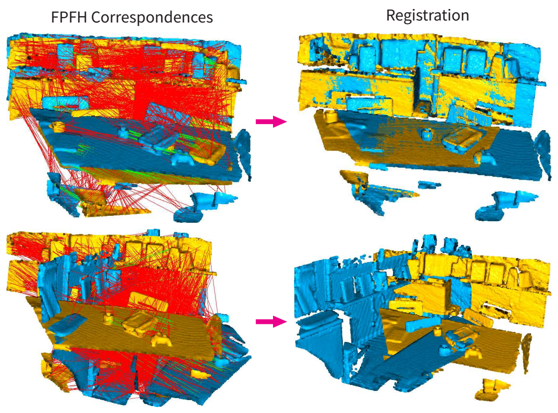
```
Input: A corpus of visual measurements: $\{\boldsymbol{a}_i, \boldsymbol{b}_i\}_{i=1}^M$; a preprocessing
  module: $\phi$; an initial correspondence matching method: $\mathcal{B}$; an architecture
  for a learned correspondence prediction function: $\mathcal{C}$, with initial weights:
  $\boldsymbol{\theta}^{(0)}$ (default: randParam); Number of iterations: $T$; boolean: verifyLabel
  (default True); boolean: retrain (default False);
Output: final weights of $\mathcal{C}$: $\hat{\boldsymbol{\theta}}$; estimated geometric models: $\{\hat{\boldsymbol{x}}_i\}_{i=1}^M$;
% Compute keypoint locations
$\boldsymbol{p}_i^a = \phi(\boldsymbol{a}_i)$, $\boldsymbol{p}_i^b = \phi(\boldsymbol{b}_i)$, $\forall i \in [M]$;
% Bootstrap (Initialize pseudo-labels)
$\boldsymbol{x}_i^{(0)} = \mathsf{teach}(\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{p}_i^a, \boldsymbol{p}_i^b, \mathcal{B}), \ \forall i \in [M]$;
% Alternating minimization
for $\tau = 1 : T$ do
    if verifyLabel = True then
        % Verify correctness of labels
        $\mathcal{S} = \mathsf{verify}(\{\boldsymbol{x}_i^{(\tau-1)}, \boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{p}_i^a, \boldsymbol{p}_i^b\}_{i=1}^M)$;
    else
        $\mathcal{S} = [M]$;
    end
    % Feature learning (problem (4.7))
    if retrain = True then
        $\boldsymbol{\theta} = \mathsf{randParam}$; % retrain
    else
        $\boldsymbol{\theta} = \boldsymbol{\theta}^{(\tau-1)}$; % finetune
    end
    $\boldsymbol{\theta}^{(\tau)} = \mathsf{learn}(\{\boldsymbol{x}_i^{(\tau-1)}, \boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{p}_i^a, \boldsymbol{p}_i^b\}_{i\in\mathcal{S}}, \boldsymbol{\theta})$;
    % Robust estimation (problem (4.6))
    $\boldsymbol{x}_i^{(\tau)} = \mathsf{teach}(\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{p}_i^a, \boldsymbol{p}_i^b, \mathcal{C}_{\boldsymbol{\theta}^{(\tau)}}), \ \forall i \in [M]$;
end
return: $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(T)}$, $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i^{(T)}, i = 1, \ldots, M$.
```

	Easy (%)		*Moderate* (%)		*Hard* (%)	
Methods	Rotation	Translation	Rotation	Translation	Rotation	Translation
SIFT+RANSAC10K [147]	80.9	48.8	58.1	43.5	40.4	34.0
SIFT+Wang-CAPS [232]	70.0	30.5	50.2	24.8	36.8	16.1
SuperPoint+Wang-CAPS [232]	72.9	30.5	53.5	27.9	38.1	19.2
SIFT+CAPS$^{\circ}$+RANSAC10K	**87.1**	52.7	**72.5**	**53.8**	**52.7**	45.6
SIFT+S-CAPS^{T}+RANSAC10K	86.3	53.1	69.2	50.3	51.3	**47.1**
SIFT+S-CAPS*+RANSAC10K	**87.1**	**53.5**	70.4	53.3	51.8	**47.1**

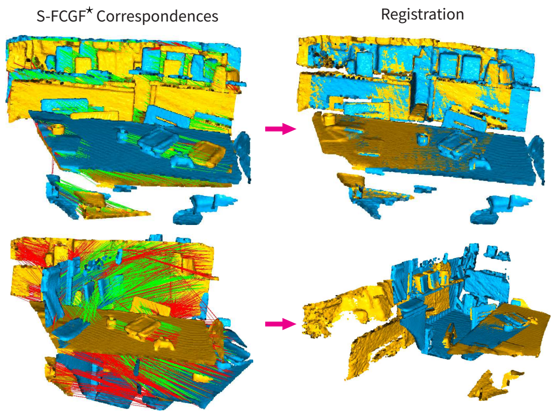
Table 4.1: Rotation and translation recalls on the MegaDepth [143] test dataset using different methods. S-CAPS^{T}: last CAPS trained by SGP. S-CAPS*: best CAPS trained by SGP. SIFT and RANSAC implemented in OpenCV [27]. SIFT uses 0.75 ratio test. All RANSAC use 99.9% confidence. Row 2-3, recall statistics are adapted from the original CAPS paper [232]. Row 4: recall is computed by using RANSAC10K with the pretrained CAPS$^{\circ}$(*i.e.*, the supervised oracle).

Methods	Kitchen (%)	Home 1 (%)	Home 2 (%)	Hotel 1 (%)	Hotel 2 (%)	Hotel 3 (%)	Study (%)	MIT (%)	Overall (%)
FPFH+RANSAC10K [198]	80.6	84.6	69.2	88.1	76.9	**88.9**	71.2	70.1	78.4
FCGF [50][7]	93.0	91.0	71.0	91.0	87.0	69.0	75.0	80.0	82.0
DGR [51]	94.5	89.7	77.9	92.9	85.6	79.6	69.9	72.7	85.2
DGR+RANSAC80K [51]	**98.8**	96.2	**81.7**	97.3	91.2	87.0	81.9	79.2	91.3
FCGF$^{\circ}$+RANSAC10K	97.2	**97.4**	77.9	97.8	**91.3**	83.3	86.3	76.6	91.1
S-FCGF^{T}+RANSAC10K	98.4	94.2	75.0	**98.7**	89.4	79.6	87.3	76.6	90.8
S-FCGF*+RANSAC10K	98.0	94.2	76.0	**98.7**	90.4	85.2	**88.0**	**80.5**	**91.4**

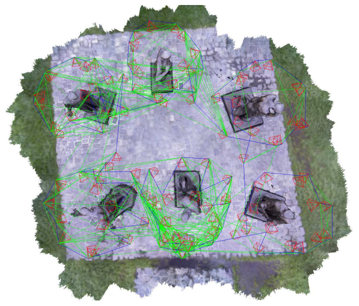
Table 4.2: Scene-wise and overall recalls on the 3DMatch [260] test dataset using different methods. S-FCGF^{T}: last FCGF trained by SGP. S-FCGF*: best FCGF trained by SGP. FPFH is implemented in Open3D [272]. All RANSAC use 99.9% confidence. Row 5: recall is computed by using RANSAC10K with the pretrained FCGF$^{\circ}$(*i.e.*, the supervised oracle).



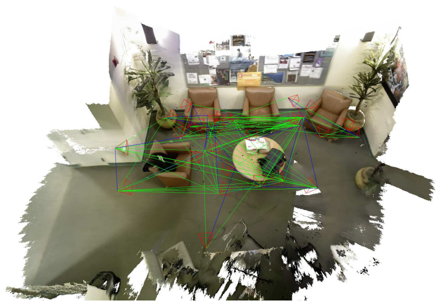
(a) Success (top) and failure (bottom) by FPFH.



(b) Successes by S-FCGF*.

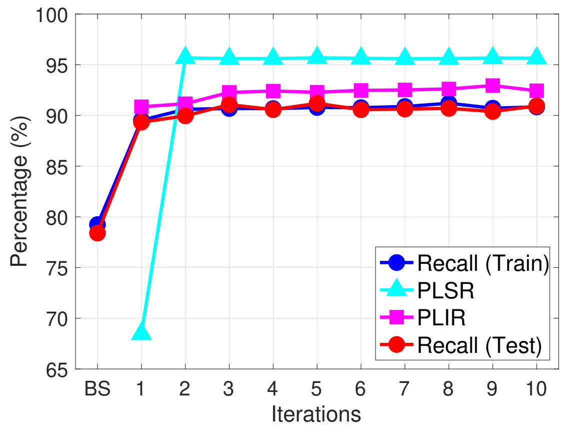


(c) Burghers with S-FCGF*.

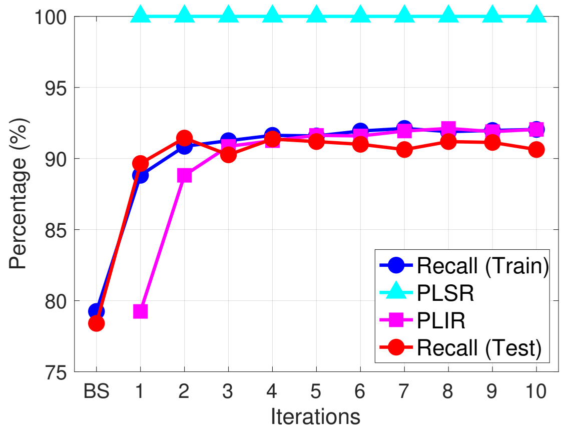


(d) Lounge with S-FCGF*.

Figure 4.5: Qualitative results showing the improved performance of (b) S-FCGF* over the bootstrap descriptor (a) FPFH for pairwise registration on 3DMatch [260], and (c)-(d) cross-dataset generalization of S-FCGF* for multi-way registration on the Stanford RGBD dataset [46]. In (a)-(b), the top pair has overlap ratio 89%, the bottom pair has overlap ratio 50%. Green lines: inlier correspondences. Red lines: outlier correspondences. In (c)-(d), Blue lines: odometry. Green lines: loop closures. [Best viewed digitally]

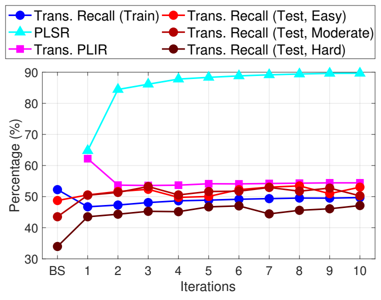


(a) retrain = True.

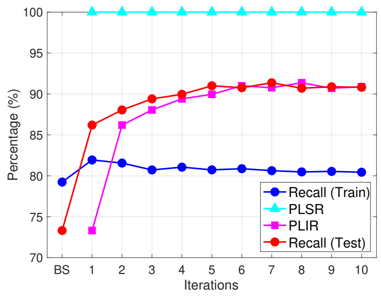


(b) verifyLabel = False.

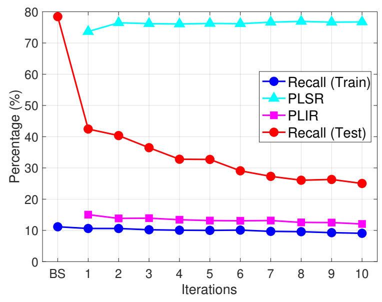
Figure 4.6: Dynamics of SGP on 3DMatch [260] with (a) retrain instead of finetune (line 17); (b) the verify (line 13) turned off. SGP still achieves over 91% overall recall on the test set.



(a) SGP on MegaDepth.



(b) SGP on 3DMatch.

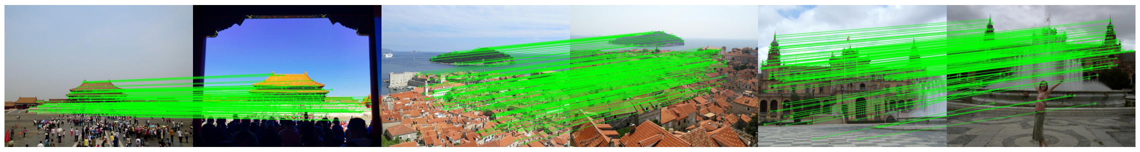


(c) SGP on 3DMatch with Horn.

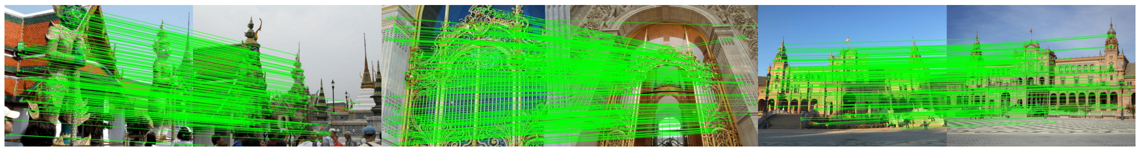
Figure 4.7: Supplementary statistics. (a) The translation statistics for using SGP on MegaDepth [143] (rotation statistics shown in Fig. 4.2 in the main text). (b) Dynamics of SGP on 3DMatch [260] with training and test sets exchanged, *i.e.*, we train SGP on the smaller test set ($1,623$ pairs), but test S-FCGF on the larger training set ($9,856$ pairs). (c) Dynamics of SGP on 3DMatch by replacing the original RANSAC10K teacher with a non-robust Horn's method [107] as the teacher.

		SIFT	SGP trained CAPS (S-CAPS)									
	Statistics (%)	Bootstrap	1	2	3	4	5	6	7	8	9	10
Train	PLSR	$**$	64.79	84.44	86.14	87.78	88.34	88.80	89.15	89.41	89.62	89.64
	Rot. PLIR	$**$	92.50	88.83	88.35	88.41	88.25	88.52	88.50	88.57	88.43	88.48
	Rot. Recall	87.75	79.33	79.69	80.62	80.70	81.20	81.34	81.57	81.57	81.56	81.68
	Trans. PLIR	$**$	62.20	53.70	53.58	53.68	54.13	54.09	54.22	54.33	54.43	54.43
	Trans. Recall	52.25	46.74	47.30	48.09	48.66	48.87	49.16	49.36	49.54	49.52	49.70
Test Recall	Rot., *Easy*	80.88	85.39	85.49	84.68	85.69	85.79	85.79	86.29	**87.09**	85.49	86.29
	Rot., *Moderate*	58.06	**70.37**	68.27	**70.37**	69.77	69.67	69.27	69.87	68.87	70.07	69.17
	Rot., *Hard*	40.35	48.36	49.38	50.31	49.59	50.10	**51.75**	50.10	50.72	51.23	51.33
	Trans., *Easy*	48.75	50.55	51.65	52.35	49.75	50.05	52.25	53.05	**53.45**	50.95	53.05
	Trans., *Moderate*	43.54	50.45	51.35	**53.25**	50.55	51.65	51.75	52.95	51.75	52.75	50.25
	Trans., *Hard*	33.98	43.53	44.35	45.28	45.17	46.71	47.02	44.46	45.60	46.10	**47.13**

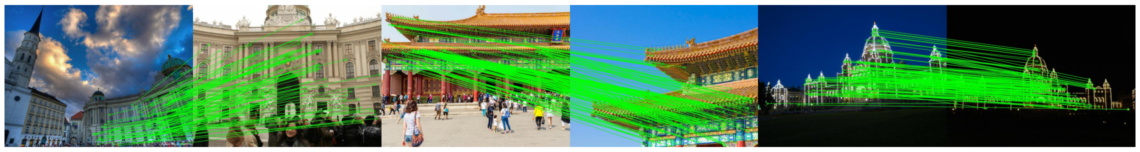
Table 4.3: Train and test statistics of running SGP on MegaDepth [143]. SGP setting: retrain = False, verifyLabel = True, verifier criteria: number of matches larger than 100 and RANSAC estimated inlier rate larger than 10%. Rotation statistics plotted in Fig. 4.2 in the main text. Translation statistics plotted in Fig. 4.7(a).



(a) *Easy*



(b) *Moderate*



(c) *Hard*

Figure 4.8: Supplementary qualitative results for relative pose estimation on the MegaDepth dataset [143] using S-CAPS^T.

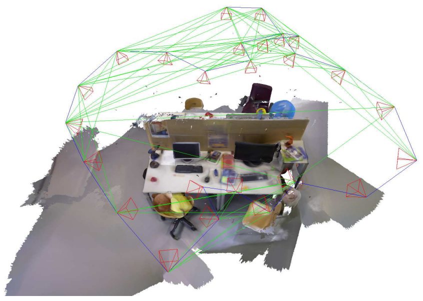
		FPFH	SGP trained FCGF (S-FCGF)									
	Statistics (%)	Bootstrap	1	2	3	4	5	6	7	8	9	10
Train	PLSR	$**$	69.98	73.91	95.53	95.69	95.74	95.73	95.75	95.73	95.76	95.77
	PLIR	$**$	92.03	93.42	92.19	92.82	93.02	93.25	93.24	93.43	93.41	93.39
	Recall	82.68	89.14	90.92	91.14	91.43	91.76	91.78	91.95	91.97	91.95	92.05
Test Recall	Kitchen	80.63	98.42	98.02	98.22	98.02	98.22	98.42	98.02	97.83	98.62	98.42
	Home 1	84.62	92.31	93.59	91.03	93.59	92.95	94.23	94.23	94.23	94.23	94.23
	Home 2	69.23	77.88	74.04	75.48	75.00	75.96	73.08	75.96	76.92	73.08	75.00
	Hotel 1	88.05	96.90	97.35	98.23	97.79	98.23	99.12	98.67	98.67	98.23	98.67
	Hotel 2	76.92	87.50	85.58	86.54	90.38	89.42	90.38	90.38	89.42	89.42	89.42
	Hotel 3	88.89	85.19	83.33	83.33	79.63	81.48	79.63	85.19	79.63	77.78	79.63
	Study	71.23	85.27	86.30	87.67	86.99	85.96	86.99	88.01	86.99	86.30	87.33
	MIT	70.13	79.22	79.22	80.52	77.92	77.92	77.92	80.52	76.62	79.22	76.62
	Overall	78.44	90.57	90.14	90.63	90.57	90.57	90.70	**91.37**	90.82	90.45	90.82

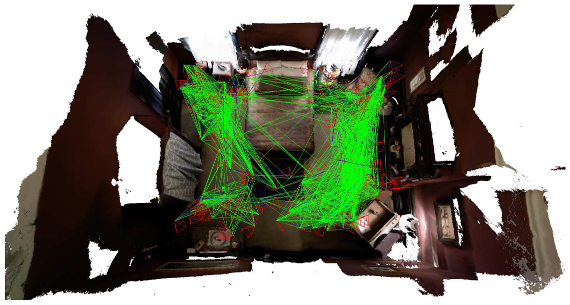
Table 4.4: Train and test statistics of running SGP on 3DMatch [260]. SGP setting: retrain = False, verifyLabel = True, verifier overlap ratio threshold η: $\eta = 30\%$ for iterations $\tau = 1, 2$, $\eta = 10\%$ for iterations $\tau = 3, \ldots, 10$. Statistics plotted in Fig. 4.4 in the main text.

		FPFH	SGP trained FCGF (S-FCGF)									
	Statistics (%)	Bootstrap	1	2	3	4	5	6	7	8	9	10
Train	PLSR	$**$	68.48	95.68	95.61	95.61	95.69	95.64	95.60	95.61	95.67	95.65
	PLIR	$**$	90.86	91.16	92.27	92.40	92.29	92.47	92.52	92.61	92.95	92.44
	Recall	79.24	89.53	90.60	90.69	90.68	90.79	90.77	90.88	91.20	90.72	90.84
Test Recall	Kitchen	$**$	97.23	97.63	98.22	97.83	98.42	97.83	97.83	97.23	98.42	98.22
	Home 1	$**$	91.67	93.59	94.23	95.51	94.87	93.59	95.51	95.51	91.03	93.59
	Home 2	$**$	73.56	71.63	76.92	73.56	75.00	74.04	72.60	76.44	75.00	75.00
	Hotel 1	$**$	96.90	96.90	96.90	96.46	96.90	96.46	96.90	98.23	97.35	96.90
	Hotel 2	$**$	85.58	89.42	92.31	88.46	87.50	90.38	88.46	88.46	86.54	91.35
	Hotel 3	$**$	85.19	88.89	83.33	81.48	83.33	83.33	83.33	85.19	85.19	83.33
	Study	$**$	82.88	84.59	86.64	88.36	88.70	87.67	87.67	86.30	87.33	86.64
	MIT	$**$	85.71	83.12	79.22	79.22	83.12	80.52	83.12	77.92	77.92	84.42
	Overall	$**$	89.34	89.96	91.07	90.57	**91.19**	90.57	90.63	90.70	90.39	90.94

Table 4.5: Train and test statistics of running SGP on 3DMatch [260]. SGP setting: retrain = True, verifyLabel = True, verifier overlap ratio threshold η: $\eta = 10\%$ for all iterations $\tau = 1, \ldots, 10$. Statistics plotted in Fig. 4.6(a) in the main text.

(a) *copyroom* from Stanford RGBD [46].


(b) *long_office* from TUM RGBD [213].


(c) *bedroom* from Indoor LIDAR RGBD [180].

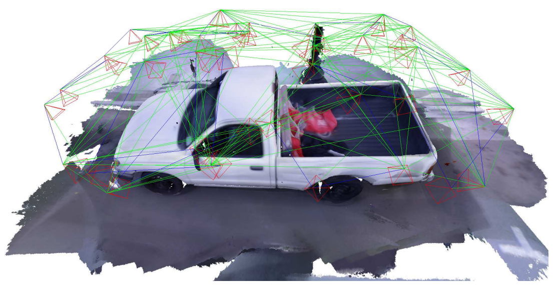

(d) *truck* from Redwood Objects [47].

Figure 4.9: Supplementary qualitative results for 3D registration. Multi-way reconstruction using S-FCGF+RANSAC10K as the global registration method succeeds on various unseen RGB-D datasets. Blue lines: odometry. Green lines: loop closures.

		FPFH	SGP trained FCGF (S-FCGF)									
	Statistics (%)	Bootstrap	1	2	3	4	5	6	7	8	9	10
Train	PLSR	∗∗	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	PLIR	∗∗	79.24	88.82	90.86	91.25	91.63	91.59	91.93	92.12	91.89	91.97
	Recall	79.24	88.82	90.86	91.25	91.63	91.59	91.93	92.12	91.89	91.97	92.05
Test Recall	Kitchen	∗∗	97.43	98.22	98.62	97.83	98.62	98.81	98.22	98.62	98.22	98.22
	Home 1	∗∗	92.31	94.23	91.67	94.23	94.23	92.95	93.59	93.59	94.87	92.95
	Home 2	∗∗	74.04	75.00	72.12	77.40	74.04	74.04	73.56	74.04	73.56	73.08
	Hotel 1	∗∗	95.58	98.23	97.35	97.79	99.12	98.23	98.67	96.90	97.79	97.35
	Hotel 2	∗∗	90.38	93.27	88.46	90.38	88.46	87.50	86.54	88.46	88.46	89.42
	Hotel 3	∗∗	88.89	85.19	83.33	87.04	85.19	85.19	81.48	85.19	83.33	81.48
	Study	∗∗	84.59	87.33	87.67	86.64	88.01	88.01	87.67	88.70	88.70	87.67
	MIT	∗∗	76.62	83.12	77.92	84.42	79.22	80.52	80.52	84.42	83.12	83.12
	Overall	∗∗	89.65	**91.44**	90.26	91.37	91.19	91.00	90.63	91.19	91.13	90.63

Table 4.6: Train and test statistics of running SGP on 3DMatch [260]. SGP setting: retrain = False, verifyLabel = False. Statistics plotted in Fig. 4.6(b) in the main text.

	Statistics (%)	FPFH Bootstrap	SGP trained FCGF (S-FCGF) 1	2	3	4	5	6	7	8	9	10
Train	PLSR	$**$	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	PLIR	$**$	73.32	86.20	88.05	89.40	89.96	91.00	90.76	91.37	90.70	90.88
	Recall	73.32	86.20	88.05	89.40	89.96	91.00	90.76	**91.37**	90.70	90.88	90.82
	Kitchen	$**$	94.66	96.84	98.42	98.81	99.21	99.60	99.41	99.01	99.21	99.21
	Home 1	$**$	91.03	89.74	93.59	95.51	95.51	94.87	94.87	95.51	96.15	95.51
	Home 2	$**$	70.67	70.67	71.63	69.23	71.15	70.19	74.04	72.60	72.12	72.60
	Hotel 1	$**$	94.69	96.02	97.35	98.67	98.67	99.12	99.12	99.12	99.12	99.12
	Hotel 2	$**$	77.88	79.81	77.88	80.77	84.62	83.65	86.54	83.65	84.62	83.65
	Hotel 3	$**$	83.33	85.19	81.48	85.19	88.89	87.04	85.19	83.33	84.19	85.19
	Study	$**$	79.79	84.93	87.33	86.64	88.36	87.33	87.67	87.33	87.67	86.99
	MIT	$**$	75.32	75.32	75.32	79.22	79.22	80.52	80.52	77.92	76.62	79.22
Test on train set		79.24	**81.94**	81.56	80.72	81.06	80.73	80.87	80.63	80.48	80.54	80.44

Table 4.7: Train and test statistics of running SGP on 3DMatch [260] with **training and test sets exchanged**, *i.e.*, we train SGP on the smaller test set ($1,623$ pairs), but test S-FCGF on the larger training set ($9,856$ pairs). SGP setting: retrain = False, verifyLabel = False. Statistics plotted in Fig. 4.7(b). We see SGP demonstrates overfitting while training on the smaller test set: S-FCGF achieves equally good (91.37%) recall on the test set, but only achieves below 82% recall on the training set (while in Tables 4.4-4.6 S-FCGF has over 92% recall on the training set). Statistics plotted in Fig. 4.7(b).

Part III

Sparsity in Scene Representation

Chapter 5

ASH: A Modern Framework for Parallel Spatial Hashing

5.1 Introduction

3D space is only one dimension higher than the 2D image space, yet the additional dimension introduces an unpredictable multiplier to the computational and storage cost. This increased dimension makes 3D perception tasks such as geometric reconstruction and appearance refinement challenging to implement. To reduce complexity, one compromise is to reuse dense data structures and constrain the 3D space by bounding the region of interest, *e.g.*, adopting a 3D array in a bounded space [168]. While this simple approach succeeds at the scale of objects, it cannot meet the demand of room or city scale perception, which is necessary for virtual tour, telepresence, and autonomous driving.

Since the 3D space is generally a collection of 2D surface manifolds, its sparsity can be exploited by *partitioning* to reduce computational cost. The general idea is to split the large 3D space into smaller regions and only proceed with the non-empty ones. There is a plethora of well-established data structures for 3D space partitioning. Examples include trees (Octree [156], KD-tree [20]) and hash maps (spatial hashing [171]). While trees are able to adaptively achieve high precision, they 1) require an initial bounding volume and 2) usually take unbalanced traversal time for a batch of spatial queries and hence 3) are less friendly to batched operations. On the other hand, spatial hashing coupled with a plain array structure is more scalable and parallelizable for reconstruction tasks.

In classical dense SLAM pipelines [171, 184], spatial hashing is used to map 3D coordinates to internal pointers that are only accessible in GPU device code. A pre-allocated memory pool allows dynamic growing hash maps, but adaptive rehashing is generally unavailable. Recent feature-grid encoders [163] apply spatial hashing to map 3D coordinates to features stored at grid points in a *static* bounded region. While differentiable spatial query is supported, *collisions* are not resolved, limiting its usage to stochastic feature optimization where incorrect key-value mappings are tolerated. A general, user-friendly, collision-free hash map is missing for efficient spatial perception at scale.

The reason for this absence is understandable. A parallel hash map on GPU has to resolve collisions and thread conflicts and preferably organize an optimized memory manager, none of which is trivial to implement. Previous studies have attempted to tackle the problem in one or more aspects, driven by their selected downstream applications. Furthermore, most of the popular parallel GPU hash maps are implemented in C++/CUDA and only expose low-level interfaces. As a result, customized extensions must start from low-level programming. While these designs usually guarantee performance under certain circumstances [4, 8, 171, 184], as of today, they leave a gap from the standpoint of the research community, which prefers to use off-the-shelf libraries for fast prototyping with a high-level scripting language using tensors and automatic differentiation. Our motivation is to bridge this gap to enable researchers to develop sophisticated 3D perception routines with less effort and drive the community towards large-scale 3D perception.

To this end, we design a modern hash map framework with the following major contributions:

1. a user-friendly *dynamic*, *generic*[1], and *collision-free* hash map interface that enables tensor I/O, *advanced indexing*, and *in-place automatic differentiation* when bridged to autodiff engines such as PyTorch;
2. an index-first adaptor that supports various state-of-the-art parallel GPU hash

[1]A dynamic hash map supports insertion and deletion after hash map construction. A generic hash map supports arbitrary dimensional keys and values in various data types.

map backends and accelerates hash map operations with an improved structure-of-array (SoA) data layout;

3. a number of downstream applications that achieve higher performance compared to state-of-the-art implementations with fewer LoC.

Experiments show that ASH achieves better performance with fewer LoC on both synthetic and real-world tasks.

5.2 Related Work

5.2.1 Parallel Hash Map

The hash map is a data structure that seeks to map *sparse keys* (*e.g*. unbounded indices, strings, coordinates) from the set $\mathcal{K}$ to values from the set $\mathcal{V}$ with amortized $O(1)$ access. It has a hash function $h : \mathcal{K} \rightarrow \mathcal{I}_n, k \mapsto h(k)$ that maps the key to the index set $\mathcal{I}_n = \{0, 1, \ldots, n-1\}$ for indexing (or addressing) that is viable on a computer.

Ideally, with a perfect injective hash function h, a hash map can be implemented by $H : \mathcal{K} \rightarrow \mathcal{V}, k \mapsto \mathbf{v}^A\big[h(k)\big]$, where $\mathbf{v}^A$ is an array of objects of type $\mathcal{V}$ and $[\cdot]$ is the trivial array element accessor. However, in practice, it is intractable to find an injective map given a sparse key distribution in $\mathcal{K}$ and a constrained index set $\mathcal{I}_n$ of size n due to the computational budget. Therefore, modifications are required to resolve inevitable *collisions*, where $i = h(k_1) = h(k_2), k_1 \neq k_2$. There are two classes of techniques for collision resolution, *open addressing* and *separate chaining*. Open addressing searches for another candidate $j \neq i, j \in \mathcal{I}_n$ via a *probing* algorithm until an empty address is found. The simplest probing, linear probing [121], computes $j = (h(k) + t) \mod n$ starting from $i = h(k)$, where t is the number of attempts. Separate chaining, on the other hand, maintains multiple entries per mapped index where a linked list is grown at i if $i = h(k_1) = h(k_2), k_1 \neq k_2$.

While hash map implementations are widely available for CPU, their GPU counterparts have only emerged in the recent decade. Most GPU hash maps use open addressing [4, 80, 117, 219], mainly due to simplicity in implementation and capability of handling highly concurrent operations. CUDPP [4] utilizes Cuckoo Hashing [177], while CoherentHash [80] adopts Robin Hood Hashing [37] – both

Table 5.1: Comparison of existing parallel GPU hash maps. ASH preserves the dynamic, generic, and atomic properties, and is extendable to the non-templated high-level Python interfaces.

	Dynamic	Generic	Collision-free	Python
SlabHash [8]	✓	✗	✓	✓
CUDPP [4]	✗	✗	✓	✗
cuDF [219]	✗	✗	✓	✓
WarpCore [117]	✓	✗	✓	✗
stdgpu [210]	✓	✓	✓	✗
InfiniTAM [184]	✓	✗	✓	✗
VoxelHashing [171]	✓	✗	✗	✗
GPURobust [65]	✓	✓	✗	✗
Instant-NGP [163]	✗	✓	✗	✓
ASH	✓	✓	✓	✓

involving advanced probing design. Although being performant when $\mathcal{K}, \mathcal{V}$ are limited to integer sets, these variations cannot be generalized to spatial hashing and only allow static input. Recently, WarpCore [117] proposes to support non-integer $\mathcal{V}$ and dynamic insertion, but the key domain is still limited to at most 64 bits.

There are also a few separate chaining implementations on GPU involving device-side linked lists. SlabHash [8] builds a linked list with a 128-bit *Slab* as the minimal unit, optimized for Single Instruction Multiple Threads (SIMT) warp operations. Although SlabHash allows dynamic insertions, similar to the aforementioned GPU hash maps, only integer $\mathcal{K}, \mathcal{V}$ are supported. stdgpu [210] follows the conventional C++ Standard Library `std::unordered_map` and builds supporting vectors, bitset lock guards, and linked lists from scratch, resulting in a generic, dynamic hash map. With these rich functionalities, however, stdgpu is not optimized for large value sets. In addition, due to its low-level templated design, users have to write device code for simple tasks.

We refer the readers to a comprehensive review of GPU hash maps [138].

5.2.2 Space Partitioning Structures

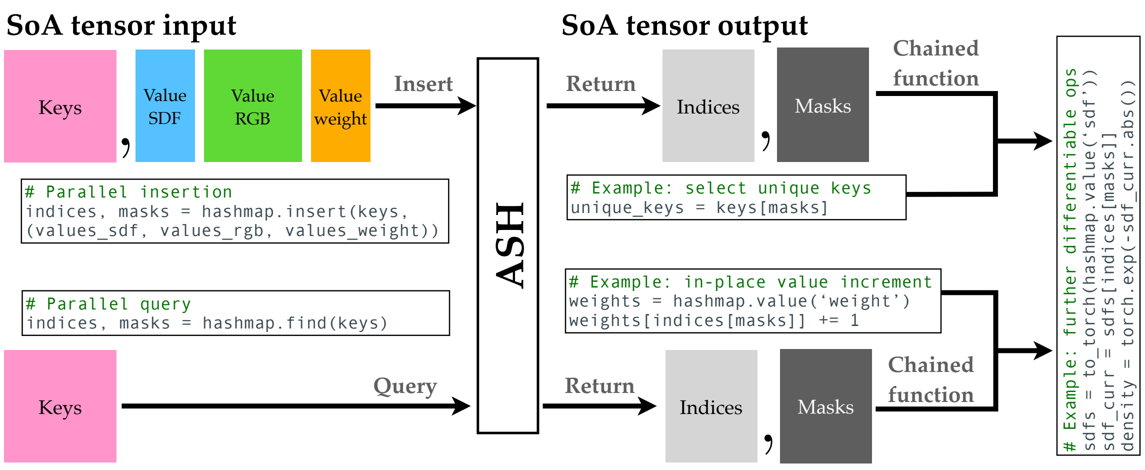
3D data is not as simple to organize as 2D images. While a 2D image can be stored in a dense matrix, exploiting sparsity in 3D data is paramount due to the limits in computer memory of the current day.

The most widely used data structures for 3D indexing are arguably trees. A KD-tree [20] recursively sorts k-dimensional data along a selected axis and partitions data at the median point. By nature, a KD-Tree is designed for neighbor search. In 3D, it is mainly used to organize 3D *points* and their *features*. Examples include normal estimation and nearest neighbor association in Iterative Closest Points (ICP) [195, 272] and 3D feature association in global registration [198, 271]. GPU adaptations exist for KD-trees[196, 268], but are not suitable for incrementally changing scenes, as they are usually constructed once and queried repeatedly.

Bounding volume hierarchy (BVH) is another hierarchical representation that organizes primitives such as *objects* and *triangles* in 3D. There are various GPU adaptations [94, 134, 229] mostly targeted at ray tracing and dynamic collision detection. While a parallel construction is possible and deformation of the nodes is allowed, the tree structure typically remains unchanged, assuming a fixed layout.

While KD-trees and BVH split the space *unevenly* by data distribution, an Octree [156], on the other hand, recursively partitions the 3D space *evenly* into 8 subvolumes according to space occupation states. It has been widely used in adaptive 3D mapping [108, 167] for robot navigation. There have been parallel implementations on GPU, from optimized data structures [105] to domain-specific languages [109]. However, these works generally focus on physics simulation within a bounded region of interest where the spatial partition is predefined. While parallel incremental division [261] is possible, an initial bounding region is still required, and the trees are not guaranteed to be balanced.

Spatial hashing is another variation of spatial management with $O(1)$ access time depending on hash maps. Bundled with small dense 3D arrays, it has been widely used in real-time volumetric scene reconstruction within *unbounded* region of interest. A handful of CPU implementations have achieved real-time performance [99, 128] at the expense of resolution. Similarly, GPU implementations [64, 65, 171, 184] reach high frame rates using GPU-based spatial hashing. However, all of the studies depend on ad hoc GPU hash maps exclusive to these specific systems. Concurrent race conditions have not been fully resolved in several implementations [65, 171], where volumes can be randomly under-allocated. Recent neural feature grids [163] apply spatial hashing in a *bounded* volume to query voxelized feature embeddings. These approaches use simplified hashing designs that are not collision-free, and thus are only compatible with stochastic optimization that tolerates noise from an incorrect query.



```
# Parallel insertion
indices, masks = hashmap.insert(keys,
(values_sdf, values_rgb, values_weight))
```

```
# Parallel query
indices, masks = hashmap.find(keys)
```

```
# Example: select unique keys
unique_keys = keys[masks]
```

```
# Example: in-place value increment
weights = hashmap.value('weight')
weights[indices[masks]] += 1
```

```
# Example: further differentiable ops
sdfs = to_torch(hashmap.value('sdf'))
sdf_curr = sdfs[indices[masks]]
density = torch.exp(-sdf_curr.abs())
```

Figure 5.1: The interface illustration of ASH. Left: ASH takes a key and/or a list of value tensors as input, with a one-liner interface. For *insertion*, tensors are organized in SoA. Right: buffer indices and masks are returned upon insertion and query, organized together in SoA. Chained functions can be easily applied to hashed data by indexing ASH buffers with indices and masks. Examples include selecting unique keys through insertion, and applying in-place value increment through query. Differentiable ops can be applied in downstream applications.

5.2.3 Spatially Varying 3D Representations

A truncated signed distance function (TSDF) [56] is an implicit representation of surfaces, recording point-wise distance to the nearest surface point. It is frequently used for dense scene reconstruction with noisy input. The distribution of surfaces is generally spatially varying and therefore, a proper parameterization is often necessary, either in a discrete [171] or neural [38, 163] form. Non-rigid deformation methods [170, 269] seek to embed point clouds in a deformable grid, where each point is anchored to and deformed by neighbor grids. They are mainly used for animation or non-rigid distortion calibration. Similar to a deformation grid, complex lighting for rendering can be approximated by spatially-varying spherical harmonics (SVSH) [150] placed at a sparse grid. These grids are natural applications of spatial hashing. A comprehensive review of spatially varying representations for real-world scene reconstruction is available [277].

While ad hoc implementations have been introduced for these representations either on CPU or GPU, ASH provides a device-agnostic interface requiring less code written and providing better performance. Table 5.1 compares various aspects of existing GPU hash maps, either as a standalone data structure (Section 5.2.1) or embedded in an application (Section 5.2.2). To the best of our knowledge, ASH is the first implementation that simultaneously supports dynamic insertion, ensures correctness via atomic collision-free operations, allows generic keys, and has a modern tensor interface and Python binding for better usability.

5.3 Overview

Before plunging into the details, we first provide a high-level overview of our framework in Fig. 5.1.

Conventional parallel hash maps reorganize the structure of arrays (SoA) input, *i.e.*, the separated key array and value array, into an array of structures (AoS) where keys and values are *paired*, inserted, and stored. Therefore, array of *pointers to pair structures* (`std::pair` in C++, `thrust::pair` in CUDA, and `tuple` in Python) are returned upon query. Consequently, the operations from insertion and query to in-place value increment require users to write device code and visit AoS at the low-level pointers.

In contrast, ASH sticks to SoA. Fig. 5.1 shows the workflow of ASH. Instead of *pointers to pairs*, ASH returns *indices* and *masks* arrays that can be directly consumed by tensor libraries such as PyTorch [182] (without memory copy) and NumPy [101] (with GPU to host memory copy). As a result, post-processing functions such as duplicate key removal and in-place modification can be chained with insertion and query in ASH via advanced indexing without writing any device code. As a general and device-agnostic interface for parallel hash maps, our framework is built upon switchable backends with details hidden from the user. Currently, *separate chain* backends are supported, including the generic stdgpu [210] backend, and the extended integer-only SlabHash [8] backend for arbitrary key-value data types. TBB's concurrent hash map [132] powers the CPU counterpart with the identical interface to GPU.

In this paper, we use calligraphic letters to represent sets, and normal lower-case letters for their elements. Normal upper-case letters denote functions. Bold lower-case letters denote vectors of elements or arrays in the programmer's perspective. Bold upper-case letters are for matrices. For instance, in a hash map, we are interested in key elements $k \in \mathcal{K}$ and their vectorized processing, *e.g*. query $Q(\mathbf{k})$. Specifically, we use $\mathcal{I}_n$ to denote a set of indices $\{0, 1, \ldots, n-1\}$, and Θ as the boolean selection $\{0, 1\}$. Given an arbitrary vector $\mathbf{x}$, we denote $\mathbf{x}(\mathbf{i})$ and $\mathbf{x}(\boldsymbol{\theta})$ as indexing and selection functions applied to $\mathbf{x}$ when $\forall i \in \mathbf{i}, i \in \mathcal{I}_n$ and $\forall \theta \in \boldsymbol{\theta}, \theta \in \Theta$. We use $\langle \mathcal{K}, \mathcal{V} \rangle$ as the key and value sets for a hash map. $h : \mathcal{K} \to \mathcal{I}_n$ is the internal hash function that converts a key to an index. $H : \mathcal{K} \to \mathcal{V}$ is the general hash map enclosing h.

5.4 The ASH framework

5.4.1 Classical Hashing

In a hash map $\langle \mathcal{K}, \mathcal{V} \rangle$, since the hash function h cannot be perfect as discussed in Section 6.2, we have to store keys to verify if collisions happen ($h(k_1) = h(k_2)$ but $k_1 \neq k_2, k_1, k_2 \in \mathcal{K}$).

In separate chaining, to resolve hash collisions, the bucket-linked list architecture is used. With n initial *buckets*, we construct the hash function $h : \mathcal{K} \to \mathcal{I}_n$ where $\mathcal{I}_n$ is defined in Section 6.2. As shown in Fig. 5.2, keys with the same hashed index $i = h(k) \in \mathcal{I}_n$ are first aggregated in the i-th bucket, where a linked list grows adaptively to accommodate different keys. A conventional hash map stores key-value pairs as the storage units. Consequently, two keys k_1, k_2 can be distinguished by checking $h(k_1) = h(k_2)$ and $k_1 = k_2$ from the pair in order, and manipulation of the keys and values can be achieved by iterating over such pairs.

With this formulation, assuming a subset $\mathcal{X} \subset \mathcal{K}$ has been inserted into the hash map with associated values $\mathcal{Y} \subset \mathcal{V}$, a query function can be described as

$$\begin{aligned} Q_{\mathcal{K},\mathcal{V}} : \mathcal{K} &\to \mathcal{K} \times \mathcal{V} \\ k &\mapsto \langle k, v \rangle, \ \forall k \in \mathcal{X}, \end{aligned} \tag{5.1}$$

where $\langle k, v\rangle$ forms a concrete pair stored in the hash map. This format is common in implementations, *e.g.* in C++ (`std::unordered_map`) and Python (`dict`).

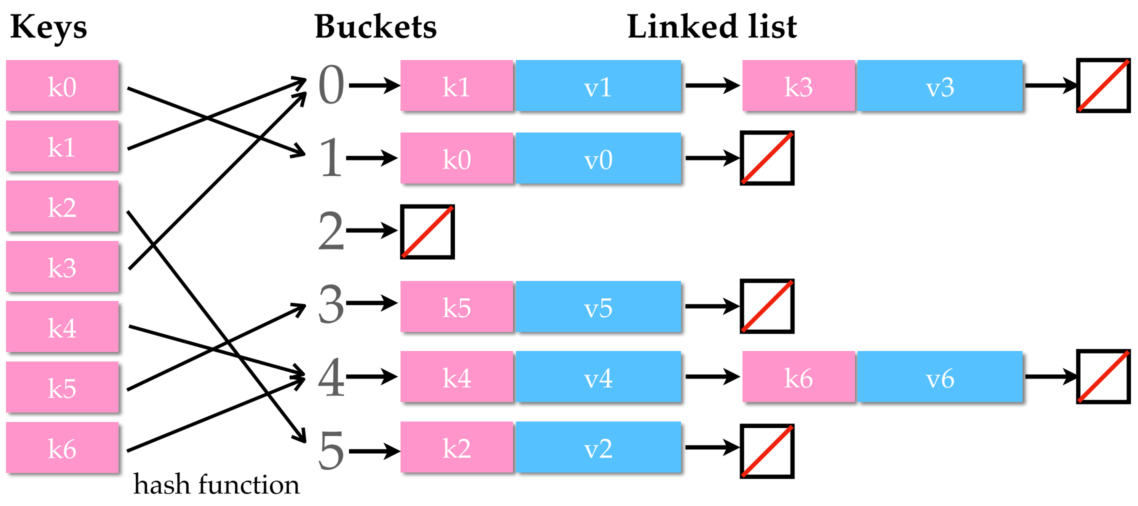


Figure 5.2: Illustration of a classical hash map using separate chaining. Keys (left) are put into corresponding buckets (middle) obtained by the hash function h. A linked list (right) is constructed per bucket to store key-value pairs within the same bucket but with unequal keys.

5.4.2 Function Chaining and Parallel Hashing

The element-wise operation in Eq. 5.1 can be extended to vectors via parallel device kernels. However, interpretation of the returned iterators of pairs is still required at the low level. In other words, although the parallel version can be implemented efficiently, results are still packed in an AoS instead of SoA:

$$Q_{\mathcal{K},\mathcal{V}}(\mathbf{k}) = \mathbf{array}\{\langle k, v\rangle\}. \tag{5.2}$$

This forms a barrier when the parallel query is located in a chain of functions. For instance, to apply any function G (*e.g.*, geometry transformation) over the result of a query, the low-level function `second` that selects the value element from a pair $\langle k, v\rangle$ must be provided to dereference the low-level structures and manipulate the keys and values *in-place*. In other words, we have to implement a non-trivial $\tilde{G}$:

$$\tilde{G}(\mathbf{k}) = (G \circ \texttt{second} \circ Q_{\mathcal{K},\mathcal{V}})(\mathbf{k}), \tag{5.3}$$

to force the conversion from AoS to SoA and chain a high-level function G with $Q_{\mathcal{K},\mathcal{V}}$ [2]. This could be tedious when prototyping geometry perception that requires hash map structures since off-the-shelf operations have to be reimplemented in device code.

We reformulate this problem by introducing two affiliate arrays, $\mathbf{k}^B$ and $\mathbf{v}^B$ (note with a superscript B for buffering, they are not the input $\mathbf{k}, \mathbf{v}$) of capacity $c \geq n$, where n is the number of buckets. These arrays are designed for explicit storage of keys and values, respectively, and serve as buffers to support natural SoA. They are exposed to users for direct access and in-place modification. Now the query function can be rewritten as

$$\begin{aligned} Q_{\mathcal{K},\mathcal{V}} &: \mathcal{K} \to \mathcal{I}_c, k \mapsto i, \\ &s.t.\ \mathbf{k}^B(i) = k,\ \mathbf{v}^B(i) = v,\ \forall \langle k, v\rangle \in \langle \mathcal{X}, \mathcal{Y}\rangle \end{aligned} \tag{5.4}$$

and this version is ready for parallelization. At this stage, to combine G and $Q_{\mathcal{K},\mathcal{V}}$, we can chain $G \circ \mathbf{v}^B \circ Q$ to manipulate values:

$$G(\mathbf{k}) = G\Big(\mathbf{v}^B(Q_{\mathcal{K},\mathcal{V}}(\mathbf{k}))\Big), \tag{5.5}$$

which retains convenient properties such as array vectorization and advanced indexing.

When the input set $\tilde{\mathcal{X}} \not\subset \mathcal{X}$ is not fully stored in the hash map, our formulation

[2]This can be achieved simply by returning a *copy* of values, but it is not feasible, especially when dealing with large-scale data, *e.g.* hierarchical voxel grids.

maintains its effectiveness by a simple masked extension:

$$\begin{aligned} & Q^{\mathcal{I}}_{\mathcal{K},\mathcal{V}} : \mathcal{K} \to \mathcal{I}_c,\ Q^{\Theta}_{\mathcal{K},\mathcal{V}} \to \{0,1\}, \\ & Q^{\mathcal{I}}_{\mathcal{K},\mathcal{V}}(\tilde{k}) = i,\ \mathcal{Q}^{\Theta}_{\mathcal{K},\mathcal{V}} = \theta, \\ & \quad s.t.\ \ \theta = 1;\ \mathbf{k}^B(i) = \tilde{k},\ \mathbf{v}^B(i) = \tilde{v},\ \text{if } \tilde{k} \in \mathcal{X}, \\ & \qquad\quad \theta = 0; i = \text{undefined},\ \text{otherwise}, \end{aligned} \tag{5.6}$$

which is also ready for parallelization. Now the chaining of functions is given by

$$G(\mathbf{k}) = G\Bigg(\mathbf{v}^B\Big(\mathbf{i}(\boldsymbol{\theta})\Big)\Bigg), \tag{5.7}$$

$$\mathbf{i} = Q^{\mathcal{I}}_{\mathcal{K},\mathcal{V}}(\mathbf{k}), \boldsymbol{\theta} = Q^{\Theta}_{\mathcal{K},\mathcal{V}}(\mathbf{k}), \tag{5.8}$$

using advanced indexing with masks. We can also select valid queries with $\mathbf{k}(\boldsymbol{\theta})$ without visiting $\mathbf{k}^B$. While our discussion was about the query function, the same applies to insertion.

In essence, by converting the pair-first AoS to an index-first SoA format with the help of array buffers, we can conveniently chain high-level functions over hash map query and insertion. This simple change enables easy development on hash maps and unleashes their potential for fast prototyping and differentiable computation. However, the layout requires fundamental changes to the hash map data structure. With this in mind, we move on to illustrate how the ASH layer converts the AoS in native backends to our SoA layout.

5.4.3 Generic Backends

We start with converting stdgpu [210], a state-of-the-art generic GPU hash map as the backend of ASH. stdgpu follows the convention of its CPU counterpart `std::unordered_map` by providing a templated interface. The underlying implementation is a classical bucket - linked list structure with locks to avoid race conditions on GPU. To exploit the power of a generic hash map without reinventing the wheel, we seek to reuse the operations over keys (*i.e.* lock-guarded bucket and linked list operations) and redirect the value mapping to our buffer $\mathbf{v}^B$.

A dynamic GPU hash map requires dynamic allocation and freeing of keys and values in device kernels. With pre-allocated key buffer $\mathbf{k}^B$ and value buffer $\mathbf{v}^B$, we maintain an additional *index heap* $\mathbf{h}$, as shown in Fig. 5.3. The index heap stores *buffer indices* i pointing to the buffers $\mathbf{k}^B, \mathbf{v}^B$ as a map $P : \mathcal{I}_c \rightarrow \mathcal{I}_c$, where the *heap top* t maintains the currently available buffer index in $\mathbf{h}[t]$. Heap top starts at $t = 0$, and is atomically increased at allocation and decreased at free. With $\mathbf{h}$ and the dynamically changing t, we instantiate a generic hash map with the templated value in stdgpu to be $\mathcal{V} =$ `Int32`, where the values are *buffer indices* i stored in $\mathbf{h}$ to access $\mathbf{k}^B, \mathbf{v}^B$ exposed to the user.

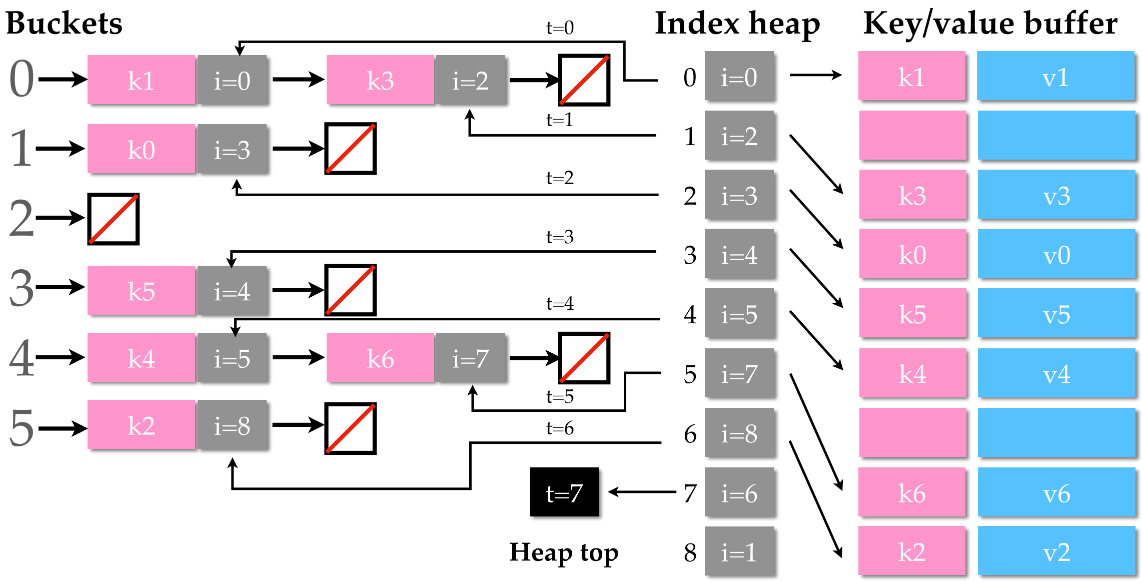


Figure 5.3: Illustration of a generic hash map bridged to tensors in ASH. *Buffer indices* i are dynamically provided by the available indices maintained in the index heap at the increasing heap top t (middle), acting as the *values* in the underlying hash map (left). It connects the hash map and the actual key values stored in the buffer (right) by accessing i. The key-bucket correspondences are the same as Fig. 5.2, omitted for simplicity.

Insertion

The insertion of a $\langle k, v \rangle \in \langle \mathcal{K}, \mathcal{V} \rangle$ pair is now decoupled into two steps, with i) insertion of $\langle k, i \rangle \in \langle \mathcal{K}, \mathcal{I}_c \rangle$ into the hash map, where i is the buffer index dynamically acquired from the heap top $\mathbf{h}[t]$ and ii) insertion of $\mathbf{k}^B(i) := k, \mathbf{v}^B(i) := v$ into buffers.

A naive implementation will acquire a buffer index i from $\mathbf{h}$ on every insertion attempt and free it if the insertion fails because the key already exists. However, when running in parallel, `atomicAdd` and `atomicSub` may be conflicting among threads, leading to race conditions. A *two-pass insertion* could resolve the issue: in the first pass, we allocate a batch of indices from $\mathbf{h}$ determined by the input size, attempt insertions, and record results; in the second pass, we free the indices to $\mathbf{h}$ from failed insertions.

We adopt a more efficient *one-pass lazy* insertion. We first attempt to insert $\langle k, -1\rangle$ with -1 as the dummy index into the backend and observe if it is successful. If not, nothing needs to be done. Otherwise, we capture the returned pointer to the pair, trigger an index i allocation from $\mathbf{h}$, and directly replace the dummy -1 with i. This significantly reduces the overhead when the key uniqueness is low (*i.e.*, many duplicates exist in the keys to be inserted).

Query

The query operation is relatively simpler. We first look up the buffer index $i \in \mathcal{I}_c$ given k in the backend. If it is a success, we end up with $k = \mathbf{k}^B(i)$, and the target $v = \mathbf{v}^B(i)$ is accessible with i by users.

5.4.4 Non-generic Backends

While the generic GPU hash map has only recently been available, the research community in parallel computation has been focusing on more controlled setups where both $\mathcal{K}$ and $\mathcal{V}$ are limited to certain dimensions or data types. We seek to generalize this non-generic setup with our index heap and verify their performance in more real-world applications. In this section, we show how ASH can be used to generalize SlabHash [8], a warp-oriented dynamic GPU hash map that only allows insertions and queries to `Int32` data type.

An extension to generic key types is non-trivial for SlabHash since its warp operations only apply to variables with limited word length. Our implementation extends the hash set variation of SlabHash, where only integers as keys are maintained in the backend.

Generalization via Index Heap

The index heap $\mathbf{h}$ is the core to generalizing the SlabHash backend. In brief, a generic key is represented by its associated buffer index i in an integer-only hash set, allocated the same way as discussed in Section 5.4.3. As illustrated in Fig. 5.4, all the insertions and queries are redirected from the buffer indices to actual keys and values via the index heap. However, the actual implementation involves more complicated changes in design.

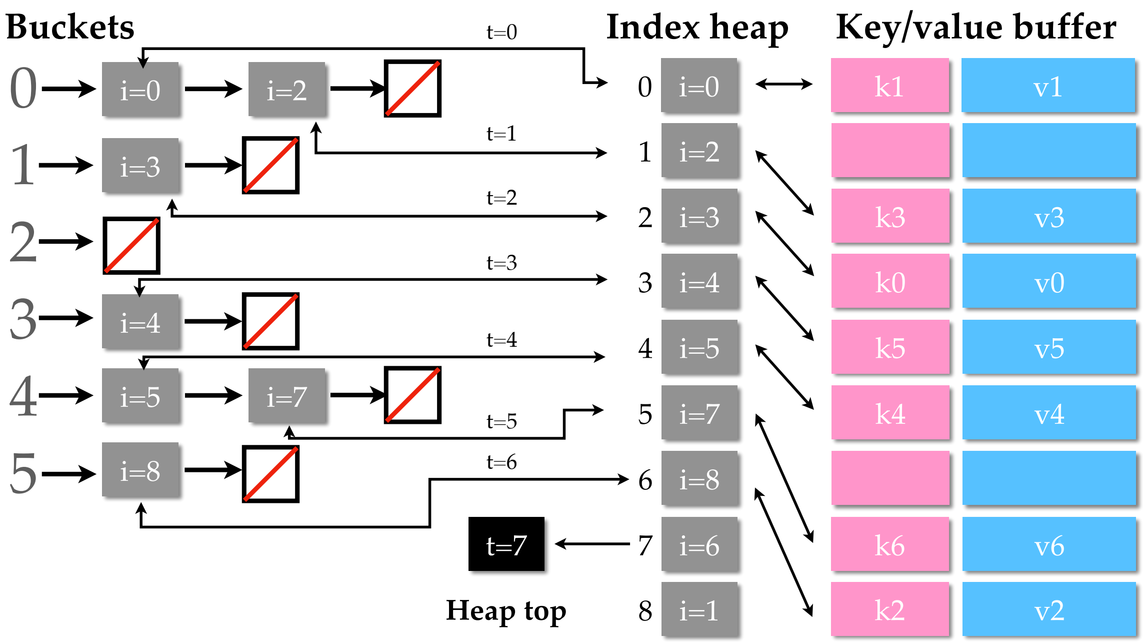


Figure 5.4: Illustration of a non-generic hash set enhanced by ASH. Integer *buffer indices* i allocated from the index heap (middle) are inserted as delegate *keys* directly into the hash set (left), associated with *actual keys* in the buffer (right) at i. The key-bucket correspondences are the same as Figs. 5.2 and 5.3, omitted for simplicity.

Given a generic key k, we first locate the bucket $b = h(k) \in \mathcal{I}_n$. Ideally, we can then allocate a buffer index i at $\mathbf{h}$'s top t and insert it into the linked list at the bucket b in the integer-only hash set. The accompanying key and value are put in $\mathbf{k}^B(i), \mathbf{v}^B(i)$. During query, we similarly first locate the bucket b then search the key in the linked list by visiting $\mathbf{k}^B$ via the stored index i.

Multi-Pass Insertion

Although query can be applied as mentioned above, lazy insertion mentioned in Section 5.4.3 is problematic in this setup. The main reason is that while the race condition in inserting index i does not occur in warp-oriented insertions, the copy of the *actual* key $k \in \mathcal{K}$ to $\mathbf{k}^B$ requires global memory write. They may not be synchronized among threads, as copying a multi-dimensional key takes several non-atomic instructions. As a result, the insertion of a key k_2 could be accidentally triggered when i) a duplicate $k_1 (= k_2)$'s index $i_1 \in \mathcal{I}_c$ has been inserted but ii) whose actual key k_1 has only been partially copied to the buffer $\mathbf{k}^B$. This would mistakenly result in $\mathbf{k}^B(i_1) = k_1 \neq k_2$ followed by the unexpected insertion of k_2 when unsynchronized. In practice, with more than 1 million keys to be inserted in parallel, these kinds of conflicts happen with probability as low as $\leq 0.1\%$. To resolve conflicts, we split insertion into three passes:

- *Pass 1:* batch insert *all* keys $\mathbf{k}$ to $\mathbf{k}^B$ by directly copying all candidates via batch allocated corresponding indices i from $\mathbf{h}$;
- *Pass 2:* perform parallel hashing with indices i from pass 1. In this pass, keys are read-only in global buffers and hence do not face race conditions. Successful insertions are marked in a mask array.
- *Pass 3:* batch insert values to $\mathbf{v}^B$ with successful masks, and free the rest to $\mathbf{h}$.

While there is overhead due to the multi-pass operation, it is still practical for a dynamic hash map. First, keys are relatively inexpensive to copy, especially for spatial coordinates, while the more expensive copying of values is done without redundancy. Second, a dynamic hash map generally reserves sufficient memory for further growth so that the *all* key insertion would not exceed the buffer capacity.

5.4.5 Rehashing and Memory Management

While buffers are represented as fixed-size arrays, growth of storage is needed to accommodate the accumulated input data, which can exceed the hash map's capacity, *e.g.* 3D points from an RGB-D stream. This triggers rehashing, where we adopt the conventional $\times 2$ strategy to double the buffer size as common in the C++ Standard Library, collect all the active keys and values, and batch insert them into the enlarged buffer.

In dynamic insertions, there can be frequent free and allocation of small memory blobs that are adjacent and mergeable. In view of this, we implement another tree-structured global GPU manager similar to PyTorch [182].

5.4.6 Dispatch Routines

To enable bindings to non-templated languages, *e.g*. Python, the tensor interface is non-templated so that it can take data types and shapes as arguments. In the context of spatial hashing, we support arbitrary dimensional keys by expanding the dispatcher macros in C++. Float types have undetermined precision behaviors on GPU. Therefore, a conversion to the integers given the desired precision is recommended to use the hash map.

We also additionally dispatch values by their element byte sizes into intrinsically supported vectors: `int, int2, int3`, and `int4`. This adaptation accelerates trivially copiable value objects such as `int3`, and supports non-trivially copiable value blocks (*e.g*. an 8^3 array pointed to a `void` pointer). This improves the insertion of large value chunks by a factor of 10 approximately.

5.4.7 Multi-value Hash Map and Hash Set

ASH supports multi-value hash maps that store values organized in SoA, as well as hash sets with only keys and no values.

Multi-value hash maps. Various applications in 3D processing require mapping coordinates to several properties. For instance, a 3D coordinate can be mapped to a normal, a color, and a label in a point cloud. While the mapped values can be packed as an array of structures (*i.e.*, AoS) to fit a hash map, code complexity could increase since structure-level functions have to be implemented. We generalize the hash map's functionality by extending the *single* value buffer $\mathbf{v}^B$ to an *array of value buffers* $\{\mathbf{v}_i^B\}$ and applying loops over properties per index during an insertion. This simple change supports the storage of complex value types in SoA that allows easy vectorized query and indexing.

Hash set. A hash set, on the other hand, is a simplified hash map – an unordered set that stores unique keys. It is generally useful in maintaining a set by rejecting duplicates, such as in point cloud voxelization. By removing $\mathbf{v}^B$ and ignoring value insertion, a hash map becomes a hash set.

5.5 Experiments

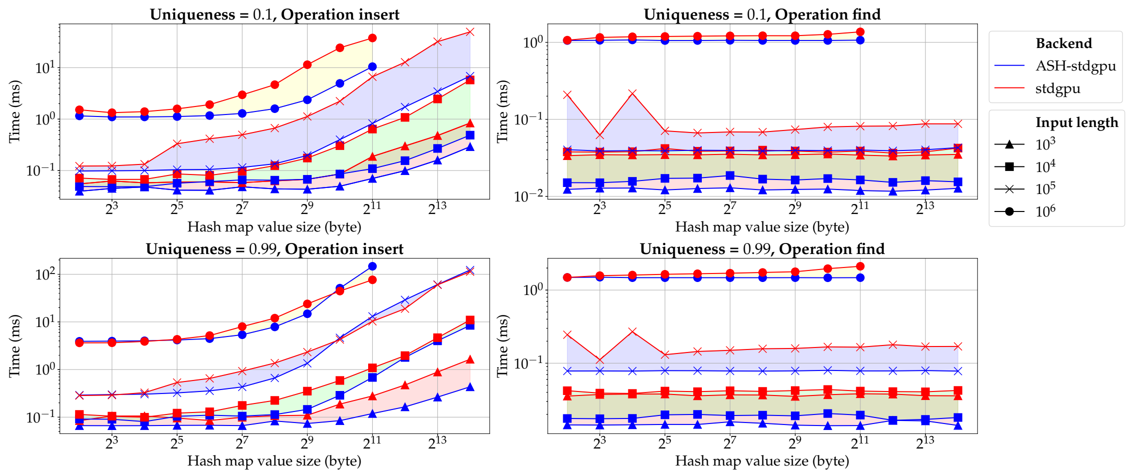


Figure 5.5: Hash map performance comparison between ASH-stdgpu and the vanilla stdgpu with 3D integer keys. Each curve shows the average operation time (*y-axis*) with varying hash map value sizes in bytes (*x-axis*), given a controlled backend, input length, and input key uniqueness ratio. *Lower is better*. Further factors are denoted by the legends on the right. ASH-stdgpu runs consistently faster than the vanilla stdgpu.

We start with synthetic experiments to show that ASH, with its optimized memory layout, increases performance while improving usability. All experiments in this section are conducted on a laptop with an Intel i7-6700HQ CPU and an Nvidia GTX 1070 GPU. In all experiments, we assume the hash map capacity is equivalent to the number of input keys (regardless of duplicates). Each reported time is an average of 10 trials.

5.5.1 Spatial Hashing with Generic Backend

The first experiment is the performance comparison between vanilla stdgpu and ASH with stdgpu backend (ASH-stdgpu). For fairness, we extend the examples of stdgpu such that an array of iterators and masks are returned for in-place manipulations. The number of buckets and the load factor are determined internally by stdgpu.

Setup 1. *We test randomly generated 3D spatial coordinates mapped to float value blocks of varying sizes. The key $\mathcal{K}$, value $\mathcal{V}$, capacity c, and uniqueness ρ are chosen as follows:*

$$\begin{aligned}\mathcal{K} &= \{\texttt{Tensor((3), Int32)}\},\\ \mathcal{V} &= \{\texttt{Tensor((2}^{\texttt{j}}\texttt{), Float32)} \mid j = 0, 1, \ldots, 12\},\\ c &= \{10^j \mid j = 3, 4, 5, 6\},\\ \rho &= \{0.1, 0.99\},\end{aligned}$$

where ρ indicates the ratio of the unique number of keys to the total number of keys being inserted or queried.

Fig. 5.5 illustrates the comparison between vanilla stdgpu and ASH-stdgpu. For *insert* operation, ASH-stdgpu is significantly faster than stdgpu when $\rho = 0.1$ is low, and the performance gain increases when the value byte size increases. This is mainly due to the SoA memory layout and the lazy insertion mechanism, where a lightweight integer $i \in \mathcal{I}_c$ is inserted in an attempt instead of the actual value $v \in \mathcal{V}$. At a high input uniqueness $\rho = 0.99$, ASH-stdgpu maintains the performance advantage with low and medium value sizes, and its performance is comparable to stdgpu with a large value size. This indicates that our dispatch pattern in copying values helps in a high throughput scenario. For *find* operation, ASH-stdgpu is consistently faster than vanilla stdgpu, under both high and low key uniqueness settings.

In addition to to *insert* and *find*, we introduce a new *activate* operation. It "activates" the input keys by inserting them into the hash map and obtaining the associated buffer indices. This is especially useful when we can pre-determine and apply the element-wise initialization. Examples include the TSDF voxel blocks (zeros) and multi-layer perceptrons (random initializations). The *activate* operation is absent in most existing hash maps and is only available as hard-coded functions [65, 171, 184].

With the *activate* operation, we conduct ablation studies to compare the insertion time of merely the keys versus the insert time of both the keys and values. Fig. 5.6

compares the runtime between *insert* and *activate* in ASH-stdgpu. The key, value, capacity, and uniqueness choices are the same as in Setup 1. We observe that while the insertion time increases as the value size increases, the activation time remains stable.

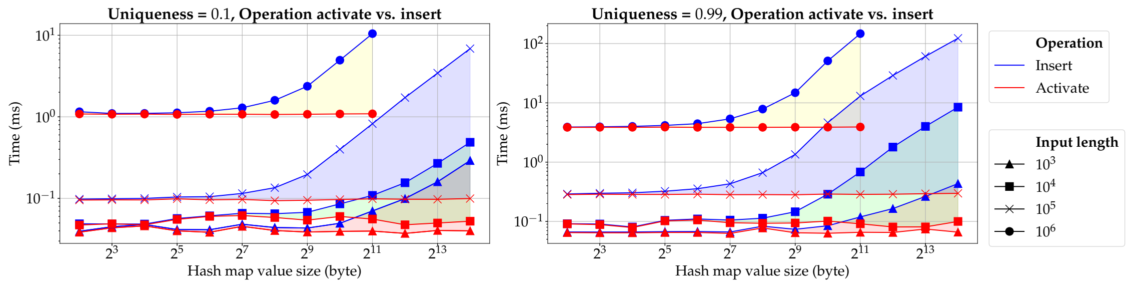


Figure 5.6: Study of the *activate* operation introduced in ASH against *insert* with 3D integer keys on the ASH-stdgpu backend. Each curve shows the average operation time (*y-axis*) with varying hash map value sizes in bytes (*x-axis*), given an input length and input key uniqueness ratio. *Lower is better*. *Activate* keeps a stable runtime in the tasks that do not require explicit value insertion, while *insert* time increases corresponding to the hash map value size.

5.5.2 Integer Hashing with Non-Generic Backend

Next, we compare ASH based on the SlabHash backend (ASH-slab) with the vanilla SlabHash. Since SlabHash only supports integers as keys and values, we limit our ASH-slab backend to the same integer types here. The number of buckets is $2\times$ capacity (load factor is approx. 0.5), since it is empirically the best factor when ASH-slab is applied to non-generic and generic tasks. Since vanilla SlabHash only supports data I/O from the host, we include the data transfer time between host and device when measuring the performance of ASH-slab.

Setup 2. *We test random scalar integer values mapped to scalar float values. The key $\mathcal{K}$,* *value $\mathcal{V}$, capacity c, and uniqueness ρ are chosen as follows:*

$$\begin{aligned}
\mathcal{K} &= \{\texttt{Tensor((1), Int32)}\},\\
\mathcal{V} &= \{\texttt{Tensor((1), Float32)}\},\\
c &= \{10^j \mid j = 3, 4, 5, 6\},\\
\rho &= \{0.1, 0.2, \ldots, 0.9, 0.99\},
\end{aligned}$$

As shown in Fig. 5.7, although ASH-slab does not make use of the non-blocking warp-oriented operations in SlabHash in order to enable support for generic key and value types, our *insert* is still comparable to vanilla SlabHash which is only optimized for integers. The drop in performance of ASH-slab when ρ increases is an expected indication that the overhead of multi-pass insertion increases correspondingly.

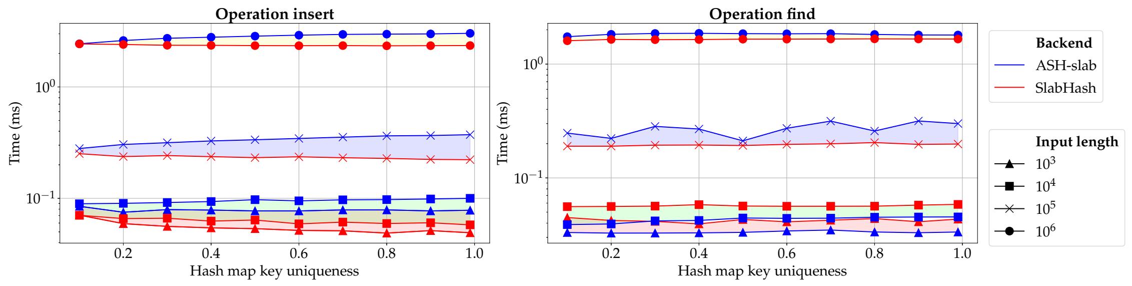


Figure 5.7: Hash map performance comparison between ASH-slab and the vanilla SlabHash with 1D integer keys and values. Each curve shows the average operation time (*y-axis*) with varying input key uniqueness ratio (*x-axis*), given an input length. *Lower is better*. ASH-slab retains a comparable performance for integers while supporting generalization to arbitrary dimensional keys and values of various data types.

It is worth mentioning that with an improved global memory manager, the construction of an ASH-slab hash map takes less than 1ms under all circumstances, while the vanilla SlabHash constantly takes 30ms for the redundant slab memory manager. In practice, where the hash map is constructed and used once (*e.g.* voxelization), ASH-slab is a more practical solution.

5.5.3 Ablation Between Backends

We now conduct an ablation study with different backends in ASH, namely ASH-stdgpu and ASH-slab, with arbitrary input key-value types beyond integers. The experimental setup follows Section 5.5.1.

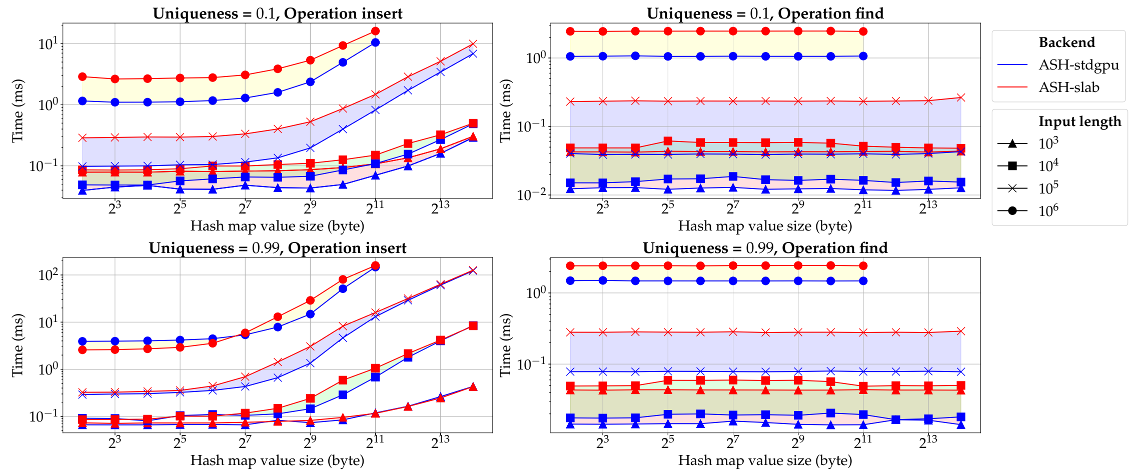


Figure 5.8: Ablation study of the hash map performance with 3D integer keys over different backends. Each curve shows the average operation time (*y-axis*) with varying hash map value sizes in bytes (*x-axis*), given a controlled backend, input length, and input key uniqueness ratio. *Lower is better*. ASH-stdgpu outperforms ASH-slab in most circumstances with the 3D integer keys and varying length values, which are common in real-world applications.

In Fig. 5.8, we can see that ASH-stdgpu outperforms ASH-slab in most circumstances with the 3D coordinate keys and varying length values that are common in real-world applications. While warp-oriented operations heavily used in Slab-Hash enjoy the benefits of intrinsic acceleration, they sacrifice the granularity of operations. Threads can only move on to the next task once all the operations in a warp (of 32) are finished. As a result, early termination when an insertion failure occurs are less likely in a warp-oriented hash map. If the data layout is not well-distributed for the intrinsic operations (*e.g.*, low-uniqueness input, keys with long word width), the performance drop could be significant.

This observation is more apparent in insertion under varying input densities. With a relatively small value size and a high uniqueness, ASH-slab performs better. When the uniqueness is low, however, each thread in ASH-slab still has to finish a

Table 5.2: Comparison of the complexity of coding (top) and LoC (bottom) of each operation among the implementations. Unlike stdgpu and SlabHash, ASH does not require that users write device code or use a CUDA compiler. It requires few LoC for construction, query, and insertion.

	stdgpu	SlabHash	ASH
Device code free?	✗	✓	✓
CUDA compiler free?	✗	✗	✓
Construct	3	9	3
Find	22	2	2
Insert	27	1	2

similar workload before termination, while ASH-stdgpu can reject many failure insertions early and move on to the following workloads. As of now, ASH-slab is suitable for the voxel downsampling application, while ASH-stdgpu is better for other tasks. Therefore, we set stdgpu as the default backend for ASH in the remaining sections.

5.5.4 Code Complexity

We now study the usage at the user end. First of all, the ASH framework, regardless of the backend used, is already compiled as a library. A C++ developer can easily include the header and build the example directly with a CPU compiler and link to the precompiled library with a light tensor engine. An equivalent Python interface is provided via pybind [113] as shown in Fig. 5.1.

In comparison, to use SlabHash's interface with an input array from host memory, a CUDA compiler is required, along with manual bucket-linked list chain configurations. For further performance improvement, detailed memory management has to be done manually via `cudaMalloc`, `cudaMemcpy` and `cudaFree`. stdgpu provides a built-in memory manager but requires writing device and host functions. In a query operation, the found values are returned *by-copy* for SlabHash, so in-place modification requires further modification of the library. stdgpu exposes iterators in an AoS fashion, therefore the device code needs to be implemented to reinterpret an array of iterators and masks for further operations.

The compilation complexity and the interface LoC required for the same functionality in C++ are listed in Table 5.2.

5.6 Applications

We now demonstrate a number of applications and ready-to-use systems in 3D perception to demonstrate the power of ASH with fewer LoC and better performance. The presented applications include:

1. Point cloud voxelization;
2. Retargetable volumetric reconstruction;
3. Non-rigid registration and deformation;
4. Joint geometry and appearance refinement.

The first two experiments are conducted on an Intel i7-6700HQ CPU and an Nvidia GeForce GTX 1070 GPU for indoor scenes. Outdoor scene experiments are run on an Intel i7-11700 CPU and an Nvidia RTX 3060 GPU. The rest are done on an Intel i7-7700 CPU and an Nvidia GeForce GTX 1080Ti GPU.

5.6.1 Point Cloud Voxelization

Setup 3. *In voxelization, a hash map maps a point cloud's discretized coordinates to its natural array indices, and the hash map capacity is the point cloud size, typically ranging from* 10^5 *to* 10^7

$$\mathcal{K} = \{\texttt{Tensor((3), Int32)}\},$$
$$\mathcal{V} = \{\texttt{Tensor((1), Int32)}\}.$$

Voxelization is a core operation for discretizing coordinates. It is essential for sparse convolution [49, 51] at the quantization preprocessing stage, and is often used to generate point cloud "pyramids" [272] in coarse-to-fine 3D registration for improved speed and robustness.

Voxelization is a natural task for parallel hashing, as the essence of the operation is to discard duplicates at grid points. To achieve this, we first discretize the input by converting the coordinates described by the continuous meter metric to the voxel units. Then a simple hash set insertion eliminates the duplicates and corresponds them to the remaining unique coordinates. The returned indices can be reused for tracing other properties such as colors and point normals associated with the input.

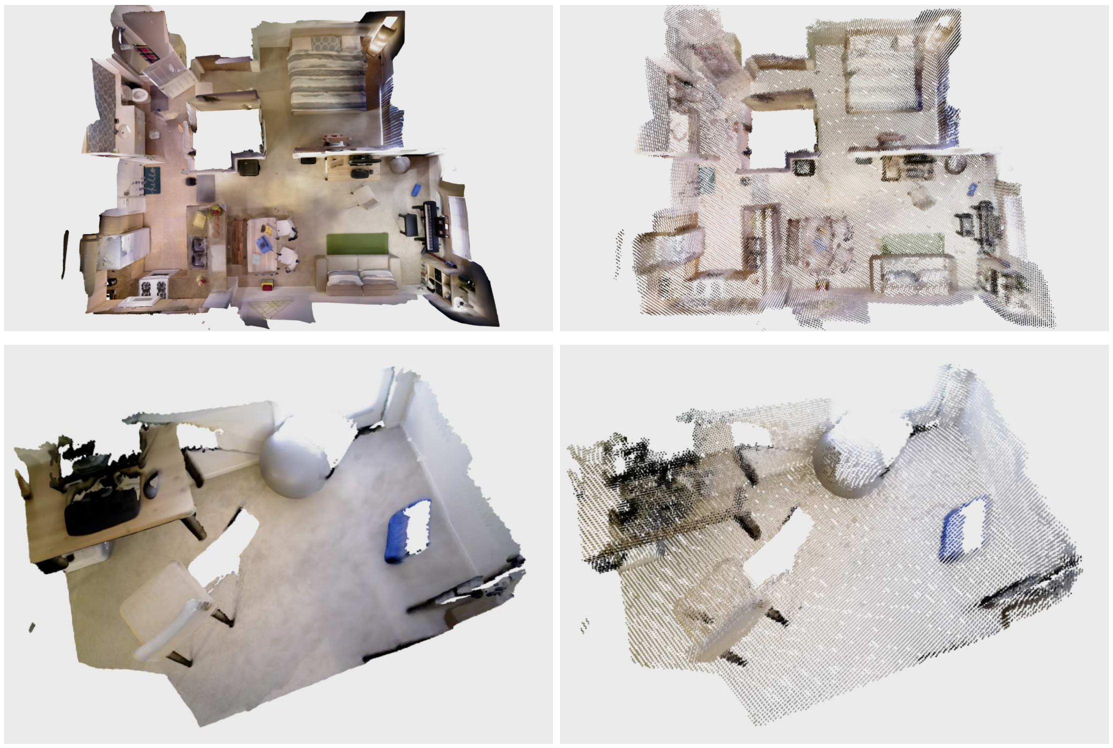


Figure 5.9: Visualization of point cloud voxelization. Top: scene-level large-scale inputs. Bottom: fragment-level small-scale inputs. Left: original point clouds. Right: voxelized point clouds.

The python code for voxelization can be found in Listing 1.

We compare voxelization implemented in ASH with two popular implementations, MinkowskiEngine [49] on CUDA and Open3D [272] on CPU. Our experiments are conducted on a large scene input with 8×10^6 points which is typical for scene perception, and a small fragment of the scene with 5×10^5 points which is typical for an RGB-D input frame, as shown in Fig. 5.9.

To evaluate the performance, we vary the parameter *voxel size* from 5mm to 5cm, which is typical in the spectrum of voxelization applications, from dense reconstruction to feature extraction. In Fig. 5.10 we can see that our implementation outperforms baselines consistently for inputs at both scales. Meanwhile, in measuring the LoC written in C++ (the Python wrappers are one-liners) required for the functionality given the hash map interface, we observe that ASH requires only 28 LoC, while MinkowskiEngine and Open3D on CPU take 71 and 72 LoC respectively.

```
import open3d.core as o3c

def voxelize(pcd, voxel_size):
    xyz = pcd.point.positions
    N = len(xyz)
    hashset = o3c.HashSet(N, o3c.int32, (3,))
    xyz_int = (xyz / voxel_size).floor().to(o3c.int32)
    _, mask = hashset.insert(xyz_int)
    # Return points with indices
    return xyz[mask], o3c.arange(N)[mask]
```

Listing 1: Voxelization

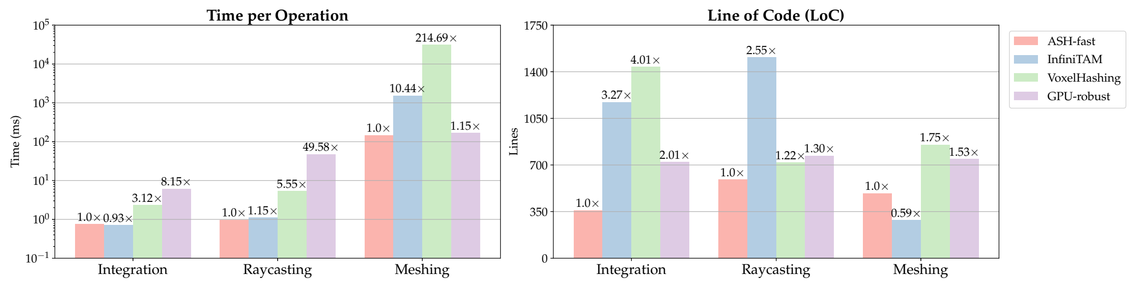


Figure 5.10: Performance comparison of voxelization. Each curve shows the run time (*y-axis*) over the varying voxel size (*x-axis*). *Lower is better*. ASH is consistently faster than Open3D's default voxelizer (CPU) and MinkowskiEngine (CUDA).

5.6.2 Retargetable Volumetric Reconstruction

Truncated Signed Distance Function

Scene representation with truncated signed distance function (TSDF) from a sequence of 3D input has been introduced [56] and adapted to RGB-D [168]. It takes a sequence of depth images $\{D^j\}$ with their poses $\{\mathbf{T}^j \in \mathbf{SE}(3)\}$ as input, and seeks to optimize the signed distance, an implicit function value d per point at $\mathbf{x} \in \mathbb{R}^3$. The signed distance measured for frame j is given by[3]

$$[u, v, r]^\top = \Pi(\mathbf{T}^{j^{-1}}\mathbf{x}), \tag{5.9}$$

$$d^j = D^j(u, v) - r, \tag{5.10}$$

[3]Details including depth masking and projective pinhole camera model are omitted for clarity and could be found in KinectFusion [168].

where Π projects the 3D point to 2D with a range reading after a rigid transformation. To reject outliers, a truncate function $\Psi_\mu(d) = \mathrm{clamp}(d, -\mu, \mu)$ is applied to d^j. There are multiple variations of Ψ and the definition of signed distance d^j [29]. For this paper, we follow the convention in KinectFusion [168].

With a sequence of inputs, per-point signed distance can be estimated in least squares with a closed-form solution

$$d = \arg\min_t \sum_j w^j \|t - d^j\|^2, d = \frac{\sum_j w^j d^j}{\sum_j w^j}, \tag{5.11}$$

where w^j is the selected weight depending on view angles and distances [29]. In other words, with a sequence of depth inputs and their poses, we can measure TSDF at any point in 3D within the camera frustums. We can also rewrite Eq. 5.11 incrementally:

$$d := \frac{w \cdot d + w^j \cdot d^j}{w + w^j}, w := w + w^j, \tag{5.12}$$

where w is the accumulated weight paired with d.

Equipped with a projection model Π that converts a point to signed distance, TSDF reconstruction can be generalized to imaging LiDARs [66] for larger scale scenes.

Spatially Hashed TSDF Blocks

Setup 4. *In a scene represented by a volumetric TSDF grid, a hash map maps the coarse voxel blocks' coordinates to the TSDF data structure of the voxel block, and the hash map capacity is typically* 10^3 *to* 10^5 *for small to large-scale indoor scenes:*

$$\mathcal{K} = \{\texttt{Tensor((3), Int32)}\},$$
$$\mathcal{V} = \{\texttt{Tensor((}\ell^3\texttt{), Float32)}, \texttt{Tensor((}\ell^3\texttt{), Float32)}\},$$

where ℓ is the voxel block resolution, which is set to 8 *or* 16.

While recent neural representations utilize multi-layer perceptrons to approximate the TSDF in continuous space [38], classical approaches use discretized voxels.

Such representations have a long history and are ready for real-world, real-time applications. They can also provide data for training neural representations.

The state-of-the-art volumetric discretization for TSDF reconstruction is spatial hashing, where points are allocated around surfaces on-demand at a voxel resolution of around $5mm$. While it is possible to hash high-resolution voxels directly, the access pattern could be less cache-friendly, as the neighbor voxels are scattered in the hash map. A hierarchical structure is a better layout, where small dense voxel grids (*e.g*. in the shape of 8^3 or 16^3) are the minimal unit in a hash map; detailed access can be redirected to simple 3D indexing. In other words, a voxel can be indexed by a coupled hash map lookup and a direct local addressing

$$\mathbf{x}_{\text{block}} = \lfloor \mathbf{x}/(s\ell) \rfloor, \tag{5.13}$$

$$\mathbf{x}_{\text{voxel}} = \lfloor (\mathbf{x} - \mathbf{x}_{\text{block}} \cdot s\ell)/s \rfloor, \tag{5.14}$$

where s is the voxel size and ℓ is the voxel block resolution as described in Setup 4.

While previous implementations [65, 171, 184] have achieved remarkable performance, modularized designs are missing. Geometry processing and hash map operations were coupled due to the absence of a well-designed parallel GPU hash map. One deficiency of this design is *unsafe* parallel insertion, where the capacity of a hash map can be exceeded. Another is ad hoc recurring low-level linked list access in geometry processing kernels that cause high code redundancy. Our implementation demonstrates the first modularized pipeline where safe hash map operations are used without any ad hoc modifications.

Voxel Block Allocation and TSDF Integration

Setup 5. *For an input depth image, a hash map maps the unprojected point coordinates to the active indices as described in Setup 4, and the capacity of a hash map is typically* 10^5 *to* 10^6*, with* 10^2 *to* 10^3 *valid entries:*

$$\mathcal{K} = \{\texttt{Tensor((3), Int32)}\},$$

$$\mathcal{V} = \{\texttt{Tensor((1), Int32)}\}.$$

In the modularized design, we first introduce a double hash map structure for voxel block allocation and TSDF estimation. Voxel block allocation identifies points from $\{D^j\}$ as surfaces and computes coordinates with Eq. 5.13. Intuitively, they can be directly inserted to the *global* hash map described in Setup 4. This is achievable in an ad hoc implementation where the core of the hash map is modified at the device code level, and *unsafe* insertion is allowed [171, 184]. However, in a modularized and *safe* setup, this could lead to problems. A VGA resolution depth input contains $640 \times 320 \approx 3 \times 10^5$ points and easily exceeds the empirical *global* hash map capacity. As we have mentioned, rehashing will be triggered under such circumstances, which is both time and memory consuming, especially for a hash map with memory-demanding voxel blocks as values.

To address this issue without changing the low-level implementation and sacrificing safety, we introduce a second hash map, the *local* hash map from Setup 5. This hash map is similar to the one used in *voxelization*: it maps discretized 3D coordinates unprojected from depths to integer indices. With this setup, a larger input capacity is acceptable, as the *local* hash map is lightweight and can be cleared or constructed from scratch per iteration.

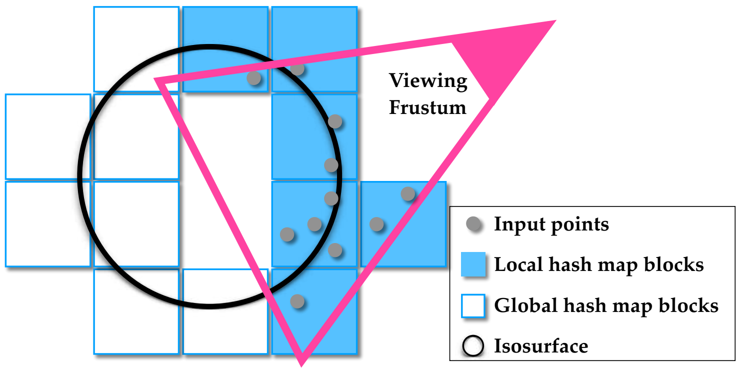


Figure 5.11: Illustration of local and global hash maps iteratively used in real-time reconstruction. The local hash map activates voxel blocks enclosing points observed in the viewing frustum. The global hashmap accumulates such activated blocks and maintains all the blocks around the isosurface.

There are two main benefits to using a *local* hash map: it converts the input from the 10^5 raw point scale to the 10^3 voxel block scale, which is safe for the *global* hash map without rehashing; as a byproduct, it keeps track of the active voxel blocks for the current frame j, which can be directly used in the following TSDF integration and ray casting. The local and global hash maps can be connected through indices, where a query of coordinate in the local map is redirected to the global map in-place. Fig. 5.11 shows the roles of the two hash maps. Listing 2 details the construction and interaction between the two hash maps.

```
import open3d.core as o3c

# Map block coords to actual storage
global_hashmap = o3c.HashMap(
    global_capacity,
    key_dtype=o3c.int32,
    key_shape=(3,),
    # Float 1-channel TSDF and 3-channel color
    value_dtypes=(o3c.float32, o3c.float32),
    values_shapes=((8, 8, 8, 1), (8, 8, 8, 3)))
# Map block coords to global hashmap indices
local_hashmap = o3c.HashMap(
    local_capacity,
    key_dtype=o3c.int32,
    key_shape=(3,)
    # Index in global hash map
    value_dtypes=(o3c.int32),
    value_shapes=((1,)))
# Discretize and insert to local map
xyz_int = (xyz / block_size).floor().to(o3c.int32)
i_local, mask = local_hashmap.activate(xyz_int)
# Remove duplicates
xyz_int = xyz_int[mask]
i_local = i_local[mask]
# Activate and query in the global map
global_hashmap.activate(xyz_int)
i_global, mask = global_hashmap.find(xyz_int)
# Associate local and global maps via indices
local_v = local_hashmap.value()
local_v[i_local] = i_global
```

Listing 2: Double hash map allocation

By accessing the global hash map's TSDF and colors through returned indices, TSDF integration can then be implemented following Eq. 5.12 in a pure geometry function, either in a low-level GPU kernel or a high-level vectorized Python script. Spatial hash map is detached from the core geometric computation, providing more flexibility in performance optimization.

Surface Extraction

A volumetric scene reconstruction is not usable for most software and solutions until the results are exported to point clouds or triangle meshes. Hence we implement a variation of Marching Cubes [146] that extracts vertices with triangle faces at zero crossings in a volume. In a spatially hashed implementation, boundary voxels in voxel blocks are harder to process since queries of neighbor voxel blocks are frequent, and shared vertices among triangles are hard to track. One common approach is to simply visit vertices at isosurfaces and disregard duplicates, but this usually results in a heavily redundant mesh [128, 184], or time-consuming post-processing to merge vertices [171]. Another method is to introduce an assistant volumetric data structure to atomically record the vertex-voxel association, but the implementations are over-complex and require frequent low-level hash map queries coupled with surface extraction [63, 65].

Now that we have a unified hash map interface, we simplify the voxel block neighbor search routine [65] and set up a *1-radius neighbor* lookup table in advance, as described in Listing 3. Surface extraction is then detached from hash map access and can be optimized separately. As a low-hanging fruit, point cloud extraction is implemented with the same routine by ignoring the triangle generation step. In fact, surface extraction of a median-scale scene shown in Fig. 5.13 takes less than 100ms, making interactive surface updates possible in a real-time reconstruction system.

Ray Casting

Another way to interpret a volume is through ray casting or ray marching. Given camera intrinsics and extrinsics, ray casting renders depth and color images by marching rays in the spatially hashed volumes, querying color and TSDF values, and finding zero-crossing interfaces. It allows rendering at known viewpoints, synthesizing novel views, and estimating camera poses.

Various accelerations can speed up ray casting. Adaptive spherical ray casting and a precomputed min-max range estimate [184] will constrain the search range and boost performance. The latter can be conducted by simply projecting the active keys collected in Listing 2 without the involvement of hash maps. In addition,

```
import open3d.core as o3c

def radius_nns(xyz, r):
    N = len(xyz)
    hashset = o3c.HashSet(N, o3c.int32, (3,))
    hashset.insert(xyz)
    # Get offset tensors
    # ([-r, -r, -r], ..., [r, r, r])
    offsets = enumerate_radius_offsets(r)
    # Collect neighbors
    xyz_query = xyz.clone()
    for offset in offsets:
        xyz_query.append(xyz + offset, 0)
    # Query
    indices, masks = hashset.find(xyz_query)
    # Reshape to get neighbor indices for each point
    indices = indices.view(N, -1)
    masks = indices.view(N, -1)
```

Listing 3: Radius nearest neighbor search in 3D.

we can squeeze more from our double-hash-map architecture. Conventional ray marching applies query in the *global* hash map [65, 171, 184]. Since the *local* hash map is directly associated with the *global* hash map with shared active indices, we can replace the *global* hash map with the *local* one accompanied with the $\mathbf{v}^B$ buffer in the *global* hash map. With such a simple change, we can now query the more compact *local* hash map, and access the *global* hash map in-place without touching the geometric computations. Since out-of-frustum voxel blocks are ignored, this operation slightly sacrifices rendering completeness at image boundaries, as shown in Fig. 5.12. However, it is able to boost speed by a factor of 5, and is useful for real-time systems such as dense SLAM.

Retargetable Reconstruction System

Thanks to the design where spatial hashing and geometry processing are detached, the spatially hashed volumetric representation can be reused for multiple purposes, from posed RGB-D surface reconstruction, dense RGB-D SLAM, to large-scale LiDAR surface reconstruction, with minimal modifications. For RGB-D input, a handful of existing reconstruction systems [65, 171, 184] run on GPU. We first retarget our system to *fast* SLAM setup for fair comparisons. In this setup, we use a less aggressive ray-based allocation strategy [184] and store only weighted TSDF,

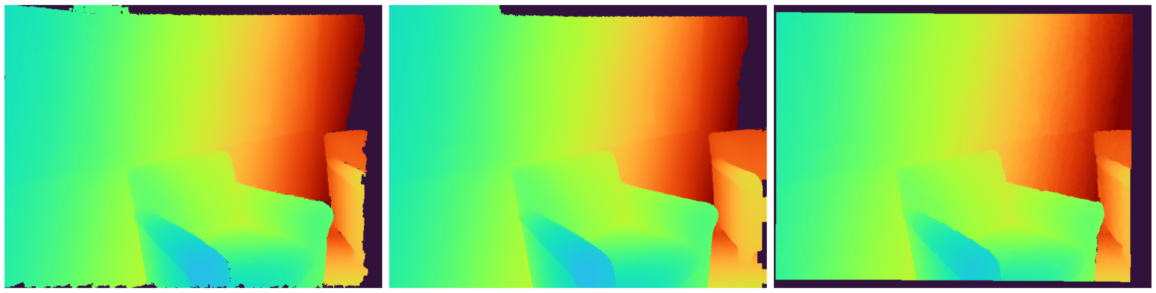


Figure 5.12: Visualization of volumetric ray casting. From left to right: rendered depth from *local*, *global* hash maps, and input ground truth depth. Note the difference at boundaries.

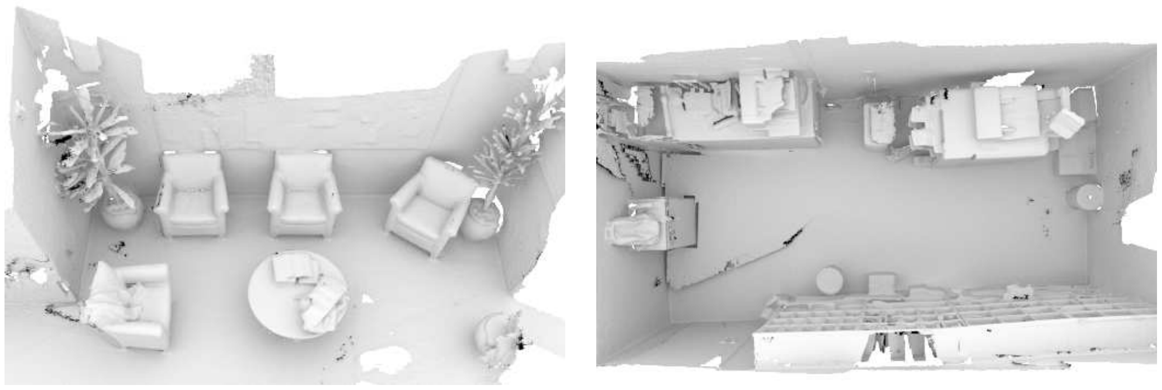


Figure 5.13: Visualization of triangle mesh extracted from the real-time dense SLAM system on scene *lounge* and *copyroom* in the *fast* mode. Rendered with Mitsuba 2 [172].

consistent with the baselines. Camera poses are estimated in real-time via frame-to-model alignment [171] between input frames and ray-casting rendering through the double-hash-map. We compare the performance of this setup against the state-of-the-art implementations with the same parameters: voxel size is 5.8mm, voxel block resolution is $\ell = 8$, TSDF truncation distance is 4cm, min/max acceptable range of depth scanning is 0.2m and 3m. Performance is profiled on the *lounge* scene shown in Fig. 5.13. Breakdown analysis of runtime and LoC[4] is shown in Fig. 5.15. In most comparisons, we can see a significant performance gain, with fewer LoC to write thanks to the elimination of redundant hash map look-ups in geometry kernels.

[4]All the code are reformatted with clang-format with a modified Google style.

Fast mode runs far beyond framerate and reconstructs pure geometry. In addition, we implement a *quality* mode for richer volumetric information including color, and reduce noise by integrating depth into 16^3 TSDF voxel blocks from a 1-radius-neighbor allocation [272]. The introduction of color requires double memory and triple computation cost in trilinear interpolation for ray casting and surface extraction, thus the *quality* mode is around $3\times$ slower. However, it still runs in real-time, and provides a better user experience. Fig. 5.14 shows the interactive reconstruction system in the *quality* mode. The system runs at 30Hz on a mid-end laptop, providing incremental volumetric reconstruction and interactive point cloud extraction and realistic rendering. Note that to retarget from a *fast* system to a user-friendly *quality* application, we only need to change the block allocation function and several parameters, in total several dozen of lines, without re-writing the core.

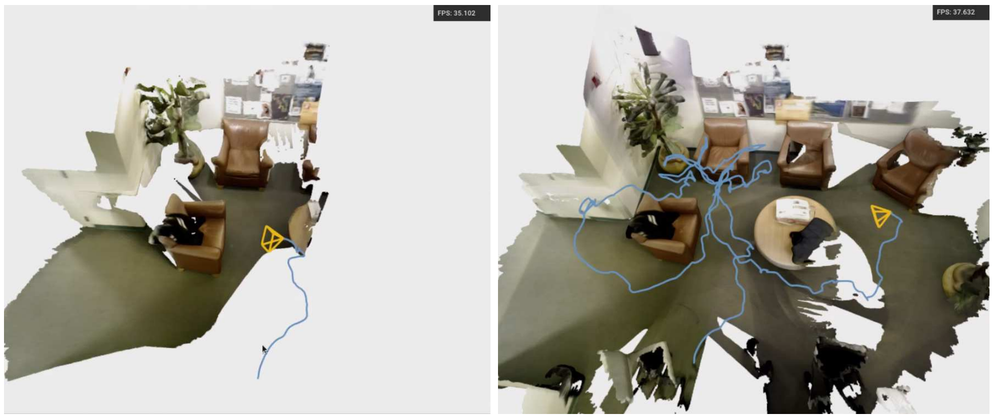


Figure 5.14: Visualization of the real-time dense SLAM system in the *quality* mode with colored and interactive surface reconstruction. Viewpoints can be changed by users to visualize the incremental reconstruction of the scene.

The system can also be adapted to LiDAR point clouds. By simply replacing the conventional pinhole camera model with a customized spherical projection model [66], we can reconstruct large-scale scenes from LiDAR data, see Chapter 6 for details.

To conclude this subsection, we presented a retargetable volumetric reconstruction system with a modular design, separating hash maps and geometric operations. With minimal changes in geometry functions, the core volumetric

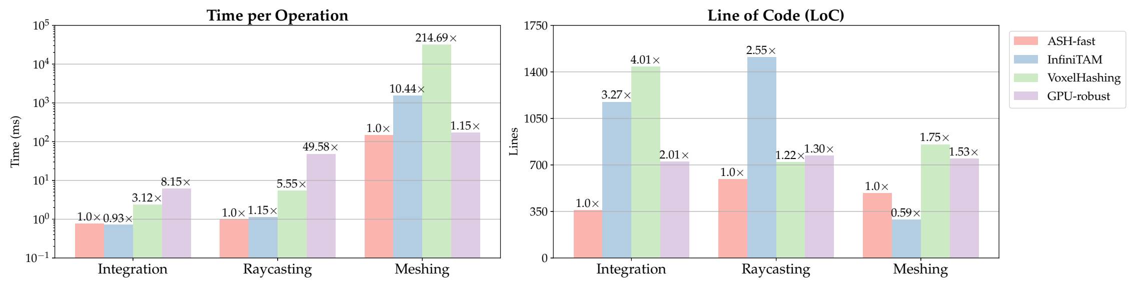


Figure 5.15: Performance and LoC comparison of our real-time dense SLAM pipeline in the *fast* mode (ASH-fast) against state-of-the-art implementations: InfiniTAM [184], VoxelHashing [171], GPU-robust [63]. Evaluated on the *lounge* scene [46]. Left: detailed comparison of separating modules. Right: corresponding LoC comparison. *Lower is better*. Note the meshing LoC in InfiniTAM is significantly fewer since the implementation is over-simplified and requires further postprocessing. ASH-fast achieves a consistent fast speed with fewer LoC.

representation can be used for RGB-D and LiDAR scene reconstruction, and interactive dense SLAM. Our system is faster, requires fewer LoC, and supports an easy switch between speed and fidelity.

5.6.3 Non-Rigid Volumetric Deformation

While fast online volumetric reconstruction is useful in exploration and visualization, offline reconstruction systems [46] are sometimes preferred when a higher quality is required for design and evaluation.

State-of-the-art offline systems adopt divide-and-conquer. Long input sequences are split into smaller subsets, each yielding a submap point cloud $\mathcal{M}^j$ reconstruction with less drift. In this setup, we can simply reuse the integration and surface extraction components in the previous subsection [46, 65]. A global pose graph is then constructed and optimized after robust registration [51, 247] of submaps. For the details, we refer the readers to state-of-the-art RGBD reconstruction systems [46, 65].

However, issues still persist in challenging scenes, *e.g.*, heavy misalignment due to the strong simulated noise in the Augmented ICL dataset [46], and the artifacts presented in the large-scale indoor RGBD LiDAR dataset [180], as shown in Fig. 5.17. To deal with this, non-rigid volumetric deformation is presented in Simultaneous Localization and Calibration (SLAC) [269, 270].

SLAC attempts to minimize the distance between correspondences from different submaps by optimizing a combination of rigid transformations and non-rigid deformations. While the rigid transformations are simply the submap poses $\{\mathbf{T}^j\}$, deformation is parameterized by a control grid $\mathbf{c}$. In $\mathbf{c}$, each grid point $\mathbf{u}$ stores a local Euclidean offset $\mathbf{c_u} \in \mathbb{R}^3$, and the accompanying function $C_\mathbf{c}(\mathbf{x}) = \mathbf{x} + \sum_{\mathbf{u}\in\mathbf{N_x}} w_\mathbf{u}(\mathbf{x})\mathbf{c_u}$ deforms a point $\mathbf{x} \in \mathbb{R}^3$ by applying interpolated neighbor grid offsets, where $w_\mathbf{u}(\mathbf{x})$ is the interpolation ratio. The loss function is then parameterized over $\{\mathbf{T}^j\}$ and $\mathbf{c}$ with as-rigid-as-possible regularizers:

$$\min_{\mathbf{c},\mathbf{T}} \sum_{\mathbf{p}\in\mathcal{M}^i,\mathbf{q}\in\mathcal{M}^j} \|(\mathbf{T}^i C_\mathbf{c}(\mathbf{p}) - \mathbf{T}^j C_\mathbf{c}(\mathbf{q})\|^2 + \lambda \sum_{\mathbf{u},\mathbf{v}\in\mathbf{N_u}} \|\mathbf{c_u} - \mathbf{R}_\mathbf{v}^{\mathbf{c_v}}\mathbf{u}\|^2, \tag{5.15}$$

where $\mathbf{p} \in \mathcal{M}^i, \mathbf{q} \in \mathcal{M}^j$ are corresponding 3D points between submaps obtained by nearest neighbor search. $\mathbf{R}_\mathbf{v}^{\mathbf{C_v}}$ is the rigid rotation that minimizes $\|\mathbf{R}_\mathbf{v}^{\mathbf{C_v}}(\mathbf{v} - \mathbf{u}) - (\mathbf{c_v} - \mathbf{c_u})\|$ locally, where $\mathbf{v}$ is a 1-ring neighbor $\mathbf{N_u}$ of $\mathbf{u}$. It controls local distortions in the as-rigid-as-possible regularizer.

This problem formulation is complicated to realize in code, and in the original implementation, the deformation grid is a simplified dense 3D array where points out-of-bound are discarded during optimization. As of today, SLAC has never been reproduced apart from the original implementation. We observe that similar operations for TSDF grids can be applied here by ASH to generate a spatially hashed control grid.

Setup 6. *A volumetric deformation hash map maps grid coordinates to position offsets, and the capacity of the hash map is typically* 10^3 *to* 10^4*:*

$$\mathcal{K} = \{\texttt{Tensor((3), Int32)}\},$$
$$\mathcal{V} = \{\texttt{Tensor((3), Float32)}\}.$$

Equipped with ASH, the non-rigid deformation can be written in several lines which results in a significant drop in LoC. We first voxelize the input point cloud with the deformation grid size following Listing 1. Then, instead of the 1-radius nearest neighbors (3^3 entries in Listing 3), we look for 1-cube nearest neighbors (2^3), where a point is enclosed in a cube formed by grid points. The interpolation ratio can be computed jointly. We also adapt the 1-radius neighbor search to 1-ring neighbors for the regularizer. The embedding of a submap point cloud is visualized in Fig. 5.16, where the edges indicate the association between points to grids, and the colors show the interpolation ratio. Note that this visualization is also made easy thanks to the simple interface of ASH.

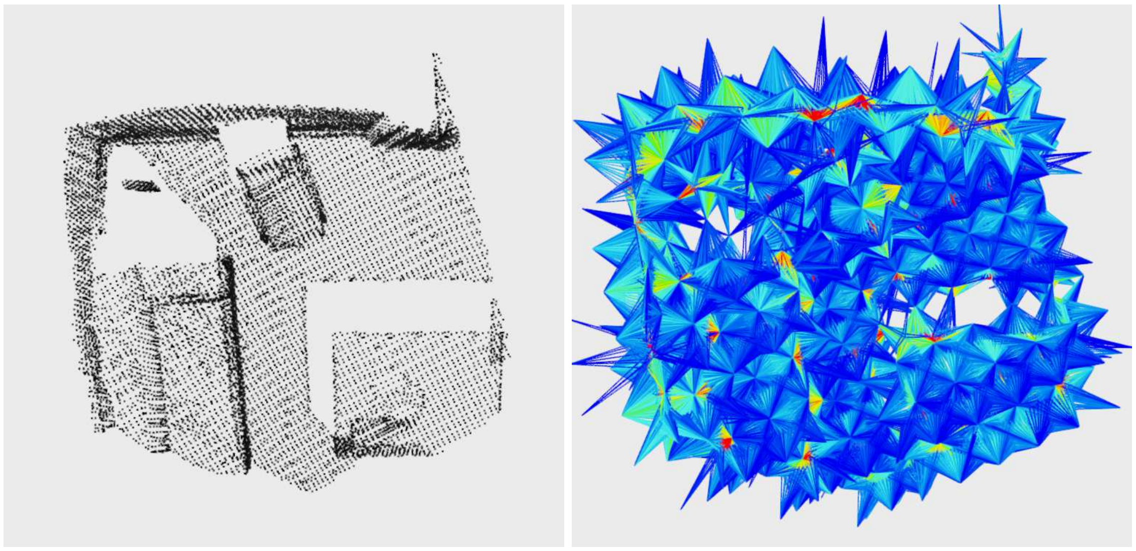


Figure 5.16: Visualization of a point cloud and its embedding in the volumetric deformation grids. Left: original point cloud. Right: embedding graph connecting the input points and associated deformation grid points. Each edge's color indicates the interpolation weight: blue shows a lower weight (closer to 0), while red shows a higher weight (closer to 1).

With this parameterization, we reproduce SLAC after rewriting the non-linear least squares solver and jointly optimizing the grid points and submap poses given the correspondences. In addition, the hash map can be saved and loaded from the disk for further processing, including deformed TSDF integration that reconstructs the scene from the deformed input depth images embedded in the grids. Experiments show that with a modularized design and a spatial hash map, we can reproduce SLAC by reducing artifacts after optimization, as shown in Fig. 5.17.

We can see a gain in performance with fewer LoC in Table 5.3 in the *livingroom 1* scene with heavy simulated noise[5]. Note while the hash map generalizes deforma-

[5]To control the experiment, we use the initial submap pose graph from the baseline implementa-

tion grids from bounded to unbounded scenes, the LoC and time contributing to the core non-linear least squares optimization are slightly reduced. Meanwhile, the deformation and integration speed *per frame* is significantly faster ($11.8\times$), which is critical for large-scale (≥ 30K frames) sequences. While being faster and easier to develop, our system achieves a higher reconstruction quality in terms of precision, recall, and F-score with a distance threshold $\tau = 20$mm [180].

Table 5.3: Performance and LoC (top) and reconstruction quality (bottom) comparison between ASH-SLAC and the original implementation [269] on the *livingroom-1* scene [46]. ASH-SLAC is faster with fewer LoC, and produces a better reconstruction.

Operation	Original SLAC		ASH-SLAC	
	Time (ms)	LoC	Time (ms)	LoC
Non-rigid optim.	2041.1	1585	**1982.1**	**1535**
Deformed integration	125.38	944	**10.62**	**446**
Reconstruction quality				
Precision (↑)	29.19		**36.10**	
Recall (↑)	51.44		**61.34**	
F-score (↑)	37.24		**45.45**	

5.6.4 Joint Geometry and Appearance Refinement

SLAC reduces artifacts for large-scale scenes. For small-scale objects, while volumetric reconstruction outputs smooth surfaces, fine details are often impaired due to the weight averaging of the TSDF.

Shape-from-Shading (SfS) refines details by jointly optimizing volumetric TSDF functions given the initial geometry and appearance [276]. It takes a reconstructed volumetric TSDF grid $\mathbf{d}^0$ with a set of high-resolution key frame RGB images I^j and their poses $\mathbf{T}^j$ as input, and outputs jointly optimized TSDF $\mathbf{d}$ and albedo a

tion.

through an image formation model

$$\min_{\mathbf{a},\mathbf{d}} \sum_{\mathbf{x},j} \|\nabla B(\mathbf{x}) - \nabla I^j(\Pi(\mathbf{T}^{j^{-1}}\mathbf{x}'))\|^2 + \lambda_{\text{smooth}} \sum_{\mathbf{x}} \|\Delta \mathbf{d}_{\mathbf{x}}\|^2 + \lambda_{\text{init}} \sum_{\mathbf{x}} \|\mathbf{d}_{\mathbf{x}} - \mathbf{d}^0_{\mathbf{x}}\|^2$$
$$+ \lambda_{\text{chrome}} \sum_{\mathbf{x},\mathbf{y} \in \mathbf{N}_{\mathbf{x}}} w(\mathbf{x},\mathbf{y}) \|\mathbf{a}_{\mathbf{x}} - \mathbf{a}_{\mathbf{y}}\|^2, \tag{5.16}$$

where the estimated voxel-wise appearance is computed by $B(\mathbf{x}) = \mathbf{a}_{\mathbf{x}}\text{SH}(\mathbf{n}_{\mathbf{x}})$ (SH stands for spherical harmonics), and associated with the closest surface point

$$\mathbf{x}' = \mathbf{x} - \mathbf{d}_{\mathbf{x}}\mathbf{n}_{\mathbf{x}}, \ \ \mathbf{n}_{\mathbf{x}} = \frac{\nabla_{\mathbf{x}}(\mathbf{d})}{\|\nabla_{\mathbf{x}}(\mathbf{d})\|}, \tag{5.17}$$

which is projected to image I^j through Π after a rigid transformation $\mathbf{T}^{j^{-1}}$. Here the voxel-wise gradient is directly derived from $\mathbf{d}$ with a finite difference

$$\nabla_{\mathbf{x}}(\mathbf{d}) = \frac{\mathbf{d}_{\mathbf{x}+\delta} - \mathbf{d}_{\mathbf{x}-\delta}}{2\delta}. \tag{5.18}$$

Similar to SLAC, we use $\mathbf{d}_{\mathbf{x}}, \mathbf{a}_{\mathbf{x}}$ to access TSDF and albedo values at grid point $\mathbf{x} \in \mathbb{R}^3$. $\lambda_{\text{smooth}}, \lambda_{\text{init}}, \lambda_{\text{chrome}}$ are coefficients for regularizing smoothness through the Laplacian, stability, and piece-wise albedo constancy via a weighted chromaticity regularizer $w(\mathbf{x},\mathbf{y})$, respectively [276].

While the image formation model is straightforward, similar to SLAC, the underlying data structure used in implementing the model can be complex mainly because of the prevalent nearest neighbor search in normal computation and neighbor voxel regularizers. As a result, to enable such a system without a modern hash map, one has to rely on low-level C++ implementation and is consequently limited to the low-level Ceres solver [2] for autodiff in optimization. Further, the spatially hashed voxels have to be bounded to reduce computation cost [150].

Now equipped with ASH, we provide a simplified solution that is built upon the hash map and advanced indexing. Unlike SLAC which requires time-consuming deformable TSDF re-integration for final scene reconstruction, SfS allows reusing accelerated surface extraction from the TSDF grids without further optimization. Therefore, we implement the SfS pipeline in *pure Python* as an example of fast prototyping of a differentiable rendering pipeline. Running on GPU, we lift the constraint of a user-defined bounding box and optimize the full reconstructed surface.

Without the requirement of extreme performance, we drop the hierarchical volumetric layout and use the simple voxel-based hash map:

Setup 7. *A voxel indexer is given by a hash set* $\mathcal{K} = \{\texttt{Tensor((3), Int32)}\}$. *The typical capacity is* 10^9 *to* 10^{10}.

With this setup, we can reuse the code in SLAC to look up the 1-ring neighbors for normal estimation and Laplacian regularization. There is, however, another lookup required since we are minimizing the difference of appearance *gradient* in Eq. 5.16: we need to find the 1-ring neighbors that also *have* 1-ring neighbors. In other words, we have to find the intersection of two sets. While `NumPy` provides the functionality for 1D arrays through *ordered* sorting, our hash map allows *unordered* intersection that can be generalized to multi-dimensional inputs:

Setup 8. *With two input sets* $\mathbf{k}_1 \subset \mathcal{K}, \mathbf{k}_2 \subset \mathcal{K}$, *the intersection* $\mathbf{k}_1 \bigcap \mathbf{k}_2$ *is given by the following operations: initialize a hash set with* $\mathbf{k}_1$; *query* $\mathbf{k}_2$ *and obtain success mask* $\boldsymbol{\theta}$; *return* $\mathbf{k}_2(\boldsymbol{\theta})$.

After data association is found and SH parameters are estimated in a preprocessing step, all the terms in Eq. 5.16 are converted to a trivial combination of indexing and arithmetic operations. We can take advantage of PyTorch's autodiff, and backpropagate the gradient through the built-in differentiable index layer. ADAM [126] with an initial learning rate 10^{-3} is used. Thus the core volumetric SfS pipeline [276] is reproduced in pure Python.

An extension can be easily implemented by introducing spatially varying lights [150], wrapped up with a hash map.

Setup 9. *Spatially varying spherical harmonics (SVSH) (bands = 3) can be described by a hash map that maps lighting subvolume coordinates to the corresponding coefficients:*

$$\mathcal{K} = \{\texttt{Tensor((3), Int32)}\},$$
$$\mathcal{V} = \{\texttt{Tensor((9), Float32)}\}.$$

The embedding of an active voxel in an SVSH map is identical to SLAC, with 1-cube neighbors for the data term and 1-ring neighbors for the regularizer. Further description is omitted here as the formulation and implementation are similar to

Eq. 5.16 [150].

Having both SfS and SVSH optimization implemented[6] [150], we show the results on the scene *lion* in Fig. 5.18. Without voxel grid upsampling, both the geometry and appearance details are sharper. Regarding performance and code complexity, we show in Table 5.4 that our code is much shorter in pure Python, and $150\times$ faster per iteration thanks to the CUDA autodiff engine in PyTorch. Note that Ceres is a 2nd-order optimizer on CPU that empirically converges faster than the 1st-order ADAM optimizer. In practice, however, we found that in 50 iterations ADAM converges well against the preset 10 iterations for the non-linear least squares solver. Thus the total optimization performance of our implementation is still $30\times$ faster with more voxels to process (remember that we do not require an additional bounding box).

We also evaluate reconstruction quality in Table 5.4. We render our optimized mesh given the keyframe camera extrinsic and intrinsic parameters and compute RMSE against the raw input images. For the baseline [150], we follow a similar procedure and render the optimized mesh (not upsampled for fairness) given refined camera parameters. We use the same mask given by the baseline to ensure the same region of interest. The results show that our implementation produces improved RMSE despite the simplified development.

Table 5.4: Performance per epoch and LoC (top), and rendering quality (bottom) comparison between ASH-Intrinsic3D and the original implementation [150] on the *lion* scene [150]. ASH-Intrinsic3D is faster with fewer LoC, and results in comparable rendering from the refined reconstruction.

Operation	Original-Intrinsic3D		ASH-Intrinsic3D	
	Time (s)	LoC	Time (s)	LoC
SVSH optim.	0.503	605	**0.092**	**254**
Joint optim.	147.323	7399	**0.916**	**1416**
Rendering quality				
RMSE mean (↓)	0.677		**0.627**	
RMSE std (↓)	**0.095**		0.120	

[6]Pose optimization and voxel upsampling are disabled at current.

5.7 Conclusions

We presented ASH, a performant and easy-to-use framework for spatial hashing. Both synthetic and real-world experiments demonstrate the power of the framework. With ASH, users can achieve the same or better performance in 3D perception tasks while writing less code.

There are various avenues for future work. At the architecture level, we seek to introduce the open address variation [4, 117] of parallel hash maps for flexibility and potential high performance static hash maps. At the low level, we plan to further optimize the GPU backend, and accelerate the CPU counterpart, potentially with cache level optimization and code generation [109, 112]. We also plan to apply ASH to sparse convolution [49, 228] and neural rendering [79, 190], where spatially varying parameterizations are exploited.

ASH accelerates a variety of 3D perception workloads. We hope that the presented framework will serve both research and production applications.

Acknowledgement

We thank Patrick Stotko for adapting stdgpu to our framework, Qian-Yi Zhou for the valuable comments and detailed math derivations that helped us implement SLAC, and Kwonyoung Ryu and Jaesik Park for providing the imaging LiDAR dataset.

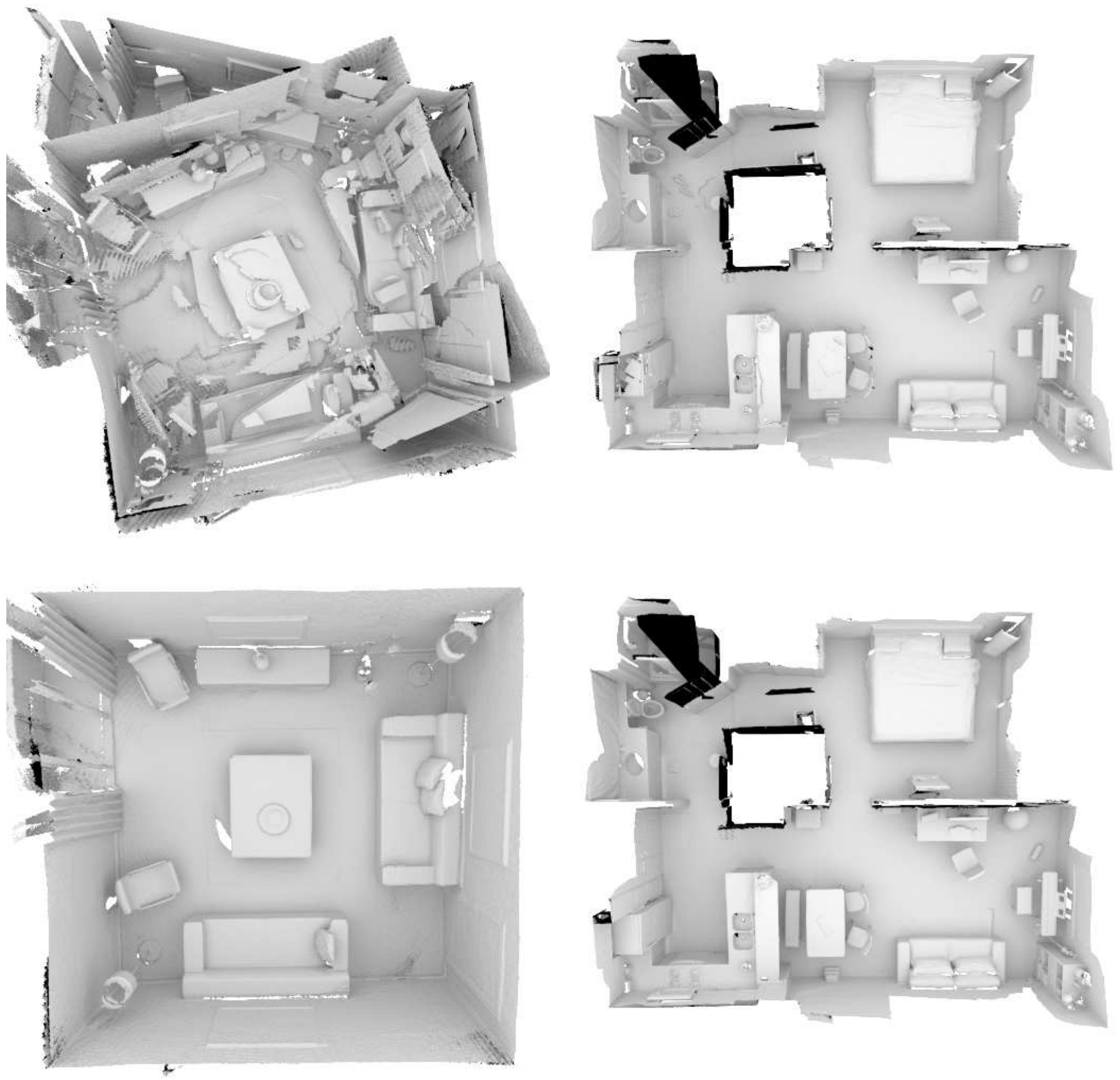
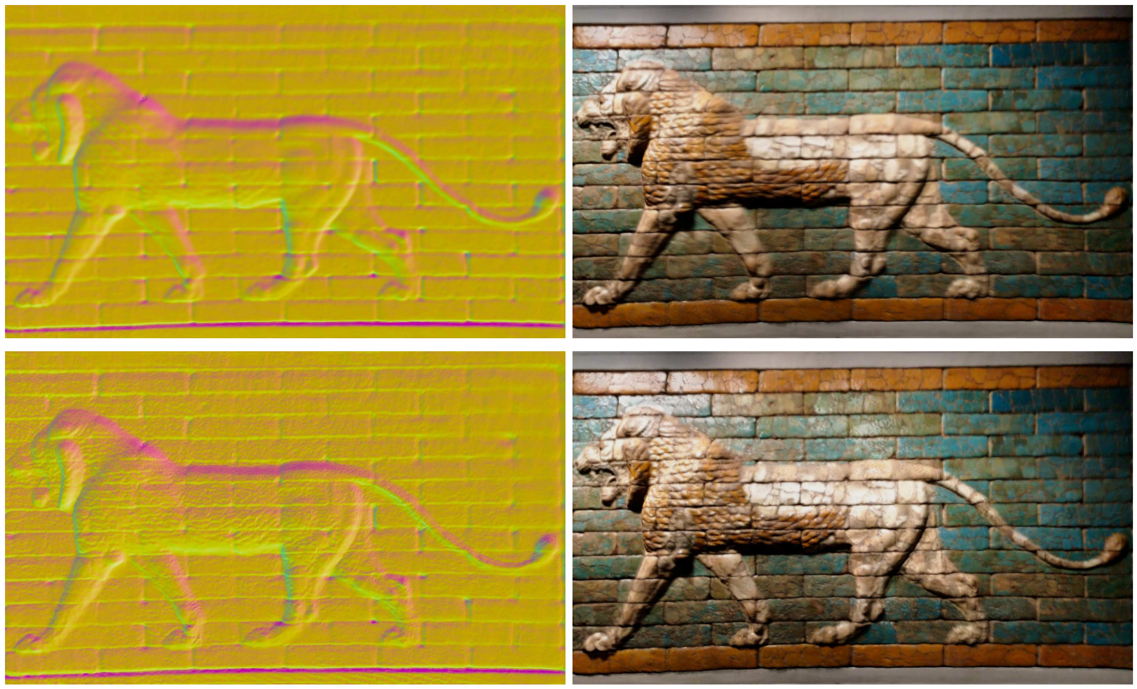


Figure 5.17: Visualization of scene reconstructions before and after ASH-SLAC. First row: before ASH-SLAC. Second row: after ASH-SLAC. Left: *livingroom-1* from Augmented ICL [46]. Right: *apartment* from Indoor LiDAR RGBD [180]. Artifacts are eliminated by global pose adjustment and local deformation via deformable TSDF integration. Rendered with Mitsuba 2 [172].



(a) Normal map (b) Color map

Figure 5.18: Appearance and geometry refinement before and after ASH-Intrinsic3D on *lion* [276]. First row: initial reconstruction from volumetric integration. Second row: refined reconstruction after optimization.

Chapter 6

Revisiting LiDAR Registration and Reconstruction

6.1 Introduction

LiDAR scanners are prevalent sensors used to obtain range data and provide 3D geometry by measuring the time of flight of modulated laser pulses. Compared with camera-like solid-state LiDARs, spinning LiDARs capture full 360° views, thus they are widely applicable to robotics, remote sensing, and autonomous driving. Popular spinning LiDARs such as Velodyne [98] and Ouster [176] are designed in a similar fashion: a line of scan is measured vertically; a complete 360° scan is formed by horizontally spinning the sensor to accumulate line scans in a consistent coordinate system.

It is clear that intrinsic geometric transformations exist in the conversion from raw scans to a 3D point cloud, consisting of spherical projective and rigid transformations. Yet the value of low-level conversions are down-weighted for convenience, and many hardware drivers and downstream datasets [82, 96, 218] only provide *3D point clouds* to the user. There is an advantage of the design, since that prevalent 3D data format is acceptable to most 3D processing pipelines. However, the intrinsic relations between the scanned points are discarded, and k-d trees [20] have to be constructed to find nearest neighbors in the Euclidean 3D

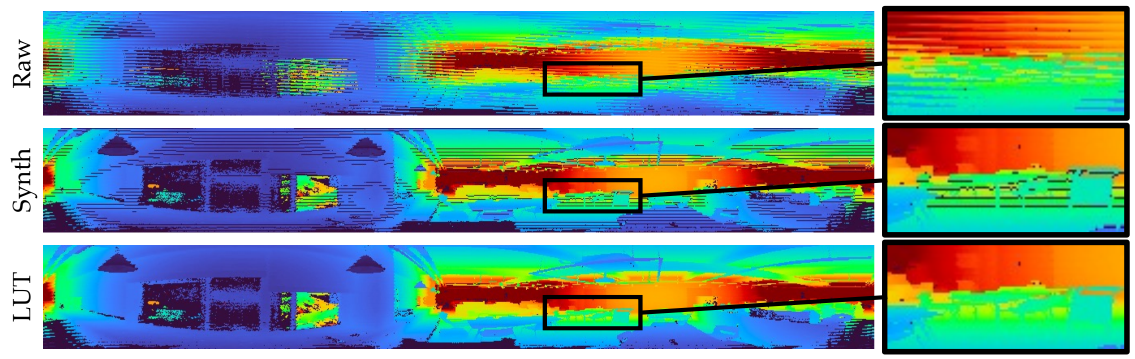


Figure 6.1: Visualization of a LiDAR scan as a cylindrical range image in various forms. Synthetically projecting a point cloud to a cylindrical image [19, 206] results in artifacts (middle) due to inaccurate altitude mapping. The cylindrical image view (bottom) of raw scans (top) with a lookup table (LUT) is loseless.

space, which require highly optimized implementation for real-time systems such as LiDAR odometry (LO) and simultaneous localization and mapping (SLAM). Recent studies [19, 206] generate proxy 2D range images from point clouds via synthetic projections to accelerate neighbor search, reducing the query complexity from $O(N \log N)$ to $O(N)$ for a point cloud of size N with the drop of a k-d tree. Yet further advantages of the image representation, from fast down-sampling to signed distance computation, are not well-studied; a loss of data quality is also inevitable due to synthetic projection, as shown in Fig. 6.1.

A cylindrical image view of spinning LiDAR's *raw scan lines*, on the other hand, is efficient without losing the data quality against its geometry-equivalent point cloud. By nature, it supports fast projective data association [237] and neighbor search in images. Therefore, direct visual odometry [168, 237] and signed distance function (SDF) reconstruction [56, 168] are applicable.

In this paper, we revisit the LiDAR[1] data formulation, and propose the range image-based representation shown in Fig. 6.1. Our contributions can be summarized as:

- A cylindrical image view of LiDAR data directly from the scan lines along with an intrinsic spherical projective model that supports accurate conversions between 2D and 3D;

[1]Without confusion, we regard camera-like solid-state LiDARs as Depth sensors in contrast to spinning LiDARs. The term *LiDAR* specifically denotes spinning LiDARs in the rest of the paper.

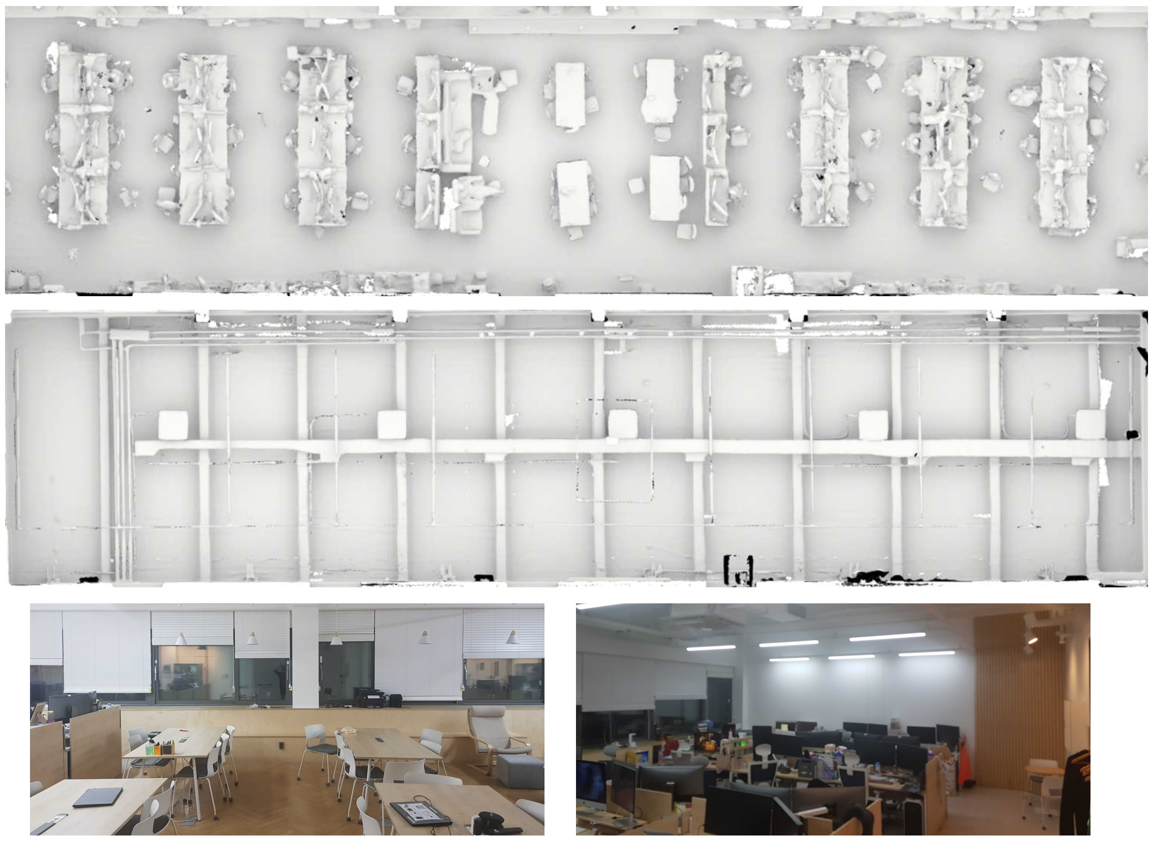


Figure 6.2: Surface reconstruction via SDF integration of the *lab* sequence from LiDAR range images. Top and bottom left are the ground and the ceiling rendered with Mitsuba 2 [172]. Bottom right are pictures of the scene.

- Fast and effective multi-scale registration and scalable SDF reconstruction for range LiDAR images, accelerated on GPU. These operations are backward compatible to synthesized range images from point clouds, *e.g.*, KITTI [82];
- A new collection of LiDAR range image sequences of both indoor and outdoor scenes with pseudo ground truth poses, along with comprehensive evaluations on the task of registration and surface reconstruction.

6.2 Related Work

Representation. LiDAR data are generally viewed as point clouds, an unordered set of 3D points, the major format in prevalent LiDAR datasets [82, 96, 218]. Several datasets and systems [19, 23, 31, 40, 43, 82, 206] project point clouds to the cylindrical image space and synthesize range images, but the data distribution is sparse with significant artifacts. In RGB-D cameras, however, 3D point clouds are densely packed as 2D images [46, 57, 180, 213], also known as organized point clouds [196]. The major difference comes from the hardware. LiDARs rely on the rotation of a line scanner, hence the output is more likely to be interpreted as an unordered set, whereas RGB-D scanners use structured sensors that by nature capture images. An image formation allows efficient operations by indexing with coordinates, while a point cloud requires trees [20, 156] or spatially hashed voxels [171] to enable fast accessing by location.

Registration. Point cloud registration is a well-studied topic that aligns two point sets with a known initial pose. In the point cloud format, variations of iterative closest points (ICP) are classical solutions [22, 180, 195, 205]. These methods depend on nearest neighbor search in 3D using trees [20], which is the bottleneck of the performance. Learning-based algorithms [11, 51, 52, 137, 233] seek to avoid nearest neighbor search via deep feature matching and/or the weighted Procrustes solver, but in practice require even more computation resources. In the range image form, projective nearest neighbor is used instead to circumvent the 3D nearest neighbor search [123, 168, 237]. This formulation is introduced to LiDAR data [19, 206] by synthetically projecting point clouds to cylindrical images.

Surface reconstruction. Conventional LiDAR reconstruction uses occupancy grids [104, 108], where the space is coarsely divided into grids recording the occupancy probability. While it preserves the coarse 3D geometry, a dense surface reconstruction is often not applicable. Several surfel based dense reconstruction algorithms exist for LiDARs [19, 179], but they are hardware or system dependent and cannot be easily generalized; time-consuming triangulation is required to generate a mesh. Truncated SDF (TSDF) reconstruction [161, 175, 193] has been adapted to LiDARs, but still relies on the point cloud representation with point-ray tracing, thus an adaptation to GPU is non-trivial due to the race conditions at ray intersections. Surface reconstruction using depth images for RGB-D sensors is more flexible due to the calibrated pinhole camera model. In addition to surfel-based reconstruction [122, 237], dense volumetric TSDF reconstruction produces water-tight surfaces for medium to large scale scenes [46, 65, 168, 171, 180] and can function alone given pose and depth image inputs.

In this paper we represent *raw scans* of LiDARs as cylindrical range images along with projective LiDAR intrinsics. We then propose efficient approaches for LiDAR range image based registration and reconstruction, accelerated on GPU. Due to the simplicity of the formulation, while retaining a similar accuracy, our approach is 15–50$\times$ faster in surface reconstruction, and 5–150$\times$ faster in registration.

6.3 Representation

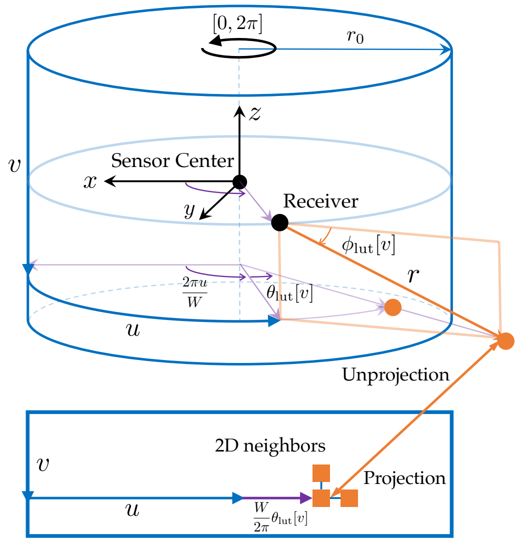


Figure 6.3: Illustration of the projection/unprojection procedure of a spinning LiDAR. Using the spherical projection model given the sensor intrinsics, 3D operations can be constrained on 2D range images with the routine of image processing.

LiDAR scanners of our interest complete scans by rotation. A fixed number (H) of points are scanned roughly in a vertical line (corresponding to elevations) through aligned laser rays, and an accumulation of W such lines form a complete scan spanning horizontally from $0°$ to $360°$. Therefore, a $H \times W$ range image can be naturally formed, where each pixel stores a range scalar associated to a ray.

However, direct use of the raw LiDAR range map is not desirable. As shown in Fig. 6.1, the interlacing artifacts occur due to the local ray offset of each scan line. Hence, we need to adopt an azimuth intrinsic look-up table (LUT) $\boldsymbol{\theta}_{\text{lut}}$ provided by the manufacturer to compensate the offset. Similarly, a nonlinear elevation distribution associated with rays is defined by hardware design, in the form of another LUT $\boldsymbol{\phi}_{\text{lut}}$. Given the ray-range image representation and the LUTs, we now analyze the spherical projection $\Pi : \mathbb{R}^3 \to S(3)$ and unprojection $\Pi^{-1} : \mathbb{R}^2 \times \Omega \to \mathbb{R}^3$ functions defined on the range image Ω. We use (u, v) to indicate a pixel coordinate, $r = \Omega(u, v)$ for the range reading, and (x, y, z) for the corresponding 3D coordinate.

6.3.1 Unprojection Π^{-1}

A spinning LiDAR's receiver in charge of range sensing is located on a cylinder of radius r_0 enclosing the sensor center, see Fig. 6.3. Therefore, the unprojection is a combination of the receiver's location on the cylinder, and a spherical transform of a ray centered at the receiver:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \Pi^{-1}(\begin{bmatrix} u \\ v \\ r \end{bmatrix}) = \begin{bmatrix} r\cos\theta(u,v)\cos\phi(v) + r_0\cos\frac{2\pi u}{W} \\ r\sin\theta(u,v)\cos\phi(v) + r_0\sin\frac{2\pi u}{W} \\ r\sin\phi(v) \end{bmatrix}, \tag{6.1}$$

$$\theta(u,v) = \frac{2\pi u}{W} + \boldsymbol{\theta}_{\text{lut}}[v], \quad \phi(v) = \boldsymbol{\phi}_{\text{lut}}[v], \tag{6.2}$$

where $\theta(u, v)$ converts the column index u to the azimuth with a linear transform by the ray's horizontal offset $\boldsymbol{\theta}_{\text{lut}}[v]$. $\phi(v)$ directly reads the elevation from $\boldsymbol{\phi}_{\text{lut}}$. As the LUTs and image size are predefined, pixel-wise LUTs can be further constructed by reorganizing Eq. 6.1 as a pixel-wise linear function of r.

6.3.2 Projection Π

While the unprojection model is straightforward, its inversion is not due to the receiver offset and the non-parametric LUTs. Assuming $r_0 \ll r$ in Eq. 6.1, we obtain an approximation:

$$\begin{bmatrix} r \\ \phi \\ \theta \end{bmatrix} \approx \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \arcsin\frac{z}{r} \\ \arctan\frac{y}{x} \end{bmatrix}, \tag{6.3}$$

then get $\hat{u} = \frac{W}{2\pi}\theta$[2] by temporarily omitting the offset $\boldsymbol{\theta}_{\text{lut}}[v]$. We then compensate x and y from the r_0 offset in Eq. 6.1 with approximated $\hat{u}$ and repeat the estimate until convergence. With the known LUT $\boldsymbol{\phi}_{\text{lut}}$, we then search v by

$$v = \underset{t \in \{0,1,\cdots,H-1\}}{\arg\min} \left\| \boldsymbol{\phi}_{\text{lut}}[t] - \phi \right\|. \tag{6.4}$$

To speed up the process, we construct an inverse LUT $\boldsymbol{\phi}_{\text{lut}}^{-1}$ with a predefined resolution, and apply $v = \boldsymbol{\phi}_{\text{lut}}^{-1}(\phi)$. In practice, for a $\boldsymbol{\phi}_{\text{lut}}$ with H entries, a $\boldsymbol{\phi}_{\text{lut}}^{-1}$ with $2H$ entries ensures the elevation index error bounded by ± 1. Given the estimated v, we finally obtain u by reverting Eq. 6.2: $u = \hat{u} - \frac{W}{2\pi}\boldsymbol{\theta}_{\text{lut}}[v]$. A chain of aforementioned operations form the imperative projection function $\begin{bmatrix} u, v, r \end{bmatrix}^\top = \Pi(\begin{bmatrix} x, y, z \end{bmatrix}^\top)$. To our best knowledge, the consideration of pixel-wise ray offset has not been presented so far in previous LiDAR unprojection and projection on range images. Note that for the popular LiDAR dataset presented in unstructured point clouds without intrinsics, such as KITTI [82], our model reduces to synthetic projection [19] with $\boldsymbol{\theta}_{\text{lut}}(\cdot) = 0$ and $\boldsymbol{\phi}_{\text{lut}}(\phi) = H \cdot \frac{\phi_{\max} - \phi}{\phi_{\max} - \phi_{\min}}$, where $(\phi_{\min}, \phi_{\max})$ indicate the sensor's field of view.

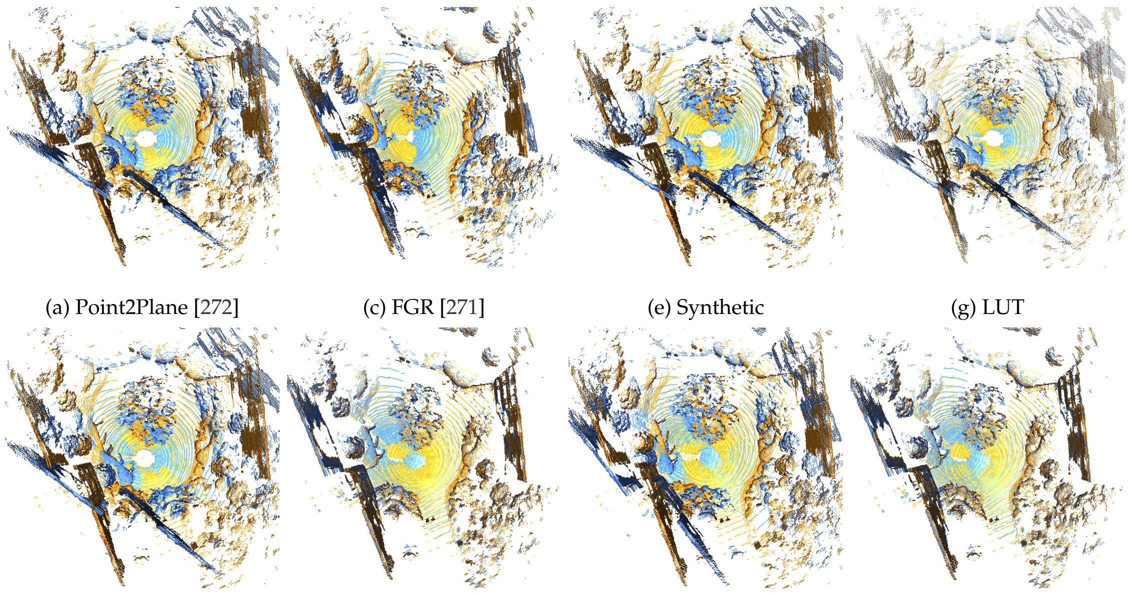
6.4 Registration and Surface Reconstruction

6.4.1 Multi-scale Cylindrical Range Image Registration

We now register two LiDAR scans through index-based projective data association. We take the source scan in the point cloud form $\mathbf{p} \in \mathbf{P}_{\text{src}}$ (by applying unprojection), and the target scan in the range image form Ω_{dst}.

With an initial transformation $\mathbf{R}_k \in SO(3), \mathbf{t}_k \in \mathbf{R}^3$ (typically $\mathbf{R}_k$ initialized to identity and $\mathbf{t}_k$ estimated by aligning two point set centers), we get the associated

[2] A warp to $[0, 2\pi]$ from $[-\pi, \pi]$ is required when using `arctan2`.



(a) Point2Plane [272] (c) FGR [271] (e) Synthetic (g) LUT

(b) G-ICP [205] (d) RANSAC [198] (f) Synthetic-Multi (h) LUT-Multi

Figure 6.4: Illustration of registration of a challenging pair in the *dormitory* sequence. While point cloud based ICP variants fail, multi-scale projective range image registration has a better convergence, and achieves comparable performance to global registration methods with known intrinsic LUTs.

point cloud $\mathbf{q} \in \mathbf{Q}_{\text{dst}}$ by[3]

$$[u, v, r]^\top = \Pi(\mathbf{R}_k \mathbf{p} + \mathbf{t}_k), \tag{6.5}$$

$$\mathbf{q} = \Pi^{-1}\Big(u, v, \Omega_{\text{dst}}(u, v)\Big), \tag{6.6}$$

where u, v, r are pixel coordinates and range, and Ω_{dst} reads the range measurements at (u, v). These operations can be easily vectorized and run in parallel. Denote the correspondence set with $\mathcal{C} = \{(\mathbf{p}_i, \mathbf{q}_j) \mid \mathbf{p}_i \in \mathbf{P}_{\text{src}},\ \mathbf{q}_j \in \mathbf{Q}_{\text{dst}}\}$, we have the nonlinear least squares estimate using Gauss-Newton from

$$\mathbf{R}_{k+1}, \mathbf{t}_{k+1} = \arg\min_{\mathbf{R},\mathbf{t}} \sum_{\mathbf{p}_i, \mathbf{q}_j \in \mathcal{C}} \rho\Big(\mathcal{L}(\mathbf{R}\mathbf{p}_i + \mathbf{t}, \mathbf{q}_j)\Big), \tag{6.7}$$

where $\mathcal{L}$ is the point-to-plane loss $\mathcal{L}(\mathbf{x}, \mathbf{y}) = \mathbf{n}_{\mathbf{y}}^\top(\mathbf{x} - \mathbf{y})$ given the normal $\mathbf{n}_{\mathbf{y}}$, attached

[3] Image boundary check is ignored for clarity. Same for SDF reconstruction.

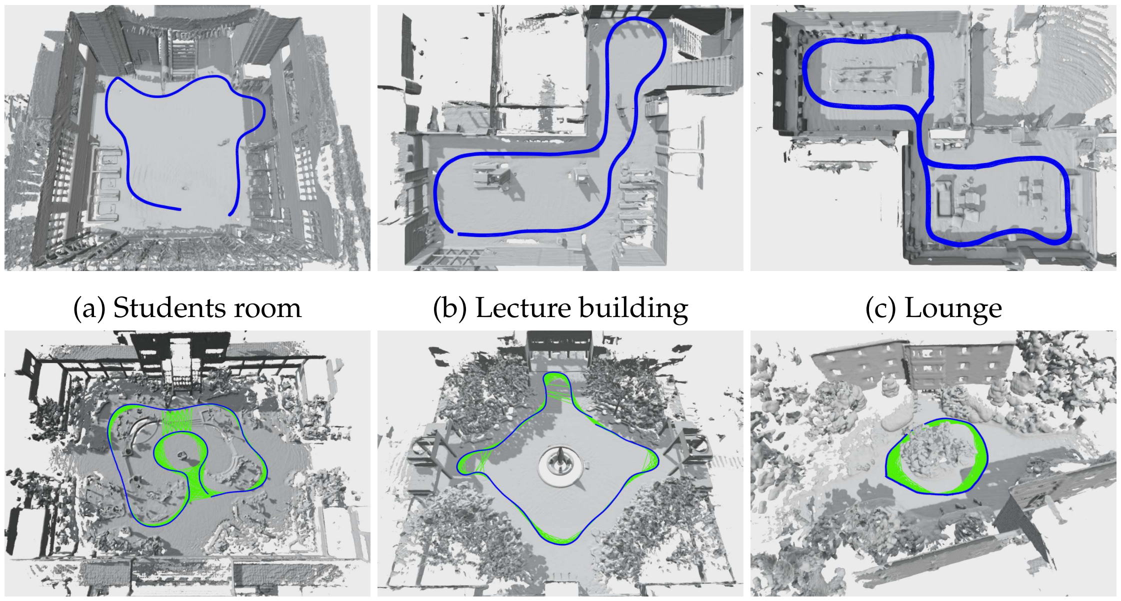
with a robust kernel ρ [14]. The normal image can be efficiently constructed by eigenvalue decomposition of nearest neighbors in a searching window, or simply a cross product of two neighbor pixels [19]. Iterating Eqs. 6.5-6.7 constructs the range image based registration algorithm. Implementation-wise, the projective data assciation discards the use of a k-d tree that requires $O(N \log N)$ construction and query time in two passes, therefore the $O(N)$ correspondence search and linear system construction can be finished in one pass in parallel.

While the cylindrical range image has a wide receptive field in the horizontal direction, putative correspondences $\mathcal{C}$ are still limited. In view of this, we propose *multi-scale registration* for the task. A range image pyramid is constructed by accessing strided range and normal images, retaining the original LiDAR intrinsics. Projective transforms are performed at the finest level, but down-sampled on coordinates at the given stride. Compared to point clouds, image-based downsampling takes no time by only changing the strides in the projection model, and does not need voxelization of point clouds that requires the $O(N)$ construction of a spatial hash map.

As a result, multi-scale registration for cylindrical images significantly boosts fidelity of registration, and it lifts the local registration algorithm in the ICP-fashion to be comparable to global registration approaches such as RANSAC. Fig. 6.4 and Fig. 6.6 show registration examples.

6.4.2 Signed Distance Function from LiDAR Range Images

One of the key benefits of using range images for LiDARs is that we can naturally apply parallel SDF estimation for dense surface reconstruction. SDF measures the distance from an arbitrary query point to its nearest surface. With a perfect watertight mesh model, signed distance per point can be computed via ray casting [217]. In real world with accumulating data, such computation is intractable especially for online usages. For LiDAR data, a common practice is to cast rays from sensor origin to scan points and update samples along the ray. This formation, however, limits the sampling distribution, and is not friendly to parallel computation due to race conditions at ray intersections. Classical volumetric reconstruction [56, 168, 171] projects arbitrary 3D points to depth images and computes weight average of


(a) Students room (b) Lecture building (c) Lounge

(d) Square (e) Fountain (f) Dormitory

Figure 6.5: Surface reconstruction via SDF integration of selected sequences from our dataset, overlaid with sensor trajectory (poses in blue, loop closures in green). Top: indoor scenes. Bottom: outdoor scenes. Loop closures for indoor scenes are omitted to avoid occlusion of geometry details.

truncated projective SDF to approximate the real SDF. It requires a range image and a projection model where our representation fits.

To estimate the projective SDF from a query point $\mathbf{x} \in \mathbb{R}^3$, we find its projective association in a range image Ω_j with pose $\mathbf{R}_j \in SO(3), \mathbf{t}_j \in \mathbb{R}^3$, and estimate the signed distance $d_j(\mathbf{x})$ along the projection ray:

$$\begin{bmatrix} u & v & r \end{bmatrix}^\top = \Pi(\mathbf{R}_j \mathbf{x} + \mathbf{t}_j), \tag{6.8}$$

$$d_j(\mathbf{x}) = \Omega_j(u, v) - r. \tag{6.9}$$

With a sequence of LiDAR range measurements $\{\Omega_{j=1}^N\}$ and their associated poses, we can get a least squares estimate at query points, typically at discretized voxel grid points:

$$d(\mathbf{x}) = \arg\min_t \|t - d_j(\mathbf{x})\|^2 = \frac{\sum d_j}{N}, \tag{6.10}$$

which can be updated incrementally [168]. While the formulation still holds when using LiDAR projective model, LiDAR has wide range, therefore a dense grid does not scale to LiDAR range images. In this regard, we use ASH [67] to generate a globally sparse locally dense hash grid for unbounded scene reconstruction. For each point unprojected from a range image, we activate dense voxel blocks in the shape of 16^3 within a certain radius; only the SDF value of activated voxel blocks in the cylindrical viewing volume will be updated. Accelerated Marching Cubes [63, 146] is applied to extract a triangle mesh at zero-crossing isosurfaces.

This approximate SDF computation at arbitrary $\mathbf{x} \in \mathbb{R}^3$ also opens the door to the online training of the neural SDF [163, 217] with incremental LiDAR inputs, where a multi-layer perceptron (MLP) is trained to predict SDF value at continuous sampled positions with SDF readings. We leave a full adaptation of neural SDF and surface reconstruction to range images as future work.

6.5 Dataset

There has been a plethora of LiDAR datasets [82, 96, 218], but most of them, if not all, are presented in point clouds. We therefore construct a new dataset in the range image format to fix this absence.

Data Collection.

We collect various indoor and outdoor sequences with an Ouster OS0 128 LiDAR. The selection of Ouster is the result of its user-friendly access to raw scans; an adaptation to Velodyne is also possible with low-level driver modifications. The LiDAR is placed on a portable cart, see supplementary for details. A cart is a good trade-off between flexibility and stability. It provides a stable platform that reduces vibration comparing to a hand-held setup, and is akin to the most prevalent vehicle-top setup but more flexible and works indoor. The easy-to-control motion pattern enriches registration patterns and improves scene coverage for surface reconstruction, in comparison to vehicle-top setups. The outdoor sequences are collected on campus, varying from squares to dormitories. The indoor sequences are collected in buildings, ranging from halls to lecture rooms. All the sequences are captured in the 128×1024 resolution at 10 Hz. The sequence names are listed in Table. 6.1 and their detailed statistics are in supplementary.

Pseudo Groundtruth Pose Generation.

To acquire poses of the range images without an available large scale motion capture system, we utilize a modified multiway registration system [46] based on Generalized ICP (G-ICP) [205], 3-pt FPFH-RANSAC [75], and robust pose graph optimization. This setup of pseudo ground truth pose generation is common in the RGB-D datasets [58, 247, 260] and has been widely used in the vision community.

For each sequence, we first apply G-ICP between adjacent frames to obtain odometry measurements and build an initial pose graph. We then select key frames every $K = 10$ frames, and exhaustively apply RANSAC (max 1M iterations with confidence 0.999) between key frames. Valid global registration results are refined with G-ICP, and inserted into the pose graph as loop closure edges. Finally, the pose graph is optimized with a robust line process [46] to filter inconsistent edges and output poses. The LiDAR scans per sequence are accumulated as a pseudo-ground truth 3D point cloud for reconstruction evaluation.

6.6 Experiments

6.6.1 Baselines and Experimental Setups

Registration.

We denote our approach with *LUT* and *LUT-Multi*, when intrinsic LUTs are available, and their simplified versions [19, 206] denoted by *Synthetic* and *Synthetic-Multi* with synthetic intrinsics. We select point-to-plane (*Pt2Pl*) and *G-ICP* [205] as ICP-variant baselines, and fast global registration (FGR) [271], RANSAC [198] as global registration baselines. We also compare against deep global registration (DGR) [51] pretrained on KITTI, one of the state-of-the-art learning-based registration approaches. In all experiments, we run 50 iterations for ICP variants and single-scale projective registration, {20, 20, 10} iterations for 3-level multi-scale registration, 1M iterations for RANSAC, and the default 64 iterations for FGR. For a controlled

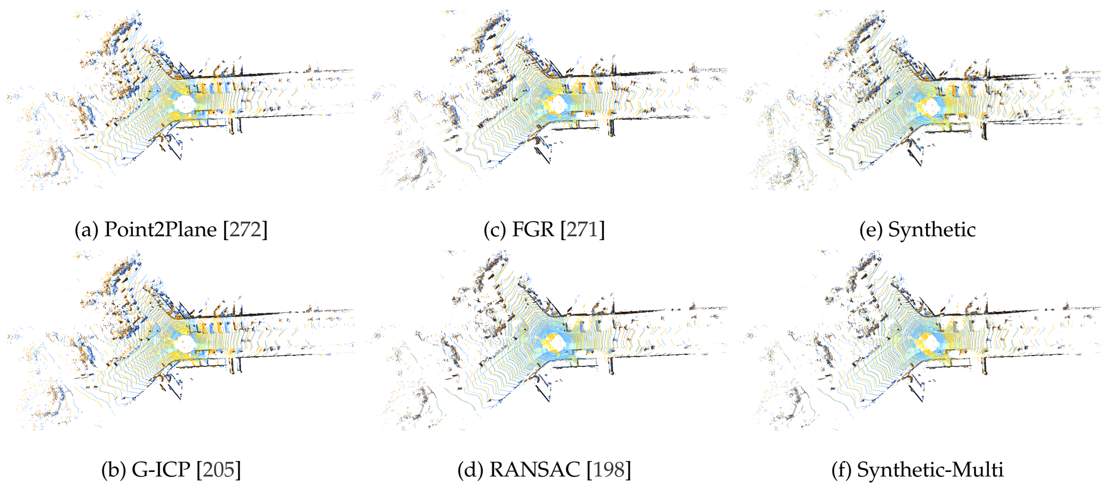


Figure 6.6: Illustration of registration results on KITTI. While ICP variants converge to inaccurate transformations, projective registration with synthetic intrinsics results in better estimates, and can be further refined by multi-scale registration.

comparison, we estimate normals in the point cloud form with radius nearest neighbor search, but organize them in the image domain. An accelerated computation of normal map directly from range image [168] is also available. We conduct experiments on real-world sequences with enumerated frame distances, defined by $|j-i|$ for frame i and j. A larger frame distance indicates a more challenging registration task. We use rotation error $e(\mathbf{R}, \mathbf{R}_{\text{gt}}) = \arccos \frac{\mathbf{R}\mathbf{R}_{\text{gt}}^\top - 1}{2}$ and translation error $e(\mathbf{t}, \mathbf{t}_{\text{gt}}) = \|\mathbf{t} - \mathbf{t}_{\text{gt}}\|^2$ as the evaluation metric. At each frame distance, we sample $M = 50$ pairs and compute the errors. The distance threshold, serving as the radius for neighbor search in baselines, and the robust psuedo-Huber kernel size for our approaches, is 0.5m for outdoor scenes (KITTI and our dataset), and 0.2m for indoor scenes (our dataset).

Surface reconstruction.

We also compare our surface reconstruction module against volumetric reconstruction pipelines that supports LiDAR data, namely voxblox [175] (outputs triangle meshes) and Octomap [108] (outputs point clouds). For evaluation, we use F-score computed by $F = \frac{2\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$, where precision defines the percentage of points in the reconstruction with valid correspondences in the GT point cloud, and recall is the opposite. Unless mentioned, we use 0.1m as the voxel size for outdoor

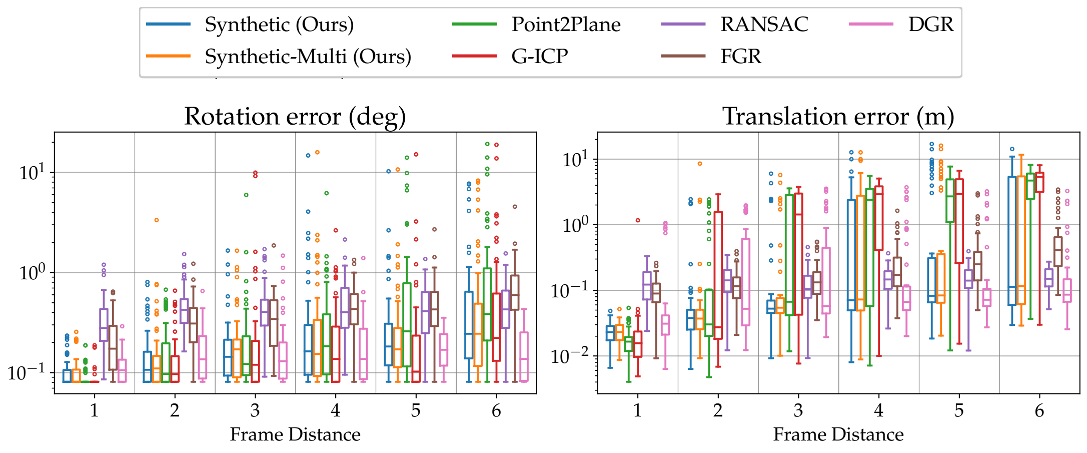


Figure 6.7: Registration accuracy evaluation on KITTI sequence 00 with sampled pairs of enumerated frame differences. For each box plot, a lower median and smaller rectangle box is better. Without a LUT, projective registration with synthetic LiDAR intrinsics achieves comparable performance to ICP variants and global approaches.

scenes, and 4cm for indoor scenes. We clip faraway points to maintain a reasonable memory footage and filter potential outliers. For indoor scenes and outdoor scenes the clipping distances are 10m and 30m, respectively.

Implementation.

All the experiments are conducted on a machine with an NVIDIA RTX 3060 graphics card and an 16 core Intel i7-11700 CPU. The code is written in C++/CUDA with modularized python bindings.

6.6.2 KITTI Dataset

Before going through the evaluation on our collected dataset, we first briefly evaluate on the KITTI dataset [82] with synthetic intrinsics to demonstrate the compatibility of our algorithms to point clouds. We deliver qualitative and quantitative registration experiments and provide qualitative reconstruction results.

Registration.

Fig. 6.6 shows the qualitative registration results. We observe that with challenging translation, the projective association ensures a wider search range for correspondences, and results in better convergence especially enhanced with multi-scale processing. We also quantitatively evaluate the registration accuracy with varying frame distances. Due to the fast moving speed, we limit the frame distance to 6 (otherwise overlaps between point clouds are limited). Here we use poses obtained from CMRNet [36, 39] as refined GT poses.

In Fig. 6.7 we can observe that in comparison to ICP variants, projective registration has a comparable performance on translation and is consistently better on rotation due to the cylindrical presentation's advantage of a wide azimuth receptive field. At large frame distances, their medium rotation and translation error are also comparable to global registration results, including learning-based DGR.

Surface Reconstruction.

In Fig. 6.8 we show the meshes extracted from TSDF reconstruction at city scale on LiDAR data. With a limited GPU memory budget, we are able to reconstruct scenes with a 20cm voxel size at 40 Hz, where points farther than 30m are clipped.

6.6.3 LiDAR Range Image Dataset

Registration.

On our LiDAR range image dataset, we select a typical indoor scene *lecture building* and outdoor scene *dormitory*. The results are shown in Fig. 6.9. In general, regardless of indoor or outdoor setups, we observe that range image based projective registration is comparable to ICP variants with small frame distances, and achieves better performance on more challenging registration tasks with large frame distances. At such setups, multi-scale registration with an LUT even outperforms global registration, including DGR, in most scenarios.

Table 6.1: F-score of surface reconstruction. Our method is consistently the best.

	Lab	*Lounge*	*Lecture Bld.*	*Lecture Rm.*	*Student Rm.*	*Campus*	*Fountain*	*Statue*	*Dormitory*	*Square*
Voxblox [175]	0.4845	0.4919	0.4923	0.4956	0.4856	0.4896	0.4916	0.4885	0.4901	0.4893
Octomap [108]	0.4762	0.4868	0.4526	0.4948	0.3883	0.4090	0.4588	0.4890	0.4898	0.4573
Ours	**0.4900**	**0.4940**	**0.4930**	**0.4972**	**0.4870**	**0.4907**	**0.4933**	**0.4895**	**0.4917**	**0.4901**

Surface Reconstruction.

In Fig. 6.5, we qualitatively show the reconstructed surfaces from SDF volumes overlaid with the camera trajectory. We observe that in indoor scenes, our algorithm reconstructs high quality surfaces, despite our range images having lower spatial density. In addition, we are able to reconstruct high quality surfaces of large scale outdoor scenes.

Quantitative results are shown in Table 6.1, where the valid correspondence searching range is set to 3$\times$ voxel size for precision and recall computation in F-score. Comparing to the baselines, our reconstruction achieves consistently the highest F-score.

Runtime Evaluation.

We then demonstrate the efficiency of our approaches by evaluating average run time on indoor and outdoor scenes separately. In Table 6.2 we observe that while achieving comparable or better accuracy than state-of-the-art approaches, our method is 15–50$\times$ faster in volumetric reconstruction, and 5–150$\times$ faster in registration.

6.6.4 Limitations

Our registration method has presented benefits both in efficiency and accuracy in the experiments, yet there are several limitations. Although it has achieved good performance with challenging rotations, it still has a reduced stability on large translations that cannot be addressed by the cylindrical representation. Since

Table 6.2: Run time evaluation. With parallel projective operations implemented on GPU, our methods are at least one magnitude faster than baselines.

	Voxblox [175]	*Octomap* [108]	*Ours*	*Pt2Pl* [272]	*G-ICP* [205]	*RANSAC* [198]	*FGR* [271]	*DGR* [51]	*Ours (LUT)*	*Ours (LUT-multi)*
Indoor	1003.26	676.20	**22.29**	62.04	118.61	445.97	977.56	347.39	14.14	**10.90**
Outdoor	1002.22	1167.68	**58.55**	59.19	118.60	408.16	1407.59	1184.51	14.20	**10.69**

fundamentally it is depending on dense nonlinear optimization, similar to other local registration approaches, it may also fall into local optima when the scene's structures are not salient. Another limitation is that the projective model can be disturbed by fast sensor motions and dynamic environments, where the projection model needs modification for moving ray centers. In the future, we would attempt to learn deep cylindrical image features for global feature matching using *e.g.* Spherical CNNs [54]; we will also consider non-rigid transform with consecutive poses assigned to each column [262].

6.7 Conclusions

We presented a range image based LiDAR data representation from raw sensor data that naturally preserves the neighbor information. With an intrinsic spherical projective model, it allows us to perform fast and accurate range image based multi-scale registration and dense reconstruction. We then collected a new LiDAR dataset in the image form, and perform comprehensive experiments for dense reconstruction and registration demonstrating the efficiency of our approaches. With the proof of concept, we humbly hope the hardware manufacturers may expose more user-friendly interfaces to generate LiDAR images for fast and accurate 3D perception, and the vision community may find it easier to transfer the knowledge from 2D to 3D.

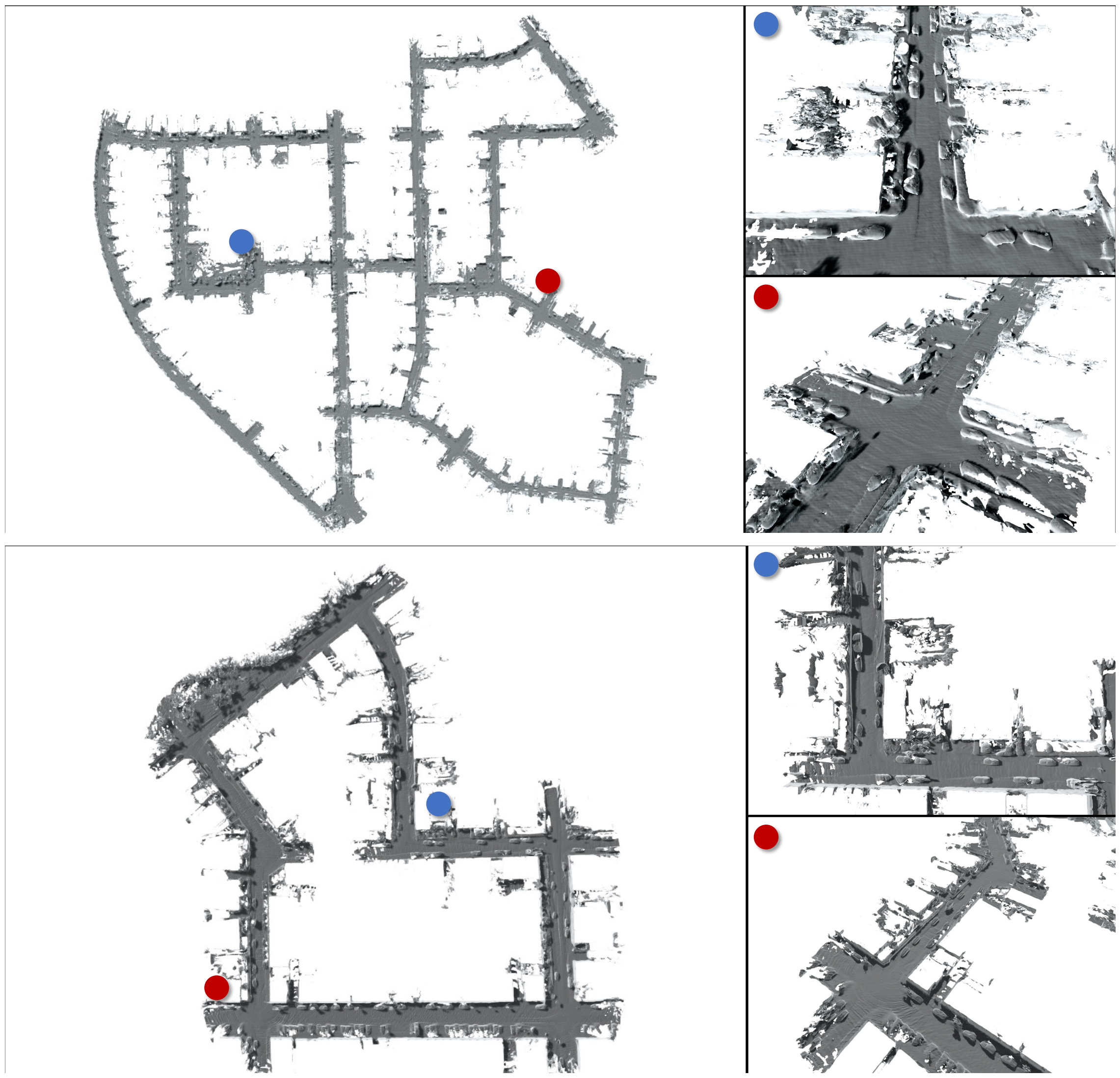


Figure 6.8: City-scale TSDF surface reconstruction of sequences 00 and 07 from KITTI. Left: full reconstruction. Right: selected details.

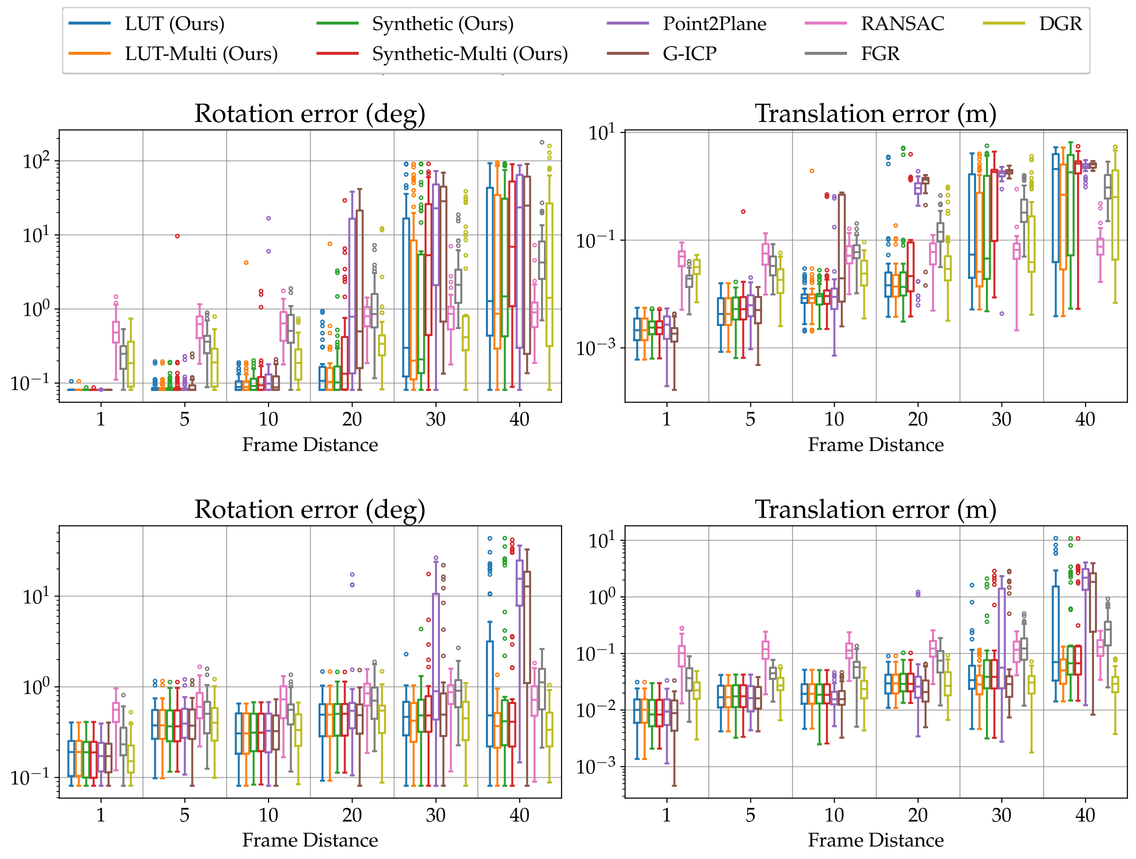


Figure 6.9: Registration accuracy evaluation on the indoor *lecture building* (top) and the outdoor *dormitory* (bottom) sequences with sampled pairs of enumerated frame differences. For each box plot, a lower median and smaller rectangle box is better. Projective registration achieves comparable performance to baselines in general. Multi-scale projective registration equipped with an LUT is the best, with a stable performance at small frame distances and a better performance than even global approaches at large frame distances.

Chapter 7

Fast Monocular Scene Reconstruction

7.1 Introduction

Reconstructing indoor spaces into 3D representations is a key requirement for many real-world applications, including robot navigation, immersive virtual/augmented reality experiences, and architectural design. Particularly useful is reconstruction from monocular cameras which are the most prevalent and accessible to causal users. While much research has been devoted to this task, several challenges remain.

Conventional monocular reconstruction from multi-view RGB images uses patch matching [202], which takes hours to reconstruct even a relatively small scene. Several 3D reconstruction methods [215, 251] have demonstrated fast reconstruction by applying 3D convolutional neural networks to feature volumes, but they have limited resolution and struggle to generalize to larger scenes.

Recently, unified neural radiance fields [160] and neural implicit representations were developed for the purpose of accurate surface reconstruction from images [174, 231, 253]. While this was successfully demonstrated on single objects, the weak photometric constraint leads to poor reconstruction and slow convergence for large-scale scenes. Guo *et al.* [95] and Yu *et al.* [257] improved the quality and convergence speed of neural field reconstruction on large-scale scenes by incorporating learned geometrical cues like depth and normal estimation [70, 189], however, training and evaluation remain inefficient. This is primarily because

🟦	wall	🟧	cabinet	🟩	sofa	🟥	door	🟫	picture
🟦	floor	🟩	chair	🟥	table	🟪	window	🟪	curtain

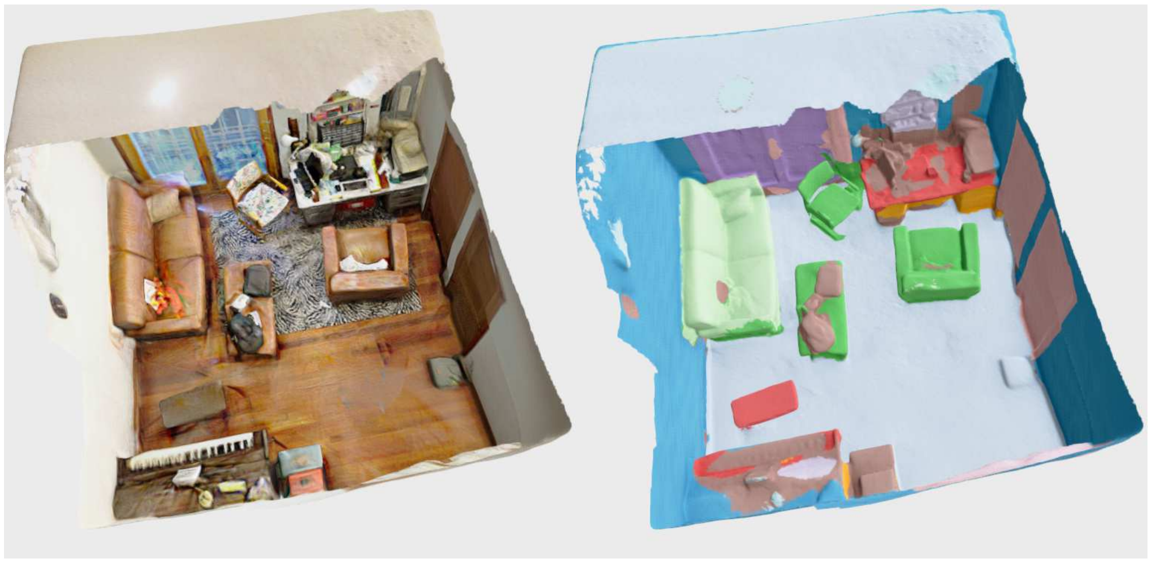
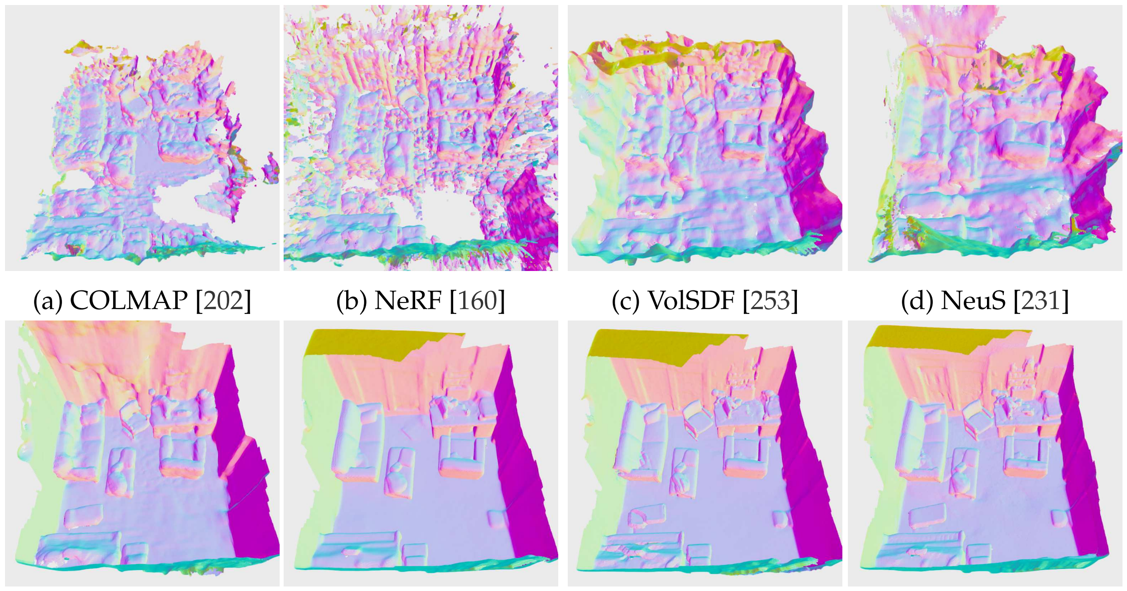


Figure 7.1: Color and semantic scene reconstruction from our system with monocular images and learned monocular priors.

these approaches rely on MLPs and feature grids [163] that encode the entire scene rather than concentrating around surfaces.

In contrast to MLPs, an explicit SDF voxel grid can be adaptively allocated around surfaces, and allows fast query and sampling. However, an efficient implementation of differentiable SDF voxel grids without MLPs is missing. Fridovich-Keil and Yu *et al.* [77] used an explicit density and color grid, but is limited to rendering small objects. Muller *et al.* [163] developed a feature grid with spatial hashing for fast neural rendering, but its backbone hash map is not collision-free, causing inevitable slow random access and inaccurate indexing at large scales. Dong *et al.* [67] proposed a collision-free spatially hashed grid following Niessner *et al.* [171], but lacks support for differentiable rendering. Several practical challenges hinder the implementation of an efficient differentiable data structure: 1. a collision-free spatial hash map on GPU that supports one-to-one indexing from positions to voxels; 2. differentiable trilinear interpolations between spatially hashed voxels; 3. parallel ray marching and uniform sampling from a spatial hash



(a) COLMAP [202] (b) NeRF [160] (c) VolSDF [253] (d) NeuS [231]

(e) ManhattanSDF [95] (f) MonoSDF-MLP [257] (g) MonoSDF-Grid [257] (h) Ours

Figure 7.2: Qualitative reconstruction comparison on ScanNet [57]. While being 10× faster in training, we achieve similar reconstruction results to state-of-the-art MonoSDF [257], with fine details (see Fig. 7.9).

map.

Our approach: we address such challenges using a differentiable *globally sparse* and *locally dense* voxel grid. We transform a collision-free GPU hash map [211] to a differentiable tensor indexer [181]. This generates a one-to-one map between positions and *globally sparse* voxel blocks around approximate surfaces, and enables skipping empty space for efficient ray marching and uniform sampling. We further manage *locally dense* voxel arrays within sparse voxel blocks for GPU cache-friendly contiguous data query via trilinear interpolation. As a result, using explicit SDF grids leads to fast SDF gradient computation in a single forward pass, which can further accelerate differentiable rendering.

This new data structure presents a new challenge — we can only optimize grid parameters if they are allocated around surfaces. To resolve this, we make use of off-the-shelf monocular depth priors [70, 189] and design a novel initialization scheme with global structure-from-motion (SfM) constraints to calibrate these unscaled predicted depths. It results in a consistent geometric initialization via volumetric fusion ready to be refined through differentiable volume rendering.

We additionally incorporate semantic monocular priors [139] to provide cues for geometric refinement in 3D. For instance, we use colors and semantics to guide the sharpening of normals around object boundaries, which in turn improves the quality of colors and semantics. We enforce these intuitive notions through our novel continuous Conditional Random Field (CRF). We use Monte Carlo samples on the SDF zero-crossings to create continuous CRF nodes and define pairwise energy functions to enforce local consistency of colors, normals, and semantics. Importantly, we define similarity in a *high dimensional space* that consists of coordinates, colors, normals, and semantics, to reject spatially close samples with contrasting properties. To make inference tractable, we follow Krahenbuhl *et al.* [131] and use variational inference, leading to a series of convolutions in a high-dimensional space. We implement an efficient permutohedral lattice convolution [1] using the collision-free GPU hashmap to power the continuous CRF inference.

The final output of our system is a scene reconstruction with geometry, colors, and semantic labels, as shown in Fig. 7.1. Experiments show that our method is $10\times$ faster in training, $100\times$ faster in inference, and has comparable accuracy measured by F-scores against state-of-the-art implicit reconstruction systems [95, 257]. In summary, we propose a fast scene reconstruction system for monocular images. Our contributions include:

- A globally sparse locally dense differentiable volumetric data structure that exploits surface spatial sparsity without an MLP;
- A scale calibration algorithm that produces consistent geometric initialization from unscaled monocular depths;
- A fast monocular scene reconstruction system equipped with volume rendering and high dimensional continuous CRFs optimization.

7.2 Related Work

Surface reconstruction from 3D data. Surface reconstruction has been well-studied from 3D scans. The general idea is to represent the space as an implicit signed distance function, and recover surfaces at zero-crossings with Marching Cubes [146]. Classical works [56, 58, 67, 168, 171] quantize the 3D space into voxels, and integrate frame-wise SDF observations into voxels. Instead of direct voxels, recent neural representations [214, 217, 275] use either a pure MLP or a feature grid to reconstruct smoother surfaces. These approaches are often fast and accurate, but heavily depends on high-quality depth input from sensors.

Surface reconstruction from RGB. A variety of classical and learning-based methods [169, 215, 251, 264] have been developed to achieve high quality multi-view depth reconstruction from monocular images. These techniques usually construct a cost volume between a reference frame and its neighbor frames, and maximize the appearance consistency. A global volume can be optionally grown from the local volumes [166, 215, 264] for scene reconstruction. While these approaches succeed on various benchmarks, they rely on fine view point selection, and the performance may be significantly reduced when the view points and surfaces are sparse in space. Training on a large datasets is also required.

Recent advances in neural rendering [15, 160, 227] and their predecessors have defined the surface geometry by a density function predicted by an MLP in 3D space. They seek to minimize ray consistency with the rendering loss using test-time optimization. While being able to achieve high rendering quality, due to the ambiguity in the underlying density representation, accurate surfaces are hard to recover. In view of this, implicit SDF representations [216, 231, 252, 253] are used to replace density, where surfaces are better-defined at zero-crossings. To enable large-scale indoor scene reconstruction, ManhattanSDF [95] and MonoSDF [257] incorporate monocular geometric priors and achieve state-of-the-art results. These approaches initialize the scene with a sphere [10], and gradually refine the details. As a result, the training time can be long, varying from hours to half a day.

Monocular priors in surface reconstruction. Priors from monocular images have been used to enhance reconstruction and neural rendering from images, by providing reasonable initialization and better constrained sampling space. A light weight prior is the structure-from-motion (SfM) supervision [202], where poses and sparse point clouds are reconstructed to provide the geometry. Similarly, dense monocular priors including depths [144, 188, 189], normals [70], and semantic segmentations [139]. Existing approaches either use only SfM [61], or enhance dense priors by SfM via finetuning depth prediction networks [236] or depth completion networks [192] for density-based neural rendering. Similarly, monocular priors

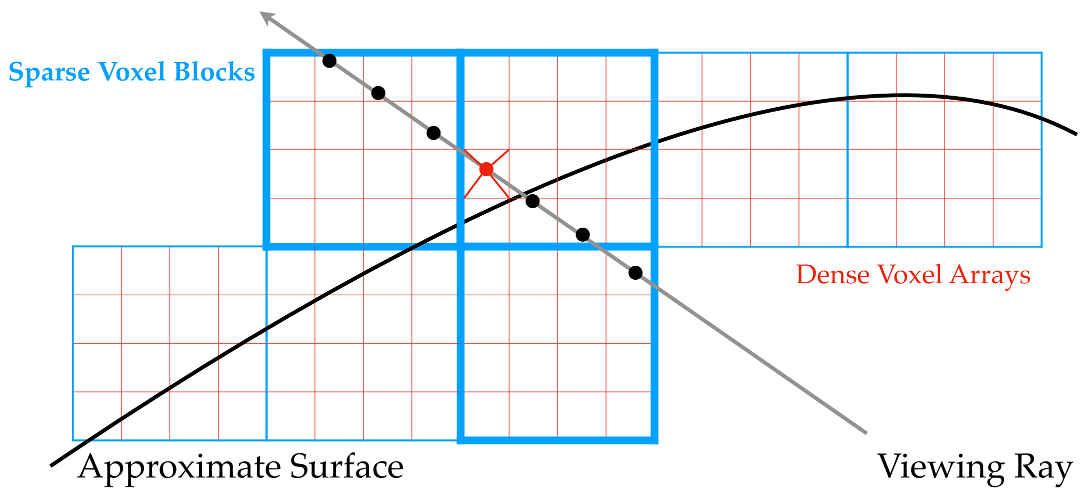


Figure 7.3: Illustration of the sparse-dense data structure. The large voxel blocks (in blue) are allocated only around approximate surfaces, and indexed by a collision-free hash map. The voxel arrays (in red) further divide the space to provide high-resolution details. Ray marching skips empty space and only activates hit blocks (in bold blue). Trilinear interpolation of neighbor voxel properties allows sampling at continuous locations.

are used to enhance SDF-based neural reconstruction [95, 257] with remarkable performance. While emphasizing the guidance in sampling, these approaches usually stick to MLPs or dense voxel grids, without being able to fully exploit the sparsity of the surface distribution.

Sparse spatial representations. Sparse spatial representations have been well-studied for 3D data, especially point clouds [49, 67, 171]. Often used data structures are hash maps, Octrees [156], or a combination [167]. These data structures have been adapted to neural 3D reconstructions and rendering to exploit spatial sparsity, but they either depend on high quality 3D input [217], or focus on object-centered reconstruction [53, 77, 163]. Their usage to scene reconstruction from monocular images is yet to be explored.

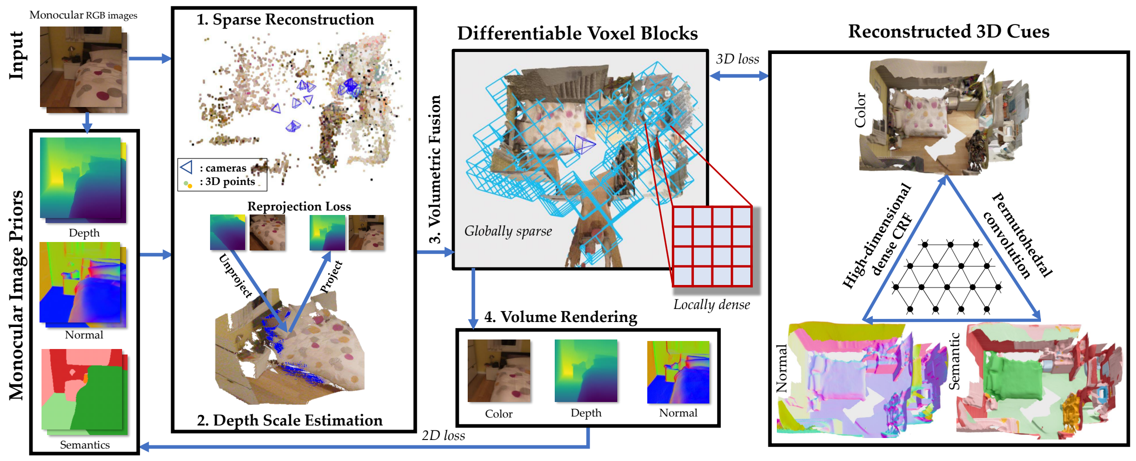


Figure 7.4: Illustration of our pipeline. We first use structure-from-motion (SfM) to obtain sparse feature-based reconstruction. With the sparse point cloud and covisibility information from SfM, we optimize the scale of predicted monocular depth images (§7.3.3), and perform volumetric fusion to construct a globally sparse locally dense voxel grid (§7.3.4). After initialization, we perform differentiable volume rendering to refine the details (§7.3.5), and apply high dimensional continuous CRFs to finetune normals, colors, and labels (§7.3.5).

7.3 Method

7.3.1 Overview

The input to our method is a sequence of monocular images $\{\mathcal{I}_i\}$. Prior to reconstruction, similar to previous works [95, 257], we generate per-image monocular priors including unscaled depth $\{\mathcal{D}_i\}$ and normal $\{\mathcal{N}_i\}$ predicted by Omnidata [70], and semantic logits $\{\mathcal{S}_i\}$ from LSeg [139]. Afterwards, the system runs in three major stages.

- Sparse SfM reconstruction [202] and initial depth scale optimization;
- Direct volumetric fusion for sparse voxel allocation and geometric initialization;
- Differentiable volume rendering and dense CRF smoothing for detail refinement.

Fig. 7.4 shows the pipeline of our framework. We will describe these stages in detail after introducing our core data structure.

7.3.2 Sparse-Dense Data Structure

In order to facilitate multi-view sensor fusion, SDF are approximated by truncated SDF (TSDF) that maintain averaged signed distances to surface in a narrow band close to the surface [56, 168, 171]. We take advantage of this property and develop a globally sparse locally dense data structure. Global sparsity is attained through allocating voxels only around approximate surfaces, which we index using a collision-free hash map. Within these sparse voxels we allocate cache-friendly small dense arrays that allow fast indexing and neighbor search storing SDF, color, and optionally labels. The data structure is visualized in Fig. 7.3.

While similar structures have been used for RGB-D data that focus on forward fusion [67, 171], our implementation supports both forward fusion via hierarchical indexing, and auto-differentiable backward optimization through trilinear interpolation, allowing refinement through volume rendering. In addition, SDF gradients can be explicitly computed along with SDF queries in the same pass, allowing efficient regularization during training.

Our data structure is akin to any neural networks that maps a coordinate $\mathbf{x} \in \mathbb{R}^3$ to a property [241], thus in the following sections, we refer to it as a function f. We use $f_{\theta_d}, f_{\theta_c}, f_{\theta_s}$ to denote functions that query SDF, color, and semantic labels from the data structure, respectively, where θ_d, θ_c, and θ_s are properties directly stored at voxels.

7.3.3 Depth Scale Optimization

Our sparse-dense structure requires approximate surface initialization at the allocation stage, hence we resort to popular monocular geometry priors [70] also used in recent works [257]. Despite the considerable recent improvement of monocular depth prediction, there are still several known issues in applications: each image's depth prediction is scale-ambiguous, often with strong distortions. However, to construct an initial structure we require a consistent scale.

To resolve this, we define a scale function ϕ_i per monocular depth image $\mathcal{D}_i$ to optimize scales and correct distortions. ϕ_i is represented by a 2D grid, where each grid point stores a learnable scale. A pixel's scale $\phi_i(\mathbf{p})$ can be obtained through bilinear interpolating its neighbor grid point scales. We optimize $\{\phi_i\}$ to achieve consistent depth across frames

$$\min_{\{\phi_i\}} \sum_{i,j \in \Omega} h(\phi_i, \phi_j) + \lambda \sum_i g(\phi_i), \tag{7.1}$$

where h and g impose mutual and binary constraints, and Ω is the set of covisible image pairs.

Previous approaches [130] use fine-grained pixel-wise correspondences to construct h via pairwise dense optical flow, and introduce a regularizer g per frame. This setup is, however, computationally intensive and opposes our initial motivation of developing an efficient system. Instead, we resort to supervision from SfM's [202] sparse reconstruction. It estimates camera poses $\{R_i, t_i\}$, produces sparse reconstruction with 3D points $\{\mathbf{x}_k\}$ and their associated 2D projections $\mathbf{p}_{\mathbf{x}_k \to i}$ at frame $\{\mathcal{I}_i, \mathcal{D}_i\}$, and provides the covisible frame pair set Ω. With such, we can define the unary constraint g via a reprojection loss

$$g(\phi_i) = \sum_{\mathbf{x}_k} \| d_{\mathbf{x}_k \to i} - \mathcal{D}_i(\mathbf{p}_{\mathbf{x}_k \to i}) \phi_i(\mathbf{p}_{\mathbf{x}_k \to i}) \|^2, \tag{7.2}$$

$$d_{\mathbf{x}_k \to i} \cdot \begin{bmatrix} \mathbf{p}_{\mathbf{x}_k \to i} & 1 \end{bmatrix}^\top \triangleq \Pi\left(\mathbf{R}_i^\top (\mathbf{x}_k - \mathbf{t}_i)\right), \tag{7.3}$$

where Π is the pinhole projection. Similarly, we define binary constraints by minimizing reprojection errors across covisible frames:

$$h(\phi_i, \phi_j) = \sum_{\mathbf{p} \in \mathcal{D}_i} \| d_{i \to j} - \mathcal{D}_j(\mathbf{p}_{i \to j}) \phi_j(\mathbf{p}_{i \to j}) \|^2 + \| \mathcal{I}_i(\mathbf{p}) - \mathcal{I}_j(\mathbf{p}_{i \to j}) \|^2, \tag{7.4}$$

$$\mathbf{x} = \Pi^{-1}\Big(\mathbf{p}, \mathcal{D}_i(\mathbf{p}) \phi_i(\mathbf{p})\Big), \tag{7.5}$$

$$d_{i \to j} \cdot \begin{bmatrix} \mathbf{p}_{i \to j} & 1 \end{bmatrix}^\top \triangleq \Pi(\mathbf{R}_{i,j} \mathbf{x} + \mathbf{t}_{i,j}), \tag{7.6}$$

where Π^{-1} unprojects a pixel $\mathbf{p}$ in frame i from deformed depth to a 3D point $\mathbf{x}$, and $\{\mathbf{R}_{i,j}, \mathbf{t}_{i,j}\}$ are relative poses. This loss enforces local consistency between visually adjacent frames.

We use a 24×32 2D grid per image, $\lambda = 10^{-3}$, and optimize $\{\phi_i\}$ via RMSprop with a learning rate of 10^{-2} for 500 steps.

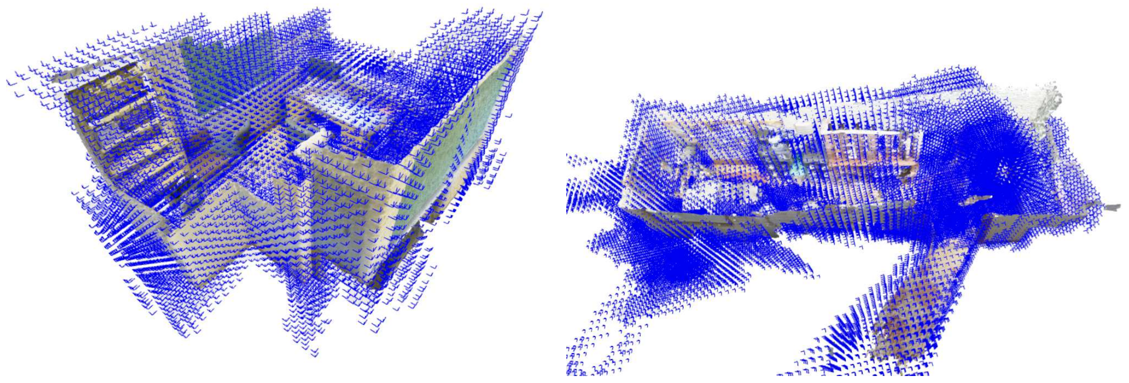


Figure 7.5: Sparse voxel grid (blue) allocation around 3D points from unprojection. The grids are adaptive to scenes with different overall surface shapes. Ground truth surface mesh are visualized for illustration.

7.3.4 Direct Fusion on Sparse Grid

Allocation

Similar to aforementioned works for online reconstruction [67, 171], the sparse blocks are allocated by the union of voxels containing the unprojected points,

$$\mathbf{X} = \cup_i \mathbf{X}_i, \mathbf{X}_i = \cup_p \Big\{ \text{Dilate}\left(\text{Voxel}(\mathbf{p})\right) \Big\}, \tag{7.7}$$

$$\text{Voxel}(\mathbf{p}) = \left\lfloor \frac{\mathbf{R}_i \Pi^{-1}(\mathbf{p}, \mathcal{D}_i(\mathbf{p})\phi_i(\mathbf{p})) + \mathbf{t}_i}{L} \right\rfloor, \tag{7.8}$$

where L is the sparse voxel block size, and the dilate operation grows a voxel block to include its neighbors to tolerate more uncertainty from depth prediction. A dynamic collision-free hash map [67] is used to efficiently aggregate the allocated voxel blocks. The dense voxel arrays are correspondingly allocated underlying the sparse voxel blocks.

Fig. 7.5 shows the surface-adaptive allocation. In contrast to popular sparse grids in a fixed bounding box used by neural rendering [77, 163], this allocation strategy is more flexible to various shapes of rooms.

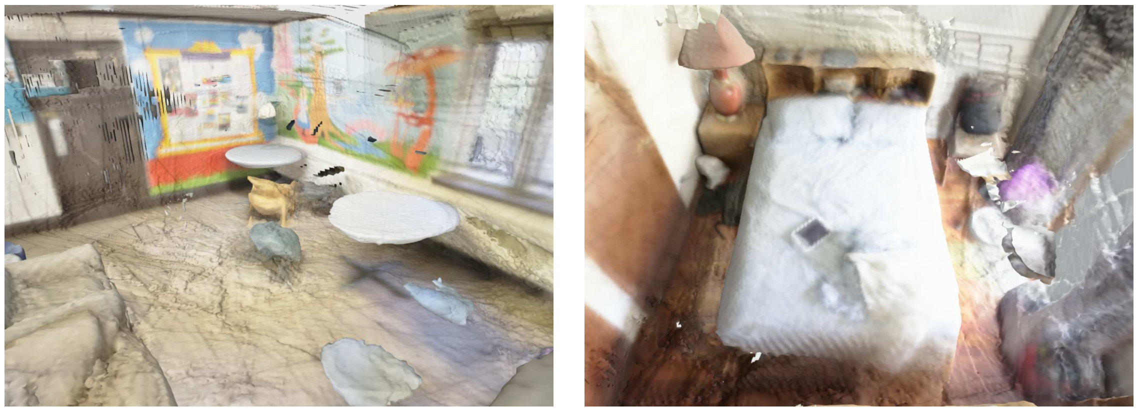


Figure 7.6: With scale calibration and volumetric fusion, room-scale geometry initialization can be achieved from monocular depth without any optimization of the voxel grid parameters. The remaining task would be refining noisy regions and prune outliers.

Depth, Color, and Semantic Fusion

Following classical volumetric fusion scheme [168, 171], we project voxels $\mathbf{v}$ back to the images to setup voxel-pixel associations, and directly optimize voxel-wise properties, namely SDF (θ_d), color (θ_c), and semantic label logits (θ_s).

$$\theta_d(\mathbf{v}) = \arg\min_{d} \sum_i \Big(- \big(d_{\mathbf{v}\to i} - \mathcal{D}_i(\mathbf{p}_{\mathbf{v}\to i})\phi_i(\mathbf{p}_{\mathbf{v}\to i})\Big)^2, \tag{7.9}$$

$$\theta_c(\mathbf{v}) = \arg\min_{\mathbf{c}} \sum_i \|\mathbf{c} - \mathcal{I}_i(\mathbf{p}_{\mathbf{v}\to i})\|^2, \tag{7.10}$$

$$\theta_s(\mathbf{v}) = \frac{\mathbf{s}^*}{\|\mathbf{s}^*\|}, \mathbf{s}^* = \arg\min_{\mathbf{s}} \sum_i \|\mathbf{s} - \mathcal{S}_i(\mathbf{p}_{\mathbf{v}\to i})\|^2, \tag{7.11}$$

where the projection $\mathbf{v} \to i$ is given by Eq. 7.3. Note by definition, only associations with SDF smaller than a truncation bound will be considered, minimizing the effect of occlusion. It is worth mentioning that we use a simple L2 loss for semantic logit instead of entropy losses, as it is considered one of the best practices in label fusion [155]. The closed-form solutions of aforementioned voxel-pixel association losses are simply averages. Therefore, with minimal processing time, we can already achieve reasonable initial surface reconstruction by classical volumetric SDF and color fusion, see Fig. 7.6.

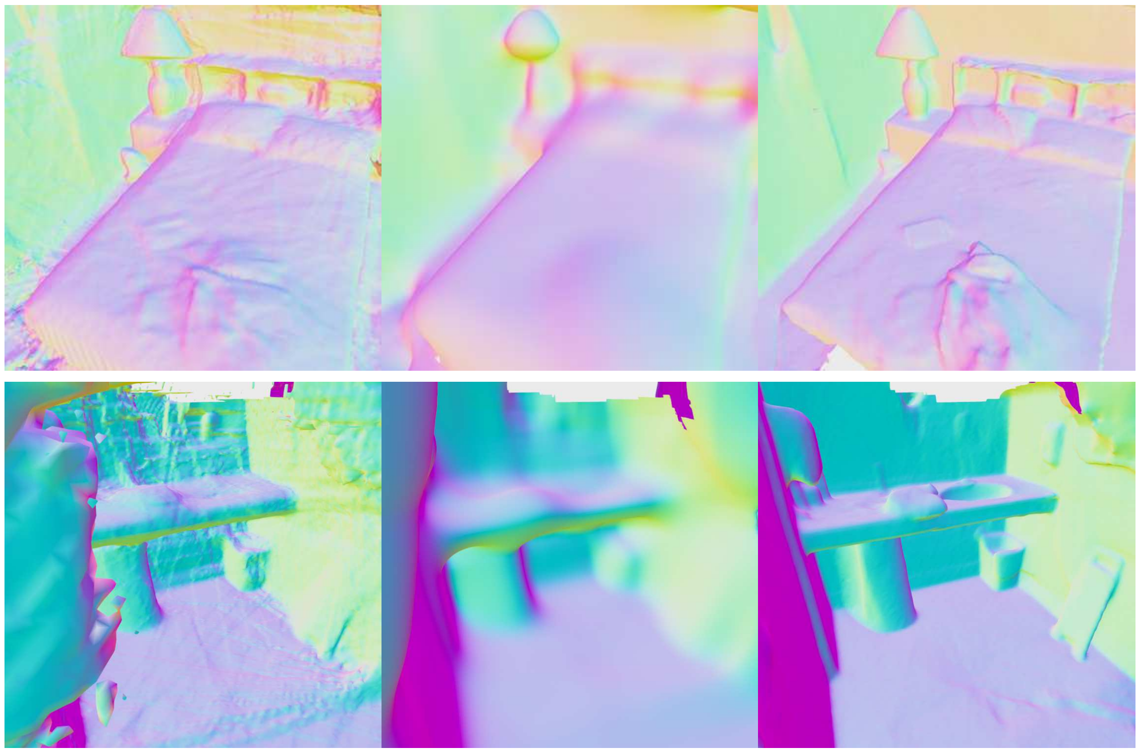


Figure 7.7: Comparison between 3 stages of reconstruction: initialization, de-noising, and volume rendering refinement.

De-noising

Direct fused properties, although being smoothed average of observations across frames *per voxel*, are spatially noisy and can result in ill-posed SDF distributions along rays. Therefore, we perform a Gaussian blurring for the voxels along all the properties. Thanks to the direct representation, with a customized forward sparse convolution followed by a property assignment, we could accomplish the filtering without backward optimizations. The effect of the de-noising operation can be observed in Fig. 7.7(a)-(b).

7.3.5 Differentiable Geometry Refinement

Volume Rendering

We follow MonoSDF [257] to refine geometry using monocular priors. For a pixel $\mathbf{p}$ from frame i, we march a ray $\mathbf{x}(t) = \mathbf{r}_o + t \cdot \mathbf{r}_d$ to the sparse voxel grid, sample a sequence of points $\{\mathbf{x}_k = \mathbf{x}(t_k)\}$, apply volume rendering, and minimize the color, depth, and normal losses respectively:

$$\mathcal{L}_c(\theta_c, \theta_d) = \left\| \sum_k w(\mathbf{x}_k) f_{\theta_c}(\mathbf{x}_k) - \mathcal{I}_i(\mathbf{p}) \right\|, \tag{7.12}$$

$$\mathcal{L}_d(\theta_d) = \left\| \sum_k w(\mathbf{x}_k) t_k - (a\mathcal{D}_i(\mathbf{p}) + b) \right\|^2, \tag{7.13}$$

$$\mathcal{L}_n(\theta_d) = \left\| \sum_k w(\mathbf{x}_k) \nabla f_{\theta_d}(\mathbf{x}_k) - \mathcal{N}_i(\mathbf{p}) \right\|, \tag{7.14}$$

$$w(\mathbf{x}_k) = \exp\big(-\sum_{j<k} \alpha(\mathbf{x}_j)\delta_j\big)\big(1 - \exp(-\alpha(\mathbf{x}_k)\delta_k)\big), \tag{7.15}$$

where $\delta_i = t_{i+1} - t_i$, depth scale a and shift b are estimated per minibatch in depth loss with least squares [189], and the density $\alpha(\mathbf{x}_k) = l(f_{\theta_d}(\mathbf{x}_k))$ is converted from SDF with a Laplacian density transform from VolSDF [253]. To accelerate, points are sampled in the sparse grid where valid voxel blocks have been allocated, and the empty space is directly skipped.

Regularization

Eikonal regularization [10] forces SDF gradients to be close to 1,

$$\mathcal{L}_{\text{Eik}} = \left(\|\nabla f_{\theta_d}(\mathbf{x})\| - 1\right)^2. \tag{7.16}$$

Similar to related works [253, 257], $\{\mathbf{x}\}$s are samples combined with ray-based samples around surfaces, and uniform samples in the sparse grids. It is worth noting that in an explicit SDF voxel grid, f_{θ_d} and ∇f_{θ_d} can be jointly computed in the same pass:

$$f_{\theta_d}(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathrm{Nb}(\mathbf{x})} r(\mathbf{x}, \mathbf{x}_i)\theta_d(\mathbf{x}_i), \tag{7.17}$$

$$\nabla_{\mathbf{x}} f_{\theta_d}(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathrm{Nb}(\mathbf{x})} \nabla_{\mathbf{x}} r(\mathbf{x}, \mathbf{x}_i)\theta_d(\mathbf{x}_i), \tag{7.18}$$

where $\theta_d(\mathbf{x}_i)$ are directly stored SDF values at voxel grid points $\mathbf{x}_i$, and r is the trilinear interpolation ratio function that is a polynomial with closed-form derivatives. This circumvents costly double backward pass for autodiff gradient estimation [253, 257], therefore speeds up training both by reducing computation burden and allowing larger batch size.

Differentiable Continuous Semantic CRF

Through differentiable volume rendering, we have achieved fine geometry reconstructions. We want to further sharpen the details at the boundaries of objects (*e.g.*, at the intersection of a cabinet and the floor). We resort to CRFs for finetuning all the properties, including colors, normals, and labels. Unlike conventional CRFs where energy functions are defined on discrete nodes, we propose to leverage our data structure and devise a continuous CRF to integrate energy over the surface

$$E(\mathbb{S}) = \int_{\mathbb{S}} \psi_u(\mathbf{x})d\mathbf{x} + \int_{\mathbb{S}}\int_{\mathbb{S}} \psi_p(\mathbf{x}_i, \mathbf{x}_j)d\mathbf{x}_i d\mathbf{x}_j, \tag{7.19}$$

where $\mathbf{x} \in \mathbb{S}$ denotes a point on the surface. ψ_u and ψ_p denote unary and pairwise Gibbs energy potentials. Following Krahenbuhl *et al.* [131], we adopt the Gaussian edge potential

$$\psi_p(\mathbf{x}_i, \mathbf{x}_j) = \mu_{\text{prop}}(\mathbf{x}_i, \mathbf{x}_j) \exp\left(-(\mathbf{f}_i - \mathbf{f}_j)^T \Lambda (\mathbf{f}_i - \mathbf{f}_j)\right), \tag{7.20}$$

where μ_{prop} denotes a learnable compatibility function of a node property (*e.g.* normal), $\exp$ computes the consistency strength between nodes from feature distances with the precision matrix Λ. A feature $\mathbf{f}_i$ concatenates 3D positions, colors, normals, and label logits queried at $\mathbf{x}_i$. We approximate the integration over the surface with Monte Carlo sampling by finding zero-crossings from random camera viewpoints.

The variational inference of the Gibbs energy potential with the mean-field approximation results in a simple update equation

$$Q(\mathbf{x}_i)^{+} \propto \exp\left(-\psi_u(\mathbf{x}_i) - \sum_j \psi_p(\mathbf{f}_i, \mathbf{f}_j) Q(\mathbf{x}_j)\right). \tag{7.21}$$

Note that the summation in Eq. 7.21 is over all the sample points and computationally prohibitive. Thus, we use a high-dimensional permutohedral lattice convolution [1] to accelerate the message passing, driven by our collision-free hash map at high dimensions.

For each of the target properties $\mathrm{prop} \in \{\mathrm{color}, \mathrm{normal}, \mathrm{label}\}$, we define a loss $\mathcal{L}_{\mathrm{prop}} = D_f\Big(\mathbf{x}_{\mathrm{prop}} \| Q(\mathbf{x}_{\mathrm{prop}})\Big)$ with f-diveregence, conditioned on the remaining properties plus the 3D positions. A joint loss is defined to optimize all the properties:

$$\mathcal{L}_{\mathrm{CRF}} = \lambda_{\mathrm{color}} \mathcal{L}_{\mathrm{CRF}}^{\mathrm{color}} + \lambda_{\mathrm{normal}} \mathcal{L}_{\mathrm{CRF}}^{\mathrm{normal}} + \lambda_{\mathrm{label}} \mathcal{L}_{\mathrm{CRF}}^{\mathrm{label}}. \tag{7.22}$$

Optimization

The overalls loss function at refinement stage is

$$\mathcal{L} = \mathcal{L}_c + \lambda_d \mathcal{L}_d + \lambda_n \mathcal{L}_n + \lambda_{\mathrm{Eik}} \mathcal{L}_{\mathrm{Eik}} + \mathcal{L}_{\mathrm{CRF}}. \tag{7.23}$$

We optimize the grid parameters $\{\theta_d, \theta_c, \theta_s\}$ with RMSProp starting with a learning rate 10^{-3}, and an exponential learning rate scheduler with $\gamma = 0.1$.

7.4 Experiments

7.4.1 Setup

We follow Manhattan SDF [95] and evaluate on 4 scenes from ScanNet [57] and 4 scenes from 7-scenes [84] in evaluation. We use reconstruction's F-score as the major metric, along with distance metrics (accuracy, completeness), precision, and recall. We compare against COLMAP [202], NeRF [160], UNISURF [174], NeuS [231], VolSDF [253], Manhattan SDF [95], and MonoSDF [257]. We train MonoSDF to obtain output mesh. For the rest of the compared approaches, we reuse reconstructions provided by the authors from Manhattan SDF [95], and evaluate them against high-resolution ground truth via TSDF fusion. The evaluation metric and implementation details are in supplementary.

For geometric priors, unlike MonoSDF [257] that generates monocular cues from 384×384 center crops, we follow DPT [189]'s resizing protocol and adapt Omnidata [70] to obtain 480×640 full resolution cues.

In all the experiments, we use a 8^3 voxel block grid with a voxel size 1.5cm. At each step, we randomly sample 1024 rays per image with a batch size of 64. Due to the reasonable geometric initialization, the loss usually drops drastically within 2×10^3 iterations, and converges at 10^4 iterations, therefore we terminate training at 10^4 steps for all scenes. Thanks to the efficient data structure, accelerated ray marching, and closed-form SDF gradient computation, it takes less than 30 mins to reconstruct a scene on a mid-end computer with an NVIDIA RTX 3060 GPU and an Intel i7-11700 CPU.

Table 7.1: Train and inference time (per image) analysis on the ScanNet scene 0084. Our approach both trains and evaluates faster.

Method	Train (h)	Inference (s)
NeuS [231]	6.64	28.32
VolSDF [253]	8.33	29.64
ManhattanSDF [95]	16.68	28.49
MonoSDF (MLP) [257]	9.89	33.80
MonoSDF (Grid) [257]	4.36	19.13
Ours	**0.47**	**0.25**

7.4.2 Runtime Comparison

We first profile the SDF query time given a collection of points on the aforementioned machine. Specifically, we sample $k^3, k \in \{2^4, \cdots, 2^9\}$ grid points of 3D resolution k, query the SDF and their gradients, and compare the run time. This is frequently used for Marching Cubes [146] (requiring SDFs) and global Eikonal

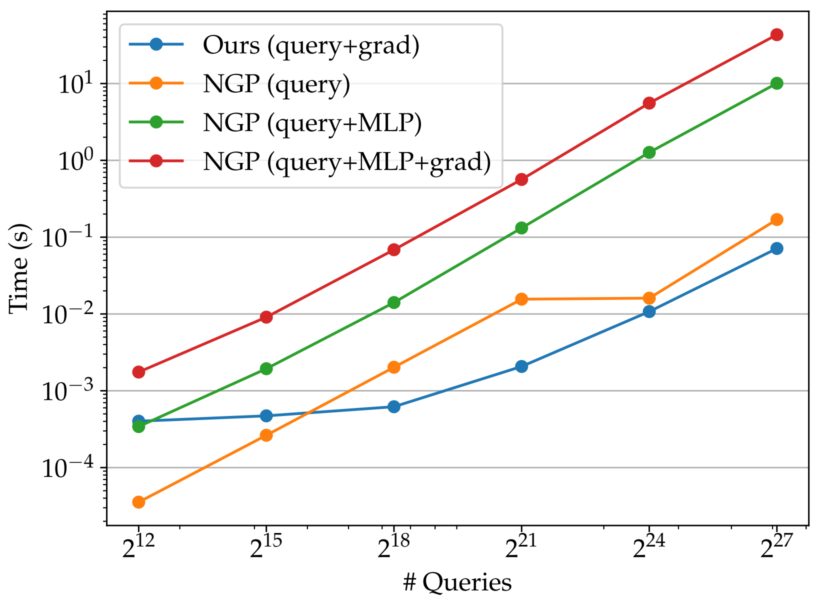


Figure 7.8: Query time comparison between ours and NGP-grid, lower is better. For end-to-end query, ours is two magnitudes faster, and maintains a high efficiency with a large number of point query. For the grid query operation itself, ours also have a better performance than multiresolution feature grids.

regularization [10] (requiring SDF gradients). We compare against MonoSDF's NGP-grid backbone that uses the multi-resolution grid from Instant-NGP [163]. In this implementation, three steps are conducted to obtain required values: query from the feature grid; SDF inference from an MLP; SDF grad computation via autograd. In contrast, ours allows its explicit computation in one forward pass, see Eq. 7.18. Fig. 7.8 shows the breakdown time comparison.

We also show the training and inference time comparison in Table 7.2. Due to the fine initialization and sparse data structure with accelerated ray sampling, our approach can complete training in less than half an hour, and allows fast rendering of color and depth at inference time.

7.4.3 Reconstruction Comparison

The comparison of reconstruction accuracy can be seen in Table 7.2. We can see that our approach achieves high accuracy at initialization that surpasses various baselines. With volume rendering and CRF refinement, it reaches comparable

Table 7.2: Quantitative comparison of reconstruction quality. While being much faster, our approach is comparable to the state-of-the-art MonoSDF [257] on ScanNet [57] and better on 7-scenes [84].

Method	ScanNet					7-Scenes				
	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑
COLMAP [202]	0.074	0.239	0.602	0.363	0.442	**0.069**	0.417	**0.536**	0.202	0.289
NeRF [160]	0.605	0.178	0.186	0.302	0.225	0.573	0.321	0.159	0.085	0.083
UNISURF [174]	0.497	0.167	0.224	0.327	0.265	0.407	0.136	0.195	0.301	0.231
NeuS [231]	0.166	0.221	0.296	0.237	0.262	0.151	0.247	0.313	0.229	0.262
VolSDF [253]	0.378	0.139	0.284	0.330	0.301	0.285	0.140	0.220	0.285	0.246
ManhattanSDF [95]	0.081	0.099	0.626	0.544	0.581	0.112	0.133	0.351	0.326	0.336
MonoSDF (MLP) [257]	**0.031**	0.057	0.783	0.652	0.710	0.097	0.192	0.441	0.311	0.361
MonoSDF (Grid) [257]	0.034	**0.046**	**0.796**	**0.711**	**0.750**	0.113	0.100	0.433	0.392	0.411
Ours (Scale Optim.)	0.058	0.064	0.655	0.605	0.627	0.151	0.080	0.367	0.462	0.409
Ours (+ Volume Rendering)	0.045	0.060	0.774	0.667	0.714	0.140	0.081	0.417	0.450	0.433
Ours (+ CRF)	0.042	0.056	0.751	0.678	0.710	0.136	**0.079**	0.436	**0.475**	**0.454**

accuracy to the state-of-the-art MonoSDF [257] on ScanNet scenes, and achieves better results on 7-scenes. The last three rows serve as the ablation study, showing a major gain from volume rendering followed by a minor refinement gain from CRF.

We also demonstrate qualitative scene-wise geometric reconstruction in Fig. 7.2, and zoomed-in details in Fig. 7.9. It is observable that while achieving similar global completeness, our method enhances details thanks to the adaptive voxel grid and direct SDF mapping from coordinates to voxels.

The control experiments of CRF's incorporated properties are visualized in Fig. 7.10, where we see that semantic labels and normals have the highest impact on reconstruction quality. Colors, on the other hand, have a lower impact mostly due to the prevalent appearance of motion blurs and exposure changes in the benchmark dataset. The same reason also affects feature-based SfM and monocular depth estimate and leads to reduced performance of our approach on certain sequences, see supplementary. We plan to incorporate more advanced semi-dense reconstruction [220, 221] for robust depth prior estimate.

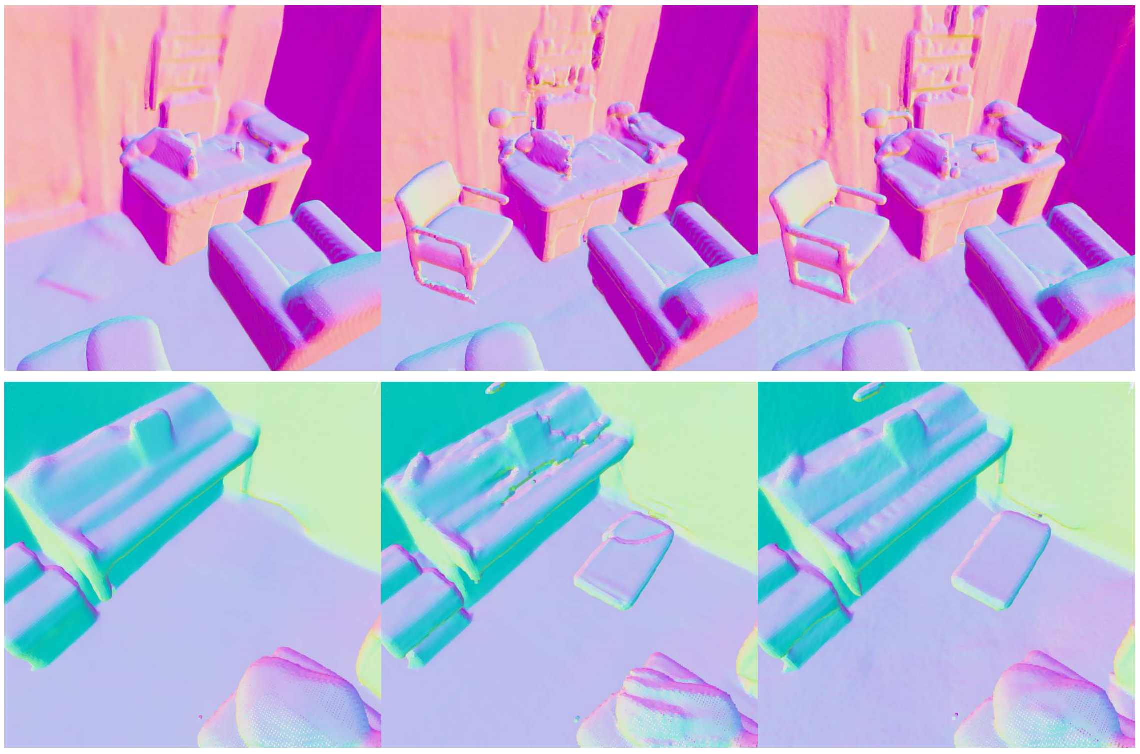


Figure 7.9: Detail comparisons between our method and current state-of-the-art neural implicit method MonoSDF [257]. We preserve better geometry details while being faster.

7.5 Conclusions

We propose an efficient monocular scene reconstruction system. Without an MLP, our model is built upon a differentiable globally sparse and locally dense data structure allocated around approximate surfaces. We develop a scale calibration algorithm to align monocular depth priors for fast geometric initialization, and apply direct refinement of voxel-level SDF and colors using differentiable rendering. We further regularize the voxel-wise properties with a high-dimensional continuous CRF that jointly refines color, geometry, and semantics in 3D. Our method is $10\times$ faster in training and $100\times$ faster in inference, while achieving similar reconstruction accuracy to state-of-the-art.

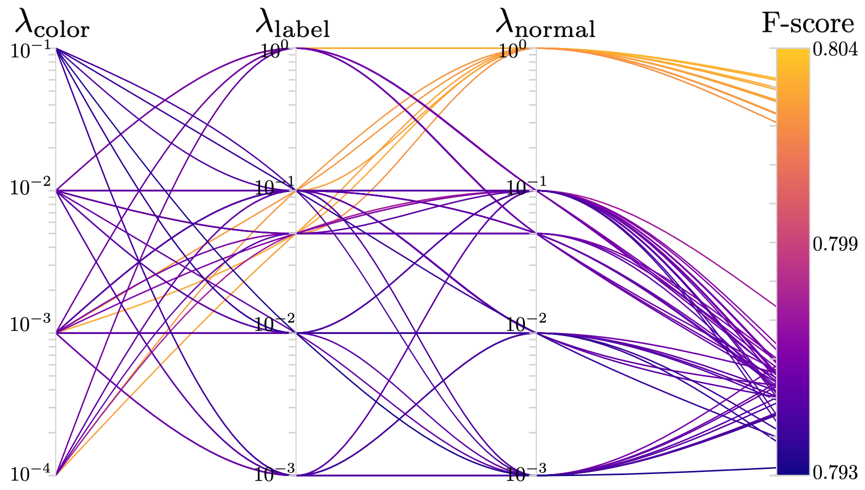


Figure 7.10: Control experiments of the CRF modules' impact to final reconstruction quality on scene 0084, see Eq. 7.22.

7.6 Appendix

7.6.1 Depth Scaler Optimization

Our system adopts monocular depth map predictions from off-the-shelf networks [70] using the DPT backbone [189]. However, these depth priors are not metric and the scale of each depth prediction is independent of others. Thus, we define the unary and binary (pairwise) constraints to estimate consistent metric scales.

Unary Constraints

Our pipeline relies on COLMAP's [202] sparse reconstruction for unary constraints. COLMAP supports sparse reconstruction with or without poses. Both modes start with SIFT [147] feature extraction and matching. The *with pose* mode then runs triangulation, while the *without pose* mode runs bundle adjustment to also estimate poses. *With pose* mode usually runs within 1 min, while the *without pose* mode often finishes around 5 mins for a sequence with several hundred frames. While our system integrates both modes, for fair comparison on the benchmark datasets, we adopt the *with pose* mode in quantitative experiments where ground truth poses from RGB-D SLAM are given. Fig. 7.11 shows the sparse reconstructions from the

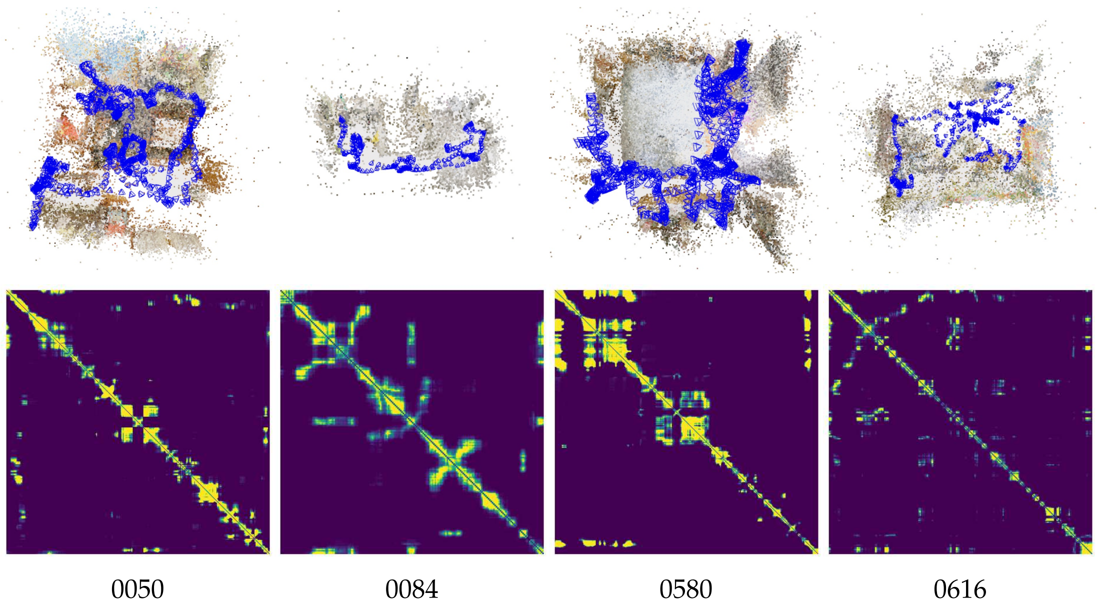


Figure 7.11: Sparse reconstruction and covisibility matrix of ScanNet scenes selected by ManhattanSDF [95].

with pose mode.

Binary Constraints

Once we have camera poses and the sparse reconstruction, we can define which triangulated feature points are visible to which cameras (covisible). Thus, we can create pairwise reprojection constraints between frames, similar to loop closures in the monocular SLAM context [165]. We directly retrieve the feature matches obtained by COLMAP, and setup such frame-to-frame covsibility constraints. Fig. 7.11 shows the covisibility matrices, where entry (i, j) indicates the number of covisible features between frame i and j. They are used to establish binary constraints between frames for refining monocular depth scales.

7.6.2 Volumetric Fusion

Eq. 9 in the main paper shows the least squares to initialize voxel-wise SDF. The more detailed implementation follows KinectFusion [168], where a truncation function ψ is used to reject associations.

$$\theta_d(\mathbf{v}) = \arg\min_d \sum_i \|d - \psi\big(d^o, \mu\big)\|^2, \tag{7.24}$$

$$d^o = d_{\mathbf{v}\to i} - \mathcal{D}_i(\mathbf{p}_{\mathbf{v}\to i})\phi_i(\mathbf{p}_{\mathbf{v}\to i}), \tag{7.25}$$

$$\psi(x, \mu) = \min(x, \mu), \tag{7.26}$$

where μ is the truncation distance. μ is associated with the *Dilate* operation and voxel block resolution in Eq. 7-8 in the main paper. Formally, we define

$$\mathrm{Dilate}_R(\mathbf{x}) = \left\{\mathbf{x}_i \mid \left\|\mathbf{x}_i - \left\lfloor\frac{\mathbf{x}}{L}\right\rfloor\right\|_0 \leq R\right\}, \tag{7.27}$$

where L is the voxel block size, $\mathbf{x}_i$ are quantized grid points around, and R is the dilation radius. We use $R = 2$ (corresponding to two 8^3 voxel blocks) to account for the uncertainty around surfaces from the monocular depth prediction. Correspondingly, we use $\mu = L \cdot R$ to truncate the SDF.

The volumetric fusion runs at 50 Hz with RGB and SDF fusion, and at 30 Hz when additional semantic labels are also fused, hence serves as a fast initializer.

7.6.3 Hyper Parameters

We followed [257]'s hyperparameter choices and used $\lambda_d = 0.1, \lambda_n = 0.05$ for the rendering loss.

For regularizors, we obtained from hyper param sweeps from the 0084 scene of ScanNet that $\lambda_{\text{eik}} = 0.1$ for the Eikonal loss, and $\lambda_{\text{color}} = 10^{-3}$, $\lambda_{\text{label}} = 0.1$, $\lambda_{\text{normal}} = 1$ for the CRF loss.

In Gaussian kernels, we fix $\sigma_{\text{sdf}} = 1.0$ and $\sigma_{\text{color}} = 0.1$.

7.6.4 Evaluation

Metrics

We follow the evaluation protocols defined by ManhattanSDF [95], where the metrics between predicted point set P and ground truth point set P^* are

$$D(p, p^*) = \|p - p^*\|, \tag{7.28}$$

$$\mathrm{D}_{\mathrm{Acc}}(P, P^*) = \operatorname*{mean}_{p \in P} \min_{p^* \in P^*} D(p, p^*), \tag{7.29}$$

$$\mathrm{D}_{\mathrm{Comp}}(P, P^*) = \operatorname*{mean}_{p^* \in P^*} \min_{p \in P} D(p, p^*), \tag{7.30}$$

$$\mathrm{Prec}(P, P^*) = \operatorname*{mean}_{p \in P} \Big(\big(\min_{p^* \in P^*} D(p, p^*) \big) < T \Big), \tag{7.31}$$

$$\mathrm{Recall}(P, P^*) = \operatorname*{mean}_{p^* \in P^*} \Big(\big(\min_{p \in P} D(p, p^*) \big) < T \Big), \tag{7.32}$$

$$\text{F-score}(P, P^*) = \frac{2 \cdot \mathrm{Prec} \cdot \mathrm{Recall}}{\mathrm{Prec} + \mathrm{Recall}}, \tag{7.33}$$

where $T = 5$cm.

Generation of P and P^*

We follow previous works [95, 257] that applied TSDF refusion to generate P for evaluation: use Marching Cubes [146] to generate a global mesh; render depth map from mesh at selected viewpoints to crop points out of viewports; apply TSDF fusion [272] to obtain the final mesh and point cloud P. For fairness, we render depth at the resolution 480×640 for all approaches to be consistent with input (in contrast to MonoSDF that uses 968×1296 in their released evaluation code), and conduct refusion to a voxel grid at the resolution of 1cm.

To ensure the same surface coverage, we generate ground truth P^* at the same viewpoints with the same image and voxel resolution, only replacing rendered depth with ground truth depth obtained by an RGB-D sensor.

7.6.5 Additional Experimental Results

Ablation of scale optimization

To further illustrate the necessity of per-frame scale optimization, we show quantitative reconstruction results without scale optimization in Table 7.3. Here, volumetric fusion is conducted on an estimated single scale factor across all frames between monocular depth and SfM, resulting in poor initial reconstruction.

Table 7.3: Initial reconstruction results without per-frame scale optimization (*c. f.* Ours (Init) in Table 7.4-7.5.)

	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑
ScanNet	0.42	0.19	0.13	0.28	0.17
7-Scenes	0.36	0.12	0.19	0.43	0.26

Fusion and Refinement

Please see video supplementary for the incremental fusion from scaled depth, and the refinement stage that converges to general shapes within several hundred steps.

Scene-wise statistics on ScanNet

We use reconstructed mesh provided by ManhattanSDF [95], and report scene-wise statistics in Table 7.4. Reconstructions and corresponding ground truths are shown in Fig. 7.12.

It is observable that our reconstructions have low error at fine details with rich textures (*e.g.* 0050, furniture in 0580), but problems exist at texture-less regions (*e.g.* walls in 0580 and 0616, floor in 0084) due to the inaccurate scale estimate from sparse reconstructions. We plan to improve these by learning-based sparse or semi-dense reconstruction, *e.g.* [220, 221].

Scene-wise statistics on 7-scenes

The reconstructed mesh and scene-wise statistics are not provided by ManhattanSDF [95] for COLMAP, NeRF, UNISURF, NeuS, VolSDF, and ManhattanSDF. Therefore, we reuse their reported averages as a reference in the main paper. Here we report scene-wise numbers in Table 7.5 for the state-of-the-art MonoSDF [257] and our method. Reconstructions and ground truths are in Fig. 7.13.

7-scenes have challenging camera motion patterns and complex scenes, thus the overlaps between viewpoints are small, leading to reduced accuracy for all the approaches. Although our approach produces less accurate floor and walls with fewer features, it achieves fine reconstruction of desktop objects in general.

Table 7.4: Scene-wise quantitative results on ScanNet.

Method	0050					0084				
	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑
COLMAP [202]	0.049	0.129	0.707	0.531	0.607	0.032	0.121	0.807	0.577	0.673
NeRF [160]	0.704	0.081	0.215	0.517	0.304	0.733	0.248	0.157	0.213	0.181
UNISURF [174]	0.432	0.087	0.309	0.482	0.376	0.594	0.242	0.218	0.339	0.266
NeuS [231]	0.091	0.103	0.528	0.455	0.489	0.231	0.365	0.159	0.090	0.115
VolSDF [253]	0.071	0.071	0.600	0.599	0.599	0.507	0.165	0.163	0.247	0.196
ManhattanSDF [95]	0.032	0.050	0.849	0.755	0.800	**0.029**	**0.041**	**0.822**	**0.784**	**0.802**
MonoSDF (MLP) [257]	**0.025**	0.054	0.865	0.713	0.781	0.036	0.048	0.700	0.646	0.672
MonoSDF (Grid) [257]	0.027	0.045	0.854	0.764	0.807	0.035	0.043	0.796	0.774	0.785
Ours (Init)	0.034	0.051	0.775	0.684	0.727	0.047	0.048	0.705	0.725	0.715
Ours (+Rendering)	0.026	**0.044**	0.875	0.780	0.825	0.038	0.046	0.762	0.748	0.755
Ours (+CRF)	0.026	**0.044**	**0.880**	**0.788**	**0.832**	0.043	0.043	0.750	0.780	0.765

Method	0580					0616				
	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑
COLMAP [202]	0.169	0.300	0.204	0.112	0.145	0.045	0.406	0.689	0.230	0.344
NeRF [160]	0.402	0.186	0.125	0.216	0.159	0.582	0.196	0.249	0.263	0.256
UNISURF [174]	0.392	0.192	0.131	0.188	0.155	0.571	0.148	0.237	0.300	0.265
NeuS [231]	0.206	0.275	0.167	0.114	0.135	0.137	0.140	0.330	0.289	0.308
VolSDF [253]	0.197	0.183	0.197	0.189	0.193	0.736	0.129	0.176	0.284	0.217
ManhattanSDF [95]	0.205	0.240	0.149	0.124	0.135	0.058	0.066	0.684	0.513	0.586
MonoSDF (MLP) [257]	**0.025**	**0.040**	**0.867**	**0.759**	**0.809**	0.039	0.087	0.702	0.488	0.576
MonoSDF (Grid) [257]	0.039	0.048	0.718	0.661	0.688	**0.033**	**0.048**	**0.815**	**0.646**	**0.721**
Ours (Init)	0.076	0.059	0.574	0.582	0.578	0.076	0.097	0.566	0.427	0.487
Ours (+Rendering)	0.070	0.080	0.760	0.636	0.692	0.046	0.070	0.699	0.504	0.586
Ours (+CRF)	0.046	0.050	0.707	0.682	0.694	0.057	0.080	0.659	0.504	0.571

Table 7.5: Scene-wise quantitative results on 7-Scenes.

Method	chess					heads				
	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑
MonoSDF (MLP) [257]	0.160	0.390	0.250	0.132	0.173	**0.068**	0.188	**0.586**	0.353	0.440
MonoSDF (Grid) [257]	**0.113**	0.143	0.324	0.267	0.293	0.133	0.099	0.305	0.327	0.315
Ours (Init)	0.164	0.108	0.278	0.350	0.310	0.186	0.083	0.288	0.401	0.335
Ours (+Rendering)	0.147	0.111	0.367	0.389	0.378	0.074	0.062	0.543	0.568	0.555
Ours (+CRF)	0.147	**0.107**	**0.368**	**0.391**	**0.379**	0.071	**0.057**	0.559	**0.626**	**0.591**

Method	office					fire				
	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑	Acc ↓	Comp ↓	Prec ↑	Recall ↑	F-score ↑
MonoSDF (MLP) [257]	**0.087**	0.128	0.338	0.236	0.278	0.075	0.064	**0.592**	0.522	**0.555**
MonoSDF (Grid) [257]	0.147	0.077	**0.539**	0.471	**0.503**	**0.061**	0.081	0.564	0.504	0.533
Ours (Init)	0.168	**0.068**	0.398	**0.483**	0.436	0.087	**0.058**	0.503	**0.616**	0.554
Ours (+Rendering)	0.180	0.081	0.330	0.400	0.362	0.160	0.072	0.426	0.445	0.435
Ours (+CRF)	0.164	0.080	0.340	0.400	0.367	0.162	0.068	0.474	0.490	0.482

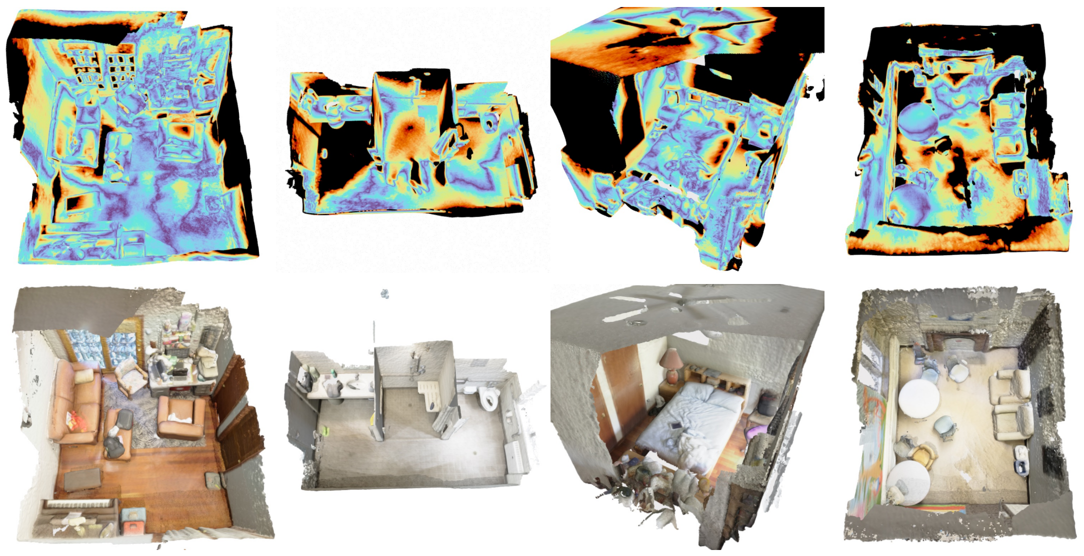


Figure 7.12: Error heatmap from our reconstruction (first row) to groundtruth (second row) for each scene in ScanNet [57]. Points are colorized by distance error ranging from 0 (blue) to 5cm (red) to its nearest neighbor in ground truth. Points with error larger than 5cm are regarded as outliers and colored in black.

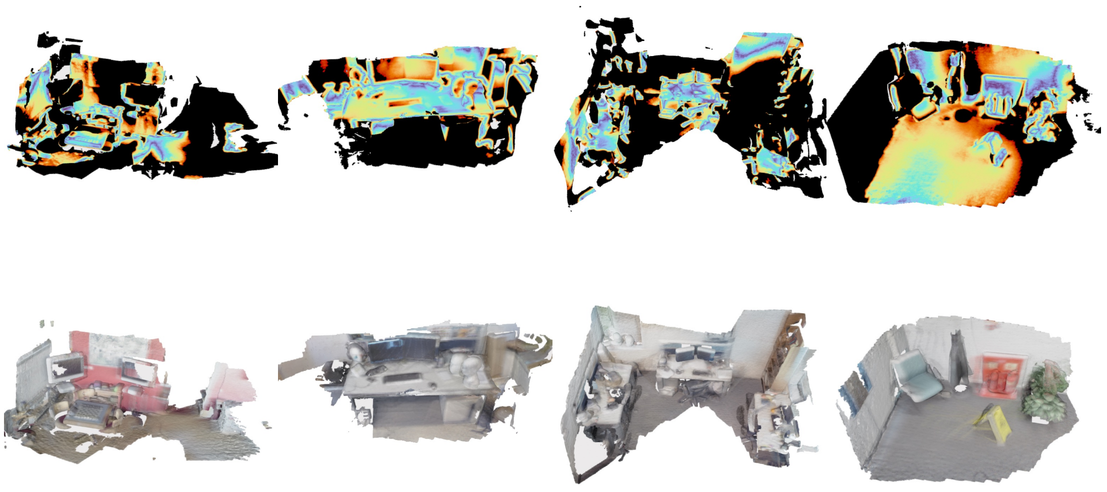


Figure 7.13: Error heatmap from our reconstruction (first row) to groundtruth (second row) for each scene in 7-Scenes [84]. The colorization is the same as Fig. 7.12.

Chapter 8

Conclusions

8.1 Contributions

In this thesis, I have developed reconstruction systems aimed at high-quality dense surface reconstruction for RGB-D, LiDAR, and monocular data. By emphasizing spatial sparsity in 3D, I have improved the quality and speed of global registration and surface reconstruction across various scenarios. The main contributions of this thesis can be summarized as follows:

- A hierarchical reconstruction system that serves as a foundation for the development and integration of various registration and reconstruction techniques.
- Novel supervised and self-supervised approaches for global registration, leveraging sparse convolutional networks to learn point-wise features and their correspondences, significantly improving registration speed and accuracy.
- A high-performance spatial hashing engine that enables efficient scene representation, demonstrated through its applications in dense RGB-D SLAM, large-scale LiDAR surface reconstruction, and fast monocular scene reconstruction with better performance and less effort of development.

8.2 Future Works and Open Problems

The immediate ongoing work is the combination of the spatial hashing engine with neural networks. In contrast to Instant-NGP [163], our approach, with collision-free hashing, can precisely capture the spatial sparsity and neighbors of the place-of-interest at the encoding stage. I hope this will lead to faster convergence and better detail recovery, especially for the large-scale surface reconstruction task. I also aim to study how the sampler from the spatial sparse data structure can be enhanced by neural samplers [15, 16], which can quickly skip *hard* empty spaces and allow *soft* sampling in occupied space simultaneously.

In the long term, there are various open problems to investigate.

Semantic Information

We can extract geometric sparsity purely from input data, through which we can extract, correspond, and register point cloud features and conduct surface reconstruction in regions of interest. However, they face problems with insufficient input. For instance, learned features can be inaccurate on smaller objects with much sparser measurements, leading to registration failure; occlusions during scans will cause incomplete reconstruction of larger scenes. To address such problems, I hope to incorporate *multi-modal* and *semantic* information so that objects and scenes can be completed, better sampled, registered, and reconstructed properly. A direct application will be densifying sparse LiDAR scans along with RGB data annotated with rich semantic labels and reconstructing complete scenes at scale.

Dynamic Scenes

While the 3D sparsity can be intuitively captured through measurements in the static setup, properly defining sparsity in the spatio-temporal 4D space is yet to be revealed. We need to determine whether we should maintain sparsity in the 3D canonical space [170] and deform it in time or simply encode it in the 4D space [185]. We also need to consider whether volumetric space division is still a good candidate or if the curse of dimensions (*e.g.*the number of 1-radius neighbors grows from $3^3 = 27$ to $3^4 = 81$) will force us to adopt another representation, such as permutohedral lattices [194]. To investigate these questions, we need theoretical proofs, controlled experiments, and comprehensive datasets.

Ultimately, we are to reconstruct the real world in 3D at scale with permanent static components (floors and walls), temporary static objects (furniture and plants), and dynamic objects (humans, animals, and interactable small objects) through semantic understandings and dynamic modeling, and develop practical interactive applications with tractable computation enabled by exploiting sparsity.

Bibliography

[1] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum*, volume 29, pages 753–762. Wiley Online Library, 2010.

[2] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[3] Dror Aiger, Niloy J. Mitra, and Daniel Cohen-Or. 4-points congruent sets for robust surface registration. *ACM Trans. on Graphics*, 27(3):85, 1–10, 2008.

[4] Dan A Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D Owens, and Nina Amenta. Real-time parallel hashing on the GPU. *ACM Trans. on Graphics*, 28(5):154, 2009.

[5] Pasquale Antonante, Vasileios Tzoumas, Heng Yang, and Luca Carlone. Outlier-robust estimation: Hardness, minimally-tuned algorithms, and applications. *arXiv preprint arXiv:2007.15109*, 2020.

[6] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. PointNetLK: Robust & efficient point cloud registration using PointNet. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2019.

[7] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-9(5):698–700, 1987.

[8] Saman Ashkiani, Martin Farach-Colton, and John D. Owens. A dynamic hash table for the gpu. In *IEEE IPDPS*, 2018.

[9] Erik Ask, Olof Enqvist, and Fredrik Kahl. Optimal geometric fitting under the truncated l2-norm. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1722–1729, 2013.

[10] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 2565–2574, 2020.

[11] Xuyang Bai, Zixin Luo, Lei Zhou, Hongbo Fu, Long Quan, and Chiew-Lan Tai. D3feat: Joint learning of dense detection and description of 3d local

features. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2020.

[12] Daniel Barath and Jiří Matas. Graph-cut RANSAC. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 6733–6741, 2018.

[13] Daniel Barath, Jana Noskova, Maksym Ivashechkin, and Jiri Matas. MAGSAC++, a fast, reliable and accurate robust estimator. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1304–1312, 2020.

[14] Jonathan T Barron. A general and adaptive robust loss function. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 4331–4339, 2019.

[15] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *arXiv preprint arXiv:2111.12077*, pages 5855–5864, 2021.

[16] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *arXiv*, 2023.

[17] Jean-Charles Bazin, Hongdong Li, In So Kweon, Cédric Demonceaux, Pascal Vasseur, and Katsushi Ikeuchi. A branch-and-bound approach to correspondence and grouping problems. *IEEE Trans. Pattern Anal. Machine Intell.*, 35(7): 1565–1576, 2012.

[18] Jean-Charles Bazin, Yongduek Seo, and Marc Pollefeys. Globally optimal consensus set maximization through rotation search. In *Asian Conf. on Computer Vision (ACCV)*, pages 539–551. Springer, 2012.

[19] Jens Behley and C. Stachniss. Efficient surfel-based SLAM using 3d laser range data in urban environments. In *Robotics: Science and Systems (RSS)*, 2018.

[20] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[21] Dimitri Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[22] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Machine Intell.*, 14(2):239–256, 1992. doi: 10.1109/34.121791.

[23] Igor Bogoslavskyi and Cyrill Stachniss. Fast range image-based segmentation of sparse 3D laser scans for online operation. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 163–169. IEEE, 2016.

[24] Michael Bosse, Gabriel Agamennoni, and Igor Gilitschenski. Robust estimation and applications in robotics. *Foundations and Trends in Robotics*, 4(4): 225–269, 2016. ISSN 1935-8253. doi: 10.1561/2300000047.

[25] Sofien Bouaziz, A. Tagliasacchi, and M. Pauly. Sparse iterative closest point.

Computer Graphics Forum, 32, 2013.

[26] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *Eur. Conf. on Computer Vision (ECCV)*, pages 536–551. Springer, 2014.

[27] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[28] Á. Parra Bustos and T. J. Chin. Guaranteed outlier removal for point cloud registration with correspondences. *IEEE Trans. Pattern Anal. Machine Intell.*, 40(12):2868–2882, 2018.

[29] Erik Bylow, Jürgen Sturm, Christian Kerl, Fredrik Kahl, and Daniel Cremers. Direct camera pose tracking and 3d reconstruction with signed distance functions. In *Robotics: Science and Systems (RSS)*, 2013.

[30] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, 32(6):1309–1332, 2016.

[31] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 11618–11628, 2020.

[32] Qin Cai, David Gallup, Cha Zhang, and Zhengyou Zhang. 3D deformable face tracking with a commodity depth camera. In *Eur. Conf. on Computer Vision (ECCV)*, 2010.

[33] Zhipeng Cai, Tat-Jun Chin, and Vladlen Koltun. Consensus maximization tree search revisited. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1637–1645, 2019.

[34] Chao Cao, Hongbiao Zhu, Howie Choset, and Ji Zhang. Tare: A hierarchical framework for efficiently exploring complex 3d environments. In *Robotics: Science and Systems (RSS)*, 2021.

[35] Heng Yang Carlone and Luca. A polynomial-time solution for robust registration with extreme outlier rates. In *Robotics: Science and Systems (RSS)*, 2019.

[36] Daniele Cattaneo, Matteo Vaghi, Augusto Luis Ballardini, Simone Fontana, Domenico Giorgio Sorrenti, and Wolfram Burgard. CMRNet: Camera to lidar-map registration. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1283–1289, Oct 2019. doi: 10.1109/ITSC.2019.8917470.

[37] Pedro Celis, Per-Ake Larson, and J Ian Munro. Robin hood hashing. In *Annual Symposium on Foundations of Computer Science*, 1985.

[38] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *Eur. Conf. on Computer Vision (ECCV)*, 2020.

[39] Ming-Fang Chang, Wei Dong, Joshua Mangelson, Michael Kaess, and Simon Lucey. Map compressibility assessment for lidar registration. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5560–5567. IEEE.

[40] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3D tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[41] Bo Chen, Jiewei Cao, Alvaro Parra, and Tat-Jun Chin. Satellite pose estimation with deep landmark regression and nonlinear pose refinement. In *Intl. Conf. on Computer Vision Workshops (ICCVW)*, 2019.

[42] Xieyuanli Chen, Andres Milioto Emanuele Palazzolo, P. Giguère, Jens Behley, and C. Stachniss. Suma++: Efficient lidar-based semantic SLAM. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4530–4537, 2019.

[43] Xieyuanli Chen, Ignacio Vizzo, Thomas Läbe, Jens Behley, and Cyrill Stachniss. Range image-based lidar localization for autonomous vehicles. *arXiv preprint arXiv:2105.12121*, 2021.

[44] T. J. Chin and D. Suter. The maximum consensus problem: recent algorithmic advances. *Synthesis Lectures on Computer Vision*, 7(2):1–194, 2017.

[45] Tat-Jun Chin, Pulak Purkait, Anders Eriksson, and David Suter. Efficient globally optimal consensus maximisation with tree search. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 2413–2421, 2015.

[46] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2015.

[47] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016.

[48] Christopher Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *Proc. Conf. on Neural Information Processing Systems*, pages 2414–2422, 2016.

[49] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal

convnets: Minkowski convolutional neural networks. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[50] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *Intl. Conf. on Computer Vision (ICCV)*, pages 8958–8966, 2019.

[51] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep global registration. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 2514–2523, 2020.

[52] Christopher Choy, Junha Lee, Rene Ranftl, Jaesik Park, and Vladlen Koltun. High-dimensional convolutional networks for geometric pattern recognition. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, June 2020.

[53] Ronald Clark. Volumetric bundle adjustment for online photorealistic scene capture. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 6124–6132, 2022.

[54] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. *arXiv preprint arXiv:1801.10130*, 2018.

[55] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *Intl. J. of Robotics Research*, 27(6):647–665, 2008.

[56] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312. ACM, 1996.

[57] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 5828–5839, 2017.

[58] Angela Dai, Matthias Nießner, Michael Zollöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. *ACM Trans. on Graphics*, 2017.

[59] Frank Dellaert and Michael Kaess. Factor graphs for robot perception. *Foundations and Trends in Robotics*, 6(1-2):1–139, 2017.

[60] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPFNet: Global context aware local features for robust 3D point matching. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2018.

[61] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, June 2022.

[62] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint:

Self-supervised interest point detection and description. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition Workshops*, pages 224–236, 2018.

[63] Wei Dong, Jieqi Shi, Weijie Tang, Xin Wang, and Hongbin Zha. An efficient volumetric mesh representation for real-time scene reconstruction using spatial hashing. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018.

[64] Wei Dong, Qiuyuan Wang, Xin Wang, and Hongbin Zha. PSDF fusion: Probabilistic signed distance function for on-the-fly 3D data fusion and scene reconstruction. In *Eur. Conf. on Computer Vision (ECCV)*, pages 701–717, 2018.

[65] Wei Dong, Jaesik Park, Yi Yang, and Michael Kaess. Gpu accelerated robust scene reconstruction. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 7863–7870. IEEE, 2019.

[66] Wei Dong, Ryu Kwonyoung, , Michael Kaess, and Jaesik Park. Revisiting LiDAR registration and reconstruction: A range image perspective. In *arXiv:2112.02779*, 2022.

[67] Wei Dong, Yixing Lao, Michael Kaess, and Vladlen Koltun. ASH: A modern framework for parallel spatial hashing in 3D perception. *IEEE Trans. Pattern Anal. Machine Intell.*, October 2022.

[68] Wei Dong, Chris Choy, Charles Loop, Yuke Zhu, Or Litany, and Anima Anandkumar. Fast monocular scene reconstruction with global-sparse local-dense grids. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2023.

[69] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[70] Ainaz Eftekhar, Alexander Sax, Jitendra Malik, and Amir Zamir. Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *Intl. Conf. on Computer Vision (ICCV)*, pages 10786–10796, 2021.

[71] Sovann En, Alexis Lechervy, and Frédéric Jurie. Rpnet: An end-to-end network for relative camera pose estimation. In *Eur. Conf. on Computer Vision Workshops (ECCV)*, pages 0–0, 2018.

[72] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Trans. Pattern Anal. Machine Intell.*, 40(3):611–625, 2017.

[73] Olof Enqvist, Erik Ask, Fredrik Kahl, and Kalle Åström. Robust fitting for multiple view geometry. In *Eur. Conf. on Computer Vision (ECCV)*, pages 738–751. Springer, 2012.

[74] Nicola Fioraio, Jonathan Taylor, Andrew Fitzgibbon, Luigi Di Stefano, and Shahram Izadi. Large-scale and drift-free surface reconstruction using online

subvolume registration. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2015.

[75] Martin Fischler and Robert Bolles. Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Graphics and Image Processing*, 24:381–395, 1981.

[76] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Proc. Conf. Robot Learning (CORL)*, 2018.

[77] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

[78] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *IEEE Trans. Pattern Anal. Machine Intell.*, 25(8):930–943, 2003.

[79] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *arXiv preprint arXiv:2103.10380*, 2021.

[80] Ismael García, Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. Coherent parallel hashing. *ACM Trans. on Graphics*, 30(6):1–8, 2011.

[81] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast K nearest neighbor search using GPU. *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition Workshops*, 2008.

[82] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2012.

[83] Donald Geman and George Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE Trans. Pattern Anal. Machine Intell.*, (3): 367–383, 1992.

[84] Ben Glocker, Shahram Izadi, Jamie Shotton, and Antonio Criminisi. Real-time rgb-d camera relocalization. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. IEEE, October 2013. URL `https://www.microsoft.com/en-us/research/publication/real-time-rgb-d-camera-relocalization/`.

[85] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *Intl. Conf. on Computer Vision (ICCV)*, pages 3828–3838, 2019.

[86] Zan Gojcic, Caifa Zhou, Jan Dirk Wegner, and Wieser Andreas. The perfect match: 3d point cloud matching with smoothed densities. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2019.

[87] Zan Gojcic, Caifa Zhou, Jan D Wegner, Leonidas J Guibas, and Tolga Birdal. Learning multiview 3d point cloud registration. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1759–1769, 2020.

[88] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[89] Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks: A new hope. *arXiv preprint arXiv:1909.04866*, 2019.

[90] John C Gower. Generalized procrustes analysis. *Psychometrika*, 1975.

[91] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2018.

[92] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Proc. Conf. on Neural Information Processing Systems*, pages 529–536, 2005.

[93] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[94] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Real time ray tracing on gpu with bvh-based packet traversal. In *Symposium on Interactive Ray Tracing*, 2007.

[95] Haoyu Guo, Sida Peng, Haotong Lin, Qianqian Wang, Guofeng Zhang, Hujun Bao, and Xiaowei Zhou. Neural 3d scene reconstruction with the manhattan-world assumption. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 5511–5520, 2022.

[96] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3D.net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017.

[97] Maciej Halber and Thomas Funkhouser. Fine-to-coarse global registration of RGB-D scans. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2017.

[98] David S. Hall. High definition lidar system, US7969558B2.

[99] Lei Han and Lu Fang. Flashfusion: Real-time globally consistent dense 3d reconstruction using cpu computing. In *Robotics: Science and Systems (RSS)*, 2018.

[100] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[101] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[102] Mark Harris. Optimizing parallel reduction in CUDA. *Developer technology report*, 2007.

[103] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[104] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D lidar SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[105] Rama Karl Hoetzlein. Gvdb: Raytracing sparse voxel database structures on the gpu. In *Proceedings of High Performance Graphics*. 2016.

[106] Dirk Holz, Alexandru E Ichim, Federico Tombari, Radu B Rusu, and Sven Behnke. Registration with the point cloud library: A modular framework for aligning in 3-D. *IEEE Robotics and Automation Letters*, 2015.

[107] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Amer.*, 4(4):629–642, Apr 1987.

[108] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[109] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Trans. on Graphics*, 38(6):1–16, 2019.

[110] Xiangru Huang, Zhenxiao Liang, Xiaowei Zhou, Yao Xie, Leonidas J Guibas, and Qixing Huang. Learning transformation synchronization. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 8082–8091, 2019.

[111] Gregory Izatt, Hongkai Dai, and Russ Tedrake. Globally optimal object pose estimation in point clouds with mixed-integer programming. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, 2017.

[112] Wenzel Jakob. Enoki: structured vectorization and differentiation on modern processor architectures, 2019. https://github.com/mitsuba-renderer/enoki.

[113] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

[114] Huaizu Jiang, Gustav Larsson, Michael Maire Greg Shakhnarovich, and Erik Learned-Miller. Self-supervised relative depth learning for urban scene understanding. In *Eur. Conf. on Computer Vision (ECCV)*, pages 19–35, 2018.

[115] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Trans. Pattern Anal. Machine Intell.*, 2020.

[116] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Trans. Pattern Anal. Machine Intell.*, 1999.

[117] Daniel Jünger, Robin Kobus, André Müller, Christian Hundt, Kai Xu, Weiguo Liu, and Bertil Schmidt. Warpcore: A library for fast hash tables on gpus. *arXiv preprint arXiv:2009.07914*, 2020.

[118] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the bayes tree. *Intl. J. of Robotics Research*, 31(2):216–235, 2012.

[119] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S Torr, and D. W. Murray. Very high frame rate volumetric integration of depth images on mobile device. *IEEE Transactions on Visualization and Computer Graphics*, 22(11), 2015.

[120] Olaf Kähler, V. A. Prisacariu, and David W. Murray. Real-time large-scale dense 3D reconstruction with loop closure. In *Eur. Conf. on Computer Vision (ECCV)*, 2016.

[121] Tomas Karnagel, René Müller, and Guy M Lohman. Optimizing gpu-accelerated group-by and aggregation. *VLDB*, 8:20, 2015.

[122] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *Proceedings of 3DV*. IEEE, 2013.

[123] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual SLAM for RGB-D cameras. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[124] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for RGB-D cameras. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2013.

[125] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. Learning compact geometric features. In *Intl. Conf. on Computer Vision (ICCV)*, 2017.

[126] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[127] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, pages 225–234. IEEE, 2007.

[128] Matthew Klingensmith, Ivan Dryanovski, Siddhartha Srinivasa, and Jizhong Xiao. CHISEL: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems (RSS)*, 2015.

[129] Laurent Kneip, Hongdong Li, and Yongduek Seo. UPnP: An optimal o(n) solution to the absolute pose problem with universal applicability. In *Eur. Conf. on Computer Vision (ECCV)*, pages 127–142. Springer, 2014.

[130] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. Robust consistent video depth estimation. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2021.

[131] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Proc. Conf. on Neural Information Processing Systems*, 2011.

[132] Alexey Kukanov and Michael J Voss. The foundations for scalable multi-core software in intel threading building blocks. *Intel Technology Journal*, 11(4), 2007.

[133] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[134] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast bvh construction on gpus. In *Computer Graphics Forum*, volume 28, pages 375–384, 2009.

[135] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 4681–4690, 2017.

[136] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, 2013.

[137] Junha Lee, Seungwook Kim, Minsu Cho, and Jaesik Park. Deep Hough voting for robust global registration. 2021.

[138] Brenton Lessley and Hank Childs. Data-parallel hashing techniques for gpu architectures. *IEEE Transactions on Parallel and Distributed Systems*, 31(1): 237–250, 2019.

[139] Boyi Li, Kilian Q Weinberger, Serge Belongie, Vladlen Koltun, and Rene Ranftl. Language-driven semantic segmentation. In *Intl. Conf. on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=RriDjddCLN.

[140] Hongdong Li. Consensus set maximization with guaranteed global optimality for robust geometry estimation. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1074–1080. IEEE, 2009.

[141] Jiaxin Li and Gim Hee Lee. USIP: Unsupervised stable interest point detection from 3d point clouds. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 361–370, 2019.

[142] Yin Li, Manohar Paluri, James M Rehg, and Piotr Dollár. Unsupervised learning of edges. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1619–1627, 2016.

[143] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 2041–2050, 2018.

[144] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. Learning the depths of moving people by watching frozen people. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 4521–4530, 2019.

[145] Pengpeng Liu, Michael Lyu, Irwin King, and Jia Xu. Selflow: Self-supervised learning of optical flow. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 4571–4580, 2019.

[146] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH*, volume 21, pages 163–169. ACM, 1987.

[147] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[148] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *DARPA Image Understanding Workshop*, 1981.

[149] João Maciel and Joao Paulo Costeira. A global solution to sparse correspon-

dence problems. *IEEE Trans. Pattern Anal. Machine Intell.*, 2003.

[150] Robert Maier, Kihwan Kim, Daniel Cremers, Jan Kautz, and Matthias Nießner. Intrinsic3d: High-quality 3d reconstruction by joint appearance and geometry optimization with spatially-varying lighting. In *Intl. Conf. on Computer Vision (ICCV)*, pages 3114–3122, 2017.

[151] Ameesh Makadia, Alexander Patterson, and Kostas Daniilidis. Fully automatic registration of 3D point clouds. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2006.

[152] Fahira Afzal Maken, Fabio Ramos, and Lionel Ott. Speeding up iterative closest point using stochastic gradient descent. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2019.

[153] Joshua G Mangelson, Derrick Dominic, Ryan M Eustice, and Ram Vasudevan. Pairwise consistent measurement set maximization for robust multi-robot map merging. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2916–2923. IEEE, 2018.

[154] Haggai Maron, Nadav Dym, Itay Kezurer, Shahar Kovalsky, and Yaron Lipman. Point registration via efficient convex relaxation. *ACM Trans. on Graphics*, 2016.

[155] John McCormac, Ronald Clark, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. Fusion++: Volumetric object-level slam. In *2018 international conference on 3D vision (3DV)*, pages 32–41. IEEE, 2018.

[156] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[157] Iaroslav Melekhov, Juha Ylioinas, Juho Kannala, and Esa Rahtu. Relative camera pose estimation using convolutional neural networks. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 675–687. Springer, 2017.

[158] Nicolas Mellado, Dror Aiger, and Niloy J Mitra. Super 4PCS fast global pointcloud registration via smart indexing. *Computer Graphics Forum*, 2014.

[159] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Eur. Conf. on Computer Vision (ECCV)*, pages 405–421. Springer, 2020.

[160] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[161] Alexander Millane, Zachary Taylor, Helen Oleynikova, Juan Nieto, Roland Siegwart, and César Cadena. C-blox: A scalable and consistent TSDF-based dense mapping approach. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[162] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Machine Intell.*, 36, 2014.

[163] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv:2201.05989*, January 2022.

[164] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras. *IEEE Trans. Robotics*, 33 (5):1255–1262, 2017.

[165] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Trans. Robotics*, 31(5): 1147–1163, 2015.

[166] Zak Murez, Tarrence van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich. Atlas: End-to-end 3d scene reconstruction from posed images. In *Eur. Conf. on Computer Vision (ECCV)*, pages 414–431. Springer, 2020. URL https://arxiv.org/abs/2003.10432.

[167] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. Openvdb: an open-source data structure and toolkit for high-resolution volumes. In *ACM SIGGRAPH Courses*, pages 1–1. 2013.

[168] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. IEEE, 2011.

[169] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Intl. Conf. on Computer Vision (ICCV)*, page Intl. Conf. on Computer Vision (ICCV). IEEE, 2011.

[170] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2015.

[171] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. on Graphics*, 32(6):169, 2013.

[172] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. on Graphics*, 38(6):1–17, 2019.

[173] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Machine Intell.*, 26(6):756–770, 2004.

[174] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Intl. Conf. on Computer Vision (ICCV)*, pages 5589–5599, 2021.

[175] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1366–1373. IEEE, 2017.

[176] Angus Pacala, Mark Frichtl, Marvin Shu, and Eric Younge. Rotating compact light ranging system, US10481269B2.

[177] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[178] G. Dias Pais, Pedro Miraldo, Srikumar Ramalingam, Jacinto C. Nascimento, Venu Madhav Govindu, and Rama Chellappa. 3DRegNet: A deep neural network for 3D point registration. *arXiv*, 2019.

[179] Chanoh Park, Peyman Moghadam, Soohwan Kim, Alberto Elfes, Clinton Fookes, and Sridha Sridharan. Elastic lidar fusion: Dense map-centric continuous-time SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1206–1213. IEEE, 2018.

[180] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Colored point cloud registration revisited. In *Intl. Conf. on Computer Vision (ICCV)*, 2017.

[181] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[182] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proc. Conf. on Neural Information Processing Systems*. 2019.

[183] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. PVNet: Pixel-wise voting network for 6dof pose estimation. In *Proc. IEEE Int. Conf.*

Computer Vision and Pattern Recognition, pages 4561–4570, 2019.

[184] Victor Adrian Prisacariu, Olaf Kähler, Stuart Golodetz, Michael Sapienza, Tommaso Cavallari, Philip HS Torr, and David W Murray. Infinitam v3: A framework for large-scale 3d reconstruction with loop closure. *arXiv preprint arXiv:1708.00783*, 2017.

[185] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2020.

[186] Chen Qian, Xiao Sun, Yichen Wei, Xiaoou Tang, and Jian Sun. Realtime and robust hand tracking from depth. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2014.

[187] René Ranftl and Vladlen Koltun. Deep fundamental matrix estimation. In *Eur. Conf. on Computer Vision (ECCV)*, 2018.

[188] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Trans. Pattern Anal. Machine Intell.*, 2020.

[189] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Intl. Conf. on Computer Vision (ICCV)*, pages 12179–12188, 2021.

[190] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *arXiv preprint arXiv:2103.13744*, 2021.

[191] Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *Intl. Conf. on Computer Vision (ICCV)*, 2017.

[192] Barbara Roessle, Jonathan T Barron, Ben Mildenhall, Pratul P Srinivasan, and Matthias Nießner. Dense depth priors for neural radiance fields from sparse input views. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 12892–12901, 2022.

[193] Luis Roldão, Raoul de Charette, and Anne Verroust-Blondet. 3D surface reconstruction from voxel-based lidar data. In *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2681–2686. IEEE, 2019.

[194] Radu Alexandru Rosu and Sven Behnke. Permutosdf: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2023.

[195] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and*

Modeling, pages 145–152, 2001. doi: 10.1109/IM.2001.924423.

[196] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[197] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[198] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2009.

[199] S. Salti, F. Tombari, and L. di Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 2014.

[200] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters*, 2(2):420–427, 2016.

[201] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 2007.

[202] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2016.

[203] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2016.

[204] Thomas Schops, Torsten Sattler, and Marc Pollefeys. BAD SLAM: Bundle adjusted direct RGB-D SLAM. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2019.

[205] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: Science and Systems (RSS)*, volume 2, page 435. Seattle, WA, 2009.

[206] Tixiao Shan and Brendan Englot. Lego-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[207] Jingnan Shi, Heng Yang, and Luca Carlone. ROBIN: a Graph-Theoretic Approach to Reject Outliers in Robust Estimation using Invariants. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021.

[208] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. A micro lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*, 2018.

[209] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual

odometry from dense RGB-D images. In *Proceedings of ICCV Workshops*. IEEE, 2011.

[210] Patrick Stotko. stdgpu: Efficient STL-like Data Structures on the GPU. *arXiv:1908.05936*, 2019.

[211] Patrick Stotko, Stefan Krumpen, Matthias B. Hullin, Michael Weinmann, and Reinhard Klein. SLAMCast: Large-Scale, Real-Time 3D Reconstruction and Streaming for Immersive Multi-Client Live Telepresence. *IEEE Transactions on Visualization and Computer Graphics*, 25(5):2102–2112, May 2019.

[212] Hauke Strasdat. *Local accuracy and global consistency for efficient visual SLAM*. PhD thesis, Department of Computing, Imperial College London, 2012.

[213] Jurgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2012.

[214] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. imap: Implicit mapping and positioning in real-time. In *Intl. Conf. on Computer Vision (ICCV)*, pages 6229–6238, 2021.

[215] Jiaming Sun, Yiming Xie, Linghao Chen, Xiaowei Zhou, and Hujun Bao. NeuralRecon: Real-time coherent 3D reconstruction from monocular video. *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2021.

[216] Jiaming Sun, Xi Chen, Qianqian Wang, Zhengqi Li, Hadar Averbuch-Elor, Xiaowei Zhou, and Noah Snavely. Neural 3d reconstruction in the wild. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022.

[217] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. 2021.

[218] Weikai Tan, Nannan Qin, Lingfei Ma, Ying Li, Jing Du, Guorong Cai, Ke Yang, and Jonathan Li. Toronto-3D: A large-scale mobile lidar dataset for semantic segmentation of urban roadways. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition Workshops*, pages 202–203, 2020.

[219] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. URL `https://rapids.ai`.

[220] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Proc. Conf. on Neural Information Processing Systems*, 34:16558–16569, 2021.

[221] Zachary Teed, Lahav Lipson, and Jia Deng. Deep patch visual odometry. *arXiv preprint arXiv:2208.04726*, 2022.

[222] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 292–301, 2018.

[223] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Intl. Conf. on Computer Vision (ICCV)*, 2019.

[224] Yurun Tian, Bin Fan, and Fuchao Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 661–669, 2017.

[225] Carl Toft, Will Maddern, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, Tomas Pajdla, et al. Long-term visual localization revisited. *IEEE Trans. Pattern Anal. Machine Intell.*, 2020.

[226] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique shape context for 3D data description. In *ACM Workshop on 3D Object Retrieval*, 2010.

[227] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2017.

[228] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *Intl. Conf. on Learning Representations (ICLR)*, 2020.

[229] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. on Graphics*, 26(1):6–es, 2007.

[230] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 2022–2030, 2018.

[231] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

[232] Qianqian Wang, Xiaowei Zhou, Bharath Hariharan, and Noah Snavely. Learning feature descriptors using camera pose supervision. In *Eur. Conf. on Computer Vision (ECCV)*, 2020.

[233] Yue Wang and Justin M. Solomon. Deep closest point: Learning representations for point cloud registration. In *Intl. Conf. on Computer Vision (ICCV)*,

October 2019.

[234] Yue Wang and Justin M. Solomon. PRNet: Self-supervised learning for partial-to-partial registration. In *Proc. Conf. on Neural Information Processing Systems*, 2019.

[235] Colin Wei, Kendrick Shen, Yining Chen, and Tengyu Ma. Theoretical analysis of self-training with deep networks on unlabeled data. *arXiv preprint arXiv:2010.03622*, 2020.

[236] Yi Wei, Shaohui Liu, Yongming Rao, Wang Zhao, Jiwen Lu, and Jie Zhou. Nerfingmvs: Guided optimization of neural radiance fields for indoor multi-view stereo. In *Intl. Conf. on Computer Vision (ICCV)*, pages 5610–5619, 2021.

[237] Thomas Whelan, Stefan Leutenegger, Renato F Salas-Moreno, Ben Glocker, and Andrew J Davison. ElasticFusion: Dense SLAM without a pose graph. In *Robotics: Science and Systems (RSS)*, 2015.

[238] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.

[239] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 10687–10698, 2020.

[240] Saining Xie, Jiatao Gu, Demi Guo, Charles R Qi, Leonidas J Guibas, and Or Litany. Pointcontrast: Unsupervised pre-training for 3d point cloud understanding. In *Eur. Conf. on Computer Vision (ECCV)*, 2020.

[241] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library, 2022.

[242] Heng Yang and Luca Carlone. A quaternion-based certifiably optimal solution to the Wahba problem with outliers. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1665–1674, 2019.

[243] Heng Yang and Luca Carlone. In perfect shape: Certifiably optimal 3D shape reconstruction from 2D landmarks. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2020.

[244] Heng Yang and Luca Carlone. One ring to rule them all: Certifiably robust geometric perception with outliers. In *Proc. Conf. on Neural Information Processing Systems*, 2020.

[245] Heng Yang, Pasquale Antonante, Vasileios Tzoumas, and Luca Carlone. Graduated non-convexity for robust spatial perception: From non-minimal

solvers to global outlier rejection. *IEEE Robotics and Automation Letters*, 5(2): 1127–1134, 2020.

[246] Heng Yang, Jingnan Shi, and Luca Carlone. TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Trans. Robotics*, 2020.

[247] Heng Yang, Wei Dong, Luca Carlone, and Vladlen Koltun. Self-supervised geometric perception. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 14350–14361, 2021.

[248] Jiaolong Yang, Hongdong Li, and Yunde Jia. Optimal essential matrix estimation via inlier-set maximization. In *Eur. Conf. on Computer Vision (ECCV)*, pages 111–126. Springer, 2014.

[249] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. Go-ICP: A globally optimal solution to 3D ICP point-set registration. *IEEE Trans. Pattern Anal. Machine Intell.*, 38(11):2241–2254, 2016.

[250] Nan Yang, Lukas von Stumberg, Rui Wang, and Daniel Cremers. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1281–1292, 2020.

[251] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Eur. Conf. on Computer Vision (ECCV)*, pages 767–783, 2018.

[252] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Proc. Conf. on Neural Information Processing Systems*, 33:2492–2502, 2020.

[253] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Proc. Conf. on Neural Information Processing Systems*, 34:4805–4815, 2021.

[254] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.

[255] Zi Jian Yew and Gim Hee Lee. 3dfeat-net: Weakly supervised local 3d features for point cloud registration. In *Eur. Conf. on Computer Vision (ECCV)*, 2018.

[256] Kwang Moo Yi, Eduard Trulls, Yuki Ono, Vincent Lepetit, Mathieu Salzmann, and Pascal Fua. Learning to find good correspondences. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2018.

[257] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas

Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. In *Proc. Conf. on Neural Information Processing Systems*, 2022.

[258] Wentao Yuan, Ben Eckart, Kihwan Kim, Varun Jampani, Dieter Fox, and Jan Kautz. DeepGMR: Learning Latent Gaussian Mixture Models for Registration. 2020.

[259] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. DPOD: 6d pose object detector and refiner. In *Intl. Conf. on Computer Vision (ICCV)*, pages 1941–1950, 2019.

[260] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3DMatch: Learning local geometric descriptors from rgb-d reconstructions. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2017.

[261] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. Octree-based fusion for realtime 3d reconstruction. *Graphical Models*, 75(3):126–136, 2013.

[262] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems (RSS)*, volume 2, pages 1–9. Berkeley, CA, 2014.

[263] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *Eur. Conf. on Computer Vision (ECCV)*, pages 649–666. Springer, 2016.

[264] Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. *arXiv preprint arXiv:2203.11283*, pages 5449–5458, 2022.

[265] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 1994.

[266] Chen Zhao, Zhiguo Cao, Chi Li, Xin Li, and Jiaqi Yang. NM-Net: Mining reliable neighbors for robust feature correspondences. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2019.

[267] Yu Zhong. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In *Intl. Conf. on Computer Vision Workshops (ICCVW)*, pages 689–696. IEEE, 2009.

[268] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Trans. on Graphics*, 27(5):1–11, 2008.

[269] Qian-Yi Zhou and Vladlen Koltun. Simultaneous localization and calibration: Self-calibration of consumer depth cameras. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2014.

[270] Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. Elastic fragments for

dense scene reconstruction. In *Intl. Conf. on Computer Vision (ICCV)*, 2013.

[271] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *Eur. Conf. on Computer Vision (ECCV)*, 2016.

[272] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[273] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[274] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2019.

[275] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, pages 12786–12796, 2022.

[276] Michael Zollhöfer, Angela Dai, Matthias Innmann, Chenglei Wu, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Shading-based refinement on volumetric signed distance functions. *ACM Trans. on Graphics*, 34(4):1–14, 2015.

[277] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. State of the art on 3d reconstruction with rgb-d cameras. In *Computer Graphics Forum*, volume 37, pages 625–652, 2018.

[278] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin D Cubuk, and Quoc V Le. Rethinking pre-training and self-training. *arXiv preprint arXiv:2006.06882*, 2020.