GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting

Chi Yan[1,3*] Delin Qu[1,2*] Dan Xu[3] Bin Zhao[1,4]
Zhigang Wang[1] Dong Wang[1†] Xuelong Li[1,5]
[1]Shanghai AI Laboratory [2]Fudan University [3]Hong Kong University of Science and Technology
[4]Northwestern Polytechnical University [5]TeleAI, China Telecom Corp Ltd

Abstract

*In this paper, we introduce **GS-SLAM** that first utilizes 3D Gaussian representation in the Simultaneous Localization and Mapping (SLAM) system. It facilitates a better balance between efficiency and accuracy. Compared to recent SLAM methods employing neural implicit representations, our method utilizes a real-time differentiable splatting rendering pipeline that offers significant speedup to map optimization and RGB-D rendering. Specifically, we propose an adaptive expansion strategy that adds new or deletes noisy 3D Gaussians in order to efficiently reconstruct new observed scene geometry and improve the mapping of previously observed areas. This strategy is essential to extend 3D Gaussian representation to reconstruct the whole scene rather than synthesize a static object in existing methods. Moreover, in the pose tracking process, an effective coarse-to-fine technique is designed to select reliable 3D Gaussian representations to optimize camera pose, resulting in runtime reduction and robust estimation. Our method achieves competitive performance compared with existing state-of-the-art real-time methods on the Replica, TUM-RGBD datasets. Project page: https://gs-slam.github.io/.*

1. Introduction

Simultaneous localization and mapping (SLAM) has emerged as a pivotal technology in fields such as robotics [6], virtual reality [10], and augmented reality [25, 39]. The goal of SLAM is to construct a dense/sparse map of an unknown environment while simultaneously tracking the camera pose. Traditional SLAM methods employ point/surfel clouds [20, 32, 42, 46], mesh representations [26], voxel hashing [12, 18, 23] or voxel grids [21] as scene representations to construct dense mapping, and have made considerable progress on localization accuracy. However, these methods face serious challenges in obtaining fine-grained dense maps.

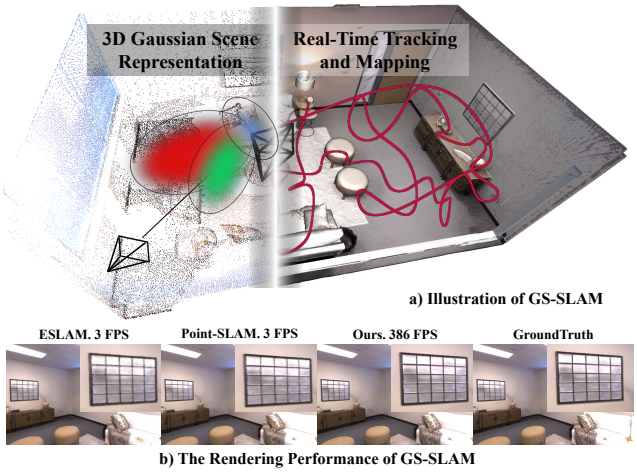


Figure 1. The illustration of the proposed GS-SLAM. It first utilizes the 3D Gaussian representation and differentiable splatting rasterization pipeline in SLAM, achieving real-time tracking and mapping performance on GPU. Besides, benefiting from the splatting rasterization pipeline, GS-SLAM achieves a 100× faster rendering FPS and more high-quality full image results than the other SOTA methods.

Recently, Neural Radiance Fields (NeRF) [19] have been explored to enhance SLAM methodologies and exhibit strengths in generating high-quality, dense maps with low memory consumption [35]. In particular, iMAP [35] uses a single multi-layer perceptron (MLP) to represent the entire scene, which is updated globally with the loss between volume-rendered RGB-D image and ground-truth observations. NICE-SLAM [55] utilizes a hierarchical neural implicit grid as scene map representation to allow local updates for reconstructing large scenes. Moreover, ESLAM [11], CoSLAM [41] and EN-SLAM [24] utilize axis-aligned feature planes and joint coordinate-parametric encoding to improve the capability of scene representation, achieving efficient and high-quality surface map reconstruction. In practical mapping and tracking steps, these methods only render a small set of pixels to reduce optimization time, which leads to the reconstructed dense maps lacking

*Equal contribution: yanchi@pjlab.org.cn.
†Corresponding author: dongwang.dw93@gmail.com.

the richness and intricacy of details. In essence, it is a trade-off for the efficiency and accuracy of NeRF-based SLAM since obtaining high-resolution images with the ray-based volume rendering technique is time-consuming and unacceptable.

Fortunately, recent work [13, 17, 47] with 3D Gaussian representation and tile-based splatting techniques has shown great superiority in the efficiency of high-resolution image rendering. It is applied to synthesize novel view RGB images of static objects, achieving state-of-the-art visual quality for 1080p resolution at real-time speed. Inspired by this, we extend the rendering superiority of 3D Gaussian scene representation and real-time differentiable splatting rendering pipeline for the task of dense RGB-D SLAM and manage to jointly promote the speed and accuracy of NeRF-based dense SLAM, as shown in Fig. 1.

To this end, we propose GS-SLAM, the first RGB-D dense SLAM system that first utilizes 3D Gaussian scene representation coupled with the splatting rendering technique to better balance speed and accuracy. Our system optimizes camera tracking and mapping with a novel RGB-D rendering approach that processes 3D Gaussians quickly and accurately through sorting and α-blending. We enhance scene reconstruction by introducing an adaptive strategy for managing 3D Gaussian elements, which optimizes mapping by focusing on current observations and minimizes errors in dense maps and images. Moreover, we propose a coarse-to-fine approach, starting with low-resolution image analysis for initial pose estimation and refining it with high-resolution rendering using select 3D Gaussians, to boost speed and accuracy. We perform extensive evaluations on a selection of indoor RGB-D datasets and demonstrate state-of-the-art performance on dense neural RGB-D SLAM in terms of tracking, rendering, and mapping. Overall, our contributions include:

- We propose GS-SLAM, the first 3D Gaussian Splatting(3DGS)-based dense RGB-D SLAM approach, which takes advantage of the fast splatting rendering technique to boost the mapping optimizing and pose tracking, achieving real-time and photo-realistic reconstruction performance.
- We present an adaptive 3D Gaussian expansion strategy to efficiently reconstruct new observed scene geometry and develop a coarse-to-fine technique to select reliable 3D Gaussians to improve camera pose estimation.
- Our approach achieves competitive performance on Replica and TUM-RGBD datasets in terms of tracking, and mapping and runs at 8.43 FPS, resulting in a better balance between efficiency and accuracy.

2. Related Work

Dense Visual SLAM. The existing real-time dense visual SLAM systems are typically based on discrete handcrafted features or deep-learning embeddings, and follow the mapping and tracking architecture in [16]. DTAM [22] first introduces a dense SLAM system that uses photometric consistency to track a handheld camera and represent the scene as a cost volume. KinectFusion [44] performs camera tracking by iterative-closest-point and updates the scene via TSDF-Fusion. BAD-SLAM [29] proposes to jointly optimize the keyframe poses and 3D scene geometry via a direct bundle adjustment (BA) technique. In contrast, recent works integrate deep learning with the traditional geometry framework for more accurate and robust camera tracking and mapping, such as DROID-SLAM [37], CodeSLAM [1], SceneCode [54], and NodeSLAM [34], have made significant advances in the field, achieving more accurate and robust camera tracking and mapping performance.

Neural Implict Radiance Field based SLAM. For NeRF-based SLAM, existing methods can be divided into three main types: *MLP-based methods*, *Hybrid representation methods*, and *Explicit methods*. MLP-based method iMAP [35] offers scalable and memory-efficient map representations but faces challenges with catastrophic forgetting in larger scenes. Hybrid representation methods combine the advantages of implicit MLPs and structure features, significantly enhancing the scene scalability and precision. For example, NICE-SLAM [55] integrates MLPs with multi-resolution voxel grids, enabling large scene reconstruction, and Vox-Fusion [48] employs octree expansion for dynamic map scalability, while ESLAM [11] and Point-SLAM [27] utilize tri-planes and neural point clouds respectively to improve the mapping capability. As for the explicit method proposed in [38], it stores map features in voxel directly, without any MLPs, enabling faster optimization. Instead of representing maps with implicit features, GS-SLAM utilizes the 3D Gaussian representation, efficiently renders images using splatting-based rasterization, and optimizes parameters directly with backward propagation.

3D Gaussian Representation. Several recent approaches have use 3D Gaussians for shape reconstruction, such as Fuzzy Metaballs [14, 15], VoGE [40], 3DGS [13]. Notably, 3DGS [13] demonstrates great superiorities in high-quality real-time novel-view synthesis. This work represents the scene with 3D Gaussians and develops a NeRF-style fast rendering algorithm to support anisotropic splatting, achieving SOTA visual quality and fast high-resolution rendering performance. Beyond the rendering superiorities, Gaussian splatting holds an explicit geometry scene structure and appearance, benefiting from the exact modeling of scenes representation [50]. This promising technology has been rapidly applied in several fields, including 3D generation [3, 36, 51], dynamic scene modeling [17][47][49], and photorealistic drivable avatars [56]. However, currently, there is no research addressing camera pose estimation or real-time mapping using 3D Gaussian models due to the in-

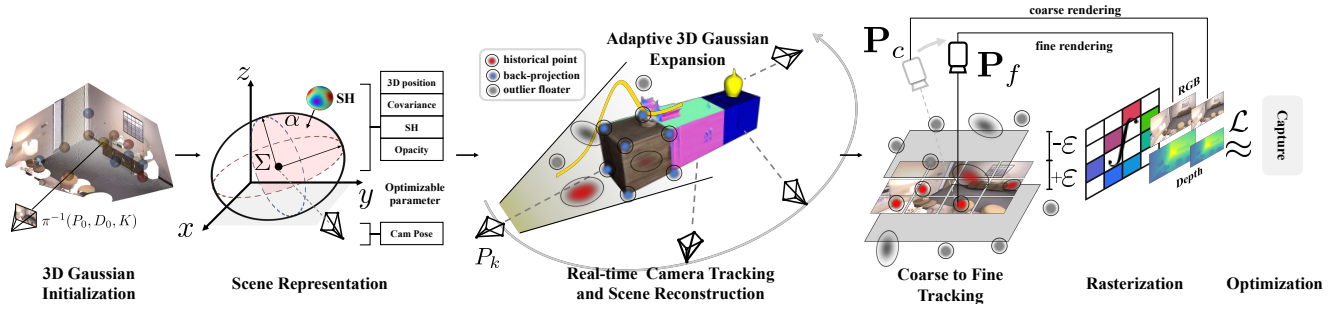


Figure 2. Overview of the proposed method. We aim to use 3D Gaussians to represent the scene and use the rendered RGB-D image for inverse camera tracking. GS-SLAM proposes a novel Gaussian expansion strategy to make the 3D Gaussian feasible to reconstruct the whole scene and can achieve real-time tracking, mapping, and rendering performance on GPU.

herent limitations of the prime pipeline [13], *i.e.*, prerequisites of initialized point clouds or camera pose inputs [28]. In contrast, we derive the analytical derivative equations for pose estimation in the Gaussian representation and implement efficient CUDA optimization.

3. Methodology

Fig. 2 shows the overview of the proposed GS-SLAM. We aim to estimate the camera poses $\{\mathbf{P}_i\}_{i=1}^{N}$ of every frame and simultaneously reconstruct a dense scene map by giving an input sequential RGB-D stream $\{\mathbf{I}_i, \mathbf{D}_i\}_{i=1}^{M}$ with known camera intrinsic $\mathbf{K} \in \mathbb{R}^{3\times3}$. In Sec. 3.1, we first introduce 3D Gaussian as the scene representation $\mathbf{S}$ and the RGB-D render by differentiable splatting rasterization. With the estimated camera pose of the keyframe, in Sec. 3.2, an adaptive expansion strategy is proposed to add new or delete noisy 3D Gaussians to efficiently reconstruct new observed scene geometry while improving the mapping of the previously observed areas. For camera tracking of every input frame, we derive an analytical formula for backward optimization with rendering RGB-D loss. We further introduce an effective coarse-to-fine technique to minimize rendering losses to achieve efficient and accurate pose estimation in Sec. 3.3.

3.1. 3D Gaussian Scene Representation

Our goal is to optimize a scene representation that captures the geometry and appearance of the scene, resulting in a detailed dense map and high-quality novel view synthesis. To do this, we model the scene as a set of 3D Gaussians coupled with opacity and spherical harmonics

$$\mathbf{G} = \{G_i : (\mathbf{X}_i, \mathbf{\Sigma}_i, \Lambda_i, \boldsymbol{Y}_i) | i = 1, ..., N\}. \tag{1}$$

Each 3D Gaussian scene representation G_i is defined by position $\mathbf{X}_i \in \mathbb{R}^3$, 3D covariance matrix $\mathbf{\Sigma}_i \in \mathbb{R}^{3\times3}$, opacity $\Lambda_i \in \mathbb{R}$ and 1-degree spherical harmonics ($\boldsymbol{Y}$) per color channel, a total of 12 coefficients for $\boldsymbol{Y}_i \in \mathbb{R}^{12}$. To reduce the learning difficulty of the 3D Gaussians [57], we parameterize the 3D Gaussian's covariance as:

$$\mathbf{\Sigma} = \mathbf{R}\mathbf{S}\mathbf{S}^T\mathbf{R}^T, \tag{2}$$

where $\mathbf{S} \in \mathbb{R}^3$ is a 3D scale vector, $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix, storing as a 4D quaternion.

Color and depth splatting rendering. With the optimized 3D Gaussian scene representation parameters, given the camera pose $\mathbf{P} = \{\mathbf{R}, \mathbf{t}\}$, the 3D Gaussians G are projected into 2D image plane for rendering with:

$$\mathbf{\Sigma}' = \mathbf{J}\mathbf{P}^{-1}\mathbf{\Sigma}\mathbf{P}^{-T}\mathbf{J}^T, \tag{3}$$

where $\mathbf{J}$ is the Jacobian of the affine approximation of the projective function. After projecting 3D Gaussians to the image plane, the color of one pixel is rendered by sorting the Gaussians in depth order and performing front-to-back α-blending rendering as follows:

$$\hat{\mathbf{C}} = \sum_{i\in N} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{4}$$

where $\mathbf{c}_i$ represents the color of the i-th 3D Gaussian obtained by learned spherical harmonics coefficients $\boldsymbol{Y}$, α_i is the density computed by learned opacity Λ_i and 2D Gaussian with covariance Σ'. Similarly, the depth is rendered by

$$\hat{D} = \sum_{i\in N} d_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{5}$$

where d_i denotes the depth of the center of the i-th 3D Gaussian, which is obtained by projecting to z-axis in the camera coordinate.

3.2. Adaptive 3D Gaussian Expanding Mapping

The 3D Gaussian scene representations are updated and optimized on each selected keyframe for stable mapping. Given the estimated pose of each selected keyframe, we first

apply the proposed adaptive expansion strategy to add new or delete noisy 3D Gaussians from the whole scene representations to render RGB-D images with resolution $H \times W$, and then the updated 3D Gaussian scene representations are optimized by minimizing the geometric depth loss $\mathcal{L}_d$ and the photometric color loss $\mathcal{L}_\mathbf{c}$ to the sensor observation depth D and color $\mathbf{C}$,

$$\mathcal{L}_\mathbf{c} = \sum_{m=1}^{HW} \left| \mathbf{C}_m - \hat{\mathbf{C}}_m \right|, \quad \mathcal{L}_d = \sum_{m=1}^{HW} \left| D_m - \hat{D}_m \right|. \quad (6)$$

The loss optimizes the parameters of all 3D Gaussians that contribute to the rendering of these keyframe images.

Adaptive 3D Gaussian Expansion Strategy. At the first frame of the RGB-D sequence, we first uniformly sample half pixels from a whole image with $H \times W$ resolution and back-projecting them into 3D points $\mathbf{X}$ with corresponding depth observation D. The 3D Gaussian scene representations are created by setting position as $\mathbf{X}$ and initializing zero degree Spherical Harmonics coefficients with RGB color $\mathbf{C}_i$. The opacities are set to pre-defined values, and the covariance is set depending on the spatial point density, *i.e.*,

$$\{G_i = (\mathbf{P}_i, \mathbf{\Sigma}_{init}, \Lambda_{init}, \mathbf{C}_i) | i = 1, ..., M\}, \quad (7)$$

where M equals to $HW/2$. The 3D Gaussians are initialized and then optimized using the first RGB-D image with rendering loss. Note that only half of the pixels are used to initialize the scene, leaving space to conduct adaptive density control of Gaussians that splits large points into smaller ones and clones them with different directions to capture missing geometric details.

Adding step: To obtain a complete map of the environment, the 3D Gaussian scene representations should be able to model the geometry and appearance of newly observed areas. Specifically, at every keyframe, we add first rendered RGB-D images using historical 3D Gaussians and calculate cumulative opacity $T = \sum_{i \in N} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$ for each pixel. We label one pixel as un-reliable x^{un} if its cumulative opacity T is too low or its rendered depth $\hat{D}$ is far away from observed depth D, *i.e.*,

$$T < \tau_T \quad \text{or} \quad |D - \hat{D}| > \tau_D. \quad (8)$$

These selected un-reliable pixels mostly capture new observed areas. Then we back-project these un-reliable pixels to 3D points $\mathbf{P}^{un}$, and a set of new 3D Gaussians at $\mathbf{P}^{un}$ initialized as Eq. 7 are added into scene representations to model the new observed areas.

Deleting step: As shown in Fig. 3, there are some floating 3D Gaussians due to the unstable adaptive control of Gaussians after optimization with Eq. 6. These floating 3D Gaussians will result in a low-quality dense map and a rendered image containing lots of artifacts. To address this issue, after adding new 3D Gaussians, we check all visible 3D Gaussians in the current camera frustum and significantly decrease opacity Λ_i of 3D Gaussians whose position is not near the scene surfaces. Formally, for each visible 3D Gaussian, we draw a ray $\mathbf{r}(t)$ from camera origin $\mathbf{o}$ and its position $\mathbf{X}_i = (x_i, y_i, z_i)$, *i.e.*, $\mathbf{r}(t) = \mathbf{o} + t(\mathbf{X}_i - \mathbf{o})$. Then, we can find a pixel with coordinate (u, v) where this ray intersects the image plane and corresponding depth observation D. The 3D Gaussians are deleted by degenerating its opacity as follows:

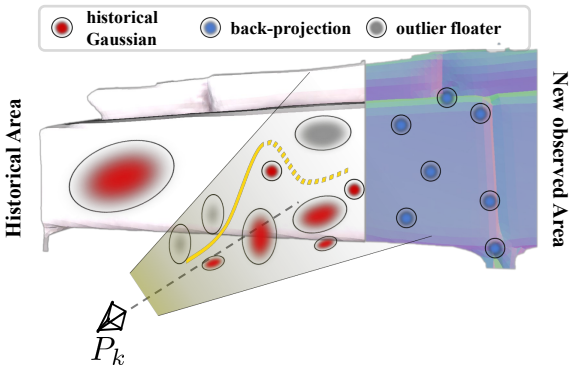


Figure 3. Illustration of the proposed adaptive 3D Gaussian expansion strategy. GS-SLAM inhibits the low-quality 3D Gaussian floaters in the current frustum according to depth.

$$G_i : \Lambda_i \Rightarrow G_i : \eta \Lambda_i, \text{ if } D - dist(\mathbf{X}_i, \mathbf{P}_{uv}) > \gamma, \quad (9)$$

where P_{uv} is the world coordinates of the intersected pixel calculated with the camera intrinsic and extrinsic. $dist(\cdot, \cdot)$ is the Euclidean distance, and η (much smaller than 1) and γ are the hyper-parameters. Note that we decrease the opacity of floating 3D Gaussians in front of the scene surfaces to make our newly added 3D Gaussians well-optimized.

3.3. Tracking and Bundle Adjustment

In the parallel camera tracking phase of our work, we first employ a common straightforward constant velocity assumption to initialize new poses. This assumption transforms the last known pose based on the relative transformation between the second-to-last pose and the last pose. Then, the accurate camera pose $\mathbf{P}$ is optimized by minimizing rendered color loss, *i.e.*,

$$\mathcal{L}_{track} = \sum_{m=1}^{M} \left| \mathbf{C}_m - \hat{\mathbf{C}}_m \right|_1, \quad \min_{\mathbf{R},\mathbf{t}} (\mathcal{L}_{track}), \quad (10)$$

where M is the number of sampled pixels for rendering.

Differentiable pose estimation. According to Eqs. (3) and (4), we observe that the gradient of the camera pose $\mathbf{P}$ is related to three intermediate variables: $\mathbf{\Sigma}'$, $\mathbf{c}_i$, and the projected coordinate $\mathbf{m}_i$ of Gaussian G_i. By applying the chain rule of derivation, we obtain the analytical formula-

tion of camera pose $\mathbf{P}$:

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathbf{c}}}{\partial \mathbf{P}} &= \frac{\partial \mathcal{L}_{\mathbf{c}}}{\partial \mathbf{C}}\frac{\partial \mathbf{C}}{\partial \mathbf{P}} = \frac{\partial \mathcal{L}_{\mathbf{c}}}{\partial \mathbf{C}}\left(\frac{\partial \mathbf{C}}{\partial \mathbf{c}_i}\frac{\partial \mathbf{c}_i}{\partial \mathbf{P}} + \frac{\partial \mathbf{C}}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial \mathbf{P}}\right) \\ &= \frac{\partial \mathcal{L}_{\mathbf{c}}}{\partial \mathbf{C}}\frac{\partial \mathbf{C}}{\partial \alpha_i}\left(\frac{\partial \alpha_i}{\partial \mathbf{\Sigma}'}\frac{\partial \mathbf{\Sigma}'}{\partial \mathbf{P}} + \frac{\partial \alpha_i}{\partial \mathbf{m_i}}\frac{\partial \mathbf{m_i}}{\partial \mathbf{P}}\right) \\ &= \frac{\partial \mathcal{L}_{\mathbf{c}}}{\partial \mathbf{C}}\frac{\partial \mathbf{C}}{\partial \alpha_i}\left(\frac{\partial \alpha_i}{\partial \mathbf{\Sigma}'}\frac{\partial (\mathbf{J}\mathbf{P}^{-1}\mathbf{\Sigma}\mathbf{P}^{-T}\mathbf{J}^T)}{\partial \mathbf{P}} + \frac{\partial \alpha_i}{\partial \mathbf{m_i}}\frac{\partial (\mathbf{K}\mathbf{P}\mathbf{X}_i)}{\partial \mathbf{P} d_i}\right)\end{aligned}, \quad (11)$$

where d_i denotes the z-axis coordinate of projection $\mathbf{m}_i$. The item $\frac{\partial \mathbf{C}}{\partial \mathbf{c}_i}\frac{\partial \mathbf{c}_i}{\partial \mathbf{P}}$ can be eliminated because we are only concerned about the view-independent color in our tracking implementation. In addition, we find that the intermediate gradient $\frac{\partial (\mathbf{K}\mathbf{P}\mathbf{X}_i)}{\partial \mathbf{P} d_i}$ is the deterministic component for the camera pose $\mathbf{P}$. So we simply ignore the back-propagation of $\frac{\partial (\mathbf{J}\mathbf{P}^{-1}\mathbf{\Sigma}\mathbf{P}^{-T}\mathbf{J}^T)}{\partial \mathbf{P}}$ for efficiency. More details can be found in the supplemental materials.

Coarse-to-fine camera tracking. It would be problematic to optimize the camera pose with all image pixels since artifacts in images can cause drifted camera tracking. To address this issue, as shown in Fig. 2, in the differentiable pose estimation step for each frame, we first take advantage of image regularity to render only a sparse set of pixels and optimize tracking loss to obtain a coarse camera pose. This coarse optimization step significantly eases the influence of detailed artifacts. Further, we use this coarse camera pose and depth observation to select reliable 3D Gaussians, which guides GS-SLAM to render informative areas with clear geometric structures to refine coarse camera pose via further optimizing tracking loss on new rendering pixels.

Specifically, in the coarse stage, we first render a coarse image $\hat{\mathbf{I}}_c$ with resolution $H/2 \times W/2$ at uniformly sampled image coordinates and optimize tracking loss in Eq. 10 for T_c iterations, and the obtained camera pose is denoted as $\mathbf{P}_c$. In the fine stage, we use a similar technique with adaptive 3D Gaussian expansion strategy in Section 3.2 to select reliable 3D Gaussian to render full-resolution images while ignoring noisy 3D Gaussians that cause artifacts. In detail, we check all visible 3D Gaussians under coarse camera pose $\mathbf{P}_c$, and remove 3D Gaussians whose position is far away from the scene surface. Formally, for each visible 3D Gaussians G_i with position $\mathbf{X}_i$, we project it to the camera plane using coarse camera pose $\mathbf{P}_c$ and camera intrinsic. Given the projected pixel's depth observation D_i and the distance d_i that is between 3D Gaussians G_i and the camera image plane, the reliable 3D Gaussians are selected as follows:

$$\begin{aligned}\mathbf{G}_{selected} &= \{G_i | G_i \in \mathbf{G} \text{ and } abs(D_i - d_i) \leq \varepsilon\}, \\ \hat{\mathbf{I}}_f &= \mathcal{F}(u, v, \mathbf{G}_{selected}),\end{aligned} \quad (12)$$

where we use the selected reliable 3D Gaussians to render full-resolution images $\hat{\mathbf{I}}_f$. u, v denote the pixel coordinates in $\hat{\mathbf{I}}_f$, and $\mathcal{F}$ represents the color splatting rendering function. The final camera pose $\mathbf{P}$ is obtained by optimizing tracking loss in Eq. 10 with $\hat{\mathbf{I}}_f$ for other T_f iterations. Note that $\hat{\mathbf{I}}_c$ and $\hat{\mathbf{I}}_f$ are only rendered at previously observed areas, avoiding rendering areas where 3D scene representations have not been optimized in the mapping process. Also, we add keyframes based on the proportion of the currently observed image's reliable region to the overall image. At the same time, when the current tracking frame and most recent keyframe differ by more than a threshold value μ_k, this frame will be inserted as a keyframe.

Bundle adjustment. In the bundle adjustment (BA) phase, we optimize the camera poses $\mathbf{P}$ and the 3D Gaussian scene representation $\mathbf{S}$ jointly. We randomly select K keyframes from the keyframe database for optimization, using the loss function similar to the mapping part. For pose optimization stability, we only optimize the scene representation $\mathbf{S}$ in the first half of the iterations. In the other half of the iterations, we simultaneously optimize the map and the poses. Then, the accurate camera pose $\mathbf{P}$ is optimized by minimizing rendering color loss, *i.e.*,

$$\mathcal{L}_{ba} = \frac{1}{K}\sum_{k=1}^{K}\sum_{m=1}^{HW}\left|D_m - \hat{D}_m\right|_1 + \lambda_m\left|\mathbf{C}_m - \hat{\mathbf{C}}_m\right|_1, \ \min_{\mathbf{R},\mathbf{t},\mathbf{S}}(\mathcal{L}_{ba}). \quad (13)$$

4. Experiment

4.1. Experimental Setup

Dataset. To evaluate the performance of GS-SLAM, we conduct experiments on the Replica [31], and TUM-RGBD [33]. Following [11, 27, 41, 48, 55], we use 8 scenes from the Replica dataset for localization, mesh reconstruction, and rendering quality comparison. The selected three subsets of TUM-RGBD datasets are used for localization.

Baselines. We compare our method with existing SOTA NeRF-based dense visual SLAM: NICE-SLAM [55], Vox-Fusion [48], CoSLAM [41], ESLAM [11] and Point-SLAM [27]. The rendering performance of CoSLAM [41] and ESLAM [11] is conducted from the open source code with the same configuration in [27].

Metric. For mesh reconstruction, we use the 2D Depth L1 (cm) [55], the Precision (P, %), Recall (R, %), and F-score with a threshold of 1 cm to measure the scene geometry. For localization, we use the absolute trajectory (ATE, cm) error [33] to measure the accuracy of the estimated camera poses. We further evaluate the rendering performance using the peak signal-to-noise ratio (PSNR), SSIM [43], and LPIPS [52] by following [27]. To be fair, we run all the methods on a dataset 10 times and report the average results. More details can be found in the supplemental materials.

Implementation details. GS-SLAM is implemented in Python using the PyTorch framework, incorporating CUDA code for Gaussian splatting and trained on a desktop PC with a 5.50GHz Intel Core i9-13900K CPU and NVIDIA RTX 4090 GPU. We extended the existing code for differentiable Gaussian splatting rasterization with additional func-

tionality for handling depth, pose, and cumulative opacity during both forward and backward propagation. More details can be found in the supplemental materials.

4.2. Evaluation of Localization and Mapping

Evaluation on Replica. ***Tracking ATE***: Tab. 1 illustrates the tracking performance of our method and the state-of-the-art methods on the Replica dataset. Our method achieves the best or second performance in 7 of 8 scenes and outperforms the second-best method Point-SLAM [27] by 0.4 cm on average at 8.34 FPS. It is noticeable that the second best method, Point-SLAM [27] runs at 0.42 FPS, which is 20× slower than our method, indicating that GS-SLAM achieves a better trade-off between the tracking accuracy and the runtime efficiency. ***Mapping ACC:*** Tab. 3 report the mapping evaluation results of our method with other current state-of-the-art visual SLAM methods. GS-SLAM achieves the best performance in Depth L1 (1.16cm) and Precision (74.0%) metrics on average. For Recall and F1 scores, GS-SLAM performs comparably to the second best method CoSLAM [41]. The visualization results in Fig. 4 show that GS-SLAM achieves satisfying construction mesh with clear boundaries and details.

Evaluation on TUM-RGBD. Tab. 2 compares GS-SLAM with the other SLAM systems in TUM-RGBD dataset. Our method surpasses iMAP [35], NICE-SLAM [55] and Vox-fusion [48], and achieves a comparable performance, average 3.7 cm ATE RSME, with the SOTA methods. A gap to traditional methods still exists between the neural vSLAM and the traditional SLAM systems, which employ more sophisticated tracking schemes [27].

Table 1. Tracking comparison (ATE RMSE [cm]) of the proposed method vs. the SOTA methods on the Replica dataset. The running speed of methods in the upper part is lower than 5 FPS, * denotes the reproduced results by running officially released code.

Method	Rm0	Rm1	Rm2	Off0	Off1	Off2	Off3	Off4	avg
Point-SLAM [27]	0.56	**0.47**	**0.30**	**0.35**	0.62	**0.55**	0.72	0.73	0.54
NICE-SLAM [55]	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.13	1.06
Vox-Fusion* [48]	1.37	4.70	1.47	8.48	2.04	2.58	1.11	2.94	3.09
ESLAM [11]	0.71	0.70	0.52	0.57	0.55	0.58	0.72	**0.63**	0.63
CoSLAM [41]	0.70	0.95	1.35	0.59	0.55	2.03	1.56	0.72	1.00
Ours	**0.48**	0.53	0.33	0.52	**0.41**	0.59	**0.46**	0.7	**0.50**

Table 2. Tracking ATE [cm] on TUM-RGBD [33]. Our method achieves a comparable performance among the neural vSLAMs. * denotes the reproduced results by running officially released code.

Method	fr1_desk	fr2_xyz	fr3_off	Avg.
DI-Fusion [9]	4.4	2.0	5.8	4.1
ElasticFusion [46]	2.5	1.2	2.5	2.1
BAD-SLAM [30]	1.7	1.1	1.7	1.5
Kintinuous [45]	3.7	2.9	3.0	3.2
ORB-SLAM2 [20]	**1.6**	**0.4**	**1.0**	**1.0**
iMAP* [35]	7.2	2.1	9.0	6.1
NICE-SLAM [55]	4.3	31.7	3.9	13.3
Vox-Fusion* [48]	3.5	1.5	26.0	10.3
CoSLAM [41]	2.7	1.9	2.6	2.4
ESLAM [11]	**2.3**	**1.1**	**2.4**	**2.0**
Point-SLAM	2.6	1.3	3.2	2.4
Ours	3.3	1.3	6.6	3.7

4.3. Rendering Evaluation

We compare the rendering performance of the proposed GS-SLAM with the neural visual SLAM methods in Tab. 6. The results show that GS-SLAM achieves the best performance

Table 3. Reconstruction comparison of the proposed method vs. the SOTA methods on Replica dataset.

Method	Metric	Rm 0	Rm 1	Rm 2	Off 0	Off 1	Off 2	Off 3	Off 4	Avg.
NICESLAM [55]	Depth L1 ↓	1.81	1.44	2.04	1.39	1.76	8.33	4.99	2.01	2.97
	Precision ↑	45.86	43.76	44.38	51.40	50.80	38.37	40.85	37.35	44.10
	Recall↑	44.10	46.12	42.78	48.66	53.08	39.98	39.04	35.77	43.69
	F1↑	44.96	44.84	43.56	49.99	51.91	39.16	39.92	36.54	43.86
VoxFusion [48]	Depth L1↓	1.09	1.90	2.21	2.32	3.40	4.19	2.96	1.61	2.46
	Precision↑	75.83	35.88	63.10	48.51	43.50	54.48	69.11	55.40	55.73
	Recall↑	64.89	33.07	56.62	44.76	38.44	47.85	60.61	46.79	49.13
	F1↑	69.93	34.38	59.67	46.54	40.81	50.95	64.56	50.72	52.20
CoSLAM [41]	Depth L1↓	0.99	0.82	2.28	1.24	1.61	7.70	4.65	1.43	2.59
	Precision↑	**81.71**	77.95	73.30	79.41	80.67	55.64	57.63	**79.76**	73.26
	Recall↑	74.03	70.79	65.73	71.46	70.35	52.96	56.06	71.22	66.58
	F1↑	77.68	74.20	69.31	75.23	75.16	54.27	56.83	75.25	69.74
ESLAM [11]	Depth L1↓	**0.63**	**0.62**	**0.98**	**0.57**	1.66	7.32	3.94	**0.88**	2.08
	Precision↑	74.33	75.94	**82.48**	72.20	65.74	70.73	**72.48**	72.24	73.27
	Recall↑	**87.37**	**87.01**	**84.99**	**88.36**	**84.38**	**81.92**	**79.18**	**80.63**	**84.23**
	F1↑	**80.32**	**81.10**	**83.72**	79.47	73.90	**75.92**	**75.68**	**76.21**	**78.29**
Ours	Depth L1↓	1.31	0.82	1.26	0.81	**0.96**	**1.41**	**1.53**	1.08	**1.16**
	Precision↑	64.58	**83.11**	70.13	**83.43**	**87.77**	**70.91**	63.18	68.88	**74.00**
	Recall↑	61.29	76.83	63.84	76.90	76.15	61.63	62.91	61.50	67.63
	F1↑	62.89	79.85	66.84	**80.03**	**81.55**	65.95	59.17	64.98	70.15

in all the metrics. Our method significantly outperforms the second-best methods CoSLAM [41], ESLAM [11] and NICE-SLAM [55] by 1.52 dB in PSNR, 0.027 in SSIM and 0.12 in LPIPS, respectively. It is noticeable that GS-SLAM achieves 386 FPS rendering speed on average, which is 100× faster than the second-best method Vox-Fusion [48]. This excellent rendering performance is attributed to the efficient 3D Gaussian rendering pipeline and can be further applied to real-time downstream tasks, such as VR [5], robot navigation [7] and autonomous driving [2]. The visualization results in Fig. 5 show that GS-SLAM can generate much more high-quality and realistic images than the other methods, especially in edge areas with detailed structures. While NICE-SLAM [55] causes severe artifacts and blurs, CoSLAM [41] and ESLAM [11] generate blur around the image boundaries.

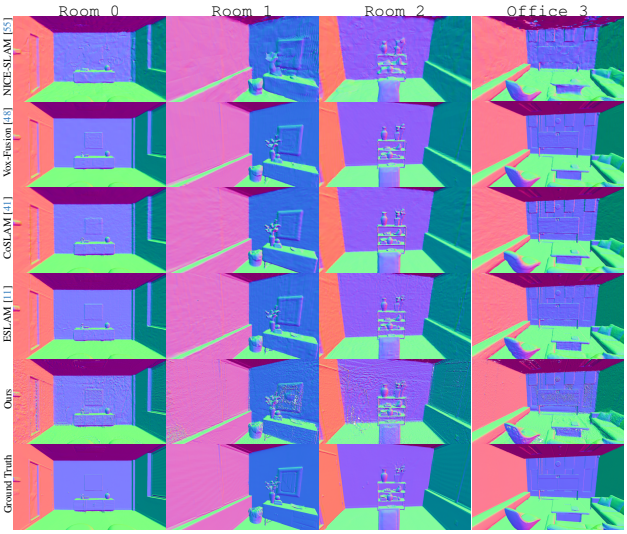


Figure 4. Reconstruction performance comparation of the proposed GS-SLAM and SOTA methods on the Replica dataset.

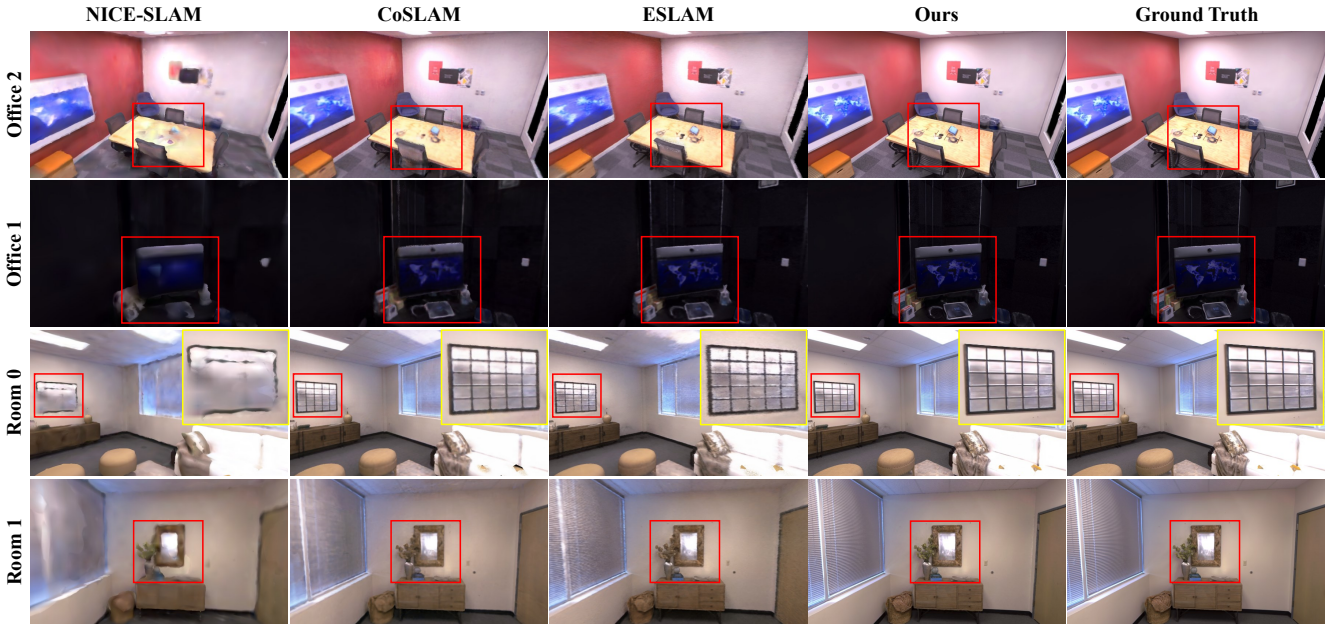


Figure 5. The render visualization results on the Replica dataset of the proposed GS-SLAM and SOTA methods. GS-SLAM can generate much more high-quality and realistic images than the other methods, especially around the object boundaries.

4.4. Runtime Analysis

Tab. 4 and Tab. 5 illustrate the runtime and memory usage of GS-SLAM and the state-of-the-art methods on the Replica and TUM-RGBD, respectively. We report the parameters of the neural networks and the memory usage of the scene representation. Note that Point-SLAM uses extra memory dynamic radius to improve performance (mark as †). The results show that GS-SLAM achieves a competitive running speed with 8.34 FPS compared to the other Radiance Fields-based vSLAMs. Note that we do not use any neural network decoder in our system, which results in the zero learnable parameter. However, the 3D Gaussian scene representations of GS-SLAM consume 198.04 MB memory, 4$\times$ larger than the second large method NICE-SLAM [55]. Memory usage is mainly caused by spherical harmonic coefficients in training, which is a common constraint among Gaussian splatting-based reconstruction methods. Despite this, we still achieve a 20 $\times$ faster FPS compared to the similar point-based method Point-SLAM [27]. Besides, we also provide a light version of GS-SLAM with zero-order spherical harmonic coefficients, significantly reducing memory usage while maintaining stable performance.

4.5. Ablation Study

We perform the ablation of GS-SLAM on the Replica dataset `#Room0` subset to evaluate the effectiveness of coarse-to-fine tracking, and expansion mapping strategy.

Effect of our expansion strategy for mapping. Tab. 7 shows the ablation of our proposed expansion strategy for mapping. The results illustrate that the expansion strategy can significantly improve the tracking and mapping performance. The implementation w/o adding means that we only initialize 3D Gaussians in the first frame and optimize the scene without adding new points. However, this strategy completely crashes because the density control in [13] can not handle real-time mapping tasks without an accurate point cloud input. Besides, the implementation w/o deletion suffers from a large number of redundant and noisy 3D Gaussian, which causes undesirable supervision. In contrast, the proposed expansion strategy effectively improves the tracking and mapping performance by 0.1 in ATE and 11.97 in Recall by adding more accurate constraints for the optimization. According to the visualization results in Fig. 6, our full implementation achieves more high-quality and detailed rendering and reconstruction results than the w/o delete strategy.

Table 4. Runtime and memory usage on Replica `#Room0`. The decoder parameters and embedding denote the parameter number of MLPs and the memory usage of the scene representation.

Method	Tracking [ms×it] ↓	Mapping [ms×it] ↓	System FPS ↑	Decoder param ↓	Scene Embedding↓
Point-SLAM [27]	0.06 × 40	34.81 × 300	0.42	0.127 M	55.42 (+12453.2)†MB
NICE-SLAM [55]	6.64 × 10	28.63 × 60	2.91	0.06 M	48.48 MB
Vox-Fusion [48]	0.03 × 30	66.53 × 10	1.28	0.054 M	1.49 MB
CoSLAM [55]	6.01 × 10	13.18 × 10	16.64	1.671 M	—
ESLAM [11]	6.85 × 8	19.87 × 15	13.42	0.003 M	27.12 MB
GS-SLAM	11.9 × 10	12.8 × 100	8.34	**0 M**	198.04 MB

Table 5. Runtime and memory usage on TUM-RGBD dataset `#fr1_desk` and `#fr2_xyz`.

Method	`#fr1_desk`		`#fr2_xyz`	
	FPS ↑	Memory ↓	FPS ↑	Memory ↓
Point-SLAM	0.10	18.3 (+160.7)†MB	0.12	14.2 (+7687.4)†MB
NICE-SLAM [55]	0.11	178.8MB	0.12	484.0MB
ESLAM [11]	0.31	27.2MB	0.31	51.6MB
GS-SLAM	1.83(ATE:3.3)	40.8MB	1.51(ATE:1.3)	48.4MB
GS-SLAM (light)	1.92(ATE:4.3)	18.8MB	1.68(ATE:2.7)	22.3MB

Table 6. Rendering performance on Replica dataset. We outperform existing dense neural RGB-D methods on the commonly reported rendering metrics. Note that GS-SLAM achieves 386 FPS on average, benefiting from the efficient Gaussian scene representation.

Method	Metric	Room 0	Room 1	Room 2	Office 0	Office 1	Office 2	Office 3	Office 4	Avg.	FPS.
NICE-SLAM [55]	PSNR [dB] ↑	22.12	22.47	24.52	29.07	30.34	19.66	22.23	24.94	24.42	
	SSIM ↑	0.689	0.757	0.814	0.874	0.886	0.797	0.801	0.856	0.809	0.30
	LPIPS ↓	0.330	0.271	0.208	0.229	0.181	0.235	0.209	0.198	0.233	
Vox-Fusion* [48]	PSNR [dB] ↑	22.39	22.36	23.92	27.79	29.83	20.33	23.47	25.21	24.41	
	SSIM ↑	0.683	0.751	0.798	0.857	0.876	0.794	0.803	0.847	0.801	3.88
	LPIPS ↓	0.303	0.269	0.234	0.241	0.184	0.243	0.213	0.199	0.236	
CoSLAM [41]	PSNR [dB] ↑	27.27	28.45	29.06	34.14	34.87	28.43	28.76	30.91	30.24	
	SSIM ↑	0.910	0.909	0.932	0.961	0.969	0.938	0.941	0.955	0.939	3.68
	LPIPS ↓	0.324	0.294	0.266	0.209	0.196	0.258	0.229	0.236	0.252	
ESLAM [11]	PSNR [dB] ↑	25.32	27.77	29.08	33.71	30.20	28.09	28.77	29.71	29.08	
	SSIM ↑	0.875	0.902	0.932	0.960	0.923	0.943	0.948	0.945	0.929	2.82
	LPIPS ↓	0.313	0.298	0.248	0.184	0.228	0.241	0.196	0.204	0.336	
Ours	PSNR [dB] ↑	**31.56**	**32.86**	**32.59**	**38.70**	**41.17**	**32.36**	**32.03**	**32.92**	**34.27**	
	SSIM ↑	**0.968**	**0.973**	**0.971**	**0.986**	**0.993**	**0.978**	**0.970**	**0.968**	**0.975**	**386.91**
	LPIPS ↓	**0.094**	**0.075**	**0.093**	**0.050**	**0.033**	**0.094**	**0.110**	**0.112**	**0.082**	

Table 7. Ablation of the adaptive 3D Gaussian expansion strategy on Replica #Room0.

Setting	#Room0							
	ATE↓	Depth L1↓	Precision↑	Recall ↑	F1↑	PSNR↑	SSIM↑	LPIPS↓
w/o add	✗	✗	✗	✗	✗	✗	✗	✗
w/o delete	0.58	1.68	53.55	49.32	51.35	31.22	0.967	0.094
w/ add & delete	**0.48**	**1.31**	**64.58**	**61.29**	**62.89**	**31.56**	**0.968**	**0.094**

Table 8. Ablation of the coarse-to-fine tracking strategy on Replica #Room0.

Setting	#Room0							
	ATE↓	Depth L1↓	Precision↑	Recall ↑	F1↑	PSNR↑	SSIM↑	LPIPS↓
Coarse	0.91	1.48	59.68	57.54	56.50	29.13	0.954	0.120
Fine	0.49	1.39	62.61	59.18	61.29	30.84	0.964	0.096
Coarse-to-fine	**0.48**	**1.31**	**64.58**	**61.29**	**62.89**	**31.56**	**0.968**	**0.094**

Effect of coarse-to-fine tracking. According to the results in Tab. 8, the proposed coarse-to-fine tracking strategy performs best in all tracking, mapping, and rendering metrics. Compared with fine tracking, the coarse-to-fine tracking strategy significantly improves the performance by 0.01 in tracking ATE, 2.11 in Recall, and 0.72 in PSNR. Although the fine strategy surpasses the coarse strategy in precision, it suffers from the artifacts and noise in the reconstructed scene, leading to a fluctuation optimization. The coarse-to-fine strategy effectively avoids noise reconstruction and improves accuracy and robustness.

4.6. Efficiency-to-accuracy trade-off.

3DGS-based SLAM trade-off focuses not only on the efficiency of tracking and mapping but also emphasizes high-quality ultra-real-time rendering. As shown in Fig. 7b, GS-SLAM achieves ≈ 400 FPS ultra-fast speed and highest PSNR in map rendering. At the same time, our method remains a competitive system FPS and lowest tracking ATE in Fig. 7a. Moreover, GS-SLAM shows great potential in memory reduction in Tab. 5, and comparable in mesh reconstruction. Note that baselines directly use 3DGS in Fig. 7, resulting in inferior performances.

5. Conclusion and Limitations

We introduced GS-SLAM, a novel dense visual SLAM method leveraging 3D Gaussian Splatting for efficient mapping and accurate camera pose estimation, striking a better speed-accuracy balance. However, its reliance on high-quality depth data may limit performance in certain conditions. Additionally, the approach's high memory requirements for large scenes suggest future improvements could focus on optimizing memory use, potentially via techniques such as quantization and clustering. We believe GS-SLAM has the potential to extend to larger scale with some improvements and will explore this in future work.

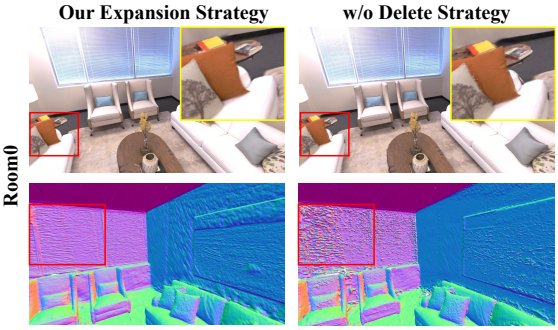
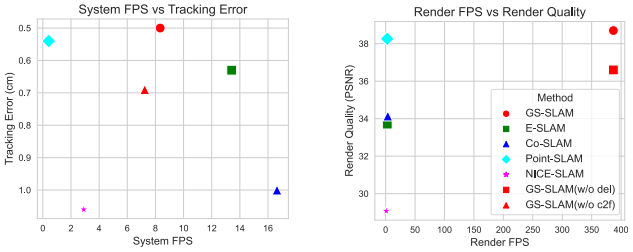


Figure 6. Rendering and mesh visualization of the adaptive 3D Gaussian expansion ablation on Replica #Room0.



(a) Tracking performance (b) Render performance

Figure 7. Bi-criteria figure of tracking/render performance and system FPS on Replica #Office0.

Acknowledgements. This work is supported by the Shanghai AI Laboratory, National Key R&D Program of China (2022ZD0160101), the National Natural Science Foundation of China (62376222), Young Elite Scientists Sponsorship Program by CAST (2023QNRC001) and the Early Career Scheme of the Research Grants Council (RGC) of the Hong Kong SAR under grant No. 26202321.

References

[1] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J. Davison. Codeslam - learning a compact, optimisable representation for dense visual slam. *CVPR*, pages 2560–2568, 2018. 2

[2] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *TIV*, 2:194–220, 2017. 6

[3] Zilong Chen, Feng Wang, and Huaping Liu. Text-to-3d using gaussian splatting. *ArXiv*, abs/2309.16585, 2023. 2

[4] Brian Curless and Marc Levoy. Volumetric method for building complex models from range images. In *SIGGRAPH*. ACM, 1996. 4

[5] Parth Rajesh Desai, Pooja Nikhil Desai, Komal Deepak Ajmera, and Khushbu Mehta. A review paper on oculus rift-a virtual reality headset. *ArXiv*, abs/1408.1173, 2014. 6

[6] Hugh F. Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *RAM*, 13:99–110, 2006. 1

[7] Christian Häne, Christopher Zach, Jongwoo Lim, Ananth Ranganathan, and Marc Pollefeys. Stereo depth map fusion for robot navigation. *IROS*, pages 1618–1625, 2011. 6

[8] Caner Hazirbas, Andreas Wiedemann, Robert Maier, Laura Leal-Taixé, and Daniel Cremers. Tum rgb-d scribble-based segmentation benchmark. https://github.com/tum-vision/rgbd_scribble_benchmark, 2018. 6

[9] Jiahui Huang, Shi-Sheng Huang, Haoxuan Song, and Shi-Min Hu. Di-fusion: Online implicit 3d reconstruction with deep priors. In *CVPR*, pages 8932–8941, 2021. 6

[10] Xudong Jiang, Lifeng Zhu, Jia Liu, and Aiguo Song. A slam-based 6dof controller with smooth auto-calibration for virtual reality. *TVC*, 39:3873 – 3886, 2022. 1

[11] Mohammad Mahdi Johari, Camilla Carta, and Franccois Fleuret. Eslam: Efficient dense slam system based on hybrid representation of signed distance fields. *CVPR*, 2023. 1, 2, 5, 6, 7, 8, 4

[12] Olaf Kähler, Victor Adrian Prisacariu, Julien P. C. Valentin, and David William Murray. Hierarchical voxel block hashing for efficient integration of depth images. *RAL*, 1:192–197, 2016. 1

[13] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *TOG*, 42(4), 2023. 2, 3, 7, 4

[14] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *ECCV*, 2022. 2

[15] Leonid Keselman and Martial Hebert. Flexible techniques for differentiable rendering with 3d gaussians. *arXiv preprint arXiv:2308.14737*, 2023. 2

[16] Georg S. W. Klein and David William Murray. Parallel tracking and mapping on a camera phone. *ISMAR*, pages 83–86, 2009. 2

[17] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. *ArXiv*, abs/2308.09713, 2023. 2

[18] Robert Maier, Raphael Schaller, and Daniel Cremers. Efficient online surface correction for real-time large-scale 3d reconstruction. *ArXiv*, abs/1709.03763, 2017. 1

[19] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[20] Raul Mur-Artal and Juan D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *TRO*, 33:1255–1262, 2016. 1, 6

[21] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew William Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. *ISMAR*, pages 127–136, 2011. 1

[22] Richard A. Newcombe, S. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. *ICCV*, pages 2320–2327, 2011. 2

[23] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *TOG*, 32:1 – 11, 2013. 1

[24] Delin Qu, Chi Yan, Dong Wang, Jie Yin, Dan Xu, Bin Zhao, and Xuelong Li. Implicit event-rgbd neural slam. *CVPR*, 2024. 1

[25] Gerhard Reitmayr, Tobias Langlotz, Daniel Wagner, Alessandro Mulloni, Gerhard Schall, Dieter Schmalstieg, and Qi Pan. Simultaneous localization and mapping for augmented reality. *ISUVR*, pages 5–8, 2010. 1

[26] Fabio Ruetz, Emili Hernández, Mark Pfeiffer, Helen Oleynikova, Mark Cox, Thomas Lowe, and Paulo Vinicius Koerich Borges. Ovpc mesh: 3d free-space representation for local ground vehicle navigation. *ICRA*, pages 8648–8654, 2018. 1

[27] Erik Sandström, Yue Li, Luc Van Gool, and Martin R. Oswald. Point-slam: Dense neural point cloud-based slam. In *ICCV*, 2023. 2, 5, 6, 7, 4

[28] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 3

[29] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. *CVPR*, pages 134–144, 2019. 2

[30] Thomas Schops, Torsten Sattler, and Marc Pollefeys. BAD SLAM: Bundle adjusted direct RGB-D SLAM. In *CVPR*, 2019. 6

[31] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Yuheng Ren, Shobhit Verma, Anton Clarkson, Ming Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke Malte Strasdat, Renzo De Nardi, Michael Goesele, S. Lovegrove, and Richard A. Newcombe. The replica dataset: A digital replica of indoor spaces. *ArXiv*, abs/1906.05797, 2019. 5, 3, 4

[32] J. Stückler and Sven Behnke. Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *JVCIR*, 25: 137–147, 2014. 1

[33] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IROS*. IEEE/RSJ, 2012. 5, 6

[34] Edgar Sucar, Kentaro Wada, and Andrew J. Davison. Nodeslam: Neural object descriptors for multi-view shape reconstruction. *3DV*, pages 949–958, 2020. 2

[35] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison. imap: Implicit mapping and positioning in real-time. *ICCV*, 2021. 1, 2, 6

[36] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *ArXiv*, abs/2309.16653, 2023. 2, 5

[37] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. In *NIPS*, 2021. 2

[38] Andreas Langeland Teigen, Yeonsoo Park, Annette Stahl, and Rudolf Mester. Rgb-d mapping and tracking in a plenoxel radiance field. *ArXiv*, abs/2307.03404, 2023. 2

[39] Charalambos Theodorou, Vladan Velisavljevic, Vladimir Dyo, and Fredi Nonyelu. Visual slam algorithms and their application for ar, mapping, localization and wayfinding. *Array*, 15:100222, 2022. 1

[40] Angtian Wang, Peng Wang, Jian Sun, Adam Kortylewski, and Alan Yuille. Voge: a differentiable volume renderer using gaussian ellipsoids for analysis-by-synthesis. *arXiv preprint arXiv:2205.15401*, 2022. 2

[41] Hengyi Wang, Jingwen Wang, and Lourdes de Agapito. Co-slam: Joint coordinate and sparse parametric encodings for neural real-time slam. *CVPR*, 2023. 1, 5, 6, 8, 2, 4

[42] Kaixuan Wang, Fei Gao, and Shaojie Shen. Real-time scalable dense surfel mapping. *ICRA*, pages 6919–6925, 2019. 1

[43] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 13(4):600–612, 2004. 5

[44] Thomas Whelan, Michael Kaess, Maurice F. Fallon, Hordur Johannsson, John J. Leonard, and John B. McDonald. Kintinuous: Spatially extended kinectfusion. In *AAAI*, 2012. 2

[45] Thomas Whelan, John McDonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J. Leonard. Kintinuous: Spatially extended kinectfusion. In *RSS Workshop on RGB-D*, 2012. 6

[46] Thomas Whelan, Stefan Leutenegger, Renato Salas-Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. In *RSS*, 2015. 1, 6

[47] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. *ArXiv*, abs/2310.08528, 2023. 2

[48] Xingrui Yang, Hai Li, Hongjia Zhai, Yuhang Ming, Yuqian Liu, and Guofeng Zhang. Vox-fusion: Dense tracking and mapping with voxel-based neural implicit representation. *ISMAR*, pages 499–507, 2022. 2, 5, 6, 7, 8

[49] Ziyi Yang, Xinyu Gao, Wenming Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *ArXiv*, abs/2309.13101, 2023. 2

[50] Zeyu Yang, Hongye Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. *ArXiv*, abs/2310.10642, 2023. 2

[51] Taoran Yi, Jiemin Fang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. Gaussiandreamer: Fast generation from text to 3d gaussian splatting with point cloud priors. *ArXiv*, abs/2310.08529, 2023. 2

[52] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, pages 586–595, 2018. 5

[53] Youmin Zhang, Fabio Tosi, Stefano Mattoccia, and Matteo Poggi. Go-slam: Global optimization for consistent 3d instant reconstruction. In *ICCV*, 2023. 2

[54] Shuaifeng Zhi, Michael Bloesch, Stefan Leutenegger, and Andrew J. Davison. Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations. *CVPR*, pages 11768–11777, 2019. 2

[55] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. *CVPR*, 2021. 1, 2, 5, 6, 7, 8

[56] Wojciech Zielonka, Timur M. Bagautdinov, Shunsuke Saito, Michael Zollhofer, Justus Thies, and Javier Romero. Drivable 3d gaussian avatars. 2023. 2

[57] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus H. Gross. Ewa volume splatting. *Proceedings Visualization, 2001. VIS '01.*, pages 29–538, 2001. 3