Globally Consistent RGB-D SLAM with 2D Gaussian Splatting

Xingguang Zhong[1], Yue Pan[1], Liren Jin[1], Marija Popović[2], Jens Behley[1], and Cyrill Stachniss[1,3]

Fig. 1: Reconstruction results of 2DGS-SLAM on synthetic dataset Replica [74] and real-world dataset ScanNet [10]. We present the reconstructed 2D Gaussian splatting maps, along with RGB and normal renderings from zoomed-in local views. These results demonstrate that our method achieves both high-fidelity image rendering and precise geometric reconstruction.

***Abstract*—Recently, 3D Gaussian splatting-based RGB-D SLAM displays remarkable performance of high-fidelity 3D reconstruction. However, 3D Gaussian splatting (3DGS) suffers from a lack of depth rendering consistency, which leads to suboptimal geometric reconstruction in 3DGS-based SLAM. In addition, 3DGS-based SLAM methods typically lack efficient loop closure, limiting their ability to build globally consistent maps online. In this paper, we present 2DGS-SLAM, an RGB-D SLAM system using 2D Gaussian splatting as map representation. By leveraging the depth-consistent rendering property of the 2D variant, we propose an accurate camera pose optimization method and achieve geometrically accurate 3D reconstruction. In addition, we implement efficient loop detection and camera relocalization by leveraging MASt3R, a feed-forward 3D reconstruction model, and achieve efficient map updates by maintaining a local active map. Experiments show that our 2DGS-SLAM approach achieves superior tracking accuracy, higher surface reconstruction quality, and more consistent global map reconstruction compared to existing rendering-based SLAM methods, while maintaining high-fidelity image rendering and improved computational efficiency.**

***Index Terms*—SLAM, mapping, localization, RGB-D perception**

I. Introduction

SIMULTANEOUS localization and mapping (SLAM) is a fundamental problem in computer vision and robotics. The ability to reconstruct unknown environments is a basis for various robotic tasks, including navigation [30], [47], [89] and exploration [4], [36]. Recently, radiance field-based map representations like neural radiance field (NeRF) [52] and Gaussian splatting (GS) [40], have opened up new possibilities for dense RGB-D SLAM by enabling high-fidelity reconstruction with photorealistic rendering. Among them, Gaussian splatting has gained popularity due to its fast rendering speed and flexible scalability, establishing itself as the more favorable map representation for radiance field-based RGB-D SLAM.

Most existing GS-based methods [38], [50], [97], [103] directly adopt classical 3D Gaussian splatting (3DGS) for mapping and frame-to-map camera tracking. However, the depth images rendered from 3DGS at different viewpoints often exhibit inconsistency, negatively impacting pose optimization with depth information and geometric reconstruction accuracy. Furthermore, since pose drift in long-term camera tracking is inevitable, SLAM systems need to incorporate loop closures as well as map correction and update mechanisms for global consistency [73]. Some radiance field-based RGB-D SLAM methods [46], [103] address this issue by using multiple submaps and applying global transformations to the submaps after loop closure. However, they often rely on computationally expensive point cloud registration for relocalization and typically require complex post-processing to merge all submaps, making them impractical for online robotic applications.

In this paper, we investigate the problem of realizing a RGB-D SLAM system that builds geometrically accurate and globally consistent radiance field reconstructions online. Instead of using 3DGS, we adopt 2D Gaussian splatting (2DGS) [29] as our map representation. 2DGS replaces 3D ellipsoids with 2D disks and explicitly computes ray-disk intersections, ensuring consistent depth rendering while maintaining high-fidelity radiance field reconstruction required for novel view synthesis. Leveraging these properties, we develop an accurate rendering-based method for frame-to-map camera pose esti-

[1]X. Zhong, Y. Pan, L. Jin, J. Behley, C. Stachniss are with the Center for Robotics, University of Bonn, Germany. [2]M. Popović is with the MAVLab, TU Delft, the Netherlands. [3]C. Stachniss is also with the Lamarr Institute for Machine Learning and Artificial Intelligence, Germany.

This work has partially been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 – 390732324 – PhenoRob, and by the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG). The authors thank for the access to the Marvin cluster of the University of Bonn.

mation. In addition, 2DGS represents the environment with discrete Gaussian splats distributed in 3D space, offering a point cloud-like structure allowing for elastic properties when closing loops. By associating each Gaussian splat with nearby keyframes, we can update the poses of keyframes and their corresponding splats after pose graph optimization. Building on this strategy, we further address two key challenges to achieve globally consistent map reconstruction in an online manner. First, inspired by classical surfel-based dense SLAM methods [5], [91], we maintain a continuously updated local active map and design a mechanism to transition Gaussian primitives between active and inactive states. In this way, we prevent tracking and relocalization failures due to the accumulation of new and old map structures, while removing the need for complicated submap management. Second, after detecting a loop closure, we need to accurately estimate the relative pose between existing frame and the current frame to add proper constraints to the pose graph. Unlike prior works that rely on computationally expensive 3D point cloud registration, we leverage MASt3R [44], a recently introduced feed-forward 3D reconstruction model with remarkable generalization capability, to estimate an initial relative pose. This initial estimate is then refined through further frame-to-map tracking within the active map, achieving accurate relocalization.

Based on the above design, we present 2DGS-SLAM, a 2DGS-based RGB-D SLAM system that enables online reconstruction of globally consistent radiance fields. As shown in Fig. 1, our 2DGS-SLAM achieves outstanding reconstruction results in synthetic and real-world scenes.

In summary, we make three key claims: (i) Our proposed 2DGS-SLAM achieves superior tracking accuracy compared to state-of-the-art rendering-based approaches; (ii) Our approach surpasses or is on-par with 3DGS-based methods in surface reconstruction quality and demonstrates more consistent mapping results in real-world scenes compared to other loop-closure-enabled methods. At the same time, 2DGS-SLAM maintains high-fidelity image rendering performance that is either superior to or on-par with baseline approaches. (iii) Compared to other radiance field-based methods that support loop closure, our approach is more efficient at runtime and has a more compact map representation. The open-source implementation of our 2DGS-SLAM is available at: https://github.com/PRBonn/2DGS-SLAM.

II. Related Work

A. Map-centric RGB-D SLAM

Compared to sparse feature-based visual SLAM systems [7], [20], [54], that target pose and feature location estimation, dense visual SLAM systems generate 3D maps beneficial for robotic tasks like interaction and navigation. RGB-D SLAM predominates indoor dense SLAM systems, as the depth camera enables direct acquisition of metrically-scaled dense geometry. Dense visual SLAM systems can be further classified into frame-centric and map-centric approaches based on their tracking strategies. Frame-centric methods estimate poses through either sparse feature matching [19], [43] or by minimizing photometric and geometric errors between consecutive frames [13], [41], [42], [80]. In these methods, the map is merely a by-product constructed by accumulating frame-wise point clouds. In contrast, similar to LiDAR-based SLAM systems [6], [25], [62], [84], map-centric methods incrementally build a 3D model of the environment and perform frame-to-map tracking for robust pose estimation.

In the past decade, numerous works employ truncated signed distance function (TSDF) [9], [11], [57], [59], [90], Octomap [19], [27], or surfels [39], [70], [75], [91] as map representations and use weighted moving average for efficient incremental mapping. Despite their effective mapping and localization capabilities, these methods suffer from limited scalability and map fidelity, constrained by their discrete map representations.

Recent advancements in radiance fields and implicit neural representations [3], [63], [101] have enabled high-fidelity scene modeling, offering new opportunities for map-centric SLAM. With the radiance field as the map, camera tracking can be performed by minimizating of photometric and geometric discrepancies between the current frame and the rendered image from the radiance field. iMap [77] pioneered the use of neural radiance fields (NeRF) [52] as a map representation, demonstrating the advantages of neural implicit representations in handling the sparse observations or occlusions through inpainting. However, despite being memory-efficient, the use of a single multi layer perceptron (MLP) to represent the whole scene limits its ability to capture fine-grained details in complex, large-scale environments. To improve scalability and rendering performance, subsequent works propose hybrid map representations that combine locally-defined optimizable features with a globally-shared shallow MLP. These features can be structured in various forms such as hierarchical voxel grids [104], octrees [94], spatial hashing [86], triplane grids [15], [37], or unordered points [46], [69], [98]. Nevertheless, the rendering process remains computationally intensive due to ray-wise sampling and volumetric integration.

3D Gaussian splatting [40] introduces a novel radiance field based on rasterization of optimizable Gaussian primitives, offering superior training and rendering efficiency while maintaining or exceeding the rendering quality of NeRF. These properties have facilitated various robotic applications, such as active sensing [36], scene-level mapping [35], and simulation [102], thereby encouraging the adoption of 3DGS as the map representation for SLAM.

3DGS-based visual SLAM systems can be split into coupled and decoupled ones, based on whether the online-built 3DGS map is utilized for rendering-based tracking. Decoupled systems [26], [31], [64], [68], [92] employ external trackers [7], [54], [71], [80] for camera pose estimation. However, these systems require maintaining a separate map for the external tracker, which is distinct from the 3DGS map, resulting in architectural redundancy in the system design. In contrast, coupled systems [22], [38], [50], [78], [93], [97] utilize 3DGS as the sole map representation for both tracking and mapping through rendering-based gradient descent optimization. These systems typically employ a keyframe-based strategy, where mapping is performed using keyframes, while tracking is applied to all frames.

Although achieving comparable tracking performance and superior map photorealism to previous map-centric SLAM systems, the aforementioned coupled 3DGS SLAM systems face two main challenges. First, geometric ambiguity in 3D Gaussian splatting limits the accuracy of geometry-based tracking and surface reconstruction. Second, these systems function primarily as visual odometries, lacking the capability to handle loop closures necessary to create a globally consistent map.

To address the first challenge, one solution is to flatten the 3D Gaussian ellipsoids into optimizable 2D surfels, as demonstrated in 2DGS [12], [29]. 2DGS provides enhanced geometric representation with multi-view consistent depth and normal rendering, motivating its use over 3DGS as the map representation to improve geometry-based tracking accuracy and surface reconstruction quality. While several concurrent works [32], [61], [92] adopt 2DGS as their map representation, none have implemented on-manifold camera pose optimization using the 2DGS rasterizer, as MonoGS does for 3DGS. Our work addresses this gap by explicitly deriving Jacobians for 2DGS-based camera tracking and implementing them in an efficient CUDA-based rasterizer. In the next section, we discuss related works addressing the second challenge of globally consistent mapping.

B. Visual Loop Closure and Globally Consistent Mapping

For visual SLAM, closuring loop is crucial for correcting accumulated odometry drift and ensuring a globally consistent map. Loop closure correction typically involves a place recognition step to identify loop closure candidates, followed by a relocalization step to estimate the relative pose between the current frame and the loop candidate. This relative pose is subsequently used in graph optimization to correct drift errors of trajectory and deform the map.

Compared to distance-based loop candidate search [42], [79], appearance-based place recognition is more versatile, as it can operate without prior knowledge of the camera position and remains effective even when odometry drift is significant. Early approaches primarily rely on aggregating handcrafted local features using bag-of-words [21], [24], random ferns [23], hamming distance embedding binary search tree [16], or VLAD [2] to build databases for efficient searching and matching [7], [43], [54], [91], or match image sequences [53], [85]. Recently, there has been a shift towards learning-based approaches using NetVLAD [1] and DINOv2 [33], [34], [58]

The relocalization step aims to estimate the relative pose between the current frame and the detected historical frame. This transformation serves as a loop constraint edge for pose graph optimization in graph-based SLAM systems. In cases where odometry drift is small, relocalization becomes a local pose tracking problem, i.e., tracking the current frame against the historical map. However, for larger loops, where the initial pose often lies outside the convergence basin of pose tracking, a coarse global localization step becomes necessary. This is typically achieved using the PnP or Umeyama [82] algorithm together with RANSAC, which relies on keypoint-based feature matching [66], [67].

Recent data-driven feed-forward 3D reconstruction models, particularly DUSt3R [87] and MASt3R [44], have demonstrated promising performance in various 3D vision tasks. Given a pair of RGB images, MASt3R generates a metrically-scaled 3D point map for both images in the first camera's coordinate frame, along with confidence maps. From the point map, additional properties including relative camera poses, depth images, and pixel correspondences can be derived. Features extracted by the MASt3R encoder can be aggregated using the ASMK framework [18] for efficient image retrieval. Several SLAM systems leverage MASt3R for different purposes: camera pose and Gaussian splats initialization for 3DGS SLAM [95], loop closure detection and camera pose tracking [55], and two-view loop constraint construction [45]. We employ MASt3R exclusively for loop closure correction. Unlike previous approaches that use separate features for loop closure detection and relocalization [46], [96], we utilize MASt3R for both tasks.

Although loop closure correction is a common practice in traditional SLAM systems, it has been adopted by only a few radiance field-based SLAM systems, as it is challenging to maintain a globally consistent radiance field throughout the SLAM process. Among coupled systems supporting loop closure, most existing approaches utilize a collection of submaps, treating each submap as a rigid body for pose adjustment. Within each submap, the radiance field can be represented using MLP-based [79], neural octree-based [48], or neural point-based [28], [46] implicit fields, as well as the 3DGS radiance field [96], [103]. While the submap-based strategy is efficient for pose graph optimization and map management [11], [60], [65], it presents challenges such as drift within the submap and additional effort required for merging submaps and refining the merged map, particularly for the radiance field [46], [96], [103]. Redundant memory usage occurs in overlapping submap areas, and discrepancies among the submaps are often unfavorable features of the submap-based strategy.

For map representations that are inherently elastic, such as surfels, neural points, and Gaussian splats, one can take a point-based deformation strategy [61], [62], [91] which associates each map primitive with a frame and adjusts frames instead of submaps during pose graph optimization.

As the first coupled 3DGS SLAM system with loop closure, LoopSplat [103] employs NetVLAD [1] for loop closure detection and estimates loop constraints through rendering-based keyframe-to-submap tracking. While it achieves superior performance in pose accuracy and map global consistency, the use of 3DGS submaps necessitates a computationally intensive map refinement step after submap merging. Moreover, without a coarse global localization step, LoopSplat may struggle with relocalization when closing a large loop, where the loop candidate is distant from the current frame. In contrast, our approach leverages MASt3R for both loop closure detection and coarse relocalization, while adopting a submap-free strategy that associates Gaussian surfels with keyframes. Compared to other radiance field-based SLAM methods with loop closure that utilize submaps, this design enables direct map correction after pose graph optimization, avoiding redundant memory usage and submap merging overhead while achieving superior geometric accuracy in a globally consistent 2DGS map.

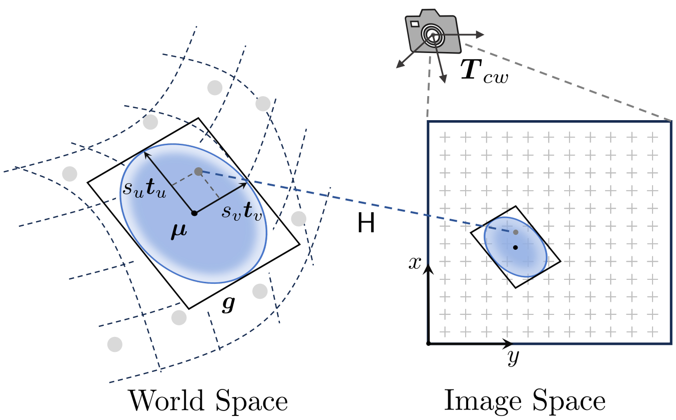


Fig. 2: Overview of 2D Gaussian splatting. The pose and shape of a splat $\boldsymbol{g}$ in the world space are defined by its center $\boldsymbol{\mu}$ and two scaled tangent vectors $s_u\boldsymbol{t}_u, s_v\boldsymbol{t}_v$. These two vectors $\boldsymbol{t}_u$ and $\boldsymbol{t}_v$ lie on this plane of $\boldsymbol{g}$. Given a camera with pose $\boldsymbol{T}_{cw}$, the splat $\boldsymbol{g}$ can be projected onto the image space. Points within the local space of the splat is mapped to their corresponding pixel on the image's x-y plane via a homography H.

III. Preliminaries

In the following sections, we first introduce the map representation used in our system, 2D Gaussian splatting [29], and derive how to backpropagate gradients to the camera pose with 2DGS-based differentiable rendering.

A. 2D Gaussian Splatting

Unlike 3DGS, 2DGS compresses one dimension of the 3D ellipsoid to zero, using 2D Gaussian disks as primitives to represent the 3D environment. By explicitly calculating the intersection of the rays from the camera with the disk's plane, 2DGS can realize multi-view consistency in depth rendering, thereby achieving a more accurate geometric representation.

As illustrated in Fig. 2, a 2D Gaussian splat $\boldsymbol{g}$ is defined within a local tangent plane in a 3D global coordinate system. This plane is determined by the splat's central point $\boldsymbol{\mu} \in \mathbb{R}^3$ and two principal tangential vectors $\boldsymbol{t}_u$ and $\boldsymbol{t}_v$, with two scale factors s_u and s_v controlling the variances along the tangential vectors, respectively. By representing the rotation matrix of the 2D Gaussian splat as $\boldsymbol{R} = [\boldsymbol{t}_u, \boldsymbol{t}_v, \boldsymbol{t}_n] \in \mathbb{R}^{3\times 3}$, where $\boldsymbol{t}_n = \boldsymbol{t}_u \times \boldsymbol{t}_v$ is the normal vector, and arranging scale factors as a 3×3 diagonal matrix $\boldsymbol{S} = \mathrm{diag}(s_u, s_v, 0)$, the 2D local frame can be parameterized as follows:

$$P(u,v) = \boldsymbol{\mu} + s_u\boldsymbol{t}_u u + s_v\boldsymbol{t}_v v = \boldsymbol{A}\,(u,v,0,1)^\top , \quad (1)$$

$$\text{where } \boldsymbol{A} = \begin{bmatrix} s_u\boldsymbol{t}_u & s_v\boldsymbol{t}_v & 0 & \boldsymbol{\mu} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{RS} & \boldsymbol{\mu} \\ 0 & 1 \end{bmatrix}. \quad (2)$$

Here, $\boldsymbol{A}$ is the affine transformation matrix from 2D local uv space to the global coordinate system.

The mapping from the uv space to the rendering image's screen space can be formulated as a 2D-to-2D homography transformation [51]. Let $\boldsymbol{W} \in \mathbb{R}^{4\times 4}$ be the transformation matrix from camera space to image space and $\boldsymbol{T}_{cw} \in SE(3)$ be the pose of the view camera, combining Eq. (1) yields:

$$\begin{aligned} \boldsymbol{x} = (xz, yz, z, 1)^\top &= \boldsymbol{W}\boldsymbol{T}_{cw}P(u,v) & (3)\\ &= \boldsymbol{W}\boldsymbol{T}_{cw}\boldsymbol{A}\,(u,v,0,1)^\top , & (4) \end{aligned}$$

where $\boldsymbol{T}_{cw}$ transforms the splat in the world space to the camera space and then $\boldsymbol{W}$ transforms it to the image space, and the $\boldsymbol{x}$ represents the ray corresponding to pixel (x,y) intersecting the 2D Gaussian splat at depth of z. For convenience, we define:

$$\begin{aligned} \boldsymbol{A}_c &= \boldsymbol{T}_{cw}\boldsymbol{A} & (5)\\ &= \boldsymbol{T}_{cw}\begin{bmatrix} \boldsymbol{RS} & \boldsymbol{\mu} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} s_u\boldsymbol{t}_{uc} & s_v\boldsymbol{t}_{vc} & 0 & \boldsymbol{\mu}_c \\ 0 & 0 & 0 & 1 \end{bmatrix}, & (6) \end{aligned}$$

where $\boldsymbol{t}_{uc}$, $\boldsymbol{t}_{vc}$ and $\boldsymbol{\mu}_c$ are Gaussian splat's tangential vectors and central point in camera space. Furthermore, we define the whole homography H as:

$$\mathsf{H} = \boldsymbol{W}\boldsymbol{T}_{cw}\boldsymbol{A} = \boldsymbol{W}\boldsymbol{A}_c\,. \quad (7)$$

To render the value of pixel $\boldsymbol{p} = (x,y)^\top$ from the splats, 2DGS solves the inverse problem of Eq. (4), computing the intersection of ray $\boldsymbol{x}$ with the 2D Gaussian splat in uv space, while avoiding the need to compute the inverse of H. For further details, we refer the reader to the original paper [29].

Apart from these geometric parameters mentioned above, each splat also contains color feature $\boldsymbol{c}$ and view-independent opacity α to represent its visual appearance. After computing the ray-splat intersections of all splats within field of view, 2DGS sorts them by depth and uses volumetric alpha blending to integrate weighted appearance values $\boldsymbol{V}_{\boldsymbol{p}}$ of pixel $\boldsymbol{p}$, as follows:

$$\boldsymbol{V}_{\boldsymbol{p}} = \sum_{i=0}^{N} \boldsymbol{v}_i \alpha_i \mathcal{G}(\boldsymbol{u}_i^{\boldsymbol{p}}) \prod_{j=0}^{i-1}\left(1 - \alpha_j \mathcal{G}(\boldsymbol{u}_j^{\boldsymbol{p}})\right), \quad (8)$$

where $\mathcal{G}(\boldsymbol{u}) = \mathcal{G}(u,v) = \exp\left(-\frac{u^2+v^2}{2}\right)$ represent the Gaussian weight of the intersection $\boldsymbol{u}$ in the uv space, $\boldsymbol{u}_i^{\boldsymbol{p}}$ means the i-th intersection along the ray of pixel $\boldsymbol{p}$, and N denotes the number of Gaussian splats that intersect with the ray. It should be noted that the appearance value $\boldsymbol{v}$ can be a view-dependent color generated from color feature $\boldsymbol{c}$, depth d and normal vector $\boldsymbol{t}_{nc} = \boldsymbol{t}_{uc} \times \boldsymbol{t}_{vc}$, meaning that the color image, depth image and the normal image are rendered in the same way. In addition, we can also render an opacity image $\boldsymbol{O}$ if we set $\boldsymbol{v} = 1$.

To summarize, each 2D Gaussian splat $\boldsymbol{g}$ contains parameters $(\boldsymbol{\mu}, \boldsymbol{t}_u, \boldsymbol{t}_v, s_u, s_v, \boldsymbol{c}, \alpha)$ to describe its geometric and visual information. These parameters can be progressively optimized through differentiable rasterization using a rendering loss to achieve high-fidelity reconstruction. 2DGS implements both the forward rendering and backward gradient propagation in CUDA, enabling efficient and scalable operation.

B. Camera Pose Optimization

Our proposed SLAM system does not rely on external visual odometry. Instead, we directly use rendering-based frame-to-map tracking to estimate the pose of each frame. The core problem of rendering-based tracking is computing the gradient of the rendering loss with respect to the camera pose. However, similar to 3DGS, the original 2DGS assumes that the camera poses of input frames are fixed and the loss generated from the forward rendering cannot propagate to them. Some 3DGS-based SLAM systems [38], [97], [103] apply the pose matrix directly to all Gaussian splats, and derive the gradient with respect to each element of the matrix by automatic differentiation. They then leverage differentiable transformation between quaternion and rotation matrix to obtain the quaternion's gradient. However, these methods cannot guarantee that the gradient remains in $SE(3)$ during the optimization process, resulting in a method that is neither efficient nor accurate.

To address this limitation, MonoGS [50] derives analytical Jacobians of camera pose in $SE(3)$ for 3DGS and achieves efficient tracking. However, due to the difference of rendering mechanism, this derivation cannot be transfered to 2DGS directly. In our work, we bridge the gap and derive the camera Jacobians explicitly based on Lie algebra for 2DGS. To save memory overhead, the 2DGS map in our system does not use a spherical harmonic function to generate view dependent colors, so spherical harmonic function is not considered in the derivation below.

Let L denote the rendering loss, which measures the difference between the rendered image and the input image. Since both the ray-splat intersection and alpha blending-based rendering are differentiable, given the rendering loss L, the per-element gradients of L with respect to the homography H, denoted as $\frac{\partial L}{\partial \mathsf{H}}$, can be obtained from 2DGS's original implementation. Based on Eq. (6) and Eq. (7), we can derive the gradients of $\boldsymbol{A}_c$ from $\frac{\partial L}{\partial \mathsf{H}}$ by applying the chain rule:

$$\frac{\partial L}{\partial \boldsymbol{A}_c} = \begin{bmatrix} \frac{\partial L}{s_u \partial \boldsymbol{t}_{uc}} & \frac{\partial L}{s_v \partial \boldsymbol{t}_{vc}} & 0 & \frac{\partial L}{\partial \boldsymbol{\mu}_c} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \boldsymbol{W}^\top \frac{\partial L}{\partial \mathsf{H}}. \tag{9}$$

From this gradient matrix, we can directly extract the gradients with respect to $\boldsymbol{t}_{uc}$, $\boldsymbol{t}_{vc}$, and $\boldsymbol{\mu}_c$, which are given by $\frac{\partial L}{\partial \boldsymbol{t}_{uc}}$, $\frac{\partial L}{\partial \boldsymbol{t}_{vc}}$, and $\frac{\partial L}{\partial \boldsymbol{\mu}_c}$. Furthermore, according to Eq. (8), 2DGS can render normal images from splats' normal vector $\boldsymbol{t}_{nc}$ in the camera space. Then, the gradient of loss L with respect to $\boldsymbol{t}_{nc}$, i.e., $\frac{\partial L}{\partial \boldsymbol{t}_{nc}}$, can be computed through the backpropagation of alpha blending. Combining these results, we obtain the full gradient of $\boldsymbol{R}_c$:

$$\frac{\partial L}{\partial \boldsymbol{R}_c} = \left[\frac{\partial L}{\partial \boldsymbol{t}_{uc}}, \frac{\partial L}{\partial \boldsymbol{t}_{vc}}, \frac{\partial L}{\partial \boldsymbol{t}_{nc}} \right]. \tag{10}$$

The camera pose $\boldsymbol{T}_{cw}$ affects the rendered image by transforming each 2D Gaussian splat from world space to camera space. This transformation impacts both the center $\boldsymbol{\mu}$ and the orientation $\boldsymbol{R}$ of the splat, producing their transformed counterparts $\boldsymbol{\mu}_c$ and $\boldsymbol{R}_c$ in the camera coordinate system. Accordingly, the gradient of $\boldsymbol{T}_{cw}$ is composed of two distinct components:

$$\frac{\partial L}{\partial \boldsymbol{T}_{cw}} = \frac{\partial L}{\partial \boldsymbol{\mu}_c} \frac{\mathcal{D} \boldsymbol{\mu}_c}{\mathcal{D} \boldsymbol{T}_{cw}} \oplus \frac{\partial L}{\partial \boldsymbol{R}_c} \frac{\mathcal{D} \boldsymbol{R}_c}{\mathcal{D} \boldsymbol{T}_{cw}}, \tag{11}$$

where $\oplus$ ensures that both terms are projected into the same tangent space of $SE(3)$ before summation, guaranteeing dimensional consistency. Adopting the same notation as in MonoGS [50], we define the partial derivative on the manifold as:

$$\frac{\mathcal{D} f(\boldsymbol{T})}{\mathcal{D} \boldsymbol{T}} = \lim_{\tau \to 0} \frac{\mathrm{Log}(f(\mathrm{Exp}(\tau) \circ \boldsymbol{T}) \circ f(\boldsymbol{T})^{-1})}{\tau}, \tag{12}$$

where $\boldsymbol{T} \in SE(3)$ and $\tau \in \boldsymbol{se}(3)$, $\circ$ is a group composition operation. Then the two derivatives in Eq. (11) can be derived as following:

$$\frac{\mathcal{D} \boldsymbol{\mu}_c}{\mathcal{D} \boldsymbol{T}_{cw}} = \begin{bmatrix} I & -\boldsymbol{\mu}_c^\times \end{bmatrix}, \frac{\mathcal{D} \boldsymbol{R}_c}{\mathcal{D} \boldsymbol{T}_{cw}} = \begin{bmatrix} 0 & -\boldsymbol{R}_{c,:,1}^\times \\ 0 & -\boldsymbol{R}_{c,:,2}^\times \\ 0 & -\boldsymbol{R}_{c,:,3}^\times \end{bmatrix}, \tag{13}$$

where $\times$ denotes the skew symmetric matrix of a 3D vector, and $:, i$ refers to the i-th column of the matrix. To ensure computational efficiency, we implement the above process in CUDA as well.

IV. Our Approach

Our proposed RGB-D SLAM system aims to reconstruct a globally consistent radiance field online while maintaining the precise geometric structure of the 3D environment. In the following sections, we first describe the structure of our system and provide a detailed explanation of each module in the individual following subsections.

A. System Overview

Leveraging the depth-consistent rendering capability of 2DGS, we develop a RGB-D SLAM system to enable accurate camera pose estimation alongside geometrically precise radiance field reconstruction. To further achieve online reconstruction of a globally consistent map, we extend the parameterization of 2D Gaussian splats. More specifically, our map representation can be expressed as:

$$\mathcal{M} = \left\{ \boldsymbol{g}_i, \delta_i, t_i^c, d_i^c, t_i^l \mid i = 1, ..., N \right\}, \tag{14}$$

where $\boldsymbol{g}_i = (\boldsymbol{\mu}, \boldsymbol{t}_u, \boldsymbol{t}_v, s_u, s_v, \boldsymbol{c}, \alpha)$ is the original Gaussian splat's learnable parameters as discussed in Sec. III-A. The superscript c stands for "closest". The item t_i^c denotes the sequential ID of the frame that observed $\boldsymbol{g}_i$ at the closest distance, where the distance is measured from the Gaussian splat's center to the camera viewpoint. This is used to associate $\boldsymbol{g}_i$ with its corresponding keyframe t_i^c for global map correction, and the closest distance is stored as d_i^c. Meanwhile, t_i^l denotes the ID of the last frame that observed $\boldsymbol{g}_i$. The Boolean variable $\delta_i = \{0, 1\}$ represents $\boldsymbol{g}_i$'s active state. Based on this state, we can split all the Gaussian splats in the map $\mathcal{M}$ into two subsets $\mathcal{M}_A = \{\delta_i = 1 \mid \boldsymbol{g}_i \in \mathcal{M}\}$ and $\mathcal{M}_I = \{\delta_i = 0 \mid \boldsymbol{g}_i \in \mathcal{M}\}$, representing active and inactive Gaussian splats respectively.

As shown in Fig. 3, our system comprises two main process: the front-end and the back-end. The front-end is responsible for estimating the current camera pose and detecting potential loop closures. The back-end focuses on expanding and optimizing the map using frames with estimated poses, updating the active map, as well as globally deforming the map after a loop closure. We summarize the main components of our system as follows:

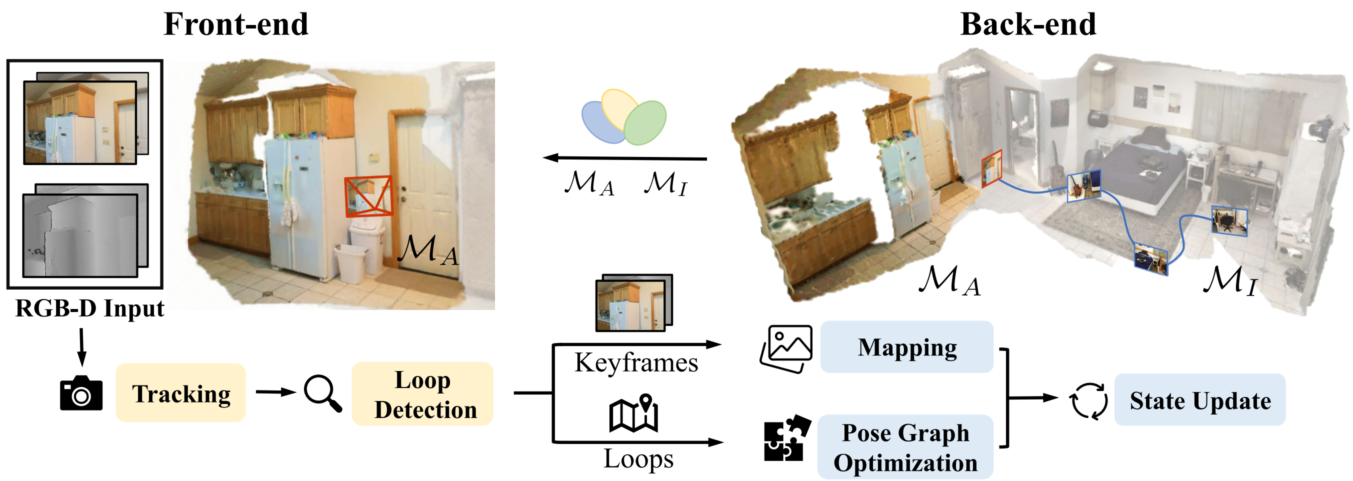


Fig. 3: System overview of 2DGS-SLAM. Our system consists of two parallel processes: a front-end and a back-end. Taking RGB-D frames as input, the front-end performs frame-to-map camera tracking using the currently active map $\mathcal{M}_A$, and searches for potential loop closures. The selected keyframe is sent to the back-end, which uses it to expand and optimize the map (mapping). If a loop closure is detected in the front-end, we send the computed loop constraint to the back-end, where pose graph optimization and map correction will be performed. After mapping or pose graph optimization, the back-end updates the active state of the map and synchronizes it with the front-end. When no message is received, the back-end keeps refining the map based on previously stored keyframes.

1) Tracking and keyframe selection (Sec. IV-B): In the front-end, upon acquiring a new RGB-D frame, we estimate the camera pose using the currently active map $\mathcal{M}_A$ through a frame-to-map tracking approach. Then, we select keyframes based on covisibility and send them to the back-end for mapping.
2) Mapping (Sec. IV-C): We expand the map by projecting 2D Gaussian splats into the world space based on keyframes. The active map $\mathcal{M}_A$ is then optimized for a few iterations using both current and historical keyframes. Afterwards, we send both $\mathcal{M}_A$ and $\mathcal{M}_I$ to the front-end for synchronization. Additionally, $\mathcal{M}_A$ and $\mathcal{M}_I$ are continuously refined using a selection of historical keyframes in the back-end, respectively.
3) Map state update (Sec. IV-D): To prevent outdated map regions from negatively impacting tracking, we mark Gaussian splat $\boldsymbol{g}_i$ that have not been observed for a certain period as inactive, i.e., $\delta_i = 0$. After a loop closure, we reactivate the observed inactive Gaussian splats to avoid redundancy.
4) Loop detection and relocalization (Sec. IV-E): In the front-end, we identify loop closures by comparing each incoming frame with previous keyframes. If a candidate keyframe is found, we estimate the relative pose between this keyframe and current frames based on MASt3R [44]. The pose is then sent to the back-end and used to add a loop constraint to the pose graph.
5) Map correction (Sec. IV-F): After receiving loop constraint from the front-end, we perform pose graph optimization, and transform every Gaussian according to the updated pose of its associating keyframe t_i^c. Finally, we reactivate observed inactive Gaussian splats and send the deformed map to the front-end.

Our system contributes several original design choices to provide a powerful 3D SLAM system using 2DGS going beyond a direct combination of existing GS-based, surfel-based, and MASt3R-based components. First, we adopt 2D Gaussian splats as the map representation, which provide multi-view consistent depth rendering. Building on this property, we explicitly derive on-manifold SE(3) pose gradients in the 2DGS rasterizer, enabling accurate frame-to-map tracking and geometrically faithful reconstruction. Second, unlike submap-based radiance-field SLAM approaches such as Loop-Splat [103], which require submap management and merging after loop closure, we introduce a submap-free formulation that maintains a single globally consistent map throughout the mapping process. This design reduces memory overhead and simplifies loop-induced map updates. Third, although we are inspired by classical surfel-based systems such as ElasticFusion [91] in dividing the map into active and inactive parts to support robust relocalization, 2DGS differs from surfels in both rendering and optimization behavior. Therefore, we propose a new transition mechanism between the active and inactive states that is specifically designed for Gaussian splats. Finally, for loop closure detection and coarse relocalization, we leverage MASt3R [44] to obtain reliable initial relative poses with strong generalization capability, and then refine them through rendering-based camera tracking. This yields a unified loop closure pipeline that combines learned coarse relocalization with frame-to-map refinement. The following sections will provide more comprehensive explanations of each component.

B. Tracking and Keyframe Selection

Based on the derivation in III-B, we can optimize the current camera's pose $\boldsymbol{T}_{cw}$ through gradient descent once the rendering loss is known. Given the input RGB image

$\boldsymbol{I}$ and depth image $\boldsymbol{D}$, we define the color rendering loss $\boldsymbol{L}_I \in \mathbb{R}^{H\times W}$ and depth rendering loss $\boldsymbol{L}_D \in \mathbb{R}^{H\times W}$ as:

$$\boldsymbol{L}_I = \|\boldsymbol{I}_r - (\alpha_l \boldsymbol{I} + \beta_l)\|_1 , \tag{15}$$

$$\boldsymbol{L}_D = \|\boldsymbol{D}_r - \boldsymbol{D}\|_1 , \tag{16}$$

where $\boldsymbol{I}_r$ and $\boldsymbol{D}_r$ are RGB image and depth image rendered from active submap $\mathcal{M}_A$ at pose $\boldsymbol{T}_{cw}$, α_l and β_l are per-frame learnable parameters to model lighting variations. and $\|\cdot\|_1$ means element-wise L1 distance.

According to Eq. (8), 2DGS can render normal vector images $\boldsymbol{N}_r \in \mathbb{R}^{H\times W\times 3}$ from 2D Gaussian splats. Utilizing this property, we can filter out the influence of back-facing splats on tracking by applying a normal mask $\boldsymbol{M}_n$, which can be calculated by:

$$\boldsymbol{M}_n(x,y) = [\![\boldsymbol{N}_r(x,y)^\top \boldsymbol{r}(x,y) > 0]\!], \tag{17}$$

where $\boldsymbol{r}(x,y) \in \mathbb{R}^3$ represents the normalized ray vector emitted from the camera's optic center and passing through pixel (x,y) on the image plane, and $[\![\cdot]\!]$ is the indicator function returning 1 if the statement is true, otherwise 0. Note that all vectors in Eq. (17) are defined in the camera space. Besides, we apply another mask $\boldsymbol{M}_o$ obtained from the rendered opacity image $\boldsymbol{o} \in \mathbb{R}^{H\times W}$ to ignore the loss generated from under-reconstructed area. The opacity mask is defined as:

$$\boldsymbol{M}_o(x,y) = [\![\boldsymbol{O}(x,y) > 0.95]\!]. \tag{18}$$

Then, the total tracking loss $L_t \in \mathbb{R}$ can be written as:

$$L_t = \frac{1}{|\Omega|}\sum_{\boldsymbol{p}\in\Omega} \boldsymbol{M}_n(\boldsymbol{p})\cdot \boldsymbol{M}_o(\boldsymbol{p})\cdot(\boldsymbol{L}_I(\boldsymbol{p}) + \lambda_d \boldsymbol{L}_D(\boldsymbol{p})), \tag{19}$$

where $\Omega = \{(u,v) \mid u \in 1,...,W, v \in 1,...,H\}$ represent all the pixels and λ_d is the weight used to balance these two losses, and $\cdot$ represents the per-element product. We directly initialize the optimization using last frame's pose and employ the Adam algorithm to iteratively optimize the pose $\boldsymbol{T}_{cw}$ and the lighting parameters α_l and β_l until convergence or reaching the maximum number of iterations n_{iter}. To ensure that the gradients are stable, during the optimization process of camera tracking, the parameters of these Gaussian splats remain fixed. It should be noted that, during tracking, we only render images from the active Gaussian splats, i.e., $\boldsymbol{g} \in \mathcal{M}_A$.

As with most SLAM frameworks, rather than using all frames for mapping, we selectively choose keyframes to improve efficiency. Similar to MonoGS [50], we primarily determine keyframes based on the covisibility.

First, from Eq. (8), the alpha-blending coefficient of Gaussian splat $\boldsymbol{g}_k$ in the rendering of pixel $\boldsymbol{p}$ is:

$$w_k(\boldsymbol{p}) = \alpha_k \mathcal{G}(\boldsymbol{u}_k^{\boldsymbol{p}}) \prod_{j=0}^{k-1}(1 - \alpha_j \mathcal{G}(\boldsymbol{u}_j^{\boldsymbol{p}})). \tag{20}$$

We define the contribution of $\boldsymbol{g}_k$ to a rendered frame from a given view V as the sum of its rendering coefficients across all pixels, expressed by:

$$\mathcal{C}_k^V = \sum_{\boldsymbol{p}\in\Omega} w_k(\boldsymbol{p}) , \tag{21}$$

where $\boldsymbol{p} \in \Omega$ denotes all pixels. Intuitively, $\mathcal{C}_k^V$ represents how many pixels the Gaussian splat $\boldsymbol{g}_k$ contributes to the rendering. Therefore, we directly define that $\boldsymbol{g}_k$ is visible in the given view V if its $\mathcal{C}_k^V$ is larger than 0.5. Furthermore, Given two camera views A, B and current active map $\mathcal{M}_A$, we define the covisibility score between these two views as follows:

$$S_{cov}(A,B) = \frac{|G_A \cap G_B|}{|G_A \cup G_B|}, \tag{22}$$

where $G_A = \{\boldsymbol{g}_k \in \mathcal{M}_A \mid \mathcal{C}_k^A > 0.5\}$ and similarly for G_B with $\mathcal{C}_k^B$. They are the sets of all visible Gaussian splats at view A and B, respectively. If the covisibility score between the current view and the last keyframe falls below a threshold c_k, or if their distance between the translation vectors of $\boldsymbol{T}_{cw}$ surpasses a threshold d_k, the current view is selected as a keyframe.

C. Mapping

After completing pose estimation in the front-end, the new keyframe K_n observed by the robot is sent to the back-end process for map expansion and optimization. To reduce memory consumption, we first convert the RGB-D data into a colored point cloud $\mathcal{P}$, and then apply random downsampling to obtain $\mathcal{P}_s$ before projecting it into the world space. Each 3D point is then initialized as a Gaussian splat, where its initial scales s_u and s_v are determined by the distance d to its nearest neighbor in $\mathcal{P}_s$, and the opacity α is set to an initial value of 0.99. The initial normal vector $\boldsymbol{t}_n$ is obtained from the normal image $\boldsymbol{N}_{\boldsymbol{D}}$, which is computed from the pixel gradient of the depth image $\boldsymbol{D}$. Specifically, we derive $\boldsymbol{N}_{\boldsymbol{D}}$ from the cross product of neighboring pixel differences in $\boldsymbol{D}$,

$$\boldsymbol{N}_{\boldsymbol{D}}(x,y) = \frac{\nabla_x \boldsymbol{D}(x,y) \times \nabla_y \boldsymbol{D}(x,y)}{\|\nabla_x \boldsymbol{D}(x,y) \times \nabla_y \boldsymbol{D}(x,y)\|}. \tag{23}$$

Here, we assign $\boldsymbol{t}_n$ to each splat based on its corresponding position. We then randomly initialize two principal tangential vectors $\boldsymbol{t}_u$ and $\boldsymbol{t}_v$, which are perpendicular to $\boldsymbol{t}_n$. In addition, the closest frame ID t_i^c and last observed frame ID t_i^l of the new splat are both initialized as the ID of the current keyframe.

To prevent the generation of excessively redundant Gaussian splats, we maintain a voxel hash table with resolution r_h for the active submap $\mathcal{M}_A$, which represents the spatial occupancy state. New Gaussian splats are only created in spatially unoccupied voxels where no existing Gaussians are present. This voxel hash table is updated whenever the map expands by adding new splats and when modifications occur in the active state, such as transitioning Gaussians between active and inactive states. Due to its relatively low resolution and restriction to active regions, the memory overhead remains minimal.

In the back-end process, we continuously optimize all the Gaussian splats $\{\boldsymbol{g} \in \mathcal{M}\}$ to ensure they not only produce high-fidelity image renderings but also align well with the actual surface, accurately capturing the geometric structure of the environment. To achieve this, we train the map with multiple loss functions. Firstly, the color image rendering loss $\boldsymbol{L}_c \in \mathbb{R}^{H\times W}$ is expressed as:

$$\boldsymbol{L}_c = \lambda_c \|\boldsymbol{I}_r - \boldsymbol{I}\|_1 + (1-\lambda_c) L_{\text{SSIM}}(\boldsymbol{I}_r, \boldsymbol{I}), \tag{24}$$

where $\lambda_c \in [0, 1]$ and the L_{SSIM} represents the structural similarity index measure (SSIM) [88]. We also apply an L1 loss $\boldsymbol{L}_D$ like Eq. (16) to directly supervise the depth rendering optimization using the input depth image. Following 2DGS, to ensure that the Gaussian splats conform to the surface locally, we add a normal consistency loss $\boldsymbol{L}_n \in \mathbb{R}^{H \times W}$ between the rendered depth image $\boldsymbol{D}_r$ and the rendered normal image $\boldsymbol{N}_r$, formulated as:

$$\boldsymbol{L}_n = \mathbf{1}_{H \times W} - \boldsymbol{N}_{\boldsymbol{D}_r} \cdot \boldsymbol{N}_r, \tag{25}$$

where $\boldsymbol{N}_{\boldsymbol{D}_r} \in \mathbb{R}^{H \times W \times 3}$ denotes the normal image estimated from the rendered depth image $\boldsymbol{D}_r$ by applying Eq. (23), and $\cdot$ indicates a per-pixel 3D vector dot product.

Finally, we optimize the map $\mathcal{M}$ using the combination of above loss functions, which can be written as:

$$L_m = \frac{1}{|\Omega|} \sum_{\boldsymbol{p} \in \Omega} (\boldsymbol{L}_c(\boldsymbol{p}) + w_d \boldsymbol{L}_D(\boldsymbol{p}) + w_n \boldsymbol{L}_n(\boldsymbol{p})), \tag{26}$$

where w_d, w_n are weights to balance the contributions of the corresponding loss terms. Utilizing the loss function L_m, we continually optimize the active map $\mathcal{M}_A$ and inactive map $\mathcal{M}_I$ separately in the back-end. For $\mathcal{M}_A$, we maintain an active frames set $\mathcal{S}_a$, which is defined as:

$$\mathcal{S}_a = \{t_i^c \mid \boldsymbol{g_i} \in \mathcal{M}_A\}, \tag{27}$$

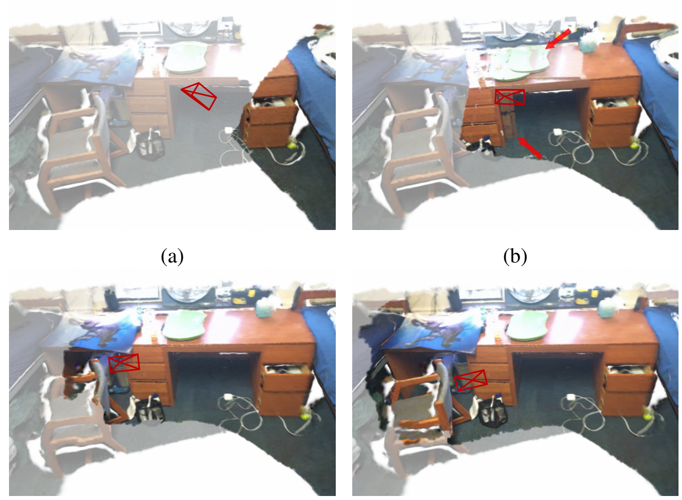
where t_i^c, as described in Eq. (14), is the ID of the closest observing frame of $\boldsymbol{g}_i$. In each iteration, we randomly sample N_a frames from $\mathcal{S}_a$ and N_i frames from the other frames to perform optimization for Gaussian splats in $\mathcal{M}_A$ and $\mathcal{M}_I$, respectively.

D. Map State Update

Due to the accumulation of tracking errors, directly integrating newly observed data into the global map can cause a misalignment between new and existing structures, negatively impacting re-localization after loop detection. To mitigate this issue, as mentioned in Sec. IV-A, we maintain two separate maps: an active map $\mathcal{M}_A$, which stores recently observed Gaussian splats, and an inactive map $\mathcal{M}_I$, which preserves historical splats.

Given the latest posed keyframe K_n sent from the front-end, where n is its sequential ID, we first assign active status ($\delta = 1$) to all new Gaussian splats generated from K_n and add them to active map $\mathcal{M}_A$. Using the visibility criterion defined in Eq. (21), we identify which Gaussian splats in $\mathcal{M}_A$ are visible from the view of K_n and then update their last observed frame ID to $t_i^l = n$. Meanwhile, we compute the distance from these Gaussian splat to the viewpoint of K_n. If the distance is smaller than the historical minimum distance d_i^c, we update their closest frame ID as $t_i^c = n$. For all $\boldsymbol{g}_i \in \mathcal{M}_A$, we mark $\boldsymbol{g}_i$ as inactive ($\delta = 0$) if $(n - t_i^l)$ exceeds a predefined time threshold, indicating that $\boldsymbol{g}_i$ has not been observed for a long time.

As illustrated in Fig. 4, to avoid accumulating redundant Gaussian splats in the same region, we reactivate inactive splats in $\mathcal{M}_I$ when the robot revisits previously observed areas. More specifically, if a loop closure is detected between the current frame $\boldsymbol{F}_c$ and a historical keyframe $\boldsymbol{K}_h$, we first perform pose graph optimization followed by map correction. For the subsequent T keyframes, where T is a predefined hyperparameters, if an inactive Gaussian splat $\boldsymbol{g}_i \in \mathcal{M}_I$ is observed, and its closed observed frame ID $t_i^c > h$, indicating that its position has been corrected by pose graph optimization, we reassign its state as active and update its last observed frame ID t_i^l to the current frame ID c. In addition, during the mapping process, we continuously sample historical keyframes, such as $\boldsymbol{K}_r$, and evaluate the contributions of the Gaussian splats associated with $\boldsymbol{K}_r$, i.e., $\boldsymbol{g} \in \{t_i^c = r \mid \boldsymbol{g}_i \in \mathcal{M}\}$, based on Eq. (21). If the contribution of a Gaussian splat falls below 0.5, we consider it occluded by surrounding splats and remove it from the map to maintain map compactness.



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

Fig. 4: Illustration of the state update process. Images (a)-(d) are shown in temporal order. The bright regions represent the active map, while the dim regions indicate the inactive map. The red cones represent the current camera views. (a) Active and inactive maps during running. (b) The camera observes part of the inactive map, but no loop closure has been detected. Due to pose drift, the active and inactive maps misalign (see red arrows). (c) The system detects a loop closure, aligns inactive and active maps, and reactivates the observed inactive Gaussian splats. (d) As the camera moves, more inactive Gaussian splats are progressively reactivated.

E. Loop Detection and Relocalization

In the back-end, we maintain a pose graph $\boldsymbol{G}$, where each keyframe serves as a vertex, and the relative pose between adjacent keyframes forms an edge in the graph. When a loop closure is detected, we compute the relative pose between the current frame and the candidate frame searched from all the keyframes to introduce a loop closure constraint for pose graph optimization.

We primarily utilize MASt3R [44] for loop detection and re-localization. Given a pair of input RGB images $\langle \boldsymbol{I}_i, \boldsymbol{I}_j \rangle$, MASt3R extracts their image features $\mathcal{F}_i, \mathcal{F}_j$ through a vision transformer-based model [17] and directly outputs pixel-wise point clouds, $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$, along with their respective confidence maps, $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$. Notably, both $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$

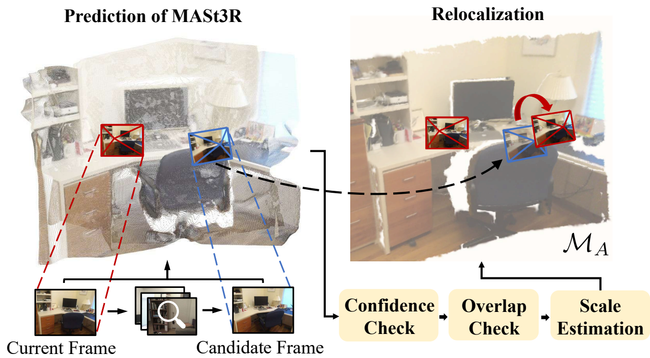


Fig. 5: We input the current frame and the loop candidate into MASt3R to estimate their relative pose and dense point clouds. After confidence and overlap checks, we optimize the scale of computed relative pose by aligning the point cloud to the local map. The scaled pose is then transfered to the world space and refined by tracking it on the active map $\mathcal{M}_A$.

are represented in the camera coordinate frame of view i. By leveraging these dense point clouds, we can estimate the camera parameters of the two frames and subsequently solve for their relative pose using the PnP algorithm [83], thereby obtaining the depth maps $\boldsymbol{D}_i^p$ and $\boldsymbol{D}_j^p$ in their respective camera space.

Inspired by recent works [18], [55], we use features $\mathcal{F}$ from the vision transformer encoder as local descriptors, and employ the aggregated selective match kernel (ASMK) [81] for image retrieval. ASMK quantizes and binarizes these features using a precomputed k-means codebook, producing high-dimensional sparse binary representations. The similarity between images can be efficiently computed via a kernel function over shared codebook elements. We integrate this process into our online system. For each keyframe K_n, we utilize MASt3R's feature encoder to extract image feature $\mathcal{F}_n$ and store it, along with the corresponding image sequence ID n, in a feature database managed using ASMK.

Fig. 5 illustrates the main pipeline of our loop closure detection. After tracking the current image $\boldsymbol{I}_c$, we extract its features and compute similarities with all keyframes. We then identify the keyframe with the highest similarity to $\boldsymbol{I}_c$. If its similarity score exceeds a predefined threshold s_r, we designate it as a loop closure candidate. Let $\boldsymbol{I}_l$ and $\boldsymbol{D}_l$ denote the RGB and depth images of the candidate keyframe $\boldsymbol{K}_l$, respectively. To further validate the loop closure, we feed the image pair $\langle \boldsymbol{I}_c, \boldsymbol{I}_l \rangle$ to MASt3R, obtaining the predicted point clouds $\boldsymbol{P}_c$ and $\boldsymbol{P}_l$, the corresponding depth maps $\boldsymbol{D}_c^p$ and $\boldsymbol{D}_l^p$, their confidence maps $\boldsymbol{C}_c$ and $\boldsymbol{C}_l$, and the estimated relative pose $\boldsymbol{T}_{lc}$. If the mean of confidence map $\boldsymbol{C}_c$ is below a predefined threshold c_s, we consider the prediction unreliable and discard this loop closure candidate. Otherwise, we estimate the overlap between the two frames as follows. Since the predicted point clouds are both expressed in the coordinate frame of $\boldsymbol{I}_c$, we directly reproject $\boldsymbol{P}_l$ onto the image plane of $\boldsymbol{I}_c$ and compare the reprojected depth $\boldsymbol{D}_c^l$ with the predicted depth $\boldsymbol{D}_c^p$. Inspired by [8], the overlap ratio O_{lc} is computed as:

$$O_{lc} = \frac{\sum_{\boldsymbol{u} \in \mathcal{V}} \mathbf{1}\left(\left| \boldsymbol{D}_c^l(\boldsymbol{u}) - \boldsymbol{D}_c^p(\boldsymbol{u}) \right| < \tau_d \right)}{|\mathcal{V}|}, \tag{28}$$

where $\mathcal{V}$ denotes the pixels falling within the image boundaries after projection, $\mathbf{1}$ is an indicator function that returns 1 if the condition inside is true and 0 otherwise. τ_d is a depth consistency threshold, which is 0.05 in our setting. If O_{lc} is smaller than a predefined threshold δ_o, we determine that the candidate frame lacks sufficient overlap for reliable relocalization and reject the loop closure attempt.

If the candidate passes the filtering, we further compute the accurate relative pose to provide a loop closure constraint for pose graph optimization. Although MASt3R is trained on a large amount of metric-scale data, its predicted depth maps remain up to scale. Therefore, we first estimate the scale factor s^* using real observed depth image $\boldsymbol{D}_c$, given by:

$$s^* = \arg\min_s \left\| \boldsymbol{C}_c \cdot (\boldsymbol{D}_c - s\boldsymbol{D}_c^p) \right\|_2 , \tag{29}$$

where $\cdot$ represents the per-element product for matrices. This is a weighted least squares problem that can be solved in closed form. Then, we multiply this scale factor with the translation component of $\boldsymbol{T}_{lc}$ to obtain the scaled relative pose $\boldsymbol{T}_{lc}^r$. Consequently, we derive the candidate keyframe's pose in the world space as $\boldsymbol{T}_l = \boldsymbol{T}_{lc}^r \boldsymbol{T}_c$, where $\boldsymbol{T}_c$ is the pose of the current camera. To obtain a more accurate estimation, we use $\boldsymbol{T}_l$ as the initial estimate and perform a scan-to-model tracking in current active map $\mathcal{M}_A$, which refines the pose by minimizing the rendering loss between the observed depth and the map-rendered depth. After convergence or reaching the maximum number of iterations n_{iter}, we re-render a depth image $\boldsymbol{D}_r$ from the active map at the optimized pose $\boldsymbol{T}_l^t$ and compute its L1 error e_t against the input depth image $\boldsymbol{D}$ using the same formulation as Eq. (19):

$$e_t = \frac{1}{|\Omega|} \sum_{\boldsymbol{p} \in \Omega} \boldsymbol{M}_n(\boldsymbol{p}) \cdot \boldsymbol{M}_o(\boldsymbol{p}) \cdot \left\| \boldsymbol{D}_r(\boldsymbol{p}) - \boldsymbol{D}(\boldsymbol{p}) \right\|_1 , \tag{30}$$

where $\boldsymbol{M}_n$ and $\boldsymbol{M}_o$ are normal and opacity masks, respectively. This error e_t serves as a criterion to filter out false positives introduced by MASt3R. Only frames with an error e_t below a threshold ε_t are retained for further refinement. Then, the successfully optimized pose $\boldsymbol{T}_l^t$ is used to calculate the accurate relative pose as $\boldsymbol{T}_{lc}^t = \boldsymbol{T}_l^t \boldsymbol{T}_c^{-1}$. Before performing pose graph optimization, we compute the error of the entire trajectory after incorporating the loop closure. If this error is larger than a threshold, we reject the loop closure attempt. With $\boldsymbol{T}_{lc}^t$ as the loop constraint, we perform pose graph optimization and update the poses of all keyframes.

To increase the number of valid loop closures and further improve the mapping accuracy, we extend our loop detection beyond image feature querying by also revisiting the inactive map. Specifically, after performing tracking based on the active map $\mathcal{M}_A$ for each incoming frame, we additionally render images using the inactive Gaussian splats $\{\boldsymbol{g} \in \mathcal{M}_I\}$. If the area of valid region in the rendered opacity image $\boldsymbol{O}_i$ exceeds a threshold a_v, indicating that the robot has observed part of the historical map. In this case, we count the occurrence numbers of all the closest observing frame ID t_i^c among all observed Gaussian splats and accordingly select the keyframe with the highest count as the loop closure candidate. We then input this candidate and the current frame into MASt3R, applying the same selection and pose refine pipeline as described earlier.

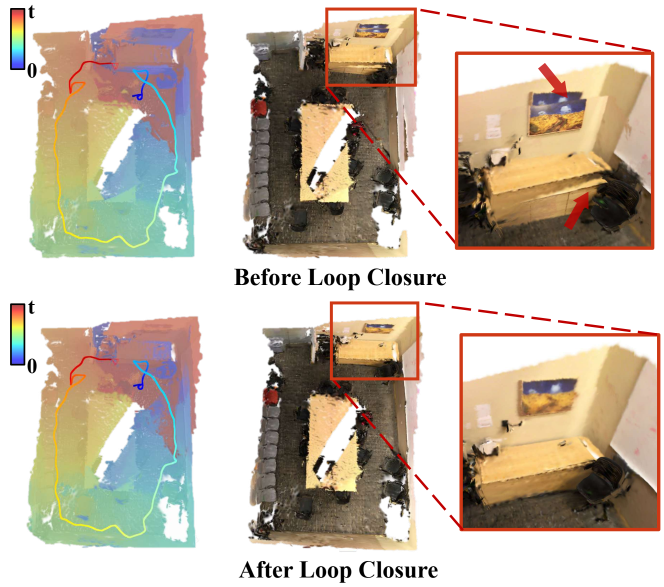


Fig. 6: Result of loop closure and map correction. The left figures illustrate the associated keyframe ID of each Gaussian splat and the camera trajectory. Both IDs and the trajectory are colored by time. The right figures compare the reconstruction results before and after loop closure. It can be observed that the map suffers from severe drift and misalignment before the loop correction (highlighted by red arrows). After that, the map structure becomes cleaner and more consistent.

F. Map Correction

After pose graph optimization, each keyframe K_i with pose $\boldsymbol{T}_i$ is updated with an optimized pose increment:

$$\Delta \boldsymbol{T}_i = \boldsymbol{T}_i^o \boldsymbol{T}_i^{-1}, \tag{31}$$

where $\boldsymbol{T}_i$ is the original pose, and $\boldsymbol{T}_i^o$ is the optimized pose.

For each Gaussian splat $\boldsymbol{g}_k \in \mathcal{M}$, let f_c^k be the frame ID of its closest observed keyframe. We apply the corresponding pose increment $\Delta \boldsymbol{T}_{f_c^k}$ to update the Gaussian's position $\boldsymbol{\mu}_k$ and orientation $\boldsymbol{R}_k$:

$$\boldsymbol{\mu}_k' = \Delta \boldsymbol{T}_{f_c^k} \boldsymbol{\mu}_k, \boldsymbol{R}_k' = \Delta \boldsymbol{R}_{f_c^k} \boldsymbol{R}_k, \tag{32}$$

where $\boldsymbol{\mu}_k$ and $\boldsymbol{R}_k$ represent the original position and rotation of the Gaussian $\boldsymbol{g}_k$, and $\boldsymbol{x}_k'$ and $\boldsymbol{R}_k'$ are the updated values. As illustrated in Fig. 6, our method ensures that the entire Gaussian map is deformed consistently with the optimized keyframe poses, preserving spatial coherence.

V. Experimental Evaluation

The main focus of this work is a rendering-based RGB-D SLAM system for building geometrically accurate and globally consistent radiance fields using 2D Gaussian splatting.

We present our experiments to show the capabilities of our method and analyze its performance. The results of our experiments also support our key claims, which are (i) Our proposed 2DGS-SLAM system demonstrates higher tracking accuracy than state-of-the-art rendering-based methods and traditional dense SLAM methods based on TSDF or surfel representations. (ii) Our method outperforms SDF-based NeRF-SLAM and 3DGS-based approaches in terms of surface reconstruction quality. The incorporation of the efficient loop closure mechanism ensures more globally consistent reconstruction results. At the same time, our method also achieves comparable or superior image rendering quality, making the reconstruction result well-suited for downstream tasks. (iii) Our 2DGS-SLAM is much more efficient in terms of runtime than other rendering-based SLAM systems with loop closures and generates a more compact map representation.

TABLE I: Hyperparameters of our approach

symbol	value	description
Tracking and Keyframe Selection, Sec. IV-B		
c_k	0.9	covisibility threshold for keyframe selection
d_k	15 (cm)	distance threshold for keyframe selection
n_{iter}	120	maximum number of iterations
Mapping, Sec. IV-C		
λ_c	0.125	weight of the SSIM loss
w_d	0.5	weight of depth loss
w_n	0.02	weight of normal loss
N_a	3	active map training frames per iteration
N_i	2	inactive map training frames per iteration
Loop Detection and Relocalization, Sec. IV-E		
s_r	0.025	similarity score threshold for image retrieval
c_s	3.0	mean confidence score threshold
a_v	0.5	valid region threshold for revisiting loop
δ_o	0.5	overlap ratio threshold

A. Experimental Setup

1) Datasets: We conduct our experiments on three public datasets that are widely adopted for performance evaluation in rendering-based SLAM methods as well as self-recorded data from a mobile robot. These datasets are the synthetic dataset Replica [74], and two real datasets, TUM-RGBD [76] and ScanNet [10]. The Replica dataset provides ground-truth camera poses along with an accurate mesh of the target environment. The TUM-RGBD dataset includes accurate camera poses captured using a motion capture system, while the reference poses in the ScanNet dataset are provided by BundleFusion [11]. It is worth noting that the depth images in the Replica dataset are rendered directly from the mesh and therefore free from noise. In contrast, both TUM-RGBD and ScanNet datasets are captured using consumer-grade structured-light-based RGB-D sensors, which introduce noticeable motion blur and depth measurement noise, presenting additional challenges for rendering-based SLAM algorithms. In addition to the three public datasets mentioned above, to evaluate the performance of our method on a real robotic platform, we recorded data using a wheeled robot in an indoor environment and conducted quantitative experiments on pose estimation.

2) Implementation details: We summarize the hyperparameters of our SLAM system, previously mentioned throughout the paper, in Tab. I. These settings are kept consistent across all experiments. In addition, optimization-related parameters for 2D Gaussian splats, such as learning rates for different components, are also fixed for all datasets. Due to variations in depth sensor accuracy, however, we adjust the tracking depth loss weight λ_d and the tracking success threshold ε_t individually for each dataset. The pose graph optimization is carried out using GTSAM [14], employing the Levenberg-Marquardt method with a maximum iteration limit of 50. We implement our system mainly using PyTorch, and all reported experiments are conducted on an NVIDIA A6000 GPU.

After completing the pose estimation of all frames, we directly merge the active map $\mathcal{M}_A$ and inactive map $\mathcal{M}_I$ to form a complete scene representation, which is then used to evaluate both reconstruction and rendering quality. Following prior works [46], [50], [69], [103], we incorporate a map refinement stage to further enhance reconstruction results. Specifically, we perform an additional optimization of the map using all keyframes for 26,000 iterations.

B. Tracking Performance

The first experiment evaluate how well our approach estimates the camera poses and compare it to existing baselines. The results of this experiment support our first claim that our 2DGS-SLAM system demonstrates higher tracking accuracy. We evaluate tracking performance on all three datasets using the ATE RMSE [76] as the metric. Among them, the Replica dataset is widely adopted for benchmarking rendering-based SLAM systems. On this synthetic dataset, we compare our 2DGS-SLAM method with several state-of-the-art approaches based on NeRF [52], 3D Gaussian splatting, and neural signed distance fields. Among the SDF-based NeRF-SLAM methods, we include GO-SLAM [100], E-SLAM [37], and Co-SLAM [86], all of which utilize neural signed distance fields for surface reconstruction. We also include MonoGS-2D [49], a concurrent work that similarly employs 2DGS as the map representation and derives on-manifold SE(3) pose gradients for 2DGS, replacing the 3DGS in MonoGS [50]. As shown in Tab. II, our proposed 2DGS-SLAM outperforms all baselines. The trajectory error of our method is only half that of the second-best method, RTG-SLAM [64], which estimates camera poses by integrating multi-level ICP with ORB-SLAM2 [54]. This strong performance is largely attributed to the high-quality, noise-free depth images provided by Replica, which enable our 2DGS representation to fully exploit the advantages of consistent depth rendering. It should be noted that all methods in Tab. II use the noise-free synthetic depth data provided by the Replica dataset as input, ensuring a fair comparison. These results also validate the effectiveness of our rendering-based camera pose optimization approach.

For the tracking results on the TUM-RGBD dataset, in addition to rendering-based methods, we also compare against classical RGB-D SLAM approaches such as Kintinuous [90], ElasticFusion [91], ORB-SLAM2 [54], and RTAB-Map [43]. As reported in Tab. III, our 2DGS-SLAM outperforms all rendering-based baselines in terms of average accuracy. In smaller-scale sequences such as `desk`, `xyz`, and `office`, our approach performs on par with state-of-the-art methods. For larger scenes like `room` and sequences with more motion blur such as `desk2`, benefiting from the strength of our efficient loop closure mechanism, our method achieves the best performance. Compared to classical methods, 2DGS-SLAM demonstrates superior performance over dense fusion approaches such as Kintinuous and ElasticFusion, but still falls slightly short of ORB-SLAM2. Furthermore, the ScanNet dataset poses additional challenges, as all eight sequences are captured in room-scale or multi-room-scale indoor environments, where robust loop closure becomes critical. As shown in Tab. IV, SLAM methods without explicit loop closure mechanisms, such as Point-SLAM [69], MonoGS [50], and SplaTAM [38], suffer from significantly higher pose estimation errors. Our method ranks second in average trajectory accuracy across all eight sequences, demonstrating the strength of our loop closure strategy. It is worth noting that the best-performing method, GO-SLAM [100], relies heavily on optical flow-based DROID-SLAM [80] for tracking and loop closure. Additionally, the ground-truth trajectories in ScanNet are generated by BundleFusion [11] rather than a high-precision motion capture system.

C. 3D Reconstruction Performance

We evaluate the reconstruction quality on the 7-Scenes dataset [72] and select representative methods with different map representations as baselines, including Co-SLAM [86], Loopy-SLAM [46], and LoopSplat [103]. We generate the ground-truth mesh by performing TSDF fusion using the reference poses and depth maps provided by the dataset, and then evaluate the reconstructed meshes by computing bidirectional point-to-surface distances between the reconstructed and ground-truth meshes. Specifically, we report Accuracy (the mean distance from reconstructed points to the ground-truth surface), Completion (the mean distance from ground-truth points to the reconstructed surface), and their average as the Chamfer distance. We use 0.1 m as distance threshold to determine matched points. As shown in Tab. V, our method achieves the highest overall performance, with the best average Accuracy, Completion, and Chamfer distance among all baselines. This experiment quantitatively demonstrates that our algorithm can produce high-quality reconstructions in real-world scenes.

We further evaluate the reconstruction quality on the Replica dataset, which provides ground-truth meshes for all sequences. We render depth images at keyframe poses using the global Gaussian splat map, followed by TSDF fusion [9] to obtain the final reconstructed mesh. Two commonly used metrics, Depth L1 error and F1 score are employed for the evaluation. Depth L1 error measures the difference between the reconstructed and ground-truth meshes by rendering depth images from 1,000 randomly sampled camera poses and computing the per-pixel L1 distance. The F1 score ($F1$) evaluates the geometric accuracy of the mesh by jointly considering precision (P) and recall (R), and is calculated as their harmonic mean: $F1 = 2\frac{PR}{P+R}$. Here, precision (P) denotes the percentage of points on the predicted mesh that lie within 1 cm of any point on the ground-truth mesh, while recall (R) measures the percentage of ground-truth points that are similarly close to the predicted mesh. Our evaluation setup is consistent with previous works [46], [69], [103], [104]. We select both Gaussian splatting-based and NeRF-style volume rendering-based methods as baselines for our quantitative experiments on the Replica dataset. As shown in Tab. VI, in terms of

TABLE II: Absolute Trajectory Error (ATE) on the Replica dataset, reported in centimeters. **LC** denotes if loop closure is enabled. We highlight the best results in **bold** and the second best results are $\underline{\text{underscored}}$.

Method	Map Representation	LC	Rm 0	Rm 1	Rm 2	Off0	Off1	Off2	Off3	Off4	Avg.
NICE-SLAM [104]	feature grids	✗	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.13	1.06
GO-SLAM [100]	feature grids	✓	0.34	0.29	0.29	0.32	0.30	0.39	0.39	0.46	0.35
E-SLAM [37]	feature planes	✗	0.71	0.70	0.52	0.57	0.55	0.58	0.72	0.63	0.63
Point-SLAM [69]	feature points	✗	0.61	0.41	0.37	0.38	0.48	0.54	0.69	0.72	0.52
Loopy-SLAM [46]	feature points	✓	0.24	0.24	0.28	0.26	0.40	0.29	0.22	0.35	0.29
PIN-SLAM [62]	feature points	✗	0.27	0.31	$\underline{0.13}$	0.22	0.30	0.28	0.16	0.28	0.24
RTG-SLAM [64]	3DGS	✓	$\underline{0.20}$	$\underline{0.18}$	$\underline{0.13}$	0.22	$\underline{0.12}$	$\underline{0.22}$	0.20	$\underline{0.19}$	$\underline{0.18}$
MonoGS [50]	3DGS	✗	0.33	0.22	0.29	0.36	0.19	0.25	$\underline{0.12}$	0.81	0.32
MonoGS-2D [49]	2DGS	✗	0.42	0.43	0.35	$\underline{0.19}$	0.19	$\underline{0.22}$	0.27	0.80	0.36
SplaTAM [38]	3DGS	✗	0.31	0.40	0.29	0.47	0.27	0.29	0.32	0.72	0.38
Gaussian-SLAM [97]	3DGS	✗	0.29	0.29	0.22	0.37	0.23	0.41	0.30	0.35	0.31
LoopSplat [103]	3DGS	✓	0.28	0.22	0.17	0.22	0.16	0.49	0.20	0.30	0.26
2DGS-SLAM (ours)	2DGS	✓	**0.06**	**0.08**	**0.10**	**0.04**	**0.07**	**0.07**	**0.06**	**0.09**	**0.07**

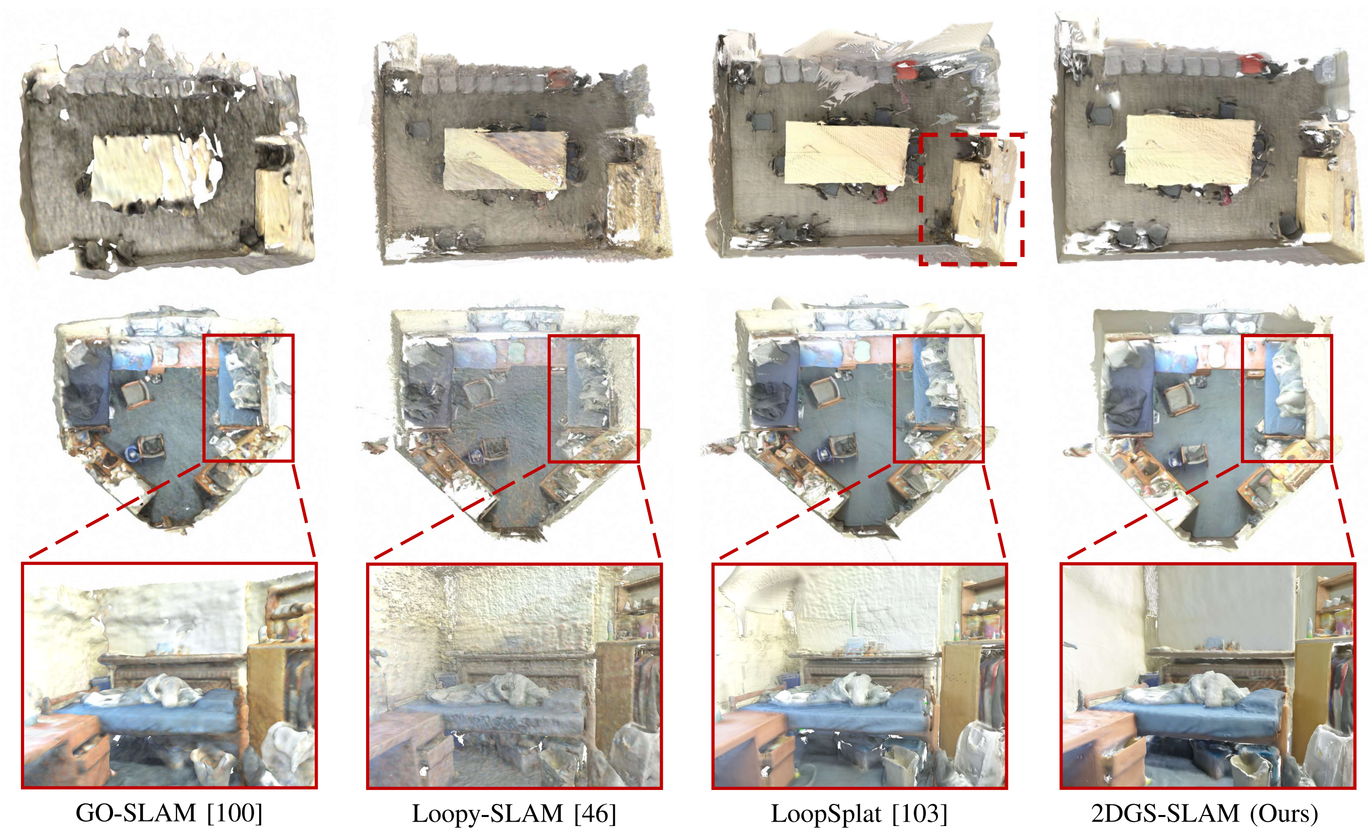


Fig. 7: Qualitative comparison of reconstruction results on the ScanNet dataset. The first row presents the reconstructed meshes on sequence `0169`. We highlight the map duplication caused by LoopSplat's pose drift by a red dashed box.The second and third rows show the results on sequence `0233`, including both the overall meshes and zoomed-in local views. It can be observed that our 2DGS-SLAM achieves the most globally consistent and smooth reconstruction among all methods.

Depth L1 error, our method ranks second, behind NeRF-based method Loopy-SLAM [46], outperforms other Gaussian Splatting-based methods. For the F1-score, our approach comes third, following Loopy-SLAM and LoopSplat [103]. It is worth noting that, during depth rendering, *Loopy-SLAM requires ground-truth depth to guide its sampling process.* While this contributes to its high accuracy on synthetic data, it limits the method's applicability in real-world scenarios where depth measurements are noisy. On the other hand, LoopSplat does not maintain a global map representation. Instead, it continuously generates local submaps during operation. For mesh reconstruction, LoopSplat renders depth images from these submaps, typically dozens per scene, and performs TSDF fusion. Since each submap only covers a limited range of viewpoints, the rendered depths tend to closely resemble the original input depth images. While this approach performs well on synthetic data, it often leads to excessive artifacts and map inconsistencies in real-world environments due to the lack of global information fusion. Moreover, maintaining a large number of overlapping submaps significantly increases memory consumption and complicates downstream robotic tasks such as planning. These results on both the 7-Scenes and Replica

TABLE III: Absolute Trajectory Error (ATE) on the TUM dataset, reported in centimeters. **LC** denotes if loop closure is enabled. We highlight the best results in **bold** and the second best results are underscored. We separately compare the performance of rendering-based methods and classical approaches.

Method	LC	desk	desk2	room	xyz	office	Avg.
Rendering-based approach							
NICE-SLAM [104]	✗	4.26	4.99	34.49	6.19	3.87	10.76
E-SLAM [37]	✗	2.47	3.69	29.73	**1.11**	2.42	7.89
Co-SLAM [86]	✗	2.77	4.54	58.3	1.84	2.6	14.1
Point-SLAM [69]	✗	4.34	4.54	30.92	1.31	3.48	8.92
Loopy-SLAM [46]	✓	3.79	3.38	7.03	1.62	3.41	3.85
MonoGS [50]	✗	**1.59**	7.03	8.55	1.44	**1.49**	4.02
MonoGS-2D [49]	✗	**1.59**	8.51	12.8	1.20	1.83	5.14
SplaTAM [38]	✗	3.35	6.54	11.13	1.24	5.16	5.48
Gaussian-SLAM [97]	✗	2.73	6.03	14.92	1.39	5.31	6.08
LoopSplat [103]	✓	2.08	3.54	6.24	1.58	3.22	3.33
2DGS-SLAM (ours)	✓	1.84	**2.76**	**5.98**	1.16	1.97	**2.74**
Classical SLAM approach							
Kintinuous [90]	✓	3.7	7.1	7.5	2.9	3.0	4.8
ElasticFusion [91]	✓	2.0	4.8	6.8	1.1	1.7	3.3
ORB-SLAM2 [54]	✓	**1.6**	**2.2**	**4.7**	**0.4**	**1.0**	**2.0**
RTAB-Map [43]	✓	2.9	4.4	6.6	0.5	2.1	3.3

TABLE IV: Absolute trajectory error (ATE) on the ScanNet dataset (cm). **LC** denotes if loop closure is enabled. We highlight the best results in **bold** and the second best results are underscored.

Method	LC	00	59	106	169	181	207	54	233	Avg.
NICE-SLAM	✗	12.0	14.0	7.9	10.9	13.4	6.2	20.9	9.0	13.0
GO-SLAM	✓	5.4	7.5	**7.0**	7.7	**6.8**	6.9	8.8	4.8	**6.9**
E-SLAM	✗	7.3	8.5	7.5	**6.5**	9.0	**5.7**	36.3	**4.3**	10.6
Point-SLAM	✗	10.2	7.8	8.7	22.0	14.8	9.5	28.0	6.1	14.3
Loopy-SLAM	✓	**4.2**	7.5	8.3	7.5	10.6	7.9	**7.5**	5.2	7.7
MonoGS	✗	9.8	32.1	8.9	10.7	21.8	7.9	17.5	12.4	15.2
SplaTAM	✗	12.8	10.1	17.7	12.1	11.1	7.5	56.8	4.8	16.6
Gaussian-SLAM	✗	21.2	12.8	13.5	16.3	21.0	14.3	37.1	11.1	18.4
LoopSplat	✓	6.2	7.1	7.4	10.6	8.5	6.6	16.0	4.7	8.4
2DGS-SLAM	✓	6.6	**6.9**	7.1	**6.5**	8.6	6.0	11.0	4.7	7.2

datasets support our second claim that our method outperforms SDF-based NeRF-SLAM and 3DGS-based approaches in terms of surface reconstruction quality and global consistency. Fig. 7 shows our qualitative results on the ScanNet dataset. We can observe that, in real-world environments with challenging lighting conditions and noisy depth measurements, Loopy-SLAM, which performs best on synthetic datasets, produces noticeably coarse mesh reconstructions. Similarly, LoopSplat suffers from issues such as more artifacts and map inconsistencies. In contrast, our approach demonstrates superior global consistency and produces smoother surface reconstructions in real-world datasets.

D. Rendering Quality

The next set of experiments is designed to evaluate the rendering quality of our method. The results support the second part of our second claim, i.e., our approach enables high-fidelity rendering suitable for online robotic applications. We evaluate the rendering quality of our method by computing the differences between the rendered images at all training views and their corresponding input images. The evaluation metrics include peak signal-to-noise ratio (PSNR), structural similarity (SSIM) [88], and learned perceptual image patch similarity (LPIPS) [99]. For baselines, we select the state-of-the-art NeRF-based methods, Point-SLAM [69], as well as Gaussian splatting based method, including SplaTAM [38], MonoGS [50], and LoopSplat [103]. We conduct quantitative evaluations on the Replica and ScanNet datasets. As shown in Tab. VII, 2DGS-SLAM achieves the best PSNR and LPIPS scores on the Replica dataset, with its SSIM score also being on par with other Gaussian splatting-based methods. On the real-world ScanNet dataset, our method ranks second in average metric scores, with PSNR and SSIM worse than LoopSplat. However, it is important to note that LoopSplat employs complex post-processing to merge its submaps. Specifically, after completing pose estimation for all frames, LoopSplat first performs TSDF fusion using depth images rendered from different submaps to obtain the global mesh, then initializes a new set of Gaussian splats from the vertices of resulting mesh and optimizes them using all RGB-D keyframes for 30,000 iterations to generate a global radiance field. To isolate the impact of post-processing, we report extra rendering results of our method, MonoGS and LoopSplat without any map refinement on Tab. IX. Since LoopSplat stores sub-maps instead of a unified global map, we directly merged its sub-maps using their respective poses to construct a global Gaussian splat map. The results show that our method outperforms the baselines in terms of PSNR, SSIM, and LPIPS, and maintains competitive performance compared to the results obtained with map refinement. Due to the severe pose drift, which can be seen in Tab. IV, MonoGS struggles to reconstruct a reliable radiance field for larger scenes online. Meanwhile, LoopSplat does not maintain a globally consistent map, leading to significant artifacts in the accumulated Gaussian splat submaps and making it unsuitable for high-quality rendering required by online robotic applications.

TABLE V: Reconstruction comparison on the 7-Scenes dataset. Columns a–g represent the sequences [chess, fire, heads, office, pumpkin, redkitchen, stairs], respectively. We highlight the best results in **bold** and the second best results are underscored. - indicates methods that failed on this sequence.

Method	Metric	a	b	c	d	e	f	g	Avg.
Co-SLAM	Acc.[cm] ↓	11.6	9.0	3.6	8.3	17.3	7.1	16.9	10.5
	Comp.[cm] ↓	11.7	2.2	2.1	5.8	12.6	4.3	9.4	6.9
	Cham.[cm] ↓	11.6	5.6	2.8	7.1	15.0	5.7	13.1	8.7
Loopy-SLAM	Acc.[cm] ↓	4.8	5.7	3.1	3.2	4.6	2.9	7.6	4.6
	Comp.[cm] ↓	3.7	4.4	**1.5**	**2.9**	**3.2**	**2.5**	4.0	3.2
	Cham.[cm] ↓	4.2	5.1	**2.3**	3.1	3.9	2.7	5.8	3.9
LoopSplat	Acc.[cm] ↓	9.1	-	3.3	5.2	-	8.0	14.5	-
	Comp.[cm] ↓	8.1	-	3.6	5.8	-	5.2	9.6	-
	Cham.[cm] ↓	8.6	-	3.5	5.5	-	6.6	12.0	-
2DGS-SLAM (ours)	Acc.[cm] ↓	**2.9**	**1.3**	**1.8**	**2.7**	**2.9**	**2.3**	**1.9**	**2.3**
	Comp.[cm] ↓	**3.0**	**1.8**	3.4	**2.9**	3.4	2.7	**2.0**	**2.7**
	Cham.[cm] ↓	**3.0**	**1.6**	2.6	**2.8**	**3.1**	**2.5**	**1.9**	**2.5**

This observation is also supported by qualitative results on TUM dataset, as illustrated in the Fig. 8. Here, we compare the Gaussian splat maps obtained directly from each method after pose estimation, without any post-processing applied to the maps. As shown, LoopSplat's rendering results suffer from severe artifacts. Moreover, the normal renderings reveal that due to inconsistencies in 3DGS-based depth rendering, LoopSplat and MonoGS fail to produce smooth surface reconstructions. In comparison, our method not only achieves high-fidelity RGB renderings but also accurately reconstructs scene geometry. While SplaTAM achieves comparable reconstruction quality, it requires a much larger number of Gaussian splats than our approach. We provide a detailed comparison of memory and time consumption in the next section.

TABLE VI: Reconstruction comparison on the Replica dataset. We highlight the best results in **bold** and the second best results are <u>underscored</u>. * indicates methods that use ground-truth depth for sampling.

Method	Map Representation	Metric	Rm 0	Rm 1	Rm 2	Off0	Off1	Off2	Off3	Off4	Avg.
NICE-SLAM [104]	feature grids	Depth L1[cm]↓	1.81	1.44	2.04	1.39	1.76	8.33	4.99	2.01	2.97
		F1 [%]↑	45.0	44.8	43.6	50.0	51.9	39.2	39.9	36.5	43.9
E-SLAM [37]	feature planes	Depth L1[cm]↓	0.97	1.07	1.28	0.86	1.26	1.71	1.43	1.06	1.18
		F1 [%]↑	81.0	82.2	83.9	78.4	75.5	77.1	75.5	79.1	79.1
Loopy-SLAM* [46]	feature points	Depth L1[cm]↓	**0.30**	**0.20**	**0.42**	**0.23**	<u>0.46</u>	**0.60**	**0.37**	**0.24**	**0.35**
		F1 [%]↑	**91.6**	**92.4**	<u>90.6</u>	**93.9**	**91.6**	<u>88.5</u>	**89.0**	**88.7**	**90.8**
SplaTAM [38]	3DGS	Depth L1[cm]↓	0.43	0.38	0.54	0.44	0.66	1.05	1.60	0.68	0.72
		F1 [%]↑	89.3	88.2	88.0	91.7	90.0	85.1	77.1	80.1	86.1
Gaussian-SLAM [97]	3DGS	Depth L1[cm] ↓	0.61	0.25	0.54	0.50	0.52	0.98	1.63	0.42	0.68
		F1 [%] ↑	88.8	91.4	90.5	91.7	90.1	87.3	84.2	87.4	88.9
LoopSplat [103]	3DGS	Depth L1[cm]↓	0.39	0.23	0.52	0.32	0.51	<u>0.63</u>	1.09	0.40	0.51
		F1 [%]↑	90.6	<u>91.9</u>	**91.1**	<u>93.3</u>	90.4	**88.9**	<u>88.7</u>	<u>88.3</u>	<u>90.4</u>
MonoGS-2D [49]	2DGS	Depth L1[cm] ↓	0.45	0.28	0.57	0.37	0.59	0.85	0.62	0.63	0.54
		F1 [%] ↑	<u>90.9</u>	91.3	90.5	93.1	<u>91.1</u>	88.2	87.6	77.7	88.8
2DGS-SLAM (ours)	2DGS	Depth L1[cm] ↓	<u>0.34</u>	<u>0.21</u>	<u>0.43</u>	<u>0.27</u>	**0.41**	1.08	<u>0.67</u>	<u>0.28</u>	<u>0.46</u>
		F1 [%] ↑	90.8	91.6	<u>90.6</u>	93.1	90.1	87.0	87.6	87.5	89.7

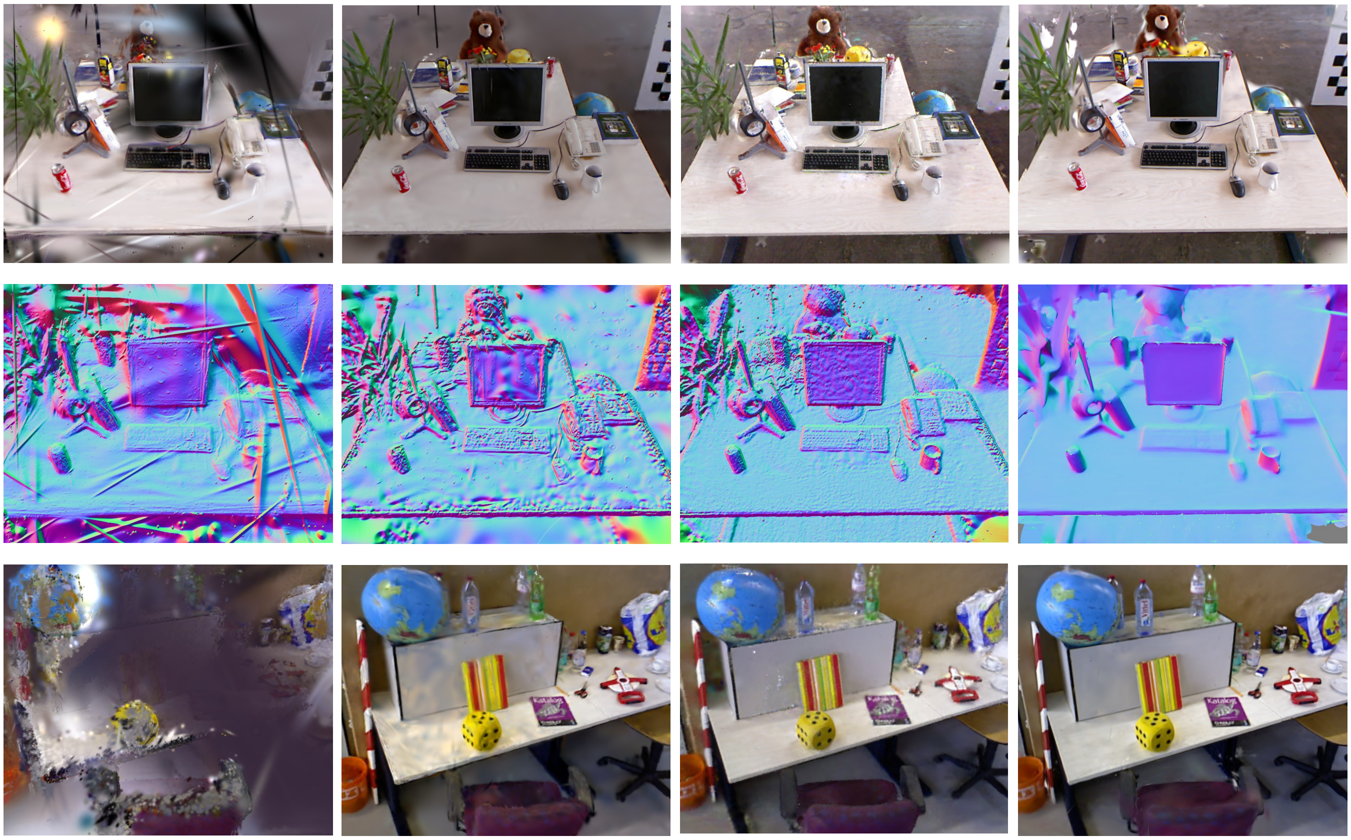
TABLE VII: Rendering performance comparison on the Replica dataset. We report three metrics: PSNR [dB], SSIM, and LPIPS. The best results are highlighted in **bold**, and the second best results are <u>underscored</u>.

Method	Metric	Rm 0	Rm 1	Rm 2	Off0	Off1	Off2	Off3	Off4	Avg.
Point-SLAM [69]	PSNR ↑	32.40	34.08	35.50	38.26	39.16	33.99	33.48	33.49	35.17
	SSIM ↑	<u>0.974</u>	<u>0.977</u>	0.982	<u>0.983</u>	<u>0.986</u>	0.960	0.960	<u>0.979</u>	<u>0.975</u>
	LPIPS ↓	0.113	0.116	0.110	0.118	0.156	0.132	0.142	0.124	0.126
SplaTAM [38]	PSNR ↑	32.86	33.89	35.25	38.26	39.17	31.97	29.70	31.81	34.11
	SSIM ↑	**0.980**	0.970	0.980	0.980	0.980	<u>0.970</u>	0.950	0.970	0.970
	LPIPS ↓	0.070	0.100	0.080	0.090	0.090	0.090	0.120	0.150	0.100
MonoGS [50]	PSNR ↑	<u>34.83</u>	<u>36.43</u>	<u>37.49</u>	39.50	<u>42.09</u>	<u>36.24</u>	**36.70**	36.07	<u>37.50</u>
	SSIM ↑	0.954	0.959	0.965	0.971	0.977	0.964	0.963	0.957	0.960
	LPIPS ↓	<u>0.068</u>	<u>0.076</u>	0.075	<u>0.072</u>	<u>0.055</u>	<u>0.078</u>	<u>0.065</u>	0.099	0.070
LoopSplat [103]	PSNR ↑	33.07	35.32	36.16	<u>40.82</u>	40.21	34.67	35.67	<u>37.10</u>	36.63
	SSIM ↑	0.973	**0.978**	**0.985**	**0.992**	**0.990**	**0.985**	**0.990**	**0.989**	**0.985**
	LPIPS ↓	0.116	0.122	0.111	0.085	0.123	0.140	0.096	0.106	0.112
2DGS-SLAM (ours)	PSNR ↑	**35.63**	**37.09**	**38.47**	**43.14**	**42.39**	**36.33**	<u>36.16</u>	**38.8**	**38.50**
	SSIM ↑	0.965	0.968	0.973	<u>0.985</u>	0.980	0.968	<u>0.966</u>	0.971	0.972
	LPIPS ↓	**0.044**	**0.048**	**0.05**	**0.029**	**0.046**	**0.049**	**0.046**	**0.049**	**0.045**

E. Runtime and Memory Evaluation

The following experiment and results support the claim that our approach is more efficient in terms of runtime and produces a more compact map compared to the other rendering-based baselines. To compare the performance of different methods, we evaluate frames per second (FPS), calculated as the total number of frames in the sequence divided by the total time, as well as the memory usage of the map without post-processing and peak GPU memory consumption on the ScanNet sequence `scene0000`, which contains a total of 5,578 frames. We selected main baselines from the previous experiments, including rendering-based methods with different map representations, such as Point-SLAM [69], Loopy-SLAM [46], MonoGS [50], SplaTAM [38], and LoopSplat [103], for comparison. As shown in Tab. X, our method is only slower than MonoGS in terms of FPS. This is expected, as our approach involves additional tasks such as image feature extraction, loop closure detection, relocalization, and map updates, which do not exist in MonoGS as it does not incorporate loop closure. In comparison with other methods that do support loop closure, such as Loopy-SLAM [46] and LoopSplat [103], our approach demonstrates significantly higher time efficiency, achieving a 6-7× speedup.

Additionally, and thanks to our efficient map management mechanism, our final map has the smallest memory footprint, suggesting that the number of redundant Gaussian splats in our system is much lower than in other Gaussian splatting-based methods. In contrast, due to the lack of removing redundant Gaussian splats, SplaTAM's map memory usage is more than 20 times higher than ours. The continuously accumulating redundant splats also lead to a decrease in its pose estimation efficiency over time. Furthermore, since LoopSplat stores overlapping submaps rather than maintaining a global map, its memory usage for map storage is very high due to the accumulation of redundant splats. In terms of peak GPU memory consumption, our method is slightly higher than LoopSplat. However, this is because LoopSplat offloads all submaps to disk in order to minimize runtime memory usage. Unfortunately, the frequent disk I/O and CPU-GPU data transfers significantly slow down its speed compared to ours. These results highlight the advantages of our submap-



LoopSplat [103] MonoGS [50] SplaTAM [38] 2DGS-SLAM (Ours)

Fig. 8: Qualitative comparison of Rendering results on the TUM dataset. For a fair comparison, we selected non-training views and used the raw Gaussian splatting maps from each method without any map refinement. The first and second rows show comparisons on sequence `fr2_xyz`, including both RGB and normal renderings. As 3DGS does not support direct normal rendering, we compute normal images from the rendered depth images using Eq. (23) for visualization. The third row shows results on sequence `fr3_office`. Our method achieves the most photorealistic RGB renderings and the smoothest normal image.

TABLE VIII: Rendering performance comparison on the ScanNet dataset. We report three metrics: PSNR [dB], SSIM, and LPIPS. The best results are highlighted in **bold**, and the second best results are underscored.

Method	Metric	0000	0059	0106	0169	0181	0207	Avg.
NICE-SLAM [104]	PSNR ↑	18.71	16.55	17.29	18.75	15.56	18.38	17.54
	SSIM ↑	0.641	0.605	0.646	0.629	0.562	0.646	0.621
	LPIPS ↓	0.561	0.534	0.510	0.534	0.602	0.552	0.548
Point-SLAM [69]	PSNR ↑	19.06	16.38	18.46	18.69	16.75	19.66	18.17
	SSIM ↑	0.662	0.615	0.753	0.650	0.666	0.696	0.673
	LPIPS ↓	0.515	0.528	0.439	0.513	0.532	0.500	0.504
SplaTAM [38]	PSNR ↑	19.33	19.27	17.73	21.97	16.76	19.8	19.14
	SSIM ↑	0.660	0.792	0.690	0.776	0.683	0.696	0.716
	LPIPS ↓	**0.438**	**0.289**	0.376	**0.281**	**0.420**	**0.341**	**0.358**
MonoGS [50]	PSNR ↑	21.13	19.70	21.35	22.44	22.02	20.95	21.26
	SSIM ↑	0.723	0.722	0.808	0.781	0.814	0.725	0.762
	LPIPS ↓	0.448	0.436	0.339	0.362	0.432	0.459	0.412
LoopSplat [103]	PSNR ↑	**24.99**	**23.23**	**23.35**	**26.80**	**24.82**	**26.33**	**24.92**
	SSIM ↑	**0.840**	**0.831**	**0.846**	**0.877**	**0.824**	**0.854**	**0.845**
	LPIPS ↓	0.450	0.400	0.409	0.346	0.514	0.430	0.425
2DGS-SLAM	PSNR ↑	23.36	19.00	20.53	24.67	21.27	23.71	22.09
	SSIM ↑	0.767	0.729	0.795	0.796	0.821	0.779	0.781
	LPIPS ↓	0.440	0.444	**0.357**	0.362	0.485	0.425	0.418

TABLE IX: Rendering performance comparison on the ScanNet dataset. All the reported results are evaluated from the raw Gaussians splatting map without any refinement. We report three metrics: PSNR [dB], SSIM, and LPIPS. The best results are highlighted in **bold**, and the second best results are underscored.

Method	Metric	0000	0059	0106	0169	0181	0207	Avg.
MonoGS [50]	PSNR ↑	15.40	15.98	**18.34**	18.75	15.43	16.34	16.70
	SSIM ↑	0.597	0.591	0.701	0.683	0.642	0.651	0.644
	LPIPS ↓	0.646	0.591	0.500	0.525	0.577	0.577	0.569
LoopSplat [103]	PSNR ↑	12.35	12.95	10.26	10.86	11.47	13.17	11.84
	SSIM ↑	0.413	0.411	0.318	0.495	0.541	0.504	0.447
	LPIPS ↓	0.840	0.724	0.798	0.791	0.698	0.704	0.759
2DGS-SLAM	PSNR ↑	**21.95**	**16.16**	17.71	**22.72**	**19.74**	**22.00**	**20.05**
	SSIM ↑	**0.740**	**0.639**	**0.710**	**0.763**	**0.793**	**0.744**	**0.731**
	LPIPS ↓	**0.453**	**0.501**	**0.456**	**0.392**	**0.464**	**0.435**	**0.450**

free approach with efficient map management. In summary, when compared to other rendering-based SLAM methods with loop closure support, 2DGS-SLAM outperforms the baselines in both memory usage and runtime efficiency.

F. Ablation Study

To systematically evaluate the contribution of each component in our MASt3R-based loop closure pipeline, we perform an ablation study on the TUM and ScanNet datasets. As shown in Tab. XII, disabling the entire loop closure module results in a significant degradation in pose estimation accuracy, especially on the ScanNet dataset that contains multiple rooms. Disabling the relocalization function provided by MASt3R also leads to severe performance drops when large drift accumulates, as the system fails to effectively relocalize and close

TABLE X: Statistics of runtime and memory. We report three metrics: FPS, map size (MB), and peak GPU memory (MB). **LC** denotes if loop closure is enabled. The best results are highlighted in **bold**, and the second best results are underscored.

Method	LC	FPS (Hz) ↑	Map size (MB) ↓	GPU Memory (MB) ↓
Point-SLAM [69]	✗	0.05	99.4	**8236**
Loopy-SLAM [46]	✓	0.13	195.3	12475
MonoGS [50]	✗	**1.92**	<u>13.2</u>	<u>9062</u>
SplaTAM [38]	✗	0.18	213.1	12939
LoopSplat [103]	✓	0.17	4608	9616
2DGS-SLAM (ours)	✓	<u>0.92</u>	**9.7**	10822

TABLE XI: Comparison of different loop closure method on the TUM and ScanNet datasets. Mean Absolute Trajectory Error (ATE) is reported in centimeters. The best and second-best results are highlighted in bold and underlined, respectively.

Method	TUM	ScanNet	Avg.
FPFH+RANSAC [46]	<u>3.2</u>	<u>8.3</u>	<u>5.75</u>
Two-stage ICP	3.97	12.5	8.24
ours	**2.74**	<u>7.1</u>	**4.92**

loops. Although MASt3R's inference results are up-to-scale, the predicted scale is generally close to the real one in most cases. Therefore, turning off the scale correction module does not cause a notable drop in trajectory accuracy. Meanwhile, enabling the revisit loop query module increases the number of effective loop closures and slightly improves pose estimation accuracy.

To verify the effectiveness of our MASt3R-based relocalization module, we compare it against two alternative approaches. The first baseline, FPFH+RANSAC, is adapted from the relocalization pipeline of Loopy-SLAM [46]. After identifying a historical frame as a loop candidate, we generate point clouds from the historical frame's depth map and the active Gaussian splat map, extract Fast Point Feature Histograms (FPFH) features, and perform correspondence matching with RANSAC outlier rejection, followed by point-to-plane ICP refinement. The second baseline, Two-stage ICP, performs a two-stage point-to-plane ICP after obtaining S^s and S^t in the same manner. The alignment comprises a coarse stage (maximum correspondence distance of 0.3 m) and a subsequent fine stage (0.03 m), to estimate the relative pose between S^s and S^t. For fair comparison, after obtaining the relative pose from either baseline, we apply the same scan-to-map tracking refinement based on the active 2DGS map as described in Sec. IV-E, to refine the estimated pose and filter out false positives from failed loop closures. As shown in Tab. XI, our MASt3R-based approach achieves the best performance on both datasets. The Two-stage ICP baseline performs the worst, especially on sequences with large accumulated drift, while FPFH+RANSAC suffers from depth noise and feature mismatches, resulting in inferior trajectory accuracy compared to our method.

G. *Experiments on Self-recorded Robot Data*

To evaluate the effectiveness of our method in real-world robotic applications beyond publicly available datasets, we also collected data using a wheeled mobile robot equipped with Intel RealSense D455 RGB-D cameras in indoor environments. The experimental scenes include (1) `corridor`, a 20-meter-long straight corridor used to evaluate the robustness of our pose estimation in low-texture, repetitive environments; (2,3) `kitchen` and `office`, two rooms measuring approximately 7 m × 6 m, where the robot performs challenging maneuvers such as rapid pure rotations during recording. As shown in Fig. 9a, we use AprilTags mounted on the ceiling to compute near ground-truth poses with approximately 1 cm global accuracy for evaluation. It is also worth noting that, compared to the structured-light-based RGB-D cameras used in datasets such as ScanNet [11] and TUM-RGBD [76], the stereo-vision-based RealSense D455 typically produces noisier depth images.

TABLE XII: Ablation study of loop closure related modules on the TUM and ScanNet datasets. Mean Absolute Trajectory Error (ATE) is reported in centimeters. The best and second-best results are highlighted in bold and underlined, respectively.

Loop Closure	MASt3R	Scale est.	Revisit	Tum	ScanNet	Avg
✗	✗	✗	✗	3.41	16.87	10.14
✓	✗	✗	✓	3.11	11.76	7.43
✓	✓	✗	✓	2.97	<u>7.67</u>	5.32
✓	✓	✓	✗	<u>2.80</u>	7.76	<u>5.28</u>
✓	✓	✓	✓	**2.74**	**7.12**	**4.93**

TABLE XIII: Absolute trajectory error (ATE) on the self-recorded dataset (cm). **LC** denotes if loop closure is enabled. The best results are highlighted in **bold**, and the second best results are underscored.

Method	LC	corridor	kitchen	office	Avg.
Point-SLAM [69]	✗	30.8	15.9	23.9	23.5
Loopy-SLAM [46]	✓	Failed	63.9	<u>7.9</u>	-
MonoGS [50]	✗	9.5	17.6	13.9	13.6
SplaTAM [38]	✗	29.8	130.5	16.7	59.0
LoopSplat [103]	✓	**2.1**	<u>10.1</u>	22.6	<u>11.6</u>
2DGS-SLAM (ours)	✓	<u>3.4</u>	**6.8**	**4.7**	**5.0**

As illustrated in Fig. 9b, our method achieves high-quality scene reconstruction, demonstrating not only a high-fidelity radiance field but also smooth surface normal rendering. We further conducted quantitative pose estimation experiments, comparing our method with the main baselines evaluated in the aforementioned public datasets. As shown in Tab. XIII, our method yields substantially lower average trajectory error than all baseline methods, highlighting its robustness to depth noise and rapid camera motion. Moreover, our method successfully performs loop closures on both the `kitchen` and `office` sequences, significantly reducing pose drift compared to competing methods. Consistent with the observations in experiment V-B, methods that lack loop closure support, such as Point-SLAM [69], MonoGS [50], and SplaTAM [38], suffer from severe pose drift, making them unsuitable for room-scale reconstruction and real-world mobile robot applications. compared with rendering-based methods with loop closure capability, including Loopy-SLAM [46] and LoopSplat [103], our approach demonstrates superior robustness in both motion estimation and loop closure, highlighting the practical value of our method in robotic applications.

H. *Experiments on Larger-Scale Scenes*

To further demonstrate the performance of our system on larger-scale environments, we conducted qualitative experiments on the BS3D dataset [56], a real-world large-scale

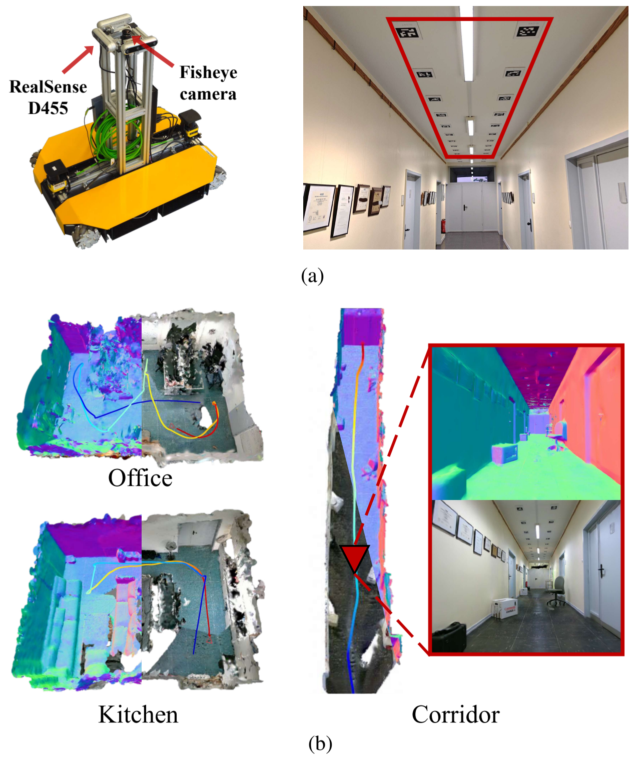


Fig. 9: (a) The wheeled robot platform used in our experiments and AprilTags mounted on the ceiling. We use the fisheye camera installed on the robot to detect AprilTags for pose evaluation. (b) Reconstructed Gaussian splat maps and camera trajectories of our method on three experimental scenes: `office`, `kitchen`, and `corridor`. For the `corridor`, we demonstrate zoomed-in views of both RGB and normal renderings. No map refinement was applied after tracking.

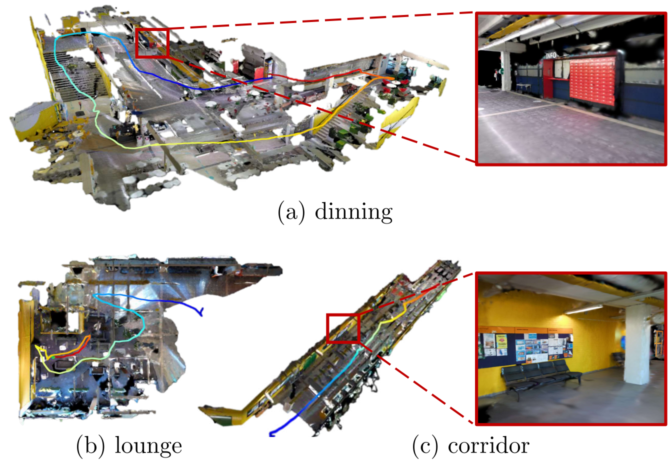


Fig. 10: Qualitative results of our method on three sequences from the BS3D dataset, including the reconstructed Gaussian splat maps, estimated camera trajectories, and local view renderings.

indoor dataset captured using an Azure Kinect depth camera. The reconstruction results are shown in Fig. 10. In particular, the sequence in Fig. 10(a) spans two floors and covers more than $1000\,\mathrm{m}^2$, while the scenes shown in Fig. 10(b) and Fig. 10(c) each cover approximately $200\,\mathrm{m}^2$. The reconstructed map for the largest sequence, called "dining", contains about 600k Gaussian splats. These results show that our method can handle not only room-scale scenes but also much larger environments, while preserving global consistency and high-fidelity rendering quality. In all cases shown in the figure, the GPU memory consumption remained below 14 GB throughout reconstruction.

VI. Limitations

While our system achieves globally consistent reconstruction through loop closure and map correction, there are certain limitations that should be acknowledged. When the map undergoes large-scale deformation after loop closure, local inconsistencies can be introduced due to pose-tracking errors and the limited degree of freedom in the deformation model. Although these effects can be gradually reduced during continuous map optimization with historical frames, the reconstruction quality inevitably degrades compared to the pre-deformation state. Furthermore, although our algorithm can be adjusted for different application scenarios by tuning parameters, for example by increasing the number of Gaussian splats to improve reconstruction quality of fine object details, it cannot yet adapt automatically to varying scene conditions. Moreover , while our linear illumination model with learnable parameters α_l and β_l can handle moderate lighting changes, abrupt illumination changes can exceed the capacity of this linear model and may lead to tracking failure. Additionally, our rendering-based camera tracking computes rendering errors over all pixels. As a result, moving objects often produce large rendering errors and can easily dominate the optimization process. Consequently, compared with traditional sparse feature-based SLAM systems, our method is more sensitive to dynamic objects in both tracking and mapping. Overall, we view these issues as important limitations of the current system and meaningful directions for future work.

VII. Conclusion

In this paper, we proposed 2DGS-SLAM, a novel RGB-D SLAM framework that enables globally consistent radiance field reconstruction based on 2D Gaussian splatting. Taking advantage of the consistent depth rendering of 2D Gaussian splatting, we propose an accurate camera tracking framework. We further introduced an efficient map management strategy and integrated a strong feed-forward 3D reconstruction model MASt3R to enable robust loop closure detection and relocalization. We implemented and evaluated our approach on different datasets and provided comparisons to other existing techniques and supported all claims made in this paper. The results demonstrate that our method achieves superior pose estimation accuracy compared to other rendering-based approaches, while delivering comparable or even better surface reconstruction quality. Moreover, our 2DGS-SLAM consistently outperforms 3D Gaussian splatting-based systems in terms of surface smoothness and global consistency. At the same time, our method maintains competitive image rendering quality with significantly improved efficiency compared with other rendering based method with loop closure support.

VIII. Acknowledgements

We thank Haofei Kuang and Niklas Trekel for providing the real-world robot-collected datasets along with ground-truth poses, and Liyuan Zhu for sharing the baseline results.

References

[1] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN Architecture for Weakly Supervised Place Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] R. Arandjelovic and A. Zisserman. All About VLAD. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[3] D. Azinović, R. Martin-Brualla, D.B. Goldman, M. Nießner, and J. Thies. Neural RGB-D Surface Reconstruction. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[4] L. Bartolomei, L. Teixeira, and M. Chli. Fast Multi-UAV Decentralized Exploration of Forests. *IEEE Robotics and Automation Letters (RA-L)*, 8(9):5576–5583, 2023.

[5] J. Behley and C. Stachniss. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[6] J.L. Blanco-Claraco. A flexible framework for accurate LiDAR odometry, map manipulation, and localization. *Intl. Journal of Robotics Research (IJRR)*, 44(9):1553–1599, 2025.

[7] C. Campos, R. Elvira, J.J.G. Rodríguez, J.M. Montiel, and J.D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Trans. on Robotics (TRO)*, 37(6):1874–1890, 2021.

[8] X. Chen, T. Läbe, A. Milioto, T. Röhling, O. Vysotska, A. Haag, J. Behley, and C. Stachniss. OverlapNet: Loop Closing for LiDAR-based SLAM. In *Proc. of Robotics: Science and Systems (RSS)*, 2020.

[9] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1996.

[10] A. Dai, A. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[11] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. BundleFusion: Real-time Globally Consistent 3D Reconstruction using Online Surface Re-integration. *ACM Trans. on Graphics (TOG)*, 36(3):1–18, 2017.

[12] P. Dai, J. Xu, W. Xie, X. Liu, H. Wang, and W. Xu. High-quality Surface Reconstruction using Gaussian Surfels. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2024.

[13] B. Della Corte, I. Bogoslavskyi, C. Stachniss, and G. Grisetti. A General Framework for Flexible Multi-Cue Photometric Point Cloud Registration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[14] F. Dellaert. Factor graphs and GTSAM: A hands-on introduction. *Georgia Institute of Technology, Tech. Rep*, 2:4, 2012.

[15] T. Deng, G. Shen, T. Qin, J. Wang, W. Zhao, J. Wang, D. Wang, and W. Chen. PLGSLAM: Progressive Neural Scene Represenation with Local to Global Bundle Adjustment. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[16] L. Di Giammarino, L. Brizi, T. Guadagnino, C. Stachniss, and G. Grisetti. Md-slam: Multi-cue direct slam. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2022.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2021.

[18] B. Duisterhof, Z. Lojze, W. Philippe, L. Vincent, C. Yohann, and R. Jerome. MASt3R-SfM: A Fully-Integrated Solution for Unconstrained Structure-from-Motion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2025.

[19] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3D Mapping with an RGB-D Camera. *IEEE Trans. on Robotics (TRO)*, 30(1):177–187, 2014.

[20] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[21] D. Galvez-López and J.D. Tardós. Bags of Binary Words for Fast Place Recognition in Image Sequences. *IEEE Trans. on Robotics (TRO)*, 28(5):1188–1197, 2012.

[22] E. Giacomini, L. Di Giammarino, L.D. Rebott, G. Grisetti, and M.R. Oswald. Splat-LOAM: Gaussian Splatting LiDAR Odometry and Mapping. *arXiv preprint*, arXiv:2503.17491, 2025.

[23] B. Glocker, J. Shotton, A. Criminisi, and S. Izadi. Real-time RGB-D camera relocalization via randomized ferns for keyframe encoding. *IEEE Trans. on Visualization and Computer Graphics*, 21(5):571–583, 2014.

[24] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth. Openfabmap: An open source toolbox for appearance-based loop closure detection. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[25] T. Guadagnino, B. Mersch, S. Gupta, I. Vizzo, G. Grisetti, and C. Stachniss. KISS-SLAM: A Simple, Robust, and Accurate 3D LiDAR SLAM System With Enhanced Generalization Capabilities. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2025.

[26] S. Ha, J. Yeon, and H. Yu. RGBD GS-ICP SLAM. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2024.

[27] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[28] J. Hu, M. Mao, H. Bao, G. Zhang, and Z. Cui. CP-SLAM: Collaborative Neural Point-based SLAM System. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2023.

[29] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2024.

[30] C. Huang, O. Mees, A. Zeng, and W. Burgard. Audio visual language maps for robot navigation. In *Proc. of the Intl. Symp. on Experimental Robotics (ISER)*, 2023.

[31] H. Huang, L. Li, C. Hui, and S.K. Yeung. Photo-SLAM: Real-time Simultaneous Localization and Photorealistic Mapping for Monocular, Stereo, and RGB-D Cameras. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[32] Y. Huang, B. Cui, L. Bai, Z. Chen, J. Wu, Z. Li, H. Liu, and H. Ren. Advancing Dense Endoscopic Reconstruction with Gaussian Splatting-driven Surface Normal-aware Tracking and Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2025.

[33] S. Izquierdo and J. Civera. Close, But Not There: Boosting Geographic Distance Sensitivity in Visual Place Recognition. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2024.

[34] S. Izquierdo and J. Civera. Optimal Transport Aggregation for Visual Place Recognition. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[35] W. Jiaxin and S. Leutenegger. GSFusion: Online RGB-D Mapping Where Gaussian Splatting Meets TSDF Fusion. *IEEE Robotics and Automation Letters (RA-L)*, 9(12):11865–11872, 2024.

[36] L. Jin, X. Zhong, Y. Pan, J. Behley, C. Stachniss, and M. Popovic. ActiveGS: Active Scene Reconstruction using Gaussian Splatting. *IEEE Robotics and Automation Letters (RA-L)*, 10(5):4866–4873, 2025.

[37] M.M. Johari, C. Carta, and F. Fleuret. ESLAM: Efficient Dense SLAM System Based on Hybrid Representation of Signed Distance Fields. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[38] N. Keetha, J. Karhade, K.M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, and J. Luiten. SplaTAM: Splat Track & Map 3D Gaussians for Dense RGB-D SLAM. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[39] M. Keller, D. Lefloch, M. Lambers, and S. Izadi. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2013.

[40] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. on Graphics (TOG)*, 42(4):1–14, 2023.

[41] C. Kerl, J. Sturm, and D. Cremers. Robust Odometry Estimation for RGB-D Cameras. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2013.

[42] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[43] M. Labbe and F. Michaud. RTAB-Map: An open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics (JFR)*, 36(1):416–446, 2019.

[44] V. Leroy, Y. Cabon, and J. Revaud. Grounding Image Matching in 3D with MASt3R. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2024.

[45] T.Y. Lim, B. Sun, M. Pollefeys, and H. Blum. Loop Closure from Two Views: Revisiting PGO for Scalable Trajectory Estimation through Monocular Priors. *arXiv preprint*, arXiv:2503.16275, 2025.

[46] L. Liso, E. Sandström, V. Yugay, L. Van Gool, and M.R. Oswald. Loopy-slam: Dense neural slam with loop closures. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[47] D. Maier, A. Hornung, and M. Bennewitz. Real-time navigation in 3D environments based on depth camera data. In *Proc. of the IEEE Intl. Conf. on Humanoid Robots*, 2012.

[48] Y. Mao, X. Yu, K. Wang, Y. Wang, R. Xiong, and Y. Liao. NGEL-SLAM: Neural Implicit Representation-based Global Consistent Low-Latency SLAM System. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2024.

[49] H. Matsuki, G. Bae, and A. Davison. 4DTAM: Non-Rigid Tracking and Mapping via Dynamic Surface Gaussians. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[50] H. Matsuki, R. Murai, P.H. Kelly, and A.J. Davison. Gaussian splatting SLAM. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[51] Z. Matthias, R. Jussi, B. Mario, D. Carsten, and P. Mark. Perspective accurate splatting. In *Proc. of Graphics Interface (GI)*, 2004.

[52] B. Mildenhall, P. Srinivasan, M. Tancik, J. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[53] M. Milford and G. Wyeth. SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[54] R. Mur-Artal and J. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Trans. on Robotics (TRO)*, 33(5):1255–1262, 2017.

[55] R. Murai, E. Dexheimer, and A.J. Davison. MASt3R-SLAM: Real-Time Dense SLAM with 3D Reconstruction Priors. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[56] J. Mustaniemi, J. Kannala, E. Rahtu, L. Liu, and J. Heikkilä. BS3D: Building-scale 3D Reconstruction from RGB-D Images. *arXiv preprint*, arXiv:2301.01057, 2023.

[57] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinect-Fusion: Real-Time Dense Surface Mapping and Tracking. In *Proc. of the Intl. Symp. on Mixed and Augmented Reality (ISMAR)*, 2011.

[58] M. Oquab, T. Darcet, T. Moutakanni, H.V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, R. Howes, P.Y. Huang, H. Xu, V. Sharma, S.W. Li, W. Galuba, M. Rabbat, M. Assran, N. Ballas, G. Synnaeve, I. Misra, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. DINOv2: Learning Robust Visual Features without Supervision. *Trans. on Machine Learning Research (TMLR)*, pages 1–31, 2024.

[59] E. Palazzolo, J. Behley, P. Lottes, P. Giguere, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[60] Y. Pan, P. Xiao, Y. He, Z. Shao, and Z. Li. MULLS: Versatile LiDAR SLAM Via Multi-Metric Linear Least Square. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[61] Y. Pan, X. Zhong, L. Jin, L. Wiesmann, M. Popović, J. Behley, and C. Stachniss. PINGS: Gaussian Splatting Meets Distance Fields within a Point-Based Implicit Neural Map. In *Proc. of Robotics: Science and Systems (RSS)*, 2025.

[62] Y. Pan, X. Zhong, L. Wiesmann, T. Posewsky, J. Behley, and C. Stachniss. PIN-SLAM: LiDAR SLAM Using a Point-Based Implicit Neural Representation for Achieving Global Map Consistency. *IEEE Trans. on Robotics (TRO)*, 40:4045–4064, 2024.

[63] J.J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[64] Z. Peng, T. Shao, L. Yong, J. Zhou, Y. Yang, J. Wang, and K. Zhou. RTG-SLAM: Real-time 3D Reconstruction at Scale using Gaussian Splatting. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2024.

[65] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto. Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps. *IEEE Robotics and Automation Letters (RA-L)*, 5(1):227–234, 2019.

[66] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2011.

[67] R. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2009.

[68] E. Sandström, K. Tateno, M. Oechsle, M. Niemeyer, L. Van Gool, M.R. Oswald, and F. Tombari. Splat-SLAM: Globally Optimized RGB-only SLAM with 3D Gaussians. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[69] E. Sandström, Y. Li, L. Van Gool, and M. R. Oswald. Point-SLAM: Dense Neural Point Cloud-based SLAM. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[70] T. Schops, T. Sattler, and M. Pollefeys. BAD SLAM: Bundle Adjusted Direct RGB-D SLAM. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[71] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[72] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[73] C. Stachniss. *Springer Handbuch der Geodäsie*, chapter Simultaneous Localization and Mapping. Springer Verlag, 2016. In German, invited.

[74] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J.J. Engel, R. Mur-Artal, C. Ren, and S.a. Verma. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint*, arXiv:1906.05797, 2019.

[75] J. Stückler and S. Behnke. Multi-Resolution Surfel Maps for Efficient Dense 3D Modeling and Tracking. *Journal of Visual Communication and Image Representation (JVCIR)*, 25(1):137–147, 2014.

[76] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[77] E. Sucar, S. Liu, J. Ortiz, and A.J. Davison. imap: Implicit mapping and positioning in real-time. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[78] S. Sun, M. Mielle, A.J. Lilienthal, and M. Magnusson. High-Fidelity SLAM Using Gaussian Splatting with Rendering-Guided Densification and Regularized Optimization. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[79] Y. Tang, J. Zhang, Z. Yu, H. Wang, and K. Xu. MIPS-Fusion: Multi-Implicit-Submaps for Scalable and Robust Online Neural RGB-D Reconstruction. *ACM Trans. on Graphics (TOG)*, 42(6):1–14, 2023.

[80] Z. Teed and J. Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[81] G. Tolias, Y. Avrithis, and H. Jégou. To aggregate or not to aggregate: Selective match kernels for image search. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2013.

[82] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 13(4):376–380, 1991.

[83] L. Vincent, M.N. Francesc, and F. Pascal. EPnP: An Accurate O(n) Solution to the PnP Problem. *Intl. Journal of Computer Vision (IJCV)*, 81:255–166, 2009.

[84] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1029–1036, 2023.

[85] O. Vysotska and C. Stachniss. Lazy Data Association For Image Sequences Matching Under Substantial Appearance Changes. *IEEE Robotics and Automation Letters (RA-L)*, 1(1):213–220, 2016.

[86] H. Wang, J. Wang, and L. Agapito. Co-SLAM: Joint Coordinate and Sparse Parametric Encodings for Neural Real-Time SLAM. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[87] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud. DUst3R: Geometric 3D Vision Made Easy. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[88] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. on Image Processing*, 13(4):600–612, 2004.

[89] A. Werby, C. Huang, M. Büchner, A. Valada, and W. Burgard. Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation. In *Proc. of Robotics: Science and Systems (RSS)*, 2024.

[90] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially Extended KinectFusion. In *Proc. of the RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[91] T. Whelan, S. Leutenegger, R.S. Moreno, B. Glocker, and A. Davison. ElasticFusion: Dense SLAM Without A Pose Graph. In *Proc. of Robotics: Science and Systems (RSS)*, 2015.

[92] K. Wu, Z. Zhang, M. Tie, Z. Ai, Z. Gan, and W. Ding. VINGS-Mono: Visual-Inertial Gaussian Splatting Monocular SLAM in Large Scenes. *arXiv preprint*, arXiv:2501.08286, 2025.

[93] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, and X. Li. GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[94] X. Yang, H. Li, H. Zhai, Y. Ming, Y. Liu, and G. Zhang. Vox-Fusion: Dense Tracking and Mapping with Voxel-based Neural Implicit Representation. In *Proc. of the Intl. Symp. on Mixed and Augmented Reality (ISMAR)*, 2022.

[95] S. Yu, C. Cheng, Y. Zhou, X. Yang, and H. Wang. OpenGS-SLAM: RGB-Only Gaussian Splatting SLAM for Unbounded Outdoor Scenes. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2025.

[96] V. Yugay, T. Gevers, and M.R. Oswald. MAGiC-SLAM: Multi-Agent Gaussian Globally Consistent SLAM. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[97] V. Yugay, Y. Li, T. Gevers, and M.R. Oswald. Gaussian-SLAM: Photo-realistic Dense SLAM with Gaussian Splatting. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[98] G. Zhang, E. Sandström, Y. Zhang, M. Patel, L. Van Gool, and M.R. Oswald. GlORIE-SLAM: Globally Optimized Rgb-only Implicit Encoding Point Cloud SLAM. *arXiv preprint*, arXiv:2403.19549, 2024.

[99] R. Zhang, P. Isola, A.A. Efros, E. Shechtman, and O. Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[100] Y. Zhang, F. Tosi, S. Mattoccia, and M. Poggi. GO-SLAM: Global Optimization for Consistent 3D Instant Reconstruction. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

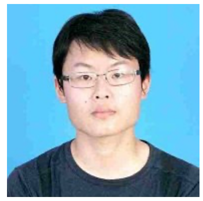
[101] X. Zhong, Y. Pan, J. Behley, and C. Stachniss. SHINE-Mapping: Large-Scale 3D Mapping Using Sparse Hierarchical Implicit Neural Representations. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[102] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.H. Yang. DrivingGaussian: Composite Gaussian Splatting for Surrounding Dynamic Autonomous Driving Scenes. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

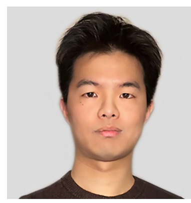
[103] L. Zhu, Y. Li, E. Sandström, S. Huang, K. Schindler, and I. Armeni. LoopSplat: Loop Closure by Registering 3D Gaussian Splats. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2025.

[104] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M.R. Oswald, and M. Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Xingguang Zhong is a Ph.D. student in Engineering at the Photogrammetry & Robotics Lab at the University of Bonn, Germany headed by Prof. Cyrill Stachniss. He obtained his B.Sc. degree in Mechanical Engineering in 2017 and his M.Sc. degree in Mechatronic Engineering in 2019 from Harbin Institute of Technology, China. His research interests include autonomous navigation and scene representations for robotics.



Yue Pan is a Ph.D. student in Engineering at the Photogrammetry & Robotics Lab headed by Prof. Cyrill Stachniss at the University of Bonn, Germany. He obtained his B.Sc. degree in Geomatics Engineering from Wuhan University, China in 2019 and received his MSc degree in Geomatics Engineering from ETH Zurich, Switzerland in 2022. His research focuses on SLAM, 3D reconstruction and navigation.



Liren Jin is a Ph.D. student at the University of Bonn since March 2021. He is advised by Prof. Marija Popović and Prof. Cyrill Stachniss. He received his Master's degree in 2020 from RWTH Aachen, Germany and Bachelor's degree in 2016 from Hefei University of Technology, China, both in Mechanical Engineering. His research focuses on active perception, 3D reconstruction and implicit neural representations.

Marija Popović is an assistant professor at TU Delft, the Netherlands. Before starting at Delft, she was a Junior Research Group Leader at the University of Bonn, Germany and a postdoctoral research associate at the Smart Robotics Lab at Imperial College London, the UK. She received her Ph.D. from the Autonomous Systems Lab at ETH Zurich, Switzerland in 2019. Her research centres around developing algorithms that enable intelligent robotic decision-making, including in planning, environmental mapping, and computer vision.

Jens Behley received his Dipl.-Inform. in computer science in 2009 and his Ph.D. in computer science in 2014, both from the Dept. of Computer Science at the University of Bonn, Germany. Since 2016, he is a postdoctoral researcher at the Photogrammetry & Robotics Lab at the University of Bonn, Germany. He finished his habilitation at the University of Bonn in 2023. His area of interest lies in the area of perception for autonomous vehicles, deep learning for semantic interpretation, and LiDAR-based SLAM.

Cyrill Stachniss is a Full Professor at the University of Bonn and heads the Lab for Photogrammetry and Robotics and is with the Lamarr Institute for Machine Learning and Artificial Intelligence. Until 2025, he was additionally a Visiting Professor in Engineering at the University of Oxford. He is the Spokesperson of the DFG Cluster of Excellence PhenoRob at the University of Bonn. His research focuses on probabilistic techniques and learning approaches for mobile robotics, perception, and navigation. The main application areas of his research are agricultural robotics, service robotics, and self-driving cars.

Certificate of Reproducibility

The authors of this publication declare that:

1) The software related to this publication is distributed in the hope that it will be useful, support open research, and simplify the reproducability of the results but it comes without any warranty and without even the implied warranty of merchantability or fitness for a particular purpose.

2) *Xingguang Zhong* primarily developed the implementation related to this paper. This was done on Ubuntu 22.04.

3) *Yue Pan* verified that the code can be executed on a machine that follows the software specification given in the Git repository available at:

   https://github.com/PRBonn/2DGS-SLAM

4) *Yue Pan* verified that the experimental results presented in this publication can be reproduced using the implementation used at submission, which is labeled with a tag in the Git repository and can be retrieved using the command:

   ```
   git checkout tro26
   ```