LONDON'S GLOBAL UNIVERSITY
Department of Computer Science

Gaussian Surfel Representation for Visual SLAM

Xingyu Chen

Supervisor: Francisco Porto Guerra E Vasconcelos

Faculty of Engineering

Department of Computer Science

University College London

A Project Report Presented in Partial Fulfillment of the Degree

MSc Robotics and Computation

Submission date: September 2024

Disclaimer: This report is submitted as part requirement for the MSc in Robotics and Computation at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Visual SLAM is a technique that uses camera images to map an area and track its position under that map. To build a digital map from the real world, the scene representation is vital. Currently, using 3D Gaussian splatting technology to construct such a map has achieved state-of-the-art visual quality in real-time, but remains a disadvantage in terms of geometry accuracy. The prevailing method directly minimizes the photometric error between camera input and rendered image, it causes inaccurate point clouds which don't fit with the actual surface and generates transparent volume artifacts which does not exist in reality. Besides, there is an important property that the point cloud in Gaussian Representation itself is explicit. Based on this, adequately using explicit map attributes to obtain spatial information from scene and designing robust function directly for optimization is important. To improve the performance of Gaussian representation in visual SLAM system, In this project, we introduce Gaussian surfel representation, a novel planar geometry instead of ellipsoid, and explore whether 2D Gaussian can provide more accurate point clouds than 3D Gaussian in SLAM setting. Then we design our own tracking module and mapping module to combine Gaussian surfels with visual SLAM system. Finally, we conduct our designed algorithm in benchmark dataset, the experimental point cloud results verify the effectiveness of our proposed method.

Contents

Chapter 1

Introduction

SLAM is a computational problem of how to construct and update a map for an unknown environment while keeping track of an agent's location at the same time. In this chapter, I will show you a big picture of visual SLAM in my understanding as well as the purpose and contribution of my thesis.

1.1 Create a cyber space from RGB-D camera

SLAM systems work on reconstructing the scene while simultaneously estimating the camera pose localization. This technology enables autonomous agents to understand the spatial relationship between themselves and their surroundings. It is a key problem across robotics and vision[1, 2, 3]. For visual SLAM system, the input is usually a video sequence, the outputs are the reconstructed map and the camera trajectory under this scene. Generally, what visual SLAM system do is try to recover the original 3D spatial information of the world from the two-dimensional images obtained by the sensor.

To begin with, let me introduce two new concepts, sampling space and reconstruction space.

- **Sampling space** is the real world as observed by the sensor. In visual SLAM setting, the sensor is an RGB-D camera, this camera projects the infinitely large real world into a finite discrete world of pixels. In other words, the sampling process allows the real world to be quantized and discretized into a pixels world. In each pixel, it used an RGB value to represent a region from the real world.
- **Reconstructed space** refers to cyberspace created by digital representation and information technology, which aim to approximate the original real world. Once given a dataset, it should be a ground truth for this sampling space. Then, the purpose of visual SLAM system is trying to make the reconstructed space as similar as possible to the original real-world space, based on the limited information from the sampling space.

Visual SLAM lies in the challenge of accurately reconstructing the real-world environment in a digital space, given the inherent limitations of the data measured from the sampling process. Reviewing the entire flow of a typical visual SLAM system, the RGB-D camera serves as the primary sensor, translating the continuous and infinitely complex real world into a finite set of discrete pixel values. This transformation process leads to a critical abstraction: the rich, multifaceted environment is distilled into a grid of pixels, each representing a small region of the scene through simplified RGB values. Consequently, the sampling space comprised of these pixel values provides only a partial and quantized view of the original world, inevitably leading to a loss of information.

The reconstructed space, on the other hand, is an artificial environment created using digital media and advanced computational techniques. The goal of visual SLAM is to ensure that this reconstructed space closely mirrors the real-world environment, despite the data limitations imposed by the sampling space. This task is inherently challenging since the SLAM system must infer and recreate aspects of the environment that are not directly captured in the sampled data, such as spatial relationships, occluded areas, and fine-grained details.

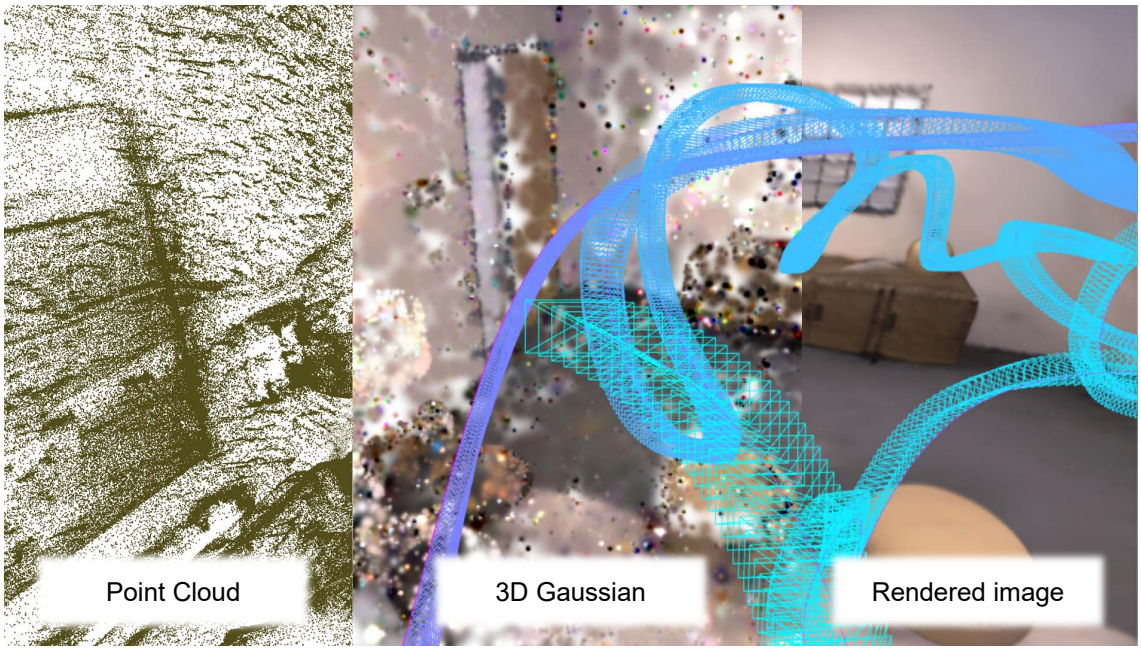


Figure 1.1: Multi-representation visualization of a digital space created by visual SLAM system. This is a coherent indoor image, the blue trajectory is the estimated pose of the camera during each frame, and the whole room is presented by three different representation layers. It shows the process of a digital scene for the real world which is rendered from basic geometric elements.

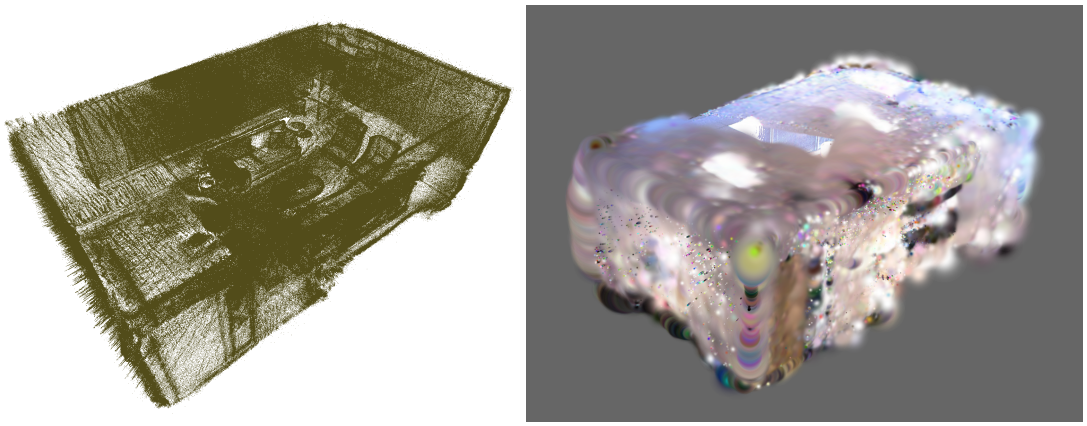
Our research motivation for developing visual SLAM is to bridge the gap between the sampling space and the real world by leveraging sophisticated algorithms and computational models. By doing so, the reconstructed space can achieve a high level of fidelity, enabling accurate localization and mapping within the environment shown in Figure 1.1. This is crucial for a wide range of applications, from autonomous navigation and robotics to augmented reality and digital twins, where the ability to interact with or navigate through a precise digital replica of the real world is essential. The ultimate aim is to enhance the capacity of machines and systems to understand and operate within the real world, making visual SLAM a critical component in the advancement of intelligent systems and immersive technologies.

With the development of visual SLAM technology, the performance of current SLAM systems is also confronted with challenges on new applications, such as adapting more advanced tasks under high-quality scenes in Mixed Reality, digital virtual avatars, detailed navigation, dexterous manipulation, legged robotic locomotion, whole-body planning for humanoid robotics.

In order to provide richer environmental information for downstream tasks as well as to enhance the user's experience, therefore, geometric modeling and high-fidelity digital representations of the scene are vital for advanced SLAM techniques. By incorporating precise geometric data, a realistic mapping SLAM system can better understand the spatial relationships and physical properties of the environment, which is very promising as a solution for complex downstream tasks. Recently, Neural Radiance Field[4] and 3D Gaussian splatting[5] solutions for photo-realistic reconstruction and rendering have received significant attention from researchers and community. Both of them can express scenes with high-fidelity rendering quality. Not only that, 3DGS achieves state-of-the-art visual quality while maintaining competitive training times, and making it become the premier mapping choice for my project. I will cover more details below.

1.2 Purpose and contribution

Currently, many researchers have proposed the SLAM system that directly reconstructs the 3D Gaussian map[6, 7]. The results show that 3D Gaussian splatting greatly enhances the high-fidelity rendering capability for visual SLAM. But remains a disadvantage in terms of geometry accuracy. For example, to achieve better rendering quality, 3D Gaussian will generate wrong point clouds, which causes inaccurate scene surface fitting. As shown in Figure 1.2, to over-fit the image quality, current 3D Gaussian methods will produce point clouds outside the edge with a very big covariance and low opacity. Recently, a lot of computer graphics works have focused on enhancing the geometric surface information of planar Gaussian representation[8, 9, 10]. Inspired by these studies, we conjecture that the 3D Gaussian is over-parameterized in the scene representation and that there is redundancy in the parametric modeling and fitting process. Therefore, one of my goals is to apply a more efficient and simple geometric model to mapping the scene.



(a) Gaussian without covariance (b) Gaussian with covariance

Figure 1.2: The room edge fitting of current 3D Gaussian SLAM.

Another core problem for applying existing Gaussian representation into SLAM system is how to track the pose according to camera sequence inputs. Besides, there is a significant property that the point cloud in Gaussian Representation itself is explicit. RGB-D image can also be considered as a point cloud with color, Each point cloud is a scan of the surface from all the objects. Based on this, adequately using the explicit point-based attribute to obtain spatial information from scene and design robust function directly for optimization is important. Also, one insight is that current methods do not fully utilize the explicit representation properties of Gaussian maps, for example, the consistency of surface normals of objects.

In this project, we introduce a novel representation of 2D Gaussian[9], an oriented planner Gaussian disks. To adapt this new Gaussian representation in SLAM system, we develop and extend the existing 3DGS SLAM approach based on the open source framework SplaTam[6], then we design a new tracking module and mapping module to explore whether 2D surface can provide more accurate point cloud results than 3D Gaussian ellipsoid. Finally, we test our proposed method in benchmark dataset. Our contribution can be summarised as:

- We introduce 2D planar geometry model Gaussian surfel into SLAM system instead of 3D Gaussian Splatting in order to obtain more accurate surface information of scene.
- We combine this with ICP algorithm to utilize the object surface information more effectively from the Gaussian representation map in order to optimize the tracking loss.
- We test the performance of our designed modules in benchmark dataset. The point cloud mapping results verify the effectiveness of our proposed method.

This thesis is mainly about Gaussian representations for visual SLAM. To summarize, for the structure of this master thesis, the content is organized as follows:

- In chapter2, I will briefly go through from traditional SLAM into differential rendering SLAM.
- In chapter3, I will introduce the definition and mathematical derivation for 3DGS and ICP.
- In chapter4, I will explain more details about my module design in tracking and mapping.
- In chapter5, I will present visualization, evaluation, and comparison with baseline.
- In chapter6, I will analyze the results and discuss about future works.

Chapter 2

Literature Review

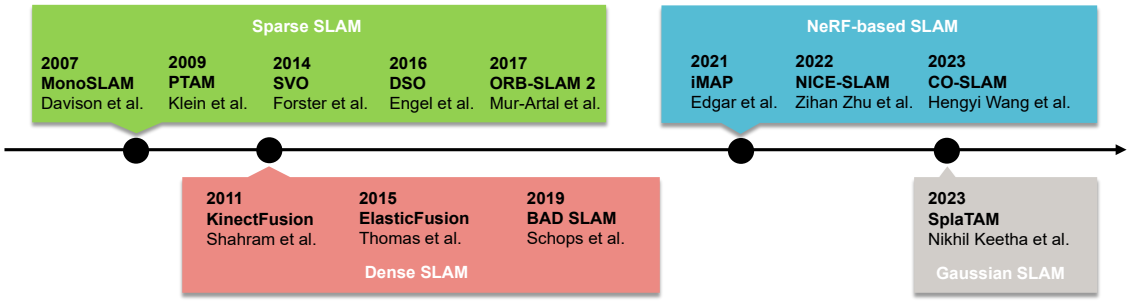


Figure 2.1: Overview of development for visual SLAM method.

Generally, traditional SLAM methods rely on classical computer vision and robotics techniques to perform localization and mapping. To the best of my knowledge, the famous classical approaches for traditional SLAM are probabilistic method, feature-based method, direct method and graph-based method. They typically use explicit geometric representations like points, meshes, or voxels. For example, sparse reconstruction identifies and uses a limited set of key feature points, such as corners and edges, to estimate camera movement and create a basic structure of the environment. This method is computationally efficient and suitable for real-time applications like robotics and augmented reality. Dense reconstruction, on the other hand, aims to create a detailed 3D model of the environment by capturing depth information for every pixel or point in the scene. This approach is more computationally demanding and is used in applications requiring detailed and accurate 3D representations, such as 3D mapping and autonomous driving.

In contrast, unlike the traditional method that divides SLAM system into front-end and back-end, and then optimizes them separately, differentiable Rendering methods adopt a back-propagation framework, which allows gradients to be computed and transferred through the whole rendering process, facilitating end-to-end optimization. This category often involves deep learning and neural network-based approaches. For example, NeRF-based SLAM employs MLP to learn volumetric scene functions from 2D images, enabling photo-realistic rendering and accurate scene understanding through implicit representation and view synthesis. Then, Gaussian-based reconstruction models the points with Gaussian distributions. These differentiable rendering approaches offer flexible and accurate scene representations, making them suitable for high-fidelity simulation and robust scene reconstruction tasks in virtual reality and augmented reality.

According to the reconstruction type, existing visual SLAM method can be broadly classified into four categories: sparse reconstruction by the landmarks[11, 12, 13, 14, 15], dense reconstruction at the pixel level[16, 17, 18], realistic reconstruction by Neural radial field[19, 20, 21], and realistic reconstruction by Gaussian Splatting[6, 7]. A timeline progression for different methodologies is shown in Figure 2.1. In this chapter, I will introduce them in chronological order.

2.1 Traditional method for visual SLAM

2.1.1 Visual SLAM with Sparse construction

Visual SLAM with sparse construction generally has two type, feature-base method and direct method. The feature-based method relies on the identification of key points or features within the environment, for example ORB-SLAM use Oriented FAST and Rotated BRIEF features points. The direct method directly estimate from intensity of the image, such as SVO and DSO. Sparse map is typically more efficient and can work in real-time on standard hardware. Here, just take a briefly review for some representative research works, which reflect a progression towards more efficient, accurate, and versatile systems.

MonoSLAM [11]:

MonoSLAM represents one of the pioneering efforts in monocular SLAM, utilizing a single camera to estimate the 3D positions of feature points in the environment. This method relies on probabilistic models to continuously estimate the state of the camera and the map, updating both as new observations are made. The core of MonoSLAM's innovation lies in its ability to operate in real-time with limited computational resources, making it suitable for applications in robotics and augmented reality where compact and efficient systems are essential.

Parallel Tracking and Mapping [12]:

Parallel Tracking and Mapping (PTAM) introduced a significant advancement in SLAM methodologies by separating the processes of tracking and mapping. This dual-threaded approach allows each process to run independently in parallel, optimizing both efficiency and accuracy. The tracking component is responsible for estimating the camera's pose using feature matching, while the mapping component constructs and refines the map using bundle adjustment techniques. PTAM's design enhances the robustness and speed of SLAM systems, particularly in scenarios with high computational demands.

Semi-direct Visual Odometry [13]:

Semi-direct Visual Odometry (SVO) combines the strengths of both feature-based methods and direct methods in visual odometry. Unlike traditional methods that rely heavily on feature extraction and matching, SVO directly utilizes the luminance values of pixels, which allows for faster processing and improved efficiency. This approach enhances the system's ability to operate in real-time while maintaining high levels of accuracy, making it ideal for applications in autonomous navigation and augmented reality.

Direct Sparse Odometry [14]:

Direct Sparse Odometry (DSO) marks a significant development in visual SLAM by using image gray values directly for optimization, thereby bypassing the need for explicit feature extraction and matching. DSO focuses on the direct alignment of pixel intensities to estimate camera motion and scene structure, which results in more detailed and accurate scene reconstruction and pose estimation. This method excels in scenarios with low-texture environments where traditional feature-based methods struggle, offering a robust solution for advanced SLAM applications.

ORB-SLAM [22, 15, 23]:

ORB-SLAM2 and ORB-SLAM3 are extended versions of the original ORB-SLAM, designed to enhance both efficiency and versatility. This system uses Oriented FAST and Rotated BRIEF features for reliable and fast feature matching, making it capable of handling monocular, stereo, and RGB-D cameras. ORB-SLAM2 integrates loop closure detection and full bundle adjustment to refine the map and camera trajectory, ensuring high accuracy and robustness. Its ability to operate across various camera setups makes it a versatile solution for a wide range of SLAM applications, from robotics to virtual reality.

In summary, MonoSLAM, PTAM, SVO, DSO, and ORB-SLAM2 each contribute to the foundational principles of sparse reconstruction by leveraging distinct strategies for feature extraction, tracking, and map optimization. MonoSLAM's pioneering monocular approach, PTAM's dual-threaded system, SVO's integration of direct luminance values, DSO's bypass of traditional feature matching, and ORB-SLAM2's utilization of ORB features collectively enhance the robustness and efficiency of sparse mapping techniques. These methods underscore a significant evolution in the field, especially focusing on the balance between computational efficiency and mapping accuracy.

2.1.2 Visual SLAM with Dense construction

Unlike sparse methods, dense SLAM aims to create a full reconstruction at pixel level, which results in detailed maps but often at a higher computational cost. And with the development of the sensor, in this stage, more and more researchers try to use RGB-D camera, which contains a depth value for each pixel.

KinectFusion [16]:

KinectFusion leverages volumetric representation to perform real-time 3D reconstruction of indoor environments. By utilizing depth data acquired from the Microsoft Kinect sensor, KinectFusion is capable of generating detailed 3D models by fusing depth information across multiple frames. The system employs a voxel grid to represent the volumetric data, which is updated continuously as new depth images are captured. This approach allows for the creation of dense 3D maps that are highly accurate and detailed, making KinectFusion suitable for applications in robotics, augmented reality, and virtual environment creation. The real-time capability of KinectFusion enables dynamic interaction with the environment, enhancing user experience and providing valuable data for downstream tasks.

ElasticFusion [16]:

ElasticFusion constructs surfel-based representations to achieve consistent room-scale mapping by detecting and correcting for loop closures. Surfels, or surface elements, provide a compact and efficient means of representing surfaces in 3D space. ElasticFusion integrates a non-rigid surface fusion technique that allows the system to maintain a globally consistent model of the environment even as it encounters and corrects loop closures. This capability is crucial for long-term mapping and navigation tasks, where maintaining an accurate and up-to-date map is essential. The use of surfels enables high-resolution surface reconstruction while managing computational efficiency, making ElasticFusion an advanced tool for applications requiring detailed environmental modeling.

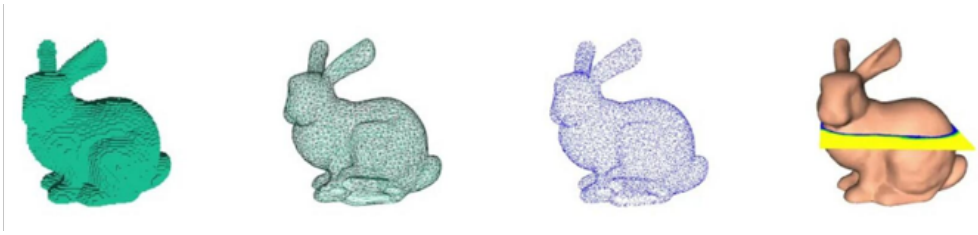
BAD SLAM [18]:

BAD SLAM employs a voxel-based graph structure to represent and optimize large-scale 3D environments. By using a voxel-based representation, BAD SLAM can handle extensive and complex environments, making it particularly suitable for large-scale mapping applications. The system optimizes the voxel graph to ensure accurate and efficient reconstruction, addressing challenges such as scalability and computational load. BAD SLAM's innovative approach allows for detailed environmental reconstruction while maintaining the ability to process large volumes of data.

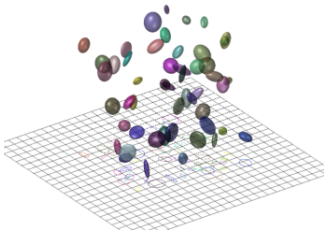
KinectFusion's volumetric representation and real-time processing capabilities, ElasticFusion's surfel-based mapping and loop closure correction, and BAD SLAM's voxel-based graph optimization collectively advance the state-of-the-art in visual SLAM, supporting a wide range of applications from indoor navigation to large-scale environmental modeling. These methods demonstrate the continuous evolution of SLAM technologies towards more robust, detailed, and scalable 3D reconstruction solutions.

2.2 Innovation in scene representation

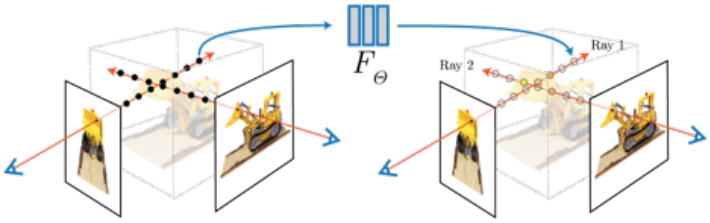
Geometric representation is a form of scene digitization. In SLAM, accurate and efficient scene representation is crucial for building and maintaining a map of the environment. Currently, there are several mainstream map representation types shown in Figure 2.2.



(a) Traditional geometry types: voxel, mesh, point cloud, SDF for Stanford bunny rabbit [24].



(b) 3D Gaussian [25]



(c) Neural Radiance Field [4]

Figure 2.2: Different geometric type for scene representation.

Different geometric representation has different properties shown as Table 2.1.

Geometric type	Differentiable	Representation	Memory cost	Comp cost
Voxel	No	Explicit	High	Medium
Mesh	No	Explicit	Medium	High
Point Cloud	No	Explicit	Low	Low
SDF	Yes	Implicit	Medium	High
NeRF	Yes	Implicit	Medium	High
3D Gaussian	Yes	Explicit	Medium	Medium

Table 2.1: Comparison of Geometric Representation

- **Voxels:** represent 3D space on a regular grid, but memory-intensive for large scene. It is widely used in autonomous navigation for obstacle avoidance, positioning and path planning.
- **Mesh:** composed of vertices, edges, and faces, provide detailed surface representation. So it is ideal for precise modeling and rendering for object surface.
- **Point clouds:** a collection of spatial points, it commonly generated by 3D scanners, so they are essential due to their flexibility and efficiency for raw sensor data.
- **Signed Distance Functions:** define surfaces via a distance function, offering smooth representations and facilitating advanced modeling techniques.
- **Neural Radiance Fields:** leverage deep learning to create continuous volumetric scene functions, enabling high-quality view synthesis and complex scene reconstruction.
- **3D Gaussian:** models each point by Gaussian distributions, give a volumetric properties for unordered point cloud. It is widely used for high quality real-time image rendering.

Each geometric type offers unique advantages and trade-offs, influencing their suitability for different SLAM applications, from efficient real-time mapping to detailed surface reconstruction and photo-realistic visualization.

Figure 2.3: The performance toward high fidelity and photo-realistic rendering.

To build these digital representation, traditional structure from motion and multi-views stereo approach are based on photographic geometry. It typically perform poorly when reconstructing unobserved regions and it can't differentiable from end-to-end and deal with changes in perspective. Recently, this field is continuously evolving with new breakthroughs and refinements by Radial Field[4] and 3D Gaussian Splatting[5]. These two type of geometry can take a realistic rendering shown in Figure 2.3.

2.2.1 NeRF: Toward photo-realistic rendering

Neural Radial field methods also denoted NeRF[4] began to emerge in 2020, revolutionising the algorithmic ecology of new view synthesis for scenes dominated by multiple photos or videos. NeRF employs deep learning to achieve photo-realistic rendering of scenes from a sparse set of input images, enabling the generation of novel views from arbitrary camera positions with remarkable accuracy. Meanwhile neural rendering algorithms greatly reduce such artefacts, and implicit representation also avoids the huge cost of storing all input images on the GPU.

NeRF directly utilizes a fully connected neural network to model the volumetric scene function, which implicitly encodes the 3D structure and appearance of a scene. Input a 3D coordinate and viewing direction, the network outputs the color and density at that point. The density values are used to simulate the accumulation of light along a ray cast through the scene, enabling the reconstruction of complex visual phenomena such as occlusions, shadows, and fine geometric details. The training process involves optimizing the neural network to minimize the difference between the rendered views and the actual observed images. This is done by leveraging differentiable volume rendering, where the network iteratively refines its predictions of color and density based on the input images. The output of this process is a neural representation that captures the radiance field of the scene, effectively allowing the generation of high-quality images from novel viewpoints.

Besides, NeRF has ability to capture fine details and complex lighting effects without the need for explicit models. Unlike traditional methods that rely on mesh representations or point clouds, NeRF provides a continuous volumetric representation that is both memory-efficient and capable of representing highly detailed scenes. Moreover, NeRF introduced importance sampling and positional encoding to improve quality, but the use of large MLP also negatively impacted processing speed. As results, subsequent research has produced a large number of methods for balancing the quality and speed of NeRF, such as the introduction of regularization strategies. The current state-of-the-art synthetic image quality for new views is Mip-NeRF360[26], which shows that although the rendering quality is good, the training and rendering time is still very long. To accelerate training and rendering, spatial data structures are used to store features and interpolate them in ray marching sessions, different encoding methods, and capacity adjustments of the MLP. For example, InstantNGP[27] uses hash grids and occupancy grids to speed up computation, and uses smaller MLPs to represent density and appearance. Plenoxels[28] use sparse voxel grids to interpolate continuous density fields, and are able to completely abandon neural networks.

2.2.2 3DGS: Toward real-time rendering

The cost of training and rendering of NeRF methods are currently expensive and inevitably sacrifice speed for quality. For unbounded and complete scenes (as opposed to isolated objects) and 1080p resolution rendering, there is currently no way to achieve real-time display rate. Even though InstantNGP[27], Plenoxels[28] achieves very competitive results, there are still bottlenecks in the speed and quality of training and rendering. So we hope that a new representation will emerge that meets or exceeds quality requirements while providing rapid training and real-time rendering. Furthermore, image quality is largely limited by the choice of structured meshes used for acceleration, and rendering speed is hampered by the need to query many samples for a given light travelling step.

To address these problem, research began to introduce point-based rendering method. That is 3D Gaussian Splatting[5], a unstructured, explicit GPU-friendly functions used achieve faster rendering speeds and better quality without the need for a neural component. Gaussian Splatting achieves state-of-the-art visual quality while maintaining competitive training times, and importantly allows for more than 30 fps high-quality real-time synthesis of new views at 1080p resolution.

2.2.3 Parameterized point-based representation

From the perspective of digital geometric representations, 3D Gaussian splatting have proposed a new point cloud representation, which I personally call parameterized point-based representation. The basic 3D point cloud representation consists of an unordered set of coordinate points, then parametric point-based representation aims to add some extra parameters to describe new attributes for each points. For example, 3DGS modeling a probability distribution with its coordinates as the mean, introducing its covariance, scaling factor, color value, opacity value, spherical harmonic function, and so on. These additional parametric attributes can greatly enrich the ability of scene representation for point clouds. Similarly, different parametric point-based representations can be constructed by different geometric modeling of the point cloud.

At early stage, researchers find that 3D Gaussian splatting does not present an ordered surface structure after reconstruction, and even it does not fit with the actual surface of the scene. Therefore, SuGaR[29] propose a surface-aligned regularity, distance function regularity and normal regularity, which constrained Gaussian point clouds better distributed on the surface. However, on the one hand, restricting the position of the Gaussian points will decrease the rendering quality of the scene, on the other hand, estimating the depth and surface directly from the center point will cause the geometry inaccurate, since ignoring other properties such as covariance and opacity. So other researchers proposed a general field for this, GOF[30] defines a Gaussian opacity field by approximating the surface normal of the Gaussian as a ray-Gaussian intersection plane. It's performs well in terms of rendering accuracy and quality, but lack in compute speed.

After that, researchers try to directly change the basic attributes of 3D Gaussian and use new parametric modeling, like planar Gaussian instead of stereo Gaussian. Gaussian surfel[8] design a new shape modeling, which compress original 3D ellipsoid into a 2D plane by defining the covariance along z-axis direction as zero. In order to compensate for the problem that missing dimensions cause the normal vector gradient propagation problem, it introduces the normal vector prior loss and the self-supervised normal-depth consistency loss. Next, 2DGS[9] directly using a 2D Gaussian disk for geometry representation, in other word, it replaces the covariance with two tangent vectors and a scaling matrix. However, project a disk into the camera coordinate, it easily degenerates to a straight line (edgewise), the affine transformation matrix inverse will be very unstable. In order to adapt to this structure, they also proposed some numerical approach called Ray-splat intersection method. It is a step further in the representation of geometry on the scene surface, while ensuring high reconstruction accuracy and speed. Then, some researchers also proposed Planar-based Gaussian Splatting called PSGR[10], which introduce an unbiased depth rendering method to obtain distance and normal map, meanwhile they utilize single-view constraints, multi-view photometry, and geometric regularization to maintain global accuracy.

2.3 Differentiable Rendering for visual SLAM

As I mention above in the last section, each of these methods brings different strengths and trade-offs, making them suitable for various applications, from robotics to augmented reality. Sparse SLAM are used for its efficiency, while dense SLAM provides higher representation. After that, NeRF-based methods continue to use more powerful geometry representation to obtain higher reconstruction solutions. So far, NeRF-based and Gaussian SLAM methods have ability to create photo-realistic renderings scene, I will introduce more details below.

2.3.1 Visual SLAM with Neural Radiance Field

Neural Radiance Fields use neural networks to learn a representation of the scene that can be rendered from arbitrary viewpoints. NeRF-based SLAM introduce deep learning method directly for scene representation and reconstruction.

iMAP [19]:

iMAP, introduced in 2021, represents a significant advancement in the real-time SLAM by introducing an implicit neural scene representation that leverages a multi-layer perceptron (MLP) for mapping and tracking using a handheld RGB-D camera. This approach marks the first instance where an MLP serves as the sole scene representation in a real-time SLAM system, showcasing the ability to build dense, scene-specific 3D models of occupancy and color without the need for prior data. iMAP achieves real-time performance through a keyframe-based structure and multi-processing computation flow, allowing it to dynamically sample the most informative pixels to reduce geometric uncertainty. This system excels in maintaining a smooth and complete reconstruction of the scene, including the ability to fill in unobserved regions with plausible estimates, a task that traditional dense SLAM systems struggle with. iMAP's contribution lies in its innovative approach to continual learning within the SLAM framework, demonstrating competitive tracking performance and efficient memory usage, which are crucial for real-time applications in dynamic environment.

NICE-SLAM [20]:

NICE-SLAM (Neural Implicit Scalable Encoding for SLAM) represents a significant advancement in the field of simultaneous localization and mapping (SLAM) by integrating neural implicit representations with a scalable, hierarchical grid-based approach for scene encoding. This method addresses the limitations of previous SLAM systems, particularly those based on neural implicit representations that often result in over-smoothed scene reconstructions and struggle to scale to large environments. NICE-SLAM introduces a novel approach by incorporating multi-level grid-based features that allow for detailed and scalable scene reconstruction, which is a key contribution to the field.

The core innovation of NICE-SLAM lies in its hierarchical scene representation, which combines the advantages of grid-based encoding with pre-trained neural implicit decoders. This structure allows the system to capture different levels of scene detail, from coarse to fine, enabling it to maintain high geometric fidelity across various scales of indoor environments. Unlike traditional SLAM systems that rely on a single neural network architecture to encode the entire scene, NICE-SLAM's multi-level grid structure allows for local updates, making it particularly suitable for large-scale applications. This capability is crucial for real-time dense SLAM systems.

The system operates by optimizing a hierarchical feature grid, which is used for both mapping and tracking. This grid encodes the scene's geometry and appearance, allowing the SLAM system to produce accurate depth and color renderings of the environment. The use of pre-trained decoders, particularly in the fine and mid-level representations, enhances the system's ability to maintain

detailed geometry and accurate texture information, even as it scales to larger scenes. The hierarchical approach not only improves the system's scalability but also contributes to its robustness, as it can effectively manage the complexities of large and dynamic environments.

NICE-SLAM takes extensive evaluations on a variety of challenging datasets, including real-world indoor scenes. The results show that NICE-SLAM consistently outperforms existing methods in both mapping accuracy and camera tracking quality. The system's ability to predict and fill in unobserved regions of the scene, a task that many traditional SLAM systems fail to accomplish effectively, further underscores its utility in real-world applications. Moreover, the system's efficiency is notable, with reduced computational demands compared to other neural implicit SLAM approaches, thanks to its innovative use of hierarchical grid-based encoding.

CO-SLAM [21]:

CO-SLAM (Coordinate and Sparse Parametric Encodings for Neural Real-Time SLAM), introduced in 2023, represents a significant leap forward in the development of real-time SLAM systems, particularly in its innovative approach to integrating both coordinate and parametric encodings for scene representation. The method addresses critical limitations in previous SLAM systems, such as the trade-off between detail preservation and computational efficiency, by proposing a hybrid representation that leverages the strengths of both encoding types.

CO-SLAM's primary contribution is its joint encoding strategy that combines the coherence and smoothness priors of coordinate-based encodings with the speed and local detail preservation capabilities of sparse parametric encodings. This hybrid approach allows CO-SLAM to achieve fast convergence and robust camera tracking while maintaining high-fidelity surface reconstruction. The use of a multi-resolution hash-grid, which is integral to the sparse parametric encoding, enables the system to efficiently manage memory while accurately representing high-frequency local features, a significant improvement over previous methods that either suffered from over-smoothing or failed to perform plausible hole-filling in unobserved areas.

Another key innovation in CO-SLAM is its approach to bundle adjustment (BA). Unlike traditional SLAM systems that restrict bundle adjustment to a limited set of keyframes, CO-SLAM performs global bundle adjustment by sampling rays from all past keyframes. This strategy significantly enhances the robustness and accuracy of camera pose estimation, leading to improved tracking performance without the need for cumbersome keyframe selection processes. The ability to perform global bundle adjustment efficiently, even in real-time, positions CO-SLAM as a state-of-the-art method in terms of both reconstruction quality and computational performance.

The system's performance has been thoroughly evaluated across multiple datasets, including Replica, ScanNet, and TUM RGB-D, where it consistently outperforms existing methods like iMAP and NICE-SLAM in both mapping accuracy and tracking speed. CO-SLAM demonstrates the ability to operate at frame rates between 10-17Hz, depending on the dataset, which is a significant improvement over its predecessors. The experimental results underscore CO-SLAM's ability to maintain high-fidelity reconstructions while ensuring real-time performance, a critical requirement for practical SLAM applications in dynamic and complex environments.

2.3.2 Visual SLAM with Gaussian Splatting

Although the NeRF-based SLAM method is different from the previous traditional SLAM methods, it can be optimized end-to-end and can directly input the original pixel values without any feature extraction. However, it also has problems such as slow rendering speed, poor image quality, and low positioning accuracy. Then, 3D Gaussian Splatting-based SLAM not only fully inherits the advantages of NeRF-based SLAM methods, but also comprehensively makes up for the shortcomings, achieving better real-time, better rendering quality, and higher positioning accuracy. Recently, plenty of new methods have been published on Arxiv, here we introduce the first open source 3DGS-based SLAM research as follow:

SplaTAM [6]:

SplaTAM (Splat, Track and Map) introduces the first approach to dense RGB-D SLAM by utilizing 3D Gaussian splatting, a method that has shown great promise in the realm of high-fidelity 3D reconstruction and real-time rendering. This method leverages the efficiency of Gaussian splatting to render and optimize volumetric representations of a scene in real-time, addressing some of the critical limitations observed in traditional radiance field-based SLAM systems.

One of the primary contributions of SplaTAM is its ability to perform real-time dense SLAM using a single unposed monocular RGB-D camera, a significant advancement over existing methods that typically rely on multiple posed cameras or suffer from inefficiencies in rendering and optimization. SplaTAM achieves this by using an explicit volumetric representation of the scene with 3D Gaussians, which allows for fast rendering (up to 400 frames per second) and efficient optimization via differentiable rendering. This capability makes SplaTAM particularly well-suited for applications that require both high accuracy in camera tracking and high fidelity in scene reconstruction.

SplaTAM's use of 3D Gaussian splatting enables several key advantages. First, it allows for the creation of explicit maps with clear spatial extents, making it easier to determine which areas of a scene have been mapped and which remain unexplored. This is crucial for efficient camera tracking, as it ensures that new frames are only compared against already mapped portions of the scene, thereby reducing computational overhead and improving tracking accuracy. Additionally, SplaTAM allows for the straightforward expansion of the map by adding new Gaussians as needed, offering flexibility in handling dynamic and complex environments.

The system's performance has been rigorously evaluated on various benchmarks, including ScanNet++, Replica, and TUM-RGBD datasets, where it consistently outperforms state-of-the-art dense SLAM methods in camera pose estimation, map construction, and novel-view synthesis. Notably, SplaTAM demonstrates superior performance in challenging scenarios such as large camera displacements and texture-less environments, where traditional SLAM methods often fail. The system's ability to render photo-realistic images at high frame rates further underscores its potential for real-time applications in areas such as robotics, augmented reality, and autonomous navigation.

For open source community, SplaTAM sets a new benchmark in the field of dense SLAM by integrating 3D Gaussian splatting with an innovative tracking and mapping pipeline. Its ability to handle challenging real-world scenarios, coupled with its high efficiency and accuracy, positions it as a significant advancement over previous approaches. SplaTAM's contributions are likely to influence future research in SLAM, particularly in developing systems that require both high fidelity and real-time performance in dynamic environments.

In conclusion, SplaTAM offers an explicit and manipulable scene representation by modeling the world with 3D Gaussians. It use silhouette images in the optimization process to determine which scene parts are new and need mapping, enhancing the robustness and accuracy of the mapping process. It performs camera pose estimation and updating the Gaussian map concurrently by the incoming RGB-D frames, facilitating fast rendering at 400 FPS and efficient optimization.

Chapter 3

Preliminaries

In this chapter, I will briefly introduce the basic mathematical description and derivation for 3D Gaussian Splatting and various point cloud registration technique. The details about the Gaussian splatting are intended as an introduction for readers who are not familiar with differentiable rendering. For senior researchers, you can quickly skip into second section, it will be used for my mapping and tracking module design.

3.1 3D Gaussian splatting

3D Gaussian splatting is a method used in graphics to represent and manipulate volumetric data through the use of Gaussian functions[31]. This technique leverages the mathematical properties of Gaussian distributions to achieve volume representations for parameterized point cloud. In this section, I will introduce 3D Gaussian splatting by focusing on how to parameterize 3D Gaussian representation, how to splatting these data into image and how to optimize these data.

3.1.1 Differentiable 3D Gaussian Volume

For arbitrary variables a, b, c follow Gaussian distribution: $a, b, c \sim \mathcal{N}(0, 1)$, the 3D probability distribution of $\mathbf{v}$ consisting of a, b, c can be expressed as:

$$p(\mathbf{v}) = p(a)p(b)p(c) = \frac{1}{(2\pi)^{3/2}} \exp\left(-\frac{a^2 + b^2 + c^2}{2}\right) \tag{3.1}$$

In 3D space, it is used to be vectorised as $\mathbf{v} = [a, b, c]^\top$:

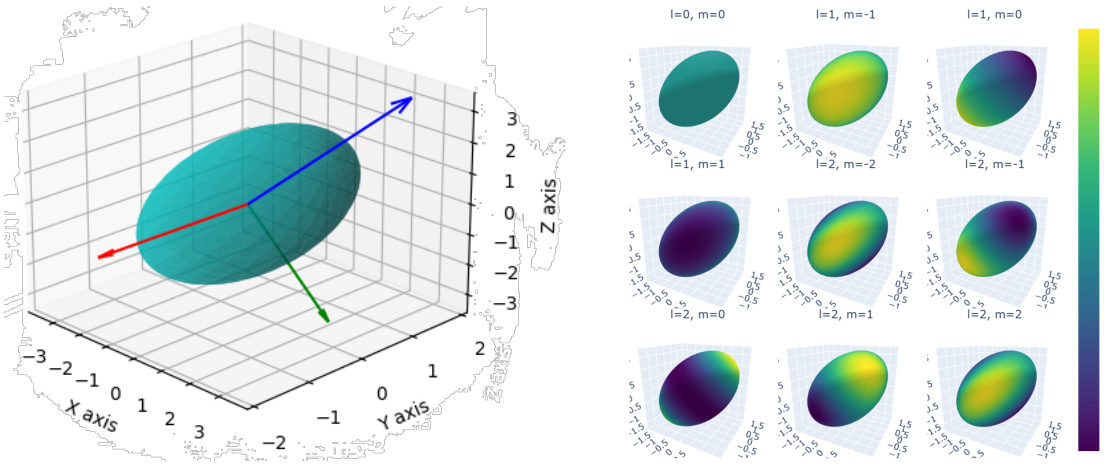
$$p(\mathbf{v}) = \frac{1}{(2\pi)^{3/2}} \exp\left(-\frac{1}{2}\mathbf{v}^\top\mathbf{v}\right) \tag{3.2}$$

For an arbitrary distribution: $\mathbf{x} = \boldsymbol{\mu} + A\mathbf{v}$, we substitute $\mathbf{v} = A^{-1}(\mathbf{x} - \boldsymbol{\mu})$:

$$p(\mathbf{v}) = \frac{1}{(2\pi)^{3/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top (A^{-1})^\top A^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{3.3}$$

Then commutative integration $dv = d(A^{-1}(\mathbf{x} - \boldsymbol{\mu})) = |\det(A)|^{-1}dx$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{3/2}|\det(A)|} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top (AA^\top)^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{3.4}$$



(a) The shape of ellipsoid defined by covariance (b) Ellipsoids with spherical harmonic functions

Figure 3.1: The visualization of a differentiable 3D Gaussian volume.

From equation(3.4), we derive the probability density function of a 3D ellipsoid which follow Gaussian probability distribution. Observe this equation, we continue substitute $\Sigma = AA^T$, and then simplified the constants to get a pure and uniform expression.

Uniform Gaussian Definition:

$$G(x) = \exp\left(-\frac{1}{2}x^T\Sigma^{-1}x\right) \tag{3.5}$$

That means a Gaussian shaped ellipsoid centered at mean x with covariance matrix Σ. This formulation allows for the efficient computation and manipulation of 3D Gaussian ellipsoid, facilitating their application in various computer graphics and visualization shown as Figure 3.1a.

Given 3D column vector coordinates variable $x = [a, b, c]^\top$, the covariance can be represented as:

$$\Sigma = \begin{bmatrix} \sigma_a^2 & \mathrm{Cov}(a,b) & \mathrm{Cov}(a,c) \\ \mathrm{Cov}(b,a) & \sigma_b^2 & \mathrm{Cov}(b,c) \\ \mathrm{Cov}(c,a) & \mathrm{Cov}(c,b) & \sigma_c^2 \end{bmatrix} \tag{3.6}$$

The covariance matrix Σ controls the expansion and rotation of an ellipsoid along the 3-axis direction, where the eigenvectors of the covariance matrix are the ellipsoid symmetry axes which on the diagonal σ^2. Meanwhile, notice that, we can use covariance to express Gaussian ellipsoid, nevertheless, generating the random numbers to fill in the covariance matrix Σ will not work. because each 3D Gaussian ellipsoid corresponds to a Σ, but not every Σ corresponds to an ellipsoid, it requiring the covariance matrix Σ to be positive semi-definite.

Spherical harmonic functions and positional encoding:

In addition to the position and shape of the Gaussian ellipsoid, there are other key properties, spherical harmonic functions and positional encoding are integral components in the Gaussian splatting framework, each contributing to the system's ability to produce high-fidelity renderings by efficiently encoding the directional and spatial characteristics of the scene[27, 28]. Positional encoding provides the network with a detailed and nuanced understanding of spatial positions within the scene, ensuring that fine-grained spatial details are captured accurately. Meanwhile, spherical harmonic functions complement this by encoding the directional properties of light at those positions.

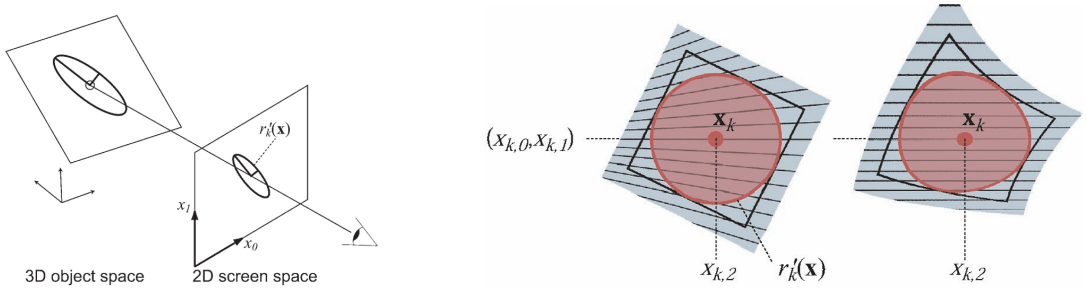
Positional encoding is a technique used to embed spatial information into neural networks, particularly in models that process data with an inherent spatial structure, such as images or 3D scenes. In NeRF, positional encoding can provide the high frequency information of 3D coordinates which enable the neural network to better learn the spatial relationships within the scene. By transforming the 3D coordinates through a series of sinusoidal functions with varying frequencies, positional encoding provides the model with a richer representation of spatial information, allowing it to capture fine details and intricate spatial patterns. This technique is particularly effective in neural radiance fields, where the precise encoding of spatial positions is crucial for accurate reconstruction of high-frequency details in the scene, such as sharp edges and fine textures. The incorporation of positional encoding in neural network facilitates the model's ability to represent complex spatial variations, resulting in more accurate and detailed 3D reconstructions.

Spherical harmonic functions are mathematical functions that represent data on the surface of a sphere shown in Figure 3.1b. They are commonly used in the fields of physics and computer graphics to model directional data, such as lighting in 3D scenes. In the context of Gaussian splatting, spherical harmonics are employed to efficiently encode the directional information of a scene's radiance. This allows the system to capture complex lighting and shading effects with a relatively compact representation. Specifically, the coefficients of spherical harmonics can be used to approximate the angular dependency of light reflecting off surfaces, enabling high-fidelity rendering of scenes with varying lighting conditions. The use of spherical harmonics in Gaussian splatting enhances the realism of the rendered images by providing an efficient way to encode and reconstruct the directional properties of light in the scene.

3.1.2 Tile-based rasterizer for image rendering

Rasterization is the process of converting the vector data of a graphic or image into pixel data that can be displayed on a computer screen. By converting graphics to the pixel level, computers can more easily process and display graphics while ensuring that the image is drawn on the screen at a high rate of speed.

In Neural Radiance field, researchers use ray-casting and volume rendering for rendering each pixel of an image. This is one of the vital factors why NeRF is very slow. In 3D Gaussian splatting, people adapt another forward rendering method called tile-based rasterizer, which is the opposite approach from NeRF. The tile-based rasterizer divides the graphics into smaller, manageable tiles that can be processed independently. This method enhances the efficiency of rendering by optimizing memory usage and computation, especially for high-resolution images for 3D Gaussian representation.



(a) The projection from volume to plane[31] (b) Local affine transform and exact transform[31]

Figure 3.2: The 3D Gaussian elliptical weighted average splatting process.

In 3D Gaussian, we project Gaussian ellipsoids from 3D space into 2D image shown as Figure 3.2a, instead of casting a ray from pixel to scene. In this step, we make a linear local approximation shown as Figure 3.2b. Then, In this way, we can project all Gaussian ellipsoids in the range of the view frustum, and finally we can obtain the rendering image by adding all their projections in order. The derivation details can be organized as follow.

Exact projective transformation:

Transform the coordinates from 3D object space $\mathbf{u}$ to camera space $\mathbf{t}$ using an affine transformation:

$$\mathbf{t} = \varphi(\mathbf{u}) = \mathbf{W}\mathbf{u} + \mathbf{d} \tag{3.7}$$

We follow notation in [31], where $\mathbf{W}$ is the viewing transformation matrix and $\mathbf{d}$ is the translation vector.

Next, we perform the projective transformation, which maps the camera coordinates to ray coordinates. Given the camera space coordinates $\mathbf{t} = (t_0, t_1, t_2)^\top$, the mapping to ray space $\mathbf{x} = (x_0, x_1, x_2)^\top$ is given by:

$$\mathbf{x} = \phi(\mathbf{t}) = \left(\frac{t_0}{t_2}, \frac{t_1}{t_2}, \kappa(t_0, t_1, t_2)\right)^\top \tag{3.8}$$

where $\kappa(t_0, t_1, t_2)$ is a function that handles the projection to the third dimension.

Solve this equation, we can get the inverse projective transformation:

$$\phi^{-1}(\mathbf{x}) = \begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix} \tag{3.9}$$

where $l = \left\|(x_0, x_1, 1)^T\right\|$.

Local affine approximation[31]:

Since the projective transformation $\phi(\mathbf{t})$ is not affine, we approximate it locally around a point $\mathbf{t}_k$ using a Taylor series expansion. The local affine approximation is given by:

$$\phi_k(\mathbf{t}) = \mathbf{x}_k + \mathbf{J}_k(\mathbf{t} - \mathbf{t}_k) \tag{3.10}$$

where $\mathbf{J}_k$ is the Jacobian matrix of ϕ at $\mathbf{t}_k$.

The full mapping from object space to ray space can now be described by combining the viewing transformation $\mathbf{t} = \phi(\mathbf{u})$ and the local affine approximation of the projective transformation:

$$\mathbf{x} = \phi_k(\varphi(\mathbf{u})) = \mathbf{J}_k\mathbf{W}\mathbf{u} + \mathbf{x}_k + \mathbf{J}_k(\mathbf{d} - \mathbf{t}_k) \tag{3.11}$$

To transform the variance matrix of the Gaussian ellipsoid from object space to ray space, we focus on the variance matrix (covariance) expression:

$$\mathbf{V}'_k = \mathbf{J}_k\mathbf{W}\mathbf{V}_k\mathbf{W}^\top\mathbf{J}_k^\top \tag{3.12}$$

where $\mathbf{V}_k$ is the variance matrix in object space, and $\mathbf{V}'_k$ is the transformed variance matrix.

To obtain the local affine transformation, we approximate the non-linear projection transformation using derivative on the Jacobian matrix. The Jacobian $\mathbf{J}_k$ is computed at the point $\mathbf{t}_k$ and used to linearly approximate the transformation in the vicinity of $\mathbf{t}_k$. Using the second-order approximation, the problem of the non-linearity of the projection transformation is solved while preserving the closed form of the Gaussian function. Thus, the projection on image is still Gaussian distribution, this mathematical property is very important.

Sorting and alpha blending:

As I mention above, but here we follow notation in [5]. To project the 3D Gaussian ellipsoid to 2D plane, the mean μ and covariance matrix Σ in camera coordinates μ', Σ' are computed as:

$$\mu' = KW \begin{bmatrix} \mu \\ 1 \end{bmatrix}^{\top}, \quad \Sigma' = JW\Sigma W^T J^T \tag{3.13}$$

where J is the Jacobian of the affine approximation of the projective transformation. W is the transformation matrix and K is the intrinsic matrix.

Then, the screen is divided into 16×16 tiles. Gaussians are culled against the view frustum and tiles, with only those intersecting the view frustum kept. Gaussians are sorted using GPU Radix sort based on view space depth and tile ID. Approximate α-blending is performed, ensuring efficient rendering and gradient computation. Finally, we can obtain the color on all pixels.

$$C = \sum_{i \in N} T_i \alpha_i c_i, \quad T_i = \prod_{j=1}^{i-1} (1 - \alpha_j), \quad \alpha_i = \text{opacity}_i \times G_i(x) \tag{3.14}$$

In this way we can implement fast and accurate back-propagation by keeping track of the desired number of traversals of sorted splats. During the backward propagation, the accumulated opacity values are used to compute gradients, storing only the final accumulated opacity α at each step.

3.1.3 Optimization with density control

Once, we get rendering image, we can compare it with ground truth, use back-propagation to make them as similar as possible. The process of optimizing 3D Gaussian splatting[5] involves controlling the density of the splats to ensure efficient and accurate rendering. Because of the uncertainty of initialization and the ambiguity of projection, geometry may inevitably be misplaced. Therefore, the optimization process needs to be able to create geometry, and also destroy geometry in the wrong place. This includes adjusting the parameters of the Gaussian functions to achieve the desired visual quality and computational performance.

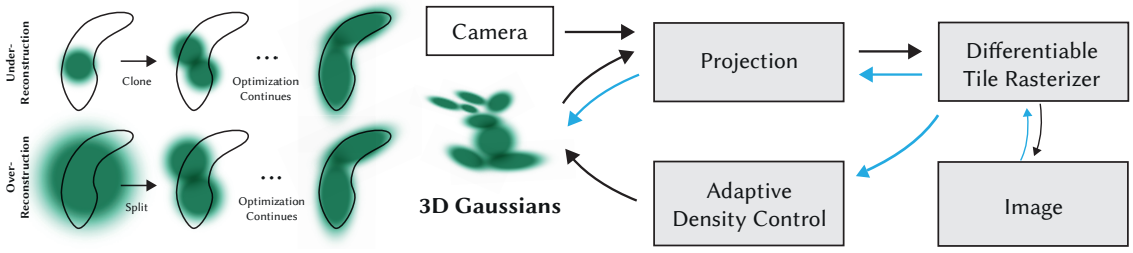


Figure 3.3: The differential rendering flow for 3D Gaussian density control and optimization[5].

Reparameterization of Covariance Matrix:

Recall the first section, an ellipsoid can be rotated by deflating the ball axially and then rotating it. So given a scaling matrix S and a rotation matrix R, we can find the corresponding Σ, and vice versa if Σ is positive semi-definite matrix. The covariance matrix Σ is represented in terms of a scaling matrix S and a rotation matrix R:

$$\Sigma = AA^T = RSS^T R^T \tag{3.15}$$

Because large homogeneous regions can be captured with a small number of large anisotropic Gaussian functions, the quality of the 3D Gaussian covariance parameter is crucial for the compactness of the geometric representation. Meanwhile, to obtain a smoother gradient, we will apply an exponential activation function to the scale of the covariance.

Objective function for optimization:

The loss function used in the forward process is a combination of L1 loss and a D-SSIM term, and optimization uses Stochastic Gradient Descent:

$$L = (1 - \lambda)L1 + \lambda L_{\text{D-SSIM}} \tag{3.16}$$

Based on extensive experimental experience, researchers typically set $\lambda = 0.2$ for all tests.

To achieve accurate scene representation and free-view synthesis, an iterative optimization process is applied for the covariance matrix and other attributes. From back-propogation process, optimization adjusts the positions p, opacities α, and covariance Σ of 3D Gaussians, along with Spherical Harmonics coefficients representing color c.

Density Control for point clouds:

Adaptive control of Gaussian density is divided into three situations:

- **Pruning**: Trimming unnecessary Gaussian by removing transparent points with opacity α less than the threshold ϵ every 100 iterations
- **Densification**: Adaptive filling of blanks in reconstructed areas shown as Figure 3.3
  - Under-recon: Cloning a new Gaussian, if small-scale geometry is under-represented. Clone the Gaussian and move along the gradient in order to cover the geometry.
  - Over-recon: Splitting into two smaller Gaussians, if a large Gaussian in a high variance region needs to be refined, and reducing the scale by a factor $\phi = 1.6$.
- **Filter**: Remove large Gaussians to avoid overlap in the periodic.

In summary, density control method provides a way to balance the trade-off between rendering quality and computational efficiency, ensuring that the splatting process is both effective and controllable.

3.2 Point cloud registration

Point cloud registration is the process of aligning two point clouds so that they can be merged into a single, unified representation. The inputs are source point cloud P_s and target point cloud P_t, which are two scans from two different perspectives. The purpose is to find a best space transfer T such that P_s and P_t can be coincide perfectly. So it is a fundamental problem in computer vision and 3D geometry processing.

Generally, current point cloud registration method can be divided into two types, the one is correspondence-based registration, the other is correspondence-free registration. The most popular methods in correspondence-based registration are feature-based approaches, which achieve alignment by first extracting distinctive features from the point clouds and then matching these features to compute the optimal transformation. These methods are particularly useful when the point clouds are only partially overlapping or when the initial alignment is unknown. For example,

FPFH (Fast Point Feature Histograms) is designed to describe the geometric properties around a point in a point cloud. SHOT (Signature of Histograms of Texture description) aims to create a robust and discriminative descriptor for point cloud features that incorporates both geometric and appearance-based information. 4PCS (4-Point Congruent Sets) is designed for global point cloud alignment by identifying congruent sets of four points between two point clouds. In addition to the methods described above, there are also some other correspondence-based registration methods, here I just give a roughly categorization as follows:

- Feature-based point alignment: FPFH[32], SHOT[33], 4PCS[34]
- Probabilistic model Alignment: Normal distributions transform[35]
- Iterative closest point alignment: Point2point, Point2plane, Generalized ICP[36]

Although the feature point-based approach is efficient, however, not all the point cloud information is used in this method. Therefore, in this section, we focus on Iterative closest point alignment approach, meanwhile, we register rigid bodies by default, that is, we consider that objects and scenes do not involve any scaling and deformation, so we can directly represent transformations between point clouds by rotation and translation. Iterative closest point algorithms is widely used in lidar sensors, which can be divided into two stages, coarse-alignment and fine-alignment. Coarse-alignment performs a quick estimation when the point cloud position is completely unknown to provide initial values for subsequent optimization because the registration is very sensitive to the initial values, and then a more accurate position is optimized by fine-alignment. However, in the visual SLAM scene, each frame is obtained from the cameras of continuous time nodes, and the camera positions of each frame are relatively close to each other, so the initial value problem of coarse alignment can be ignored and the fine alignment can be performed directly.

3.2.1 Point to point iterative closest point

The Iterative Closest Point (ICP) algorithm aligns two point clouds by minimizing the distance between corresponding points. In the point-to-point variant, the objective is to find a transformation $\mathbf{T}$ (composed of a rotation $\mathbf{R}$ and a translation $\mathbf{t}$) that minimizes the following error function:

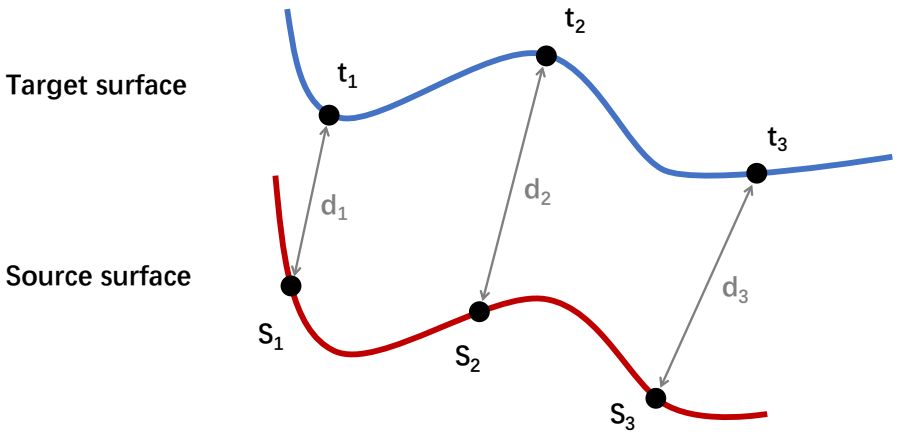


Figure 3.4: Point to point distance function between target points and source points.

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} \|\mathbf{p}_i - (\mathbf{R}\mathbf{q}_i + \mathbf{t})\|^2 \tag{3.17}$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ are corresponding points in the two point clouds.

Algorithm 1 Point-to-Point ICP

Initialize $\mathbf{R}$ and $\mathbf{t}$
repeat
    for each point $\mathbf{q}_i$ in the source point cloud **do**
        Find the closest point $\mathbf{p}_i$ in the target point cloud
    end for

    Compute the optimal transformation $\mathbf{T} = (\mathbf{R}, \mathbf{t})$
    Minimizing the error function $E(\mathbf{R}, \mathbf{t})$
    Apply the transformation to the source point cloud
until convergence

3.2.2 Point to plane iterative closest point

The point-to-plane ICP variant minimizes the distance between a point in one point cloud and the plane defined by the corresponding point's normal in the other point cloud. The error function is:

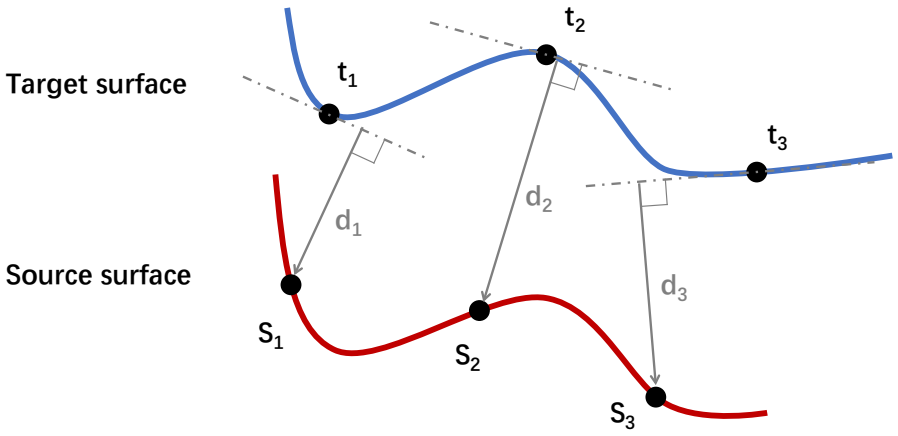


Figure 3.5: Point to plane distance function between target points and source points.

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} ((\mathbf{p}_i - (\mathbf{R}\mathbf{q}_i + \mathbf{t})) \cdot \mathbf{n}_i)^2 \tag{3.18}$$

where $\mathbf{n}_i$ is the normal at $\mathbf{p}_i$.

Pseudocode:

Algorithm 2 Point-to-Plane Iterative closest point

Initialize $\mathbf{R}$ and $\mathbf{t}$
repeat
    for each point $\mathbf{q}_i$ in the source point cloud **do**
        Find the closest point $\mathbf{p}_i$ in the target point cloud
        Compute its normal $\mathbf{n}_i$
    end for

    Compute the optimal transformation $\mathbf{T} = (\mathbf{R}, \mathbf{t})$
    Minimizing the error function $E(\mathbf{R}, \mathbf{t})$
    Apply the transformation to the source point cloud
until convergence

3.2.3 Generalized iterative closest point

Generalized ICP[36] combines the point-to-point and point-to-plane error metrics to improve robustness. The error function is a weighted combination of both metrics:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} \left(\alpha \left\| \mathbf{p}_i - (\mathbf{R}\mathbf{q}_i + \mathbf{t}) \right\|^2 + \beta \left((\mathbf{p}_i - (\mathbf{R}\mathbf{q}_i + \mathbf{t})) \cdot \mathbf{n}_i \right)^2 \right) \tag{3.19}$$

where α and β are weights that balance the contributions of the point2point and point2plane terms.

Next, we introduce a probabilistic model to the error function above and incorporate the minimization step of the standard ICP algorithm as follow:

Consider two point clouds $A = \{a_i\}_{i=1}^N$ and $B = \{b_i\}_{i=1}^N$ where a_i and b_i are corresponding points. We assume that the true underlying points $\hat{A} = \{\hat{a}_i\}$ and $\hat{B} = \{\hat{b}_i\}$ generate the observed points:

$$a_i \sim \mathcal{N}(\hat{a}_i, C_{A_i}) \tag{3.20}$$

$$b_i \sim \mathcal{N}(\hat{b}_i, C_{B_i}) \tag{3.21}$$

where C_{A_i} and C_{B_i} are the covariance matrices associated with the points. The goal is to find the transformation T that aligns the point clouds A and B.

For the correct transformation T^*, we have:

$$\hat{b}_i = T^* \hat{a}_i \tag{3.22}$$

For an arbitrary transformation T, define the residual:

$$d_i^{(T)} = b_i - T a_i \tag{3.23}$$

which is assumed to be normally distributed:

$$d_i^{(T^*)} \sim \mathcal{N}(0, C_{B_i} + T C_{A_i} T^T) \tag{3.24}$$

Then, we can estimate the transformation T by maximizing the likelihood of the observed data:

$$T = \arg\max_T \prod_{i=1}^{N} p(d_i^{(T)}) = \arg\max_T \sum_{i=1}^{N} \log p(d_i^{(T)}) \tag{3.25}$$

The log-likelihood is given by:

$$\log p(d_i^{(T)}) = -\frac{1}{2} d_i^{(T)T} (C_{B_i} + T C_{A_i} T^T)^{-1} d_i^{(T)} \tag{3.26}$$

Finally, the objective function to minimize is:

$$T = \arg\min_T \sum_{i=1}^{N} d_i^{(T)T} (C_{B_i} + T C_{A_i} T^T)^{-1} d_i^{(T)} \tag{3.27}$$

Pseudocode:

Algorithm 3 Generalized Iterative Closest Point (Generalized-ICP)

Initialize $\mathbf{R}$ and $\mathbf{t}$
repeat
    for each point $\mathbf{b}_i$ in the source point cloud B **do**
        Find the closest point $\mathbf{a}_i$ in the target point cloud A
        Compute the surface normal $\boldsymbol{\nu}_i$ at $\mathbf{a}_i$
        Compute the surface normal $\boldsymbol{\mu}_i$ at $\mathbf{b}_i$
        Compute the covariance matrices C_{A_i} and C_{B_i} based on $\boldsymbol{\nu}_i$ and $\boldsymbol{\mu}_i$
    end for

    Compute the optimal transformation $\mathbf{T} = (\mathbf{R}, \mathbf{t})$
    Minimizing the error function

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} (\mathbf{b}_i - \mathbf{R}\mathbf{a}_i - \mathbf{t})^T \left(C_{B_i} + \mathbf{R}C_{A_i}\mathbf{R}^T\right)^{-1} (\mathbf{b}_i - \mathbf{R}\mathbf{a}_i - \mathbf{t})$$

    Apply the transformation $\mathbf{T}$ to the source point cloud B
until convergence

Chapter 4

Methodology

In this chapter, I will briefly introduce the system overview in the first section by presenting the pipelines. It will show what modules are in the operation process, how each module works during the run time of the system, and how the data interact with each other. Then I will describe the details for two key modules, the mapping module and the tracking module.

4.1 System framework

4.1.1 Module pipelines

Visual SLAM system is designed to estimate the camera's trajectory and simultaneously build a map of the environment, here we focus on using RGB-D sequences as input. The pipeline operates as shown in Figure 4.1. This pipeline illustrates the iterative optimization of the SLAM process, where each new frame refines the map and the camera's trajectory in dynamic camera motion.

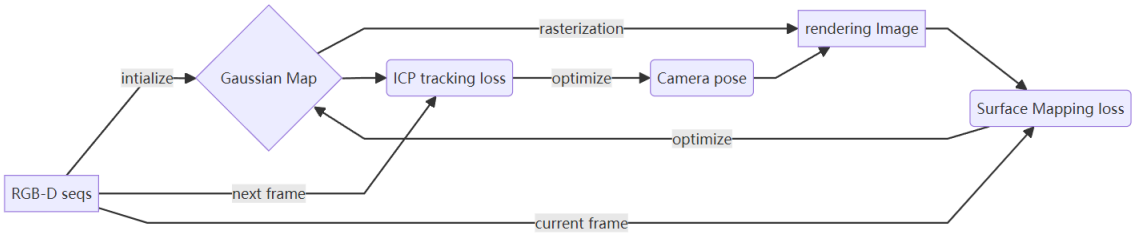


Figure 4.1: Pipeline of our method for visual SLAM.

For the whole structure, the surface mapping loss and ICP tracking loss are used to continuously optimize the Gaussian Map and the camera pose separately, improving the system's accuracy over time. To begin with my methodology, let me quickly go through the key modules:

- Initialization: The process begins with the RGB-D sequences, which consist of synchronized color and depth images. These sequences are used to initialize the Gaussian Map.
- Gaussian Map: The Gaussian Map serves as the core representation of the environment. It is continuously updated and refined as new frames are processed.
- Tracking: The Iterative Closest Point (ICP) algorithm is employed to minimize the tracking loss by aligning the current frame with the rendering image from Gaussian map. This alignment is crucial for accurate camera pose estimation.

- Camera Pose Optimization: Based on the ICP tracking loss, the camera pose is optimized to ensure accurate localization within the environment. This optimized camera pose is essential for consistent trajectory estimation. It will be used in the next mapping module.
- Rendering Image: The obtained camera pose allows for the rendering of an image from the map, providing a visual representation of the environment from the camera's perspective. We can directly get color image and depth image from Gaussian map.
- Mapping: This step involves calculating the surface mapping loss, which ensures that the rendered image aligns well with the observed data from the current frame. This loss is used to further refine the Gaussian Map.

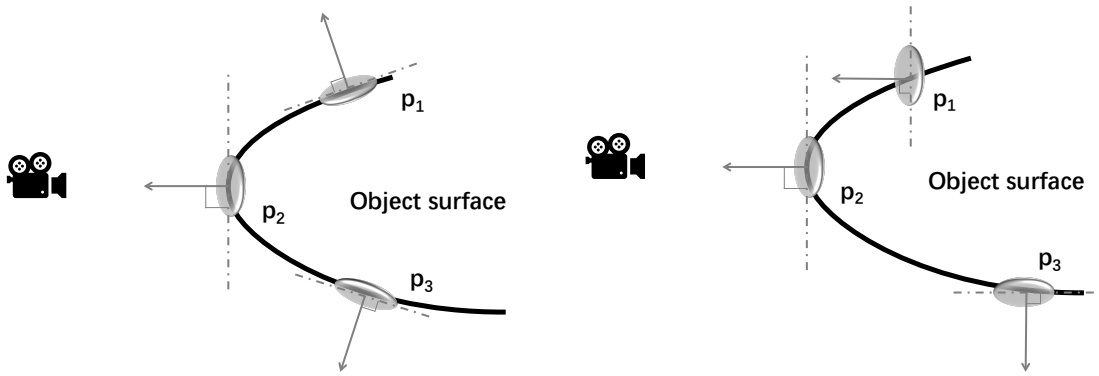
In this section, I demonstrate the important modules to show how this system works. In this pipeline, some process details are not included, such as key-frame selection, Gaussian map densification, point cloud pruning, and so on. I will introduce them in the next section.

4.2 Mapping module

4.2.1 Gaussian surfel modeling

In this project, we adopted 2DGS as the basic Gaussian Surfel Representation in the mapping module, as I mentioned in the introduction chapter, 2DGS is a planar disk consisting of 2x2 covariance matrix. It can be divided into two tangent vectors and one scaling vector. Then, we propose two different modeling strategies: surface-fitting modeling and sight-line fitting modeling.

The surface fitting modeling approach aims to let each 2DGS disk be attached to the surface of the object, meanwhile, the normal of each 2DGS element is in the same direction as the normal of that surface. This approach is very similar to tangent-space normal mapping, the disk sticks to the surface and orients to normal. However, we found that due to the lack of a dimension in the 2D Gaussian, when the angle between the line of sight and the surface of the object varies a lot, map gaps and blanks can easily appear when the Gaussian surface elements are degraded. Therefore, we hope that the Gaussian surface element can be oriented towards the camera in the non-contour region of the object so that the covariance in the line of sight can be spread out as much as possible to fully utilize the surface area of the 2D Gaussian. Therefore, here we design another modeling strategy, sight-line fitting modeling, which also aims to make each 2DGS disk stick close to the surface of the object, however, it divides the surface of the object into two categories, with the normal of the disk on the front surface facing the camera, and the tangent of the disk on the side surface facing the camera.



(a) Surface fitting modeling (b) Sight-line fitting modeling

Figure 4.2: Two type of surface fitting strategies.

Surface fitting modeling:

In computer graphics, normals are crucial for tasks such as shading, rendering, and surface reconstruction. Normals can be estimated from depth maps, which provide the distance from the camera to points on a surface. The depth information can be used to compute the gradient of the surface, and subsequently, the normal at each point. As shown in Figure 4.2a, all disk located on the surface and follow the curvature of local tangent line, and keep normal in the same direction.

The normal consistency is maintained by aligning the normals of 2D splats with the actual surface normals. This is achieved by adjusting the splat normals to match the gradients of the depth map at the intersection points, where the opacity reaches a threshold value, usually set as 0.5. The normal $\mathbf{N}(x, y)$ at a point (x, y) on the depth map $p(x, y)$ can be computed using the cross product of the gradients in the x and y directions. These gradients can be approximated using finite differences.

The gradient in the x direction $\nabla_x p$, and the gradient in the y direction $\nabla_y p$, are given by:

$$\nabla_x p \approx \frac{p(x+1, y) - p(x-1, y)}{2} \tag{4.1}$$

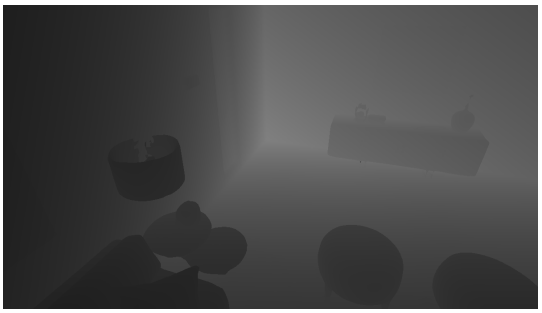
$$\nabla_y p \approx \frac{p(x, y+1) - p(x, y-1)}{2} \tag{4.2}$$

The normal $\mathbf{N}(x, y)$ is then computed as:

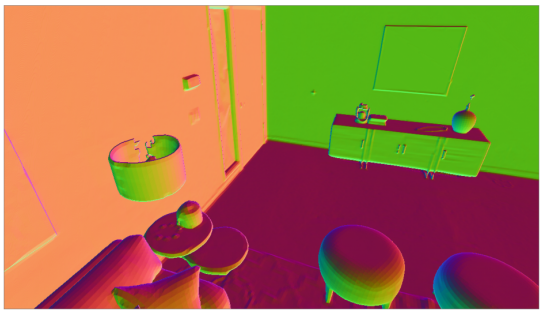
$$\mathbf{N}(x, y) = \frac{\nabla_x p \times \nabla_y p}{|\nabla_x p \times \nabla_y p|} \tag{4.3}$$

Where the cross product and the magnitude ensure that the normal is a unit vector aligned with the local surface orientation. In this way, the final surface normal map could be computed from depth shown as 4.3c.

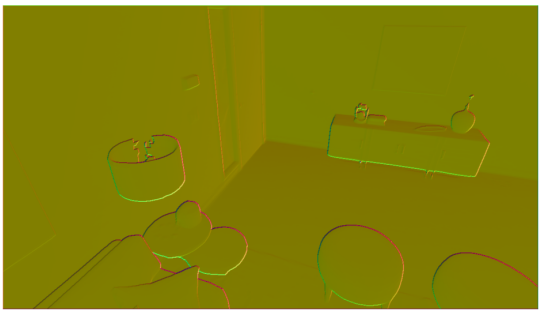
(a) RGB image



(b) Depth image



(c) Surface normal



(d) Edge gradient

Figure 4.3: Obtaining geometric information from the depth image.

Sight-line fitting modeling:

When the Gaussian normal is parallel to the line of sight, the area of the disk is maximized for projection onto the image plane. To reduce the blank, our method is applying edge detection. The Sobel operator is a popular filter in image processing. It approximates the gradient of the image intensity at each pixel, which helps in identifying edges by highlighting regions where the intensity changes sharply.

The Sobel operator uses two 3x3 convolution kernels: one for detecting the gradient in the horizontal direction (x-direction) and one for the vertical direction (y-direction). The kernel for detecting the gradient in the x-direction G_x is:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{4.4}$$

The kernel for detecting the gradient in the y-direction G_y is:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{4.5}$$

Let $I(x, y)$ represent the intensity of the image at pixel (x, y). The gradients in the x and y directions are computed by convolving the image with the Sobel kernels:

$$G_x(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} G_x(i, j) \cdot I(x + i, y + j) \tag{4.6}$$

$$G_y(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} G_y(i, j) \cdot I(x + i, y + j) \tag{4.7}$$

Where $G_x(i, j)$ and $G_y(i, j)$ are the elements of the Sobel kernels, and $I(x + i, y + j)$ represents the pixel intensities in the neighborhood of (x, y). The magnitude of the gradient at each pixel is given by:

$$|\mathbf{G}(x, y)| = \sqrt{G_x(x, y)^2 + G_y(x, y)^2} \tag{4.8}$$

In some situation, the magnitude can be approximated to reduce computational complexity:

$$|\mathbf{G}(x, y)| \approx |G_x(x, y)| + |G_y(x, y)| \tag{4.9}$$

The direction of the gradient, which indicates the edge orientation, is computed as:

$$\theta(x, y) = \arctan\left(\frac{G_y(x, y)}{G_x(x, y)}\right) \tag{4.10}$$

A large image gradient indicates a significant change in pixel intensity, typically corresponding to edges or boundaries within the image, then the normal oriented to side. When small gradient, the normal oriented to camera. As shown in Figure 4.2b, the front disk faces the camera in order to utilize all the area of the plane for rendering, and the side disks only need to provide sides to render the edge and corner.

4.2.2 Mapping loss optimization

The framework from SplaTAM[6] that performs camera pose tracking, optimizes the camera's position and orientation across frames, adds new Gaussians to the map to enhance detail, selects keyframes, and performs optimization to build and refine the 3D map, and saves intermediate results and states for robustness. This pipeline is highly customizable with various configurations, allowing different SLAM strategies to be tested and utilized. In this part, I will briefly introduce the details of mapping iteration.

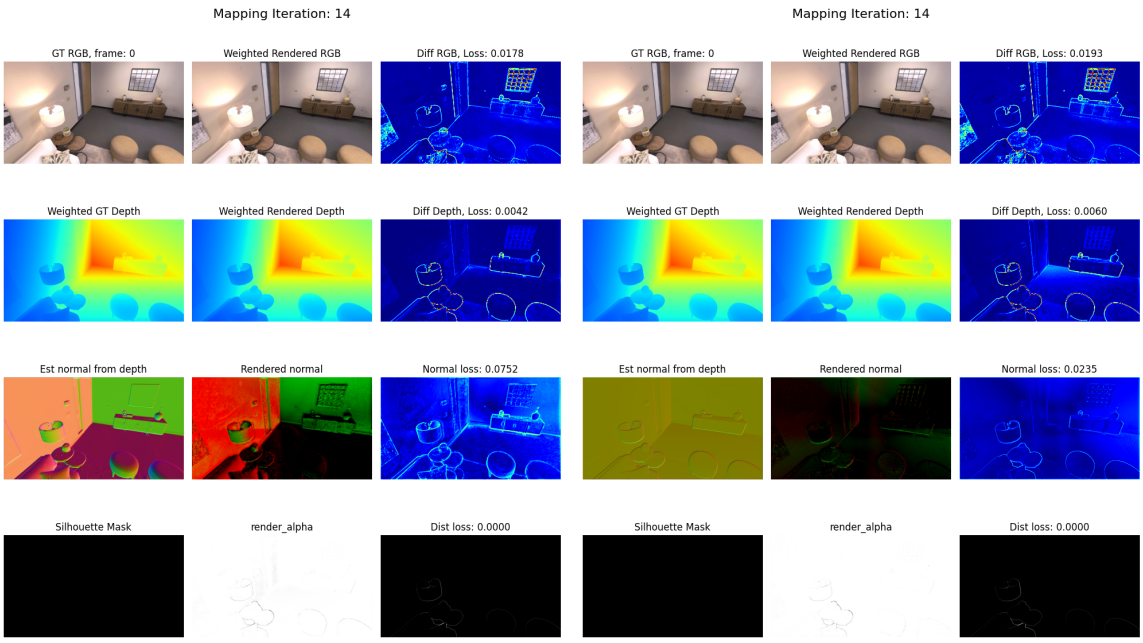
First, before mapping, there is a function to check whether there are enough Gaussian to rendering an image. If not enough, new Gaussians will be added to the scene according to the depth map. after adding new Gaussians to the scene, Then, updating the 3D Gaussian parameters as usual.

Next, during the mapping process, there are several Key-frames, which are selected based on the current camera pose, depth data, and overlapping criteria. Selected key-frames are used to optimize the map in this iteration, ensuring the selected frames contribute effectively to the map's quality. Then, the optimizer is reset again for full map optimization. Notice that, in the mapping processing, the optimized variables are only the attributes of Gaussian, for example, its means3D, RGB colors, rotation matrix, scale vector, and opacities. The camera's parameters will not take any iterative updates.

In each iteration, a keyframe or the current frame is randomly selected, and loss is computed for that frame. Then, back-propagation and the gradient updates by optimizer. That means, in the current iteration of the round, not only the current frame is optimized, but also previous frames are randomly selected to be optimized together.

After the mapping loop, the current frame is added to the keyframe list whether it met certain conditions. Meanwhile, the pipeline checks if it's time to save a checkpoint. If so, the parameters are saved, and the keyframe time indices are stored. The CUDA cache is emptied at the end of each frame to manage GPU memory efficiently.

Here is a visualization of the mapping process of Replica[37] in progress shown as Figure 4.4.



(a) Surface fitting mapping process (b) Sight-line fitting mapping process

Figure 4.4: Two type of surface fitting strategies.

4.3 Tracking module

4.3.1 Camera tracking approach

In the SpalaTAM[6], they use photometric loss to estimate the camera pose, which is the difference between rendered image and camera image. However, for our mapping process, we use 2D Gausssian disk instead of 3D Gaussian ellipsoid. For each 2D Gaussian disk, it is planar geometry which has surface property like normal information. Besides, in visual SLAM setting, there is a large overlap in the images of neighboring frames. Consider all of this, in our project, we adopted Iterative closest point algorithm to estimate the camera pose in tracking process.

First, let me start from camera model. The pinhole camera model relates a 3D point $\mathbf{P} = (X, Y, Z)$ in the camera coordinate system to its 2D projection (u, v) on the image plane through the intrinsic matrix $\mathbf{K}$. The intrinsic matrix is given by:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

Where f_x, f_y are the focal lengths along the x and y axes, respectively. And c_x, c_y are the coordinates of the principal point.

The relationship between the 3D coordinates and the image coordinates is:

$$u = \frac{f_x \cdot X}{Z} + c_x \tag{4.12}$$

$$v = \frac{f_y \cdot Y}{Z} + c_y \tag{4.13}$$

Given a depth image $Z = D(u, v)$ that provides the depth Z for each pixel (u, v), we can derive the corresponding 3D point $\mathbf{P} = (X, Y, Z)$ using the inverse of the above relationship:

$$X = \frac{(u - c_x) \cdot Z}{f_x} \tag{4.14}$$

$$Y = \frac{(v - c_y) \cdot Z}{f_y} \tag{4.15}$$

To convert the depth image into a point cloud, the steps shown as follow:

1. Extract the intrinsic parameters c_x, c_y, f_x, f_y from the intrinsics matrix.
2. Generate a grid of pixel coordinates (u, v) for the image.
3. Normalize coordinates by subtracting the principal point and dividing by the focal length.
4. Compute the 3D coordinates X, Y, Z for each pixel using the depth values $D(u, v)$.
5. Return the 3D points as a point cloud.

The resulting 3D points in camera frame are computed as:

$$\mathbf{P}(u, v) = \left(\frac{(u - c_x) \cdot D(u, v)}{f_x}, \frac{(v - c_y) \cdot D(u, v)}{f_y}, D(u, v) \right) \tag{4.16}$$

Finally, once we get point cloud, we can directly apply tracking target function in Chapter3:

- Point to point distance in Equation3.17
- Point to plane distance in Equation3.18

4.3.2 Tracking loss optimization

Before tracking, if the current frame index is greater than 0, the camera pose is initialized based on the previous pose using a constant velocity model:

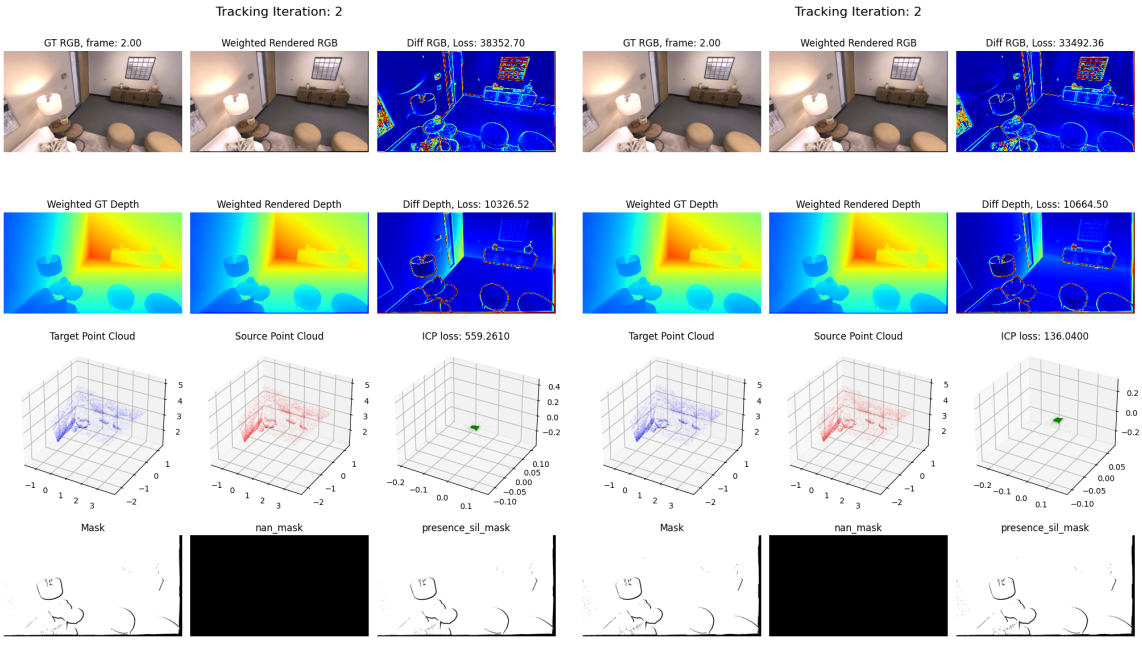
$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + (\mathbf{x}(t) - \mathbf{x}(t - \Delta t)) \tag{4.17}$$

Next, variables to store the best candidate camera rotation and translation are initialized. If the loss is lower than the previous minimum loss, the current rotation and translation are saved as the best candidates. The initial minimum loss is set to a very high value.

Then, the tracking module is a while loop for iterative optimization until a termination condition is met. Loss for the current frame is computed by ICP loss function, with other various tracking-related losses. After back-propagation by computed loss, the optimizer will update the model parameters.

During each iteration, the optimizer is reset, and learning rates are set based on the configuration. Notice that, in the tracking processing, the optimized variables are only the camera's parameters. The attributes of Gaussian, for example, its means3D, RGB colors, rotation matrix, scale vector, and opacities will not take any iterative updates.

After exiting the tracking loop, the best candidate rotation and translation are assigned to the corresponding parameters for the current time step. Here is a visualization of the tracking process of Replica[37] in progress shown as Figure 4.5.



(a) Point to point tracking process (b) Point to plane tracking process

Figure 4.5: Two type of tracking distance function.

Chapter 5

Experiment

5.1 Set up

5.1.1 Configuration

In UCL Malet Place Engineering Building Lab 105, there are 24 custom PCs with i9 processors, 128 GB RAM and Nvidia GeForce RTX 3090 graphics cards. That provides the computation resource for this project. The configuration of the our develop environment are shown as follow:

- NVIDIA GeForce RTX 3090 (24GB)
- Ubuntu 20.04 (Linux-5.4.0)

We use anaconda and pip command to install all required packages, and the dependencies packages and its version shown as follow:

- pytorch 1.12.1
- torchvision 0.13.1
- torchaudio 0.12.1
- cudatoolkit 11.6
- open3D 0.16.0
- wandb.ai
- other packages

Here, we highlight the public implementations from other authors which we used in our experiment, thanks the contributors of the following repositories for their open source code:

- SplaTAM[6]
- Differential Gaussian Rasterization[5]
- Differential Surfel Rasterization[9]
- GradSLAM[38] & ConceptFusion[39]

5.1.2 Code Implement

For our code implement, we follow the existing open-source framework in order to better compared our proposed mapping results and tracking strategies. In this project, we decided to develop our algorithm from SplaTAM[6]'s framework, that is one of most efficient and simple 3D Gaussian splatting visual SLAM systems. Here, we modify the tracking module and mapping module into our own algorithm implementation, aim to facilitate the subsequent ablation experiments and comparison experiments. The overview code structure of SplaTam are shown as follow:

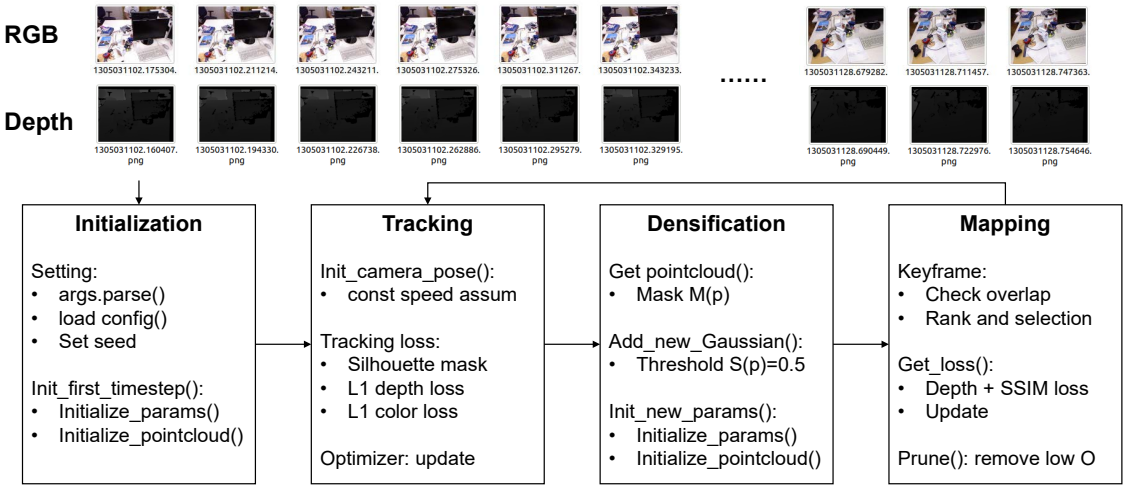


Figure 5.1: Overview of SplaTAM's framework for visual SLAM.

5.1.3 Benchmark datasets

We list some popular datasets used to test and evaluate the performance of previous NeRF-based SLAM methods and Gaussian-based SLAM methods, and then we will make a brief introduction for all these datasets.

- Replica dataset[37] (Room0, Room1, Room2, Office0, Office1, Office2, Office3, Office4)
- TUM-RGBD dataset[40] (fr1/desk, fr1/desk2, fr1/room, fr2/xyz, fr3/office)
- Original ScanNet[41]
- ScanNet++[42] (8b5caf3398 (S1) and b20a261fdf (S2))

Replica[37] is a high-quality synthetic dataset created by Facebook Reality Labs. It contains highly realistic 3D reconstructions of indoor environments, with accurate texture and geometry. It provides ground truth for depth, surface normals, and segmentation. Meanwhile, the movement of camera pose between two consecutive frames is relatively small, making it particularly useful for developing and evaluating algorithms.

TUM-RGBD[40] created by Computer Vision Group in Technical University of Munich, is one of the most widely used datasets for RGB-D SLAM. It consists of several indoor sequences recorded with a Microsoft Kinect sensor, providing both RGB images and depth maps. The dataset also includes ground truth trajectories captured with a highly accurate motion capture system, making it suitable for evaluating the performance of SLAM algorithms in terms of localization accuracy.

Original ScanNet[41]is a large-scale RGB-D video dataset with 3D ground truth. However, it is harder than Replica and TUM-RGBD datasets, since the quality of the raw RGB-D sequences it provided are poor. In its 1,500 scans of indoor scenes, its depth maps are sparse and a amount of pixels are missing, and the color images are mixed with a lot of motion blur artifacts. It also Includes instance-level and semantic-level annotations for semantic SLAM, which can be used in 3D reconstructions with semantic segmentation annotations.

ScanNet++[42] is an extension of the ScanNet dataset, offering enhanced data for more sophisticated tasks. It includes additional annotations and improved data quality, focusing on the challenges of instance segmentation and 3D object detection in cluttered indoor environments. This dataset provides high-quality color and depth images and includes a second capture loop for each scene. This second loop allows researchers to evaluate entirely new, unseen views, enhancing the dataset's utility for testing model generalization. Generally, researcher often use the DSLR captures from S1 and S2. However, the challenge with ScanNet++ lies in its camera pose movement interval. because the gap between two consecutive frames in ScanNet++ is equivalent to about a 30 times larger than that in the Replica dataset, highlighting the increased difficulty in tracking and localization.

Considering the performance breakthrough of optic sensor in recent years, the current vision camera on the market have super high resolution with impressive frequency rate, so the Replica[37] dataset is closer to the products that users use on a daily basis. Therefore, in this project, we choose it to conduct our experiment.

5.1.4 Metrics in evaluation

We follow all metrics for final performance evaluation as previous works. In visual SLAM, these evaluation metrics are used to measure the performance of the proposed method, the detailed explanations given later.

- For measuring rendering mapping performance
  - Depth L1 loss
  - PSNR (Peak Signal-to-Noise Ratio)
  - SSIM (Structural Similarity Index)
  - LPIPS (Learned Perceptual Image Patch Similarity)
- For camera pose estimation tracking performance
  - ATE RMSE (average absolute trajectory error)

Depth L1 Loss

Depth L1 loss is a metric used to measure the difference between predicted depth values and ground truth depth values in a pixel-wise manner.

$$\text{Depth L1 Loss} = \frac{1}{N} \sum_{i=1}^{N} |d_i - d'_i| \tag{5.1}$$

- d_i is the predicted depth value for pixel i.
- d'_i is the ground truth depth value for pixel i.
- N is the total number of pixels.

This metric is used in visual SLAM to evaluate the accuracy of depth map predictions. Lower Depth L1 loss indicates that the SLAM system more accurately reconstructs the depth information of the environment.

PSNR (Peak Signal-to-Noise Ratio)

PSNR is a widely used metric to measure the quality of a reconstructed or compressed image compared to its original version. It represents the ratio between the maximum possible power of a signal and the power of distorting noise that affects the fidelity of its representation.

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{5.2}$$

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left[I(i,j) - K(i,j) \right]^2 \tag{5.3}$$

- MAX_I is the maximum possible pixel value of the image (for an 8-bit image, this is 255).
- MSE is the Mean Squared Error between the original and the reconstructed image.
- $I(i,j)$ is the pixel value from the original image.
- $K(i,j)$ is the pixel value from the reconstructed image.
- m and n are the dimensions of the images.

In visual SLAM, PSNR is used to evaluate the quality of RGB rendering. Higher PSNR indicates better image quality, implying that system has effectively captured the visual details.

SSIM (Structural Similarity Index)

SSIM is a perceptual metric that measures the similarity between two images. Unlike PSNR, which is a purely mathematical metric, SSIM considers changes in structural information, which is closer to how the human visual system perceives differences.

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5.4}$$

- μ_x and μ_y are the average pixel intensities of images x and y.
- σ_x^2 and σ_y^2 are the variances of x and y.
- σ_{xy} is the covariance between x and y.
- C_1 and C_2 are constants to stabilize the division.

SSIM is used in visual SLAM to assess the perceived quality of the rendered images. A higher SSIM value indicates a more accurate reconstruction that preserves the structure of the scene.

LPIPS (Learned Perceptual Image Patch Similarity)

LPIPS is a deep learning-based metric that measures perceptual similarity between images. It is designed to align more closely with human perception compared to traditional metrics like PSNR and SSIM.

$$\text{LPIPS}(x,y) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} ||w_l \cdot (\phi_l(x) - \phi_l(y))_{hw}||_2^2 \tag{5.5}$$

- w_l are learned weights.
- H_l and W_l are the height and width of the feature map at layer l.
- $\phi_l(x)$ and $\phi_l(y)$ are feature maps from layer l of a pre-trained network for images x and y.

LPIPS is used to evaluate the perceptual quality of rendered images in visual SLAM. Lower LPIPS values indicate higher perceptual similarity to the original image, meaning the rendered image is more visually accurate.

ATE RMSE (Average Trajectory Error Root Mean Square Error)

ATE RMSE measures the deviation between the estimated trajectory by the SLAM system and the ground truth trajectory. It quantifies how accurately the SLAM system tracks the camera.

$$\text{ATE RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}||T_i^{est} - T_i^{gt}||^2} \tag{5.6}$$

- T_i^{est} is the estimated transformation (pose) at time i.
- T_i^{gt} is the ground truth transformation (pose) at time i.
- N is the total number of poses.

In visual SLAM, we will take trajectory alignment and correct the scale factor before computing ATE RMSR, in order to put the trajectory into the same frame. Lower ATE RMSE values indicate that the SLAM system more accurately follows the true camera trajectory.

5.2 Our results

Eval	Metric	R_0	R_1	R_2	Of_0	Of_1	Of_2	Of_3	Of_4	Avg.
Final State	ATE RMSE↓	0.289	0.581	0.660	0.461	0.407	0.342	0.713	0.564	0.502
	Avg Depth L1↓	0.616	0.786	0.841	0.667	0.881	1.370	1.526	1.852	1.067
	Avg LPIPS↓	0.094	0.155	0.112	0.143	0.169	0.171	0.175	0.221	0.155
	Avg SSIM↑	0.972	0.942	0.971	0.954	0.947	0.917	0.924	0.907	0.942
	Avg PSNR↑	32.51	32.52	33.58	35.67	36.93	30.46	28.70	30.44	32.60
	Latest Pose Error	0.719	1.053	3.568	1.088	0.816	0.153	1.559	0.284	1.155
	Latest Rela Error	0.029	0.041	0.985	0.053	0.027	0.023	0.078	0.108	0.168
Mapping	Depth RMSE↓	0.405	0.299	1.952	0.378	0.420	0.329	0.464	0.264	0.564
	Mapping PSNR↑	35.21	38.07	27.62	38.08	40.88	36.09	35.43	40.45	36.48
	Iter Depth Loss	0.601	0.303	0.613	0.410	0.198	0.522	0.395	0.908	0.494
	Iter RGB Loss	0.702	0.501	1.044	0.591	0.654	0.979	0.715	1.012	0.775
Tracking	Depth RMSE↓	0.421	0.279	1.582	0.402	0.390	0.324	0.500	0.264	0.520
	Tracking PSNR↑	33.49	37.28	26.83	37.95	38.52	33.96	32.72	40.38	35.14
	Iter Depth Loss	330.3	238.7	1315.0	317.7	311.8	266.7	389.8	215.4	423.2
	Iter RGB Loss	2851	1551	4720	1530	1543	1473	2529	1329	2191
System	Duration	356	418	452	369	385	294	331	463	384
	Map Iter time	102.9	123.6	137.1	109.1	117.3	85.2	95.6	129.7	112.6
	Track Iter time	94.6	109.9	113.6	96.3	95.6	75.8	87.8	134.8	101.0
	GPU Memory	13.59	16.28	18.36	13.47	17.79	15.40	12.56	15.48	15.37
	Power Usage	315.1	305.3	301.9	312.9	313.1	332.7	332.5	310.4	315.5

Table 5.1: **Overall results on Replica dataset.** From left to right, there are all room and office in the Replica. The 'final state' is evaluating all frames in the end. The 'tracking', 'mapping' and 'system' is real time monitoring during each frame. The unit as follow: Depth L1 [cm], RMSE [cm], Number of GS [million], Duration [minute], Iter time [ms], Memory [GB], Power Usage [w].

In this paper, we maintain the same units for all evaluation table as Table 5.1 under the same metric indicator.

Our SLAM mapping results are shown in the Figure 5.2. From the reconstructed maps, it is obvious that the reconstruction quality of Office 0 and Office 1 is very high, and thus the corresponding PSNR metrics are over 35. Office 3 and Office 4, on the other hand, have relatively below-average rendering quality, and we can observe a lot of ripples on the walls.

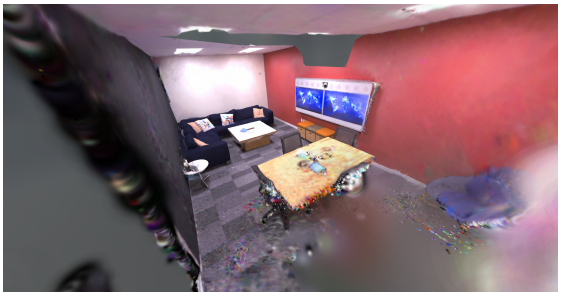
(a) Visualization for mapping room0

(b) Visualization for mapping room1

(c) Visualization for mapping room2

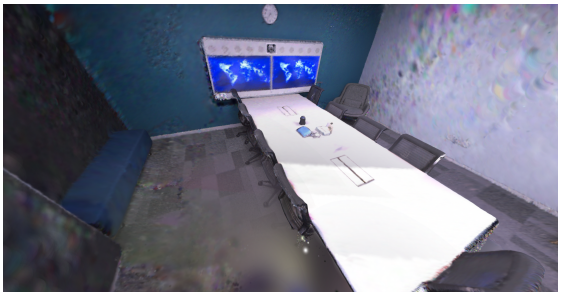
(d) Visualization for mapping office0

(e) Visualization for mapping office1



(f) Visualization for mapping office2

(g) Visualization for mapping office3



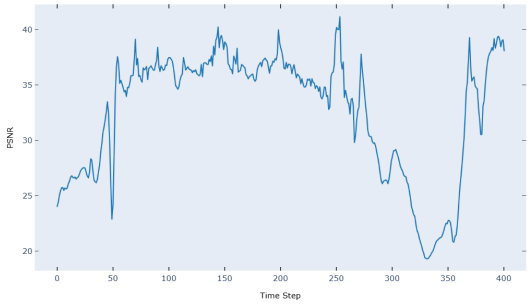
(h) Visualization for mapping office4

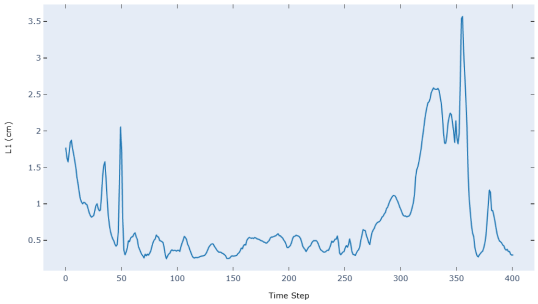
Figure 5.2: Visualization for building map

Although the 3D Gaussian and Neural radiance field have ability to fill in areas that are not seen, such as the underside of the tables in office 3, and the backs of chairs in the room 2. But because of the lack of actual RGB images to guide the rendering of the colors, we can see small colorful patches.

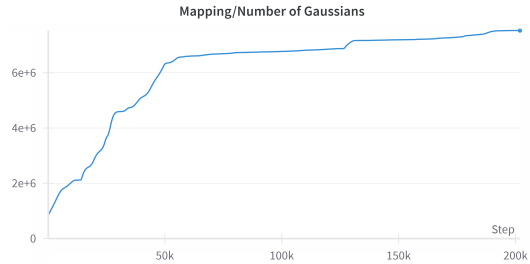
5.2.1 Mapping details

Here, we use Replica Room1 as an demonstration to present our mapping process details.

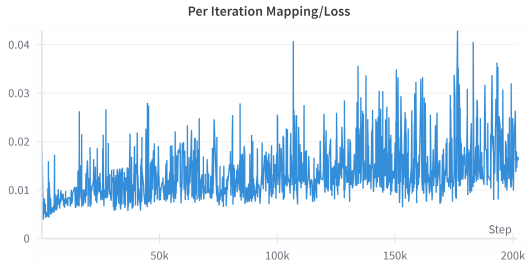




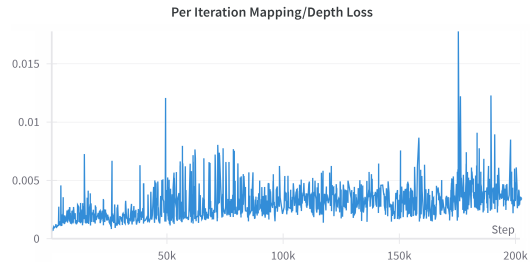
(a) Final State evaluation for every 5 frames



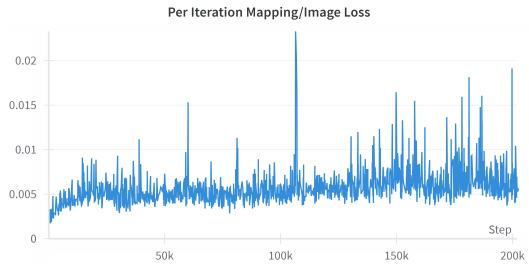
(b) The total number of point clouds



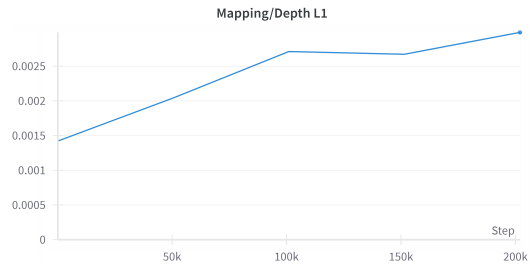
(c) The loss during each mapping iteration



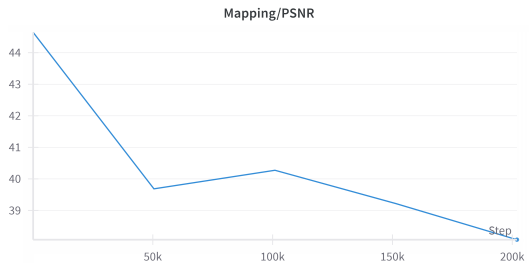
(d) Depth loss during each mapping iteration



(e) Color loss during each mapping iteration



(f) Depth L1 for rendered depth image



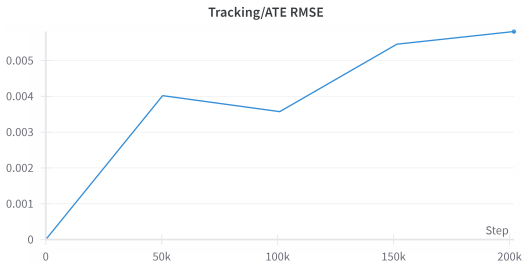
(g) PSNR for rendered color image

Figure 5.3: The results of mapping module.

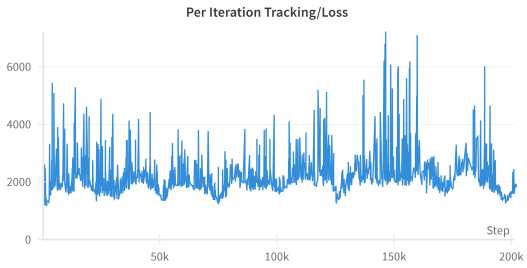
From the final results, it is clear that our method suffers from big mapping errors at 0-250 and 1500-1800 frames. It corresponds to the time when the camera motion is large in the original dataset. With the process of building the image, the traversal of the space is increasing, and the depth error and image quality are getting worse. Also from the number of point clouds, after the 500th frame, the growth trend tends to level off, corresponding to the time when the camera has looked around the room. From the loss trend figure, there are many peaks in the middle, indicating that our method does not perform smoothly enough between some frames.

5.2.2 Tracking details

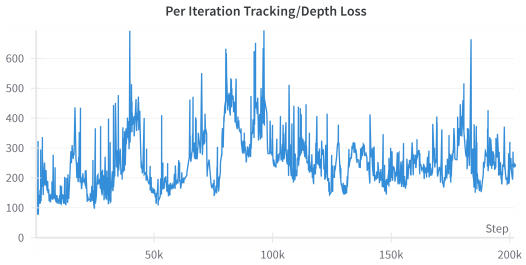
Here, we use Replica Room1 as an demonstration to present our tracking process details.



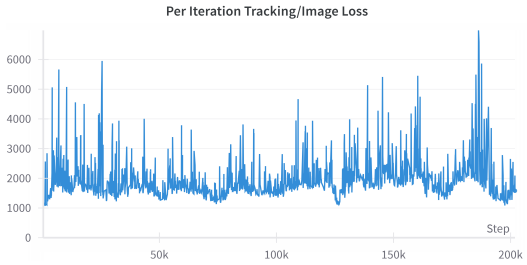
(a) ATE RMSE during tracking for every 500 frames



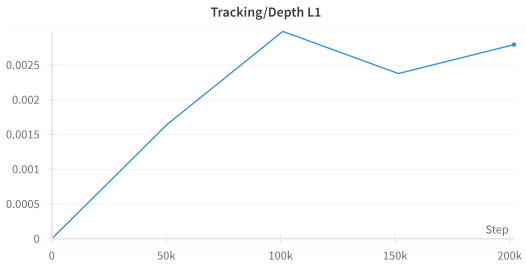
(b) The loss during each tracking iteration



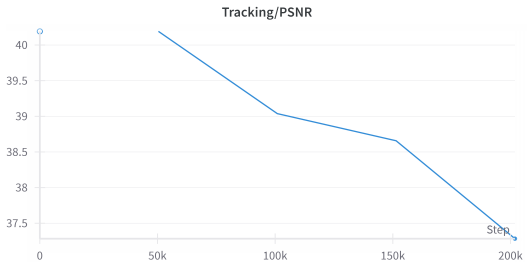
(c) Depth loss during each tracking iteration



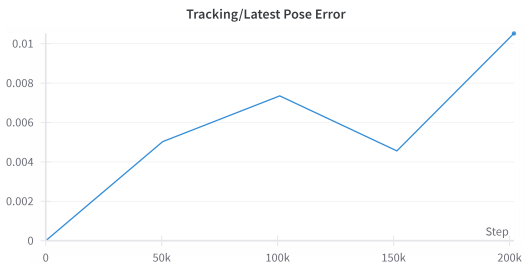
(d) Color loss during each tracking iteration



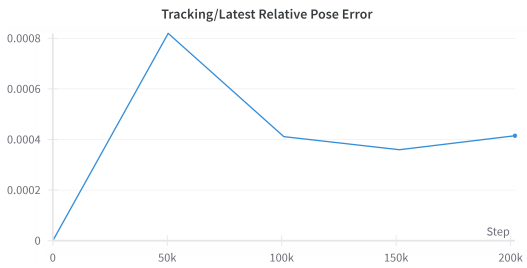
(e) Depth L1 for rendered depth image



(f) PSNR for rendered depth image



(g) Pose error for every 500 frames



(h) Relative pose error for every 500 frames

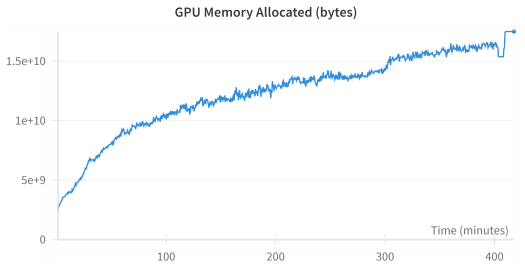
Figure 5.4: The results of tracking module.

From the trend plot of the loss, there are many peaks in the middle, indicating that our tracking method does not perform stable enough between some frames. Inaccurate camera pose estimation will lead to inaccurate map building.

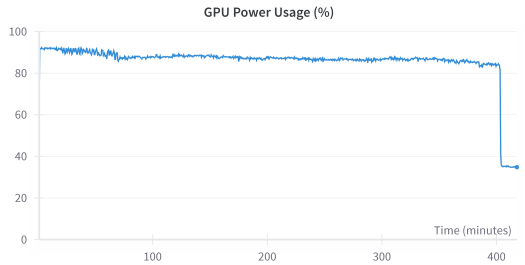
Meanwhile, as the trajectory of the camera moving in space becomes longer and longer, the trajectory error, depth error, pose error, and relative pose error, all them show a growing trend, probably due to the accumulation of drift error caused by the lack of global constraints.

5.2.3 Other results

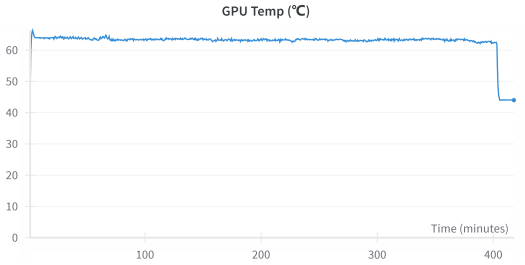
Here, we use Replica Room1 as an demonstration to present our hardware details.



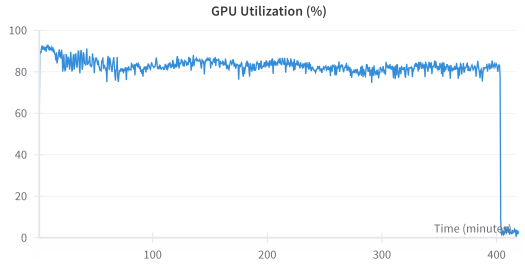
(a) Real-time monitoring GPU memory



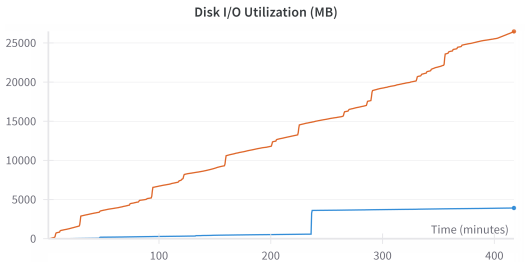
(b) Real-time monitoring GPU power



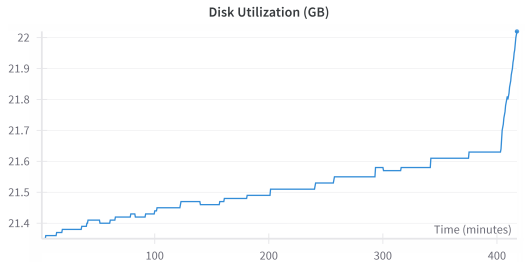
(c) Real-time monitoring GPU Temp



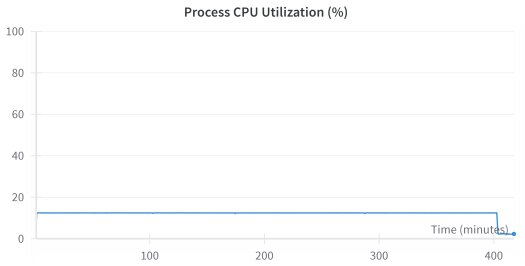
(d) Real-time monitoring GPU Usage



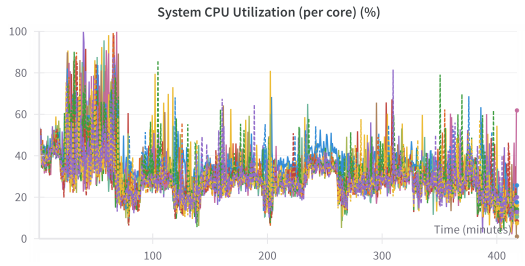
(e) Real-time monitoring Disk I/O



(f) Real-time monitoring Disk usage



(g) Real-time monitoring CPU usage



(h) Real-time monitoring core usage

Figure 5.5: The results of system hardware.

In NeRF-based SLAM and Gaussian-based SLAM methods, the model relies heavily on GPU graphics cards. We test our results on RTX 3090 with 24 GB memory and a Maximum power of 350w. During usage, this SLAM system will take 67% of the memory and 90% of the power in GPU. For CPU, the overall usage plot shows a steady and low utilization across time. This indicates the process itself is not very demanding in terms of CPU usage. However, for each core, it is high and variable, which indicates our algorithm is multi-threaded but not efficiently parallelized.

5.3 Extended experiment

5.3.1 Ablation experiment

In this section, we only replace one part of original SLAM system with different algorithmic implementations, in order to present our proposed combination is a relatively better design. Then, we also propose two nearest point search strategies: The one is taking the same pixel location as the closest point directly, and the other is using K-Nearest Neighbors to find the closest point.

Considering the size and number of point clouds (1200 x 680), we use a KD-tree to speed up the searching process, even then, performing the exact closest point matching is still slow (Approximately 5-6 times slower). Therefore, we limit all iterations to 10 times per frame during tracking and mapping for all experiments. Other hyper-parameters, such as the learning rates of all the Gaussian properties are kept consistent. The final results are shown as Table 5.2.

Eval	Metric	3DGS	2DGS	R_{Surf}	R_{Sight}	p2point	w/corr	p2plane	w/corr
Final State	ATE RMSE↓	0.553	0.571	0.665	0.564	2.263	**0.501**	1.311	0.508
	Avg Depth L1↓	0.863	**0.759**	0.987	0.811	0.965	0.761	0.935	0.799
	Avg LPIPS↓	0.161	0.117	0.146	0.145	0.142	0.118	0.135	**0.115**
	Avg SSIM↑	0.946	**0.959**	0.948	0.948	0.946	0.956	0.949	0.958
	Avg PSNR↑	29.77	29.57	29.18	29.10	28.65	**29.86**	28.88	29.85
	Latest Pose Error	1.881	**1.675**	1.816	2.232	5.843	1.980	4.511	2.019
	Latest Rela Error	0.214	0.382	0.099	0.464	0.161	0.215	0.074	**0.156**
Mapping	Depth RMSE↓	1.032	1.435	1.436	**0.967**	2.536	1.341	1.389	1.357
	Mapping PSNR↑	26.65	26.39	28.66	26.67	26.91	25.36	**28.91**	26.30
	Iter Depth Loss	0.815	0.911	**0.414**	0.883	0.625	1.072	0.821	0.855
	Iter RGB Loss	2.548	1.572	**0.851**	2.184	1.779	1.853	1.748	2.041
Tracking	Depth RMSE↓	0.581	0.640	**0.534**	0.708	1.689	0.607	1.099	0.661
	Tracking PSNR↑	27.09	27.11	**29.36**	26.67	27.84	26.01	29.12	26.77
	Iter Depth Loss	5201	5348	4261	5538	1384	5184	**902**	5411
	Iter RGB Loss	5554	5366	4432	5307	5040	5726	**4251**	5358
System	Duration	**37.13**	46.14	45.35	46.16	47.09	273.22	46.55	271.15
	Map Iter time	**24.19**	38.01	37.16	37.59	38.65	40.45	38.11	40.11
	Track Iter time	**22.01**	34.19	33.20	34.13	36.40	710.21	36.24	705.65
	GPU Memory	9.837	**8.560**	9.105	8.843	8.767	8.890	8.835	8.734
	Power Usage	**278.2**	309.0	309.7	308.6	311.7	313.7	309.6	318.1

Table 5.2: **Ablation experiment on Replica room0.** From left to right, there are original 3DGS, original 2DGS, 2DGS with surface normal regular term, sight-line fitting term, 2DGS with point2point tracking and it with correlation point matching, 2DGS with point2plane tracking, and it with correlation point matching.

After we replace the Gaussian ellipsoid with Gaussian disk, the common change is an overall decrease in image rendering quality. If add regularization, we get a lower PNSR, which are negative enhancement since forcing constraints on the surface is normal. To our surprise, 3DGS leads the 2DGS approach in both depth error and trajectory error, the planar modeling cannot give any improvement for tracking. Although 3DGS has better depth error, if we visualize its point cloud, we find that the 3D Gaussian makes a large noise in the fitted plane, this will be discussed later.

For normal regularization terms, the sight-line fitting model can achieve a smaller average depth error and ATE error than surface fitting model. For tracking objective function, point-to-plane can lead a smaller depth loss in iteration. Applying the point matching can improve trajectory accuracy, however, it is time-consuming.

5.3.2 Comparison experiment

We select three NeRF-based SLAM methods and three 3DGS-based SLAM methods as baseline:

- NeRF-based SLAM: Co-SLAM [21], NICE-SLAM [20], Point-SLAM [43]
- 3DGS-based SLAM: MonoGS [7], SplaTAM [6], GS-SLAM [44]

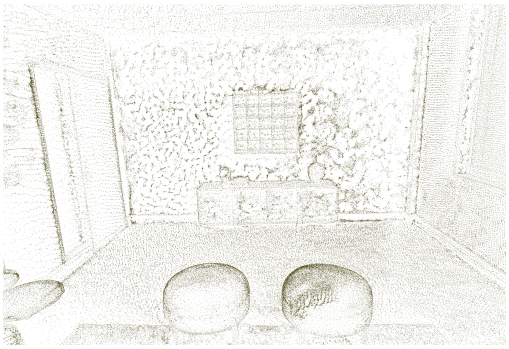
Method	Metric	R_0	R_1	R_2	Of_0	Of_1	Of_2	Of_3	Of_4	Avg.
NICE-SLAM[20]	PSNR↑	22.12	22.47	24.52	29.07	30.34	19.66	22.23	24.94	24.42
	SSIM↑	0.689	0.757	0.814	0.874	0.886	0.797	0.801	0.856	0.809
	LPIPS↓	0.33	0.271	0.208	0.229	0.181	0.235	0.209	0.198	0.233
	Depth↓	2.11	1.68	2.90	1.83	2.46	8.92	5.93	2.38	3.53
	ATE↓	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.13	1.07
Co-SLAM[21]	PSNR↑	27.27	28.45	29.06	34.14	34.87	28.43	28.76	30.91	30.24
	SSIM↑	0.910	0.909	0.932	0.961	0.969	0.938	0.941	0.955	0.939
	LPIPS↓	0.324	0.294	0.266	0.209	0.196	0.258	0.229	0.236	0.252
	Depth↓	1.05	0.85	2.37	1.24	1.48	1.86	1.66	1.54	1.51
	ATE↓	0.70	0.95	1.35	0.59	0.55	2.03	1.56	0.72	1.00
Point-SLAM[43]	PSNR↑	32.40	34.08	35.5	38.26	39.16	33.99	33.48	33.49	35.17
	SSIM↑	0.974	0.977	0.982	0.983	0.986	0.96	0.960	0.979	**0.975**
	LPIPS↓	0.113	0.116	0.111	0.1	0.118	0.156	0.132	0.142	0.124
	Depth↓	0.53	0.22	0.46	0.30	0.57	0.49	0.51	0.46	**0.44**
	ATE↓	0.61	0.41	0.37	**0.38**	0.48	0.54	0.69	0.72	0.52
MonoGS[7]	PSNR↑	34.83	36.43	37.49	39.95	42.09	36.24	36.7	36.07	**37.50**
	SSIM↑	0.954	0.959	0.965	0.971	0.977	0.964	0.963	0.957	0.960
	LPIPS↓	0.068	0.076	0.075	0.072	0.055	0.078	0.065	0.099	**0.070**
	Depth↓	-	-	-	-	-	-	-	-	-
	ATE↓	0.47	0.43	0.31	0.70	0.57	0.31	**0.31**	3.2	0.79
GS-SLAM[44]	PSNR↑	31.56	32.86	32.59	38.70	41.17	32.36	32.03	32.92	34.27
	SSIM↑	0.968	0.973	0.971	0.986	0.993	0.978	0.970	0.968	**0.975**
	LPIPS↓	0.094	0.075	0.093	0.050	0.033	0.094	0.110	0.112	0.082
	Depth↓	1.31	0.82	1.26	0.81	0.96	1.41	1.53	1.08	1.16
	ATE↓	0.48	0.53	0.33	0.52	0.41	0.59	0.46	0.7	0.50
SplaTAM[6]	PSNR↑	32.86	33.89	35.25	38.26	39.17	31.97	29.70	31.81	34.11
	SSIM↑	0.98	0.97	0.98	0.98	0.98	0.97	0.95	0.95	0.97
	LPIPS↓	0.07	0.10	0.08	0.09	0.09	0.10	0.12	0.15	0.10
	Depth↓	0.43	0.38	0.54	0.44	0.66	1.05	1.60	0.68	0.72
	ATE↓	0.31	**0.40**	**0.29**	0.47	**0.27**	**0.29**	0.32	**0.55**	**0.36**
Ours	PSNR↑	32.51	32.52	33.58	35.67	36.93	30.46	28.70	30.44	32.60
	SSIM↑	0.972	0.942	0.971	0.954	0.947	0.917	0.924	0.907	0.942
	LPIPS↓	0.094	0.155	0.112	0.143	0.169	0.171	0.175	0.221	0.155
	Depth↓	0.616	0.786	0.841	0.667	0.881	1.370	1.526	1.852	1.067
	ATE↓	**0.289**	0.581	0.660	0.461	0.407	0.342	0.713	0.564	0.502

Table 5.3: **Comparison results for RGB-D SLAM methods on Replica.**

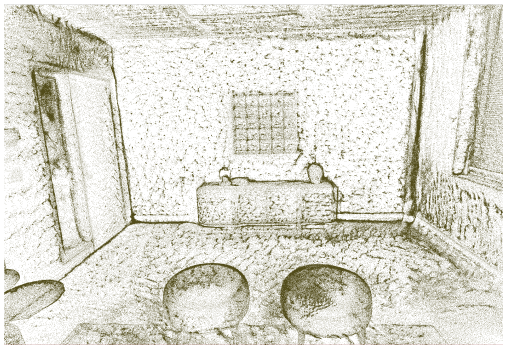
Here, we list the key results from other related methods to better make a comparison, as shown in Table 5.3. Notice that, since the differences in computing platforms and methods, these results are not compared in a unified framework, for example, Point-SLAM uses ground-truth depth. And the number is from their paper or their appendix, for reference only.

5.3.3 Point cloud visualization

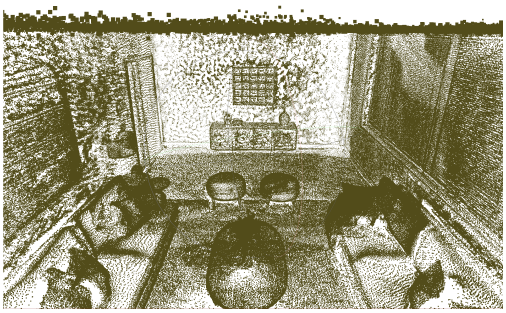
Here, we use Replica room0 as an example to visualize SLAM with 2D Gaussian and 3D Gaussian.



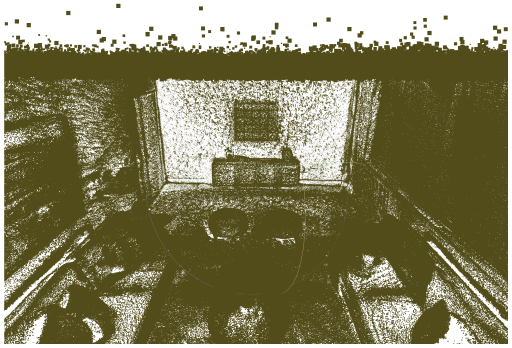
(a) Point could for 2D Gaussian



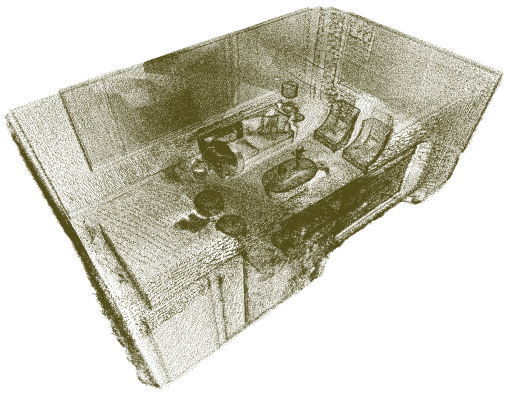
(b) Point could for 3D Gaussian



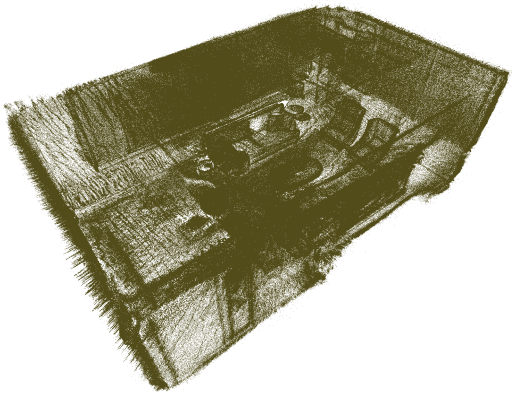
(c) Ceiling surface for 2D Gaussian



(d) Ceiling surface for 3D Gaussian



(e) Room view for 2D Gaussian



(f) Room view for 3D Gaussian

Figure 5.6: Geometric point cloud for different representation.

We visualize the Gaussian point clouds obtained from the two representation, 2D Gaussian and 3D Gaussian shown in Figure 5.6, and we can see that the 2D Gaussian is more sparsely distributed over the surface. Also in the fitting of object edges and corners, the 3D Gaussian tends to require more point clouds to fit very narrow edges. Therefore, the 3D Gaussian distribution appears more volumetric, with points filling space in all three dimensions.

We also placed the camera view parallel to the ceiling of the room as shown in Figure 5.6(c)(d), and then we observed that both the 3D Gaussian and the 2D Gaussian regressed to the exact plane. However, their difference is in the variance, and it is clear that the 3D Gaussian has a larger noise variance, significantly higher than the 2D Gaussian. These experimental results of the point cloud validate our initial conjecture that a 2D Gaussian can lead to more accurate geometric accuracy in SLAM.

Chapter 6

Conclusion

6.1 Quantitative analysis

In the last chapter, we compare the performance of our method with other methods mainly from the perspective of evaluation indicators. However, in this section, we will analyze and discuss our results from perspectives out of the metrics.

6.1.1 Scene rendering capability discussion

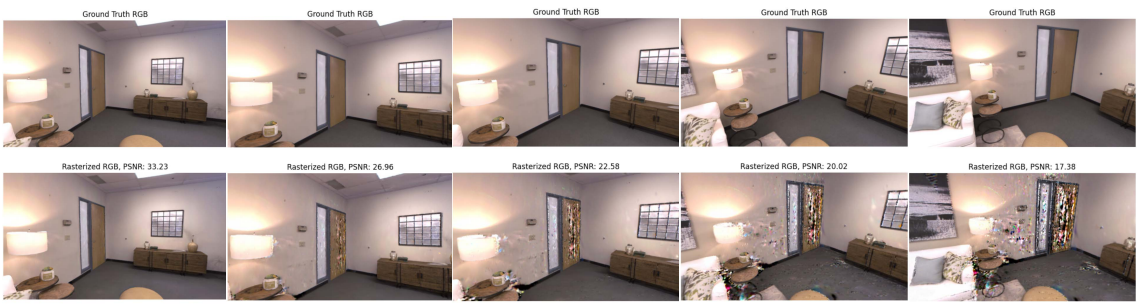
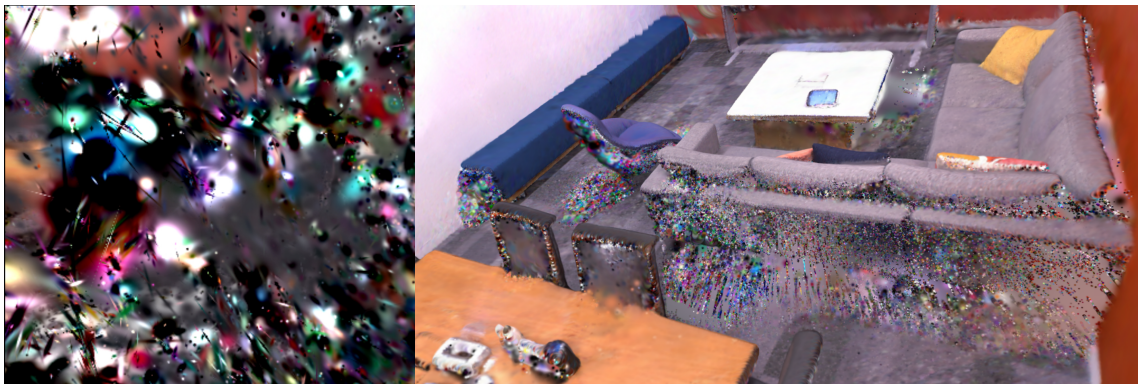


Figure 6.1: Rendering scene of 2D Gaussian for new perspectives.

For novel camera perspectives, we just fit the ground truth image in the first frame shown as the left image in Figure 6.1. Then, we stop mapping, move the camera to the next frames, and compare these new frame rendered images with ground truth. We observe the 2D Gaussian renders very poorly for new camera views and generalizes much weaker than the 3D Gaussian, much like the window shutters. Although the 2D Gaussian can lead to a clear point cloud with less point cloud and memory usage. It still misses a dimension. Therefore, it is hard to estimate the camera pose because it has weak generalization in a new perspective.

Therefore, in our ablation experiment, if we apply 2D Gaussian as map representation, the trajectory tracking accuracy will decrease. Because the 2D Gaussian is only planar, when it fits the first viewpoint, it doesn't take into account the other viewpoints, so there will be many gaps from the other viewpoints, and this structure is much like a shutter. That made the tracking process much more difficult. As a result, the ATE error has been enlarged. To solve this problem, we proposed a sight-line fitting strategy in the mapping module. We try to let the Gaussian disk in the non-contour region be oriented towards the camera, making the observed area as large as possible, thus minimizing gaps and seams. The experimental results indicated that this could reduce the depth error in building the map to some extent.

6.1.2 Hole filling capability discussion



(a) Enlarged view of the hole (b) Hole filing in the back

Figure 6.2: The hole filling for unseen region.

For hole filling capability, we observe that both 2D and 3D Gaussian do not ensure geometric consistency of the scene for regions unseen shown in Figure 6.2. On the back of the sofa and chair areas not captured by the camera, the Gaussian point clouds still hang and float in the air.

6.2 Limitation

Balancing the build quality of photo-realistic scenes with real-time operational efficiency is an important issue for modern SLAM systems. When evaluating, we choose to optimize 100 iterations per frame to ensure that the evaluation value on each metric is in a nice interval. However, in practical applications, real-time performance is more important, in which case we can control our iterations per frame to between 10-20, which ensures a decent processing speed. Of course, decreasing the iteration in tracking and mapping will also reduce accuracy.

From the ablation experiments in the previous chapter, we can see that applying correlation point matching for deep point clouds can significantly improve the trajectory estimation error. However, this method consumes a lot of time resulting in not being able to run in real time. Therefore we hope that the existing tracking and matching methods can take care of both computational efficiency and accuracy.

As I mentioned in the last section, current metric for evaluating Gaussian-based SLAM method cannot indicate the point cloud quality of the surface. Existing 3D Gaussian points have good rendering quality of the plane and are able to regress and recover depth well, nevertheless, possess great variance in the distribution of points, shown as Figure 1.2. So if we only consider the depth mean square error, it can only indicate the mean but ignore the variance, which cannot reflect surface geometric accuracy.

Therefore, we need to introduce more metrics from geometry to take evaluation. For example, in the area of scene reconstruction, the researcher compute the average distance between sampled points from the reconstructed mesh and the nearest ground-truth point to measure the accuracy. In 3D Gaussian, all point clouds are explicit, but we cannot directly use depth map as the ground-truth point to compute the distance, because many datasets have big noise and error points, for example, nan value.

Another measurable method is completion, the quality of the reconstruction is measured by computing the average distance from the sampled points of the map reconstruction to the real mesh. However, this only applies to virtual scenes, since it is difficult to sample an absolutely realistic mesh as ground-truth in a real scene.

6.3 Future works

It is still a phase of continuous refinement and development of high-quality rendering technology, which, in combination with Gaussian representation with SLAM technology, demonstrates its powerful ability to render and fit photo-realistic scenes at the photo level. In addition, using it to build maps can also fill in the holes caused by unseen regions. With the continuous iteration of the algorithm, it will be rapidly applied to related products.

For static scenes, that is to say, environments where no change in object position will be detected and no deformation will be produced, 3D Gaussian-based reconstruction has moved to the stage of focusing on the geometrical accuracy of the surface, which is the concern of this project. We hope to be able to perform fast and accurate mesh extraction from these attributes of the point cloud and covariance of the Gaussian map, which is seamlessly integrated with downstream tasks.

Meanwhile, recently, a large number of researchers have begun to experiment with modeling dynamic scenes using 3D Gaussian representations. Also, some researchers have started to try to incorporate instance detection and segmentation tasks from semantic SLAM into Gaussian maps. There are also researchers who attempt to introduce multi-modality in Gaussian maps for open vocabulary recognition or introduce large models for navigation.

Once the geometric accuracy of Gaussian maps is improved, the most promising research direction is to apply it to downstream tasks, for example, motion planning and trajectory generation. In complex scenarios, we can sample a set of trajectory points from Gaussian map that can be used to plan the robot's end-effector to perform more challenging tasks.

Bibliography

[1] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ transactions on computer vision and applications*, 9:1–11, 2017.

[2] Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual slam. *Expert Systems with Applications*, 205:117734, 2022.

[3] Andréa Macario Barros, Maugan Michel, Yoann Moline, Gwenolé Corre, and Frédérick Carrel. A comprehensive survey of visual slam algorithms. *Robotics*, 11(1):24, 2022.

[4] 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.

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023.

[6] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. Splatam: Splat, track & map 3d gaussians for dense rgb-d slam. *arXiv preprint arXiv:2312.02126*, 2023.

[7] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian splatting slam. *arXiv preprint arXiv:2312.06741*, 2023.

[8] Pinxuan Dai, Jiamin Xu, Wenxiang Xie, Xinguo Liu, Huamin Wang, and Weiwei Xu. High-quality surface reconstruction using gaussian surfels. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.

[9] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. *arXiv preprint arXiv:2403.17888*, 2024.

[10] Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *arXiv preprint arXiv:2406.06521*, 2024.

[11] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[12] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pages 83–86. IEEE, 2009.

[13] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[14] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017.

[15] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[16] 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 *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. Ieee, 2011.

[17] 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*, volume 11, page 3. Rome, Italy, 2015.

[18] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 134–144, 2019.

[19] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. imap: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6229–6238, 2021.

[20] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12786–12796, 2022.

[21] Hengyi Wang, Jingwen Wang, and Lourdes Agapito. Co-slam: Joint coordinate and sparse parametric encodings for neural real-time slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13293–13302, 2023.

[22] Raul Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[23] Cristóbal Campos, Richard Elvira, Juan J García Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[24] Stanford University Computer Graphics Laboratory. The stanford 3d scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`, 1994.

[25] Jiawei Zhang. Math derivation of 3d gaussian splatting. `https://zjwfufu.github.io/2023/11/11/3DGS_math/`, 2023.

[26] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[27] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022.

[28] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

[29] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *arXiv preprint arXiv:2311.12775*, 2023.

[30] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient and compact surface reconstruction in unbounded scenes. *arXiv preprint arXiv:2404.10772*, 2024.

[31] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.

[32] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[33] Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.

[34] Dror Aiger, Niloy J Mitra, and Daniel Cohen-Or. 4-points congruent sets for robust pairwise surface registration. In *ACM SIGGRAPH 2008 papers*, pages 1–10. 2008.

[35] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 3, pages 2743–2748. IEEE, 2003.

[36] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[37] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[38] Jatavallabhula Krishna Murthy, Soroush Saryazdi, Ganesh Iyer, and Liam Paull. gradslam: Dense slam meets automatic differentiation. In *arXiv*, 2020.

[39] Krishna Murthy Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Tao Chen, Shuang Li, Ganesh Iyer, Soroush Saryazdi, Nikhil Keetha, Ayush Tewari, Joshua B. Tenenbaum, Celso Miguel de Melo, Madhava Krishna, Liam Paull, Florian Shkurti, and Antonio Torralba. Conceptfusion: Open-set multimodal 3d mapping. *Robotics: Science and Systems (RSS)*, 2023.

[40] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580. IEEE, 2012.

[41] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[42] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12–22, 2023.

[43] Erik Sandström, Yue Li, Luc Van Gool, and Martin R Oswald. Point-slam: Dense neural point cloud-based slam. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18433–18444, 2023.

[44] Chi Yan, Delin Qu, Dong Wang, Dan Xu, Zhigang Wang, Bin Zhao, and Xuelong Li. Gs-slam: Dense visual slam with 3d gaussian splatting. *arXiv preprint arXiv:2311.11700*, 2023.