

High-Fidelity SLAM Using Gaussian Splatting with Rendering-Guided Densification and Regularized Optimization

Shuo Sun¹, Malcolm Mielle², Achim J. Lilienthal^{1,3}, and Martin Magnusson¹

Abstract—We propose a dense RGBD SLAM system based on 3D Gaussian Splatting that provides metrically accurate pose tracking and visually realistic reconstruction. To this end, we first propose a Gaussian densification strategy based on the rendering loss to map unobserved areas and refine reobserved areas. Second, we introduce extra regularization parameters to alleviate the “forgetting” problem during continuous mapping, where parameters tend to overfit the latest frame and result in decreasing rendering quality for previous frames. Both mapping and tracking are performed with Gaussian parameters by minimizing re-rendering loss in a differentiable way. Compared to recent neural and concurrently developed Gaussian splatting RGBD SLAM baselines, our method achieves state-of-the-art results on the synthetic dataset *Replica* and competitive results on the real-world dataset *TUM*. The code is released on <https://github.com/ljjtyjr/HF-SLAM>.

I. INTRODUCTION

Dense visual SLAM with RGBD inputs is essential for many downstream tasks in mobile robots, AR/VR and robot manipulation. Traditional SLAM systems have focused mainly on camera tracking [1] and geometric surface reconstruction [2]. However, surface appearance reconstruction, which contains rich information for scene understanding, is often lacking in traditional SLAM systems. Motivated by the success of *NeRF* [3] on novel view synthesis, recent RGBD SLAM methods use volumetric structures [4, 5, 6, 7] or points [8] as the map representation. Combined with neural networks as decoders, these methods can achieve both appearance and geometry reconstruction. However, limited by the computationally expensive sampling along rays in the optimization of *NeRF*, the above mentioned methods cannot render full-resolution images with high quality (commonly only 0.5% pixels are used for mapping [5], leading to low quality image rendering; see Fig. 3 and Tab. I).

The recent work *3D Gaussian Splatting* (3DGS) [9] provides a more efficient way for novel view synthesis. Instead of expensive sampling along the ray, 3DGS relies on rasterization for rendering, which accelerates the mapping process so as to handle full-resolution images. The original 3DGS needs prior camera poses, which are often estimated by *Structure-from-Motion* [10]. In this work, we extend 3DGS to the case of online tracking and mapping, removing the need

for prior camera poses. Our method processes sequential RGBD frame inputs, simultaneously optimizing Gaussian parameters and estimating camera poses. Though there are some concurrent works combining 3DGS with SLAM [11, 12], we show our method achieves better or competitive rendering and tracking results compared to recent baselines.

Our first contribution is the introduction of a novel densification strategy based on rendering. We directly densify the map utilizing rendering loss, enabling us to effectively map unobserved areas and enhance the rendering quality of reobserved regions.

Our second contribution is regularized optimization during mapping. Continuous mapping with 3DGS suffers from the “forgetting” problem, which means that Gaussian parameters tend to overfit to the latest frame and lead to decreased reconstruction quality for previous frames (see Fig. 2). To alleviate the forgetting problem, we introduce extra parameters to supervise the learning process, and our experiments show that our method can preserve the rendering quality of previously visited areas.

We demonstrate state-of-the-art reconstruction quality and tracking accuracy on *Replica* dataset and competitive results on the real-world dataset *TUM-RGBD*. We show the relative effectiveness of the two main contributions in the ablation study. We also find that continuous mapping can reconstruct high-quality images even if when faced with motion blur, which may inspire work on 3DGS reconstruction with blurry inputs.

II. RELATED WORK

A. Visual SLAM

Conventional visual SLAM methods are divided into direct and indirect methods. Indirect methods (e.g., *ORB-SLAM* [1]) first detect points of interest and attach feature descriptors. Tracking is conducted by feature matching and minimizing the re-projection error across frames. Direct methods (e.g., *DSO* [13]) consume raw pixels and construct photometric loss for tracking. These conventional visual SLAM methods focus more on tracking instead of scene reconstruction. Since *NeRF* [14] has shown powerful scene reconstruction abilities, recent work applies neural representation in SLAM to produce more complete and photorealistic scene reconstruction. For example, *iMAP* [7] uses a neural network to represent the scene; *NICE-SLAM* [4] and later work [5, 6] combine neural networks and voxel grids to improve reconstruction quality. Thanks to the generalization capabilities of neural networks, these neural representations have more powerful abilities to fill holes and construct

*This work has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 101017274 (DARKO).

¹AASS research center, Örebro University, Sweden. {shuo.sun, achim.lilienthal, martin.magnusson}@oru.se

²Independent researcher. malcolm.mielle@protonmail.com

³Technical University of Munich, Chair: Perception for Intelligent Systems. achim.j.lilienthal@tum.de

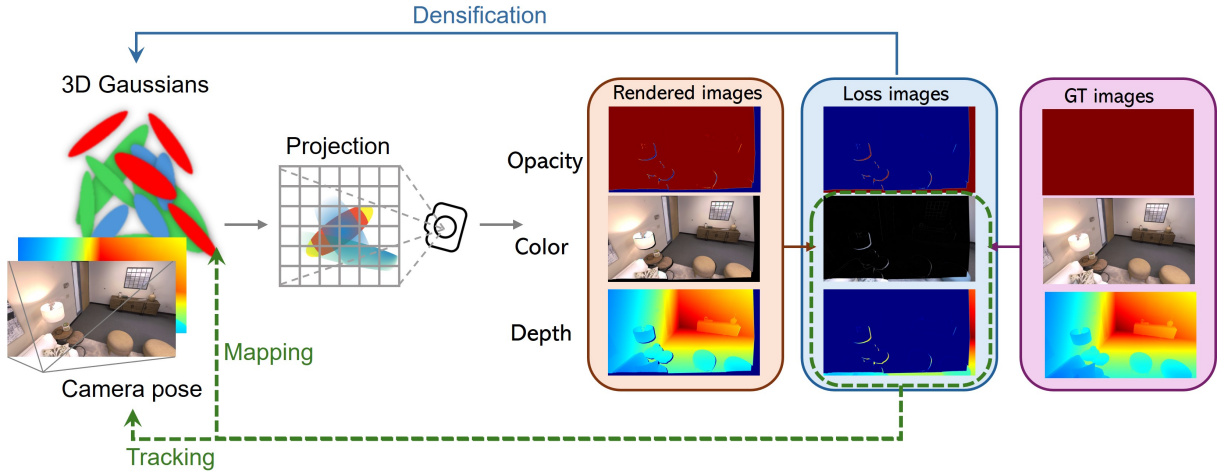


Fig. 1: Overview of the method. Our method takes RGBD frames as inputs. During mapping, when given a posed RGBD frame, we first render the opacity image, color image and depth image. Then we compare them with the ground truth to densify the existed map. During tracking, we minimize the color and depth re-rendering loss to optimize the camera pose.

more complete scenes compared to explicit representations. Although neural representations achieve good results in geometric scene reconstruction, their performance in recovering high-fidelity scenes is not satisfactory enough. In this work, we use parameterized Gaussians [9] as the map representation and achieve a more photorealistic reconstruction. Concurrent with our work, some SLAM systems are also based on Gaussian Splatting: *MonoGS* [15] expands the map by random sampling; *Gaussian-SLAM* [16] divides the scene into many submaps; *Photo-SLAM* [17] conducts tracking by minimizing the reprojection error; *SplaTAM* [11] and *GS-SLAM* [12] densify the map only considering the unobserved regions (more information can be found in [18]). Our method expands the map by considering filling holes in unobserved areas and refining reobserved areas, which can effectively improve the reconstruction quality. In addition, our regularized optimization can effectively alleviate the forgetting problem during mapping and preserve details of previously visited areas.

B. Photo-realistic reconstruction

NeRF [14] opens a convenient way for photo-realistic reconstruction from only 2D images. Based on neural networks and volumetric differentiable rendering, NeRF can render novel photorealistic views that were not observed in the inputs. Many follow-up works improve on the original NeRF by accelerating training speed [19], improving rendering quality [20], etc. Some works [21] also reveal that neural networks are not necessary for novel view synthesis. As opposed to Gaussian splatting [9], however, all the above-mentioned methods adopt expensive ray-marching when calculating one pixel, resulting in low rendering speed (lower than 15 fps). Gaussian Splatting uses parameterized Gaussians (as introduced in Sec. III) as rendering primitives. Instead of expensive ray-marching, Gaussian Splatting renders the image by tile-based rasterization, which can achieve much faster rendering speed (around 100 fps). The original Gaussian

Splatting needs known camera poses (commonly achieved by *SfM* [10]) and optimizes Gaussian parameters offline. In this work, we consider sequential RGBD frame inputs and optimize parameters online, which releases the requirement of known camera poses and brings the possibility of online applications such as exploration.

III. PRELIMINARY: GAUSSIAN SPLATTING RENDERING

We use 3D Gaussians as the map representation primitives for the color and depth image rendering. As described in [9], each Gaussian consists of parameters

$$\Theta_i := \{\mu_i, \mathbf{s}_i, \mathbf{r}_i, \mathbf{c}_i, o_i\}, \quad (1)$$

with Gaussian means $\mu_i \in \mathbb{R}^3$, scales $\mathbf{s}_i \in \mathbb{R}^3$, rotation quaternion $\mathbf{r}_i \in \mathbb{R}^4$, RGB color $\mathbf{c}_i \in \mathbb{R}^3$, and opacity $o_i \in \mathbb{R}$. All of these parameters are optimizable during training.

When rendering the image at some camera pose T , the 3D Gaussians in the field of view are projected onto the image plane as 2D Gaussians with mean μ'_i and covariance Σ'_i . To render the color C and depth D of the selected pixel x on the image plane, we blend N ordered (by depth) 2D Gaussians overlapping the pixel:

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (2)$$

$$D = \sum_{i \in N} z_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (3)$$

with

$$\alpha_i = o_i \cdot \exp\left[-\frac{1}{2}(\mathbf{x} - \mu')(\Sigma')^{-1}(\mathbf{x} - \mu')\right]. \quad (4)$$

Though Eq. (2) is the same as the volume rendering used in the classical NeRF [14], rasterization-based rendering is much more efficient. Because rasterization allows us to project and render objects in the space directly instead of

expensive sampling along the ray used in a full volumetric representation.

IV. METHODS

Figure 1 shows an overview of our method. Given RGBD frames and estimated camera poses, we update the map by comparing the rendered images and the ground truth to identify unobserved regions and areas requiring refinement. Regularization terms are incorporated into the optimization process to mitigate the issue of forgetting during mapping (Sec. IV-A). We track the camera pose in the Gaussian map by minimizing color and depth re-rendering loss (Sec. IV-B).

A. Mapping

Gaussian Densification. Differently from initializing the Gaussian parameters with all images as in the original 3D Gaussian Splatting [9], we process sequential RGBD frame inputs. For the latest RGBD input frame with an estimated pose (Sec. IV-B), we need to add new Gaussians to the map, either to *complete unobserved regions* or to *improve the rendering quality of previously mapped regions*.

First, for the unobserved regions, we need to add new Gaussians to fill holes. We judge holes by rendering opacity. That is, when receiving a new frame, we render the *opacity image* at the estimated pose by

$$O = \sum_{i \in N} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (5)$$

In the opacity image, if the rendered value is below some predefined opacity threshold τ_{opa} (i.e., $O(x, y) < \tau_{\text{opa}}$), we need to densify the corresponding regions to fill holes.

Second, due to e.g. occlusion and illumination changes, the rendered image is often view-dependent. In addition, the optimized Gaussian parameters may get trapped into local minima to fit certain frames. To account for such view-dependent errors, we propose a kind of *rendering error guided* densification. When giving the estimated pose, we render the color image \hat{C} and the depth image \hat{D} at this pose; the current reference color image and the depth image are C and D . We add new Gaussians when the error between the rendered image and the input image is large, where the color error and depth error are $E_{\text{color}}(x, y) = |\hat{C}(x, y) - C(x, y)|$ and $E_{\text{depth}}(x, y) = |\hat{D}(x, y) - D(x, y)|$ respectively. We will densify those regions where $E_{\text{color}}(x, y) > \tau_{\text{color}}$ or $\frac{E_{\text{depth}}(x, y)}{D(x, y)} > \tau_{\text{depth}}$, where τ_{color} and τ_{depth} are predefined thresholds. (We divide by $D(x, y)$ when comparing to τ_{depth} to compensate for varying scales.)

In summary, we densify the regions where $(O(x, y) < \tau_{\text{opa}}) \mid (E_{\text{color}}(x, y) > \tau_{\text{color}}) \mid (\frac{E_{\text{depth}}(x, y)}{D(x, y)} > \tau_{\text{depth}})$. With depth images, we project corresponding depths directly. The experiment reported in Tab. IV shows that our densification strategy can significantly improve map quality.

Continuous Mapping. We optimize Gaussian parameters by minimizing the loss ℓ_1 between the rendered color and

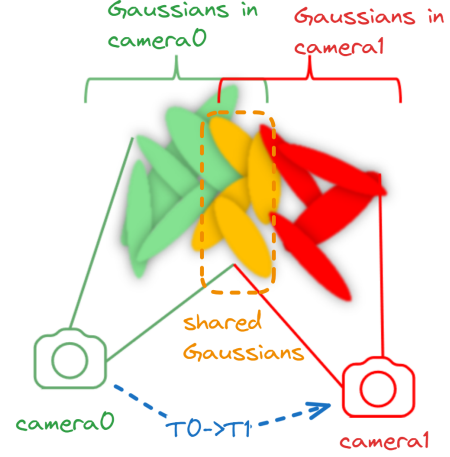


Fig. 2: Illustration of the *forgetting* problem in the context of continual mapping based on Gaussians. The Gaussians colored by yellow are shared by camera0 and camera1. However, these Gaussians tend to be optimized to overfit the latest frame camera1, resulting in drop of reconstruction quality for previous frames.

depth image and the input frames. In addition, following the original implementation in [9], we add an extra SSIM term.

$$\mathcal{L}_{\text{mapping}} = \lambda_{\text{color}} |\hat{C} - C| + \lambda_{\text{depth}} |\hat{D} - D| + \lambda_{\text{SSIM}} \text{SSIM}(\hat{C}, C) \quad (6)$$

As a continuous learning problem, the incremental mapping in our work also suffers from the *forgetting* problem [22]. That is, the same Gaussian might be “seen” in multiple camera poses, but the Gaussian parameters tend to be optimized (overfitted) to fit the latest frame, resulting in a performance drop for previous frames (as demonstrated in Fig. 2). To alleviate the overfitting problem, prior work [4, 5, 6] maintains a keyframe buffer. When optimizing the latest frame, some previous keyframes are selected from the buffer based on some criteria such as overlap ratio with the current frame. The selected keyframes will then be optimized together with the current frame. In the original 3D Gaussian Splatting, all frames are queued and picked randomly for many iterations to optimize Gaussian parameters.

We can, of course, pick many previous frames whenever we optimize the current frame; however, optimizing with multiple frames will lead to more time consumption, which is not suitable for online continual mapping. To prevent Gaussian parameters from forgetting, we add an extra regularization term \mathcal{L}_{reg} when mapping:

$$\mathcal{L}_{\text{reg}} = \sum_{i \in G} \Omega_i^s |s_i^t - s_i^*| + \Omega_i^c |r_i^t - r_i^*| + \Omega_i^d |z_i^t - z_i^*|, \quad (7)$$

where G refers to the set of Gaussians involved in the optimization for the current frame; s_i^t, r_i^t, z_i^t are scales, colors and depth to be optimized at the step t ; s_i^*, r_i^*, z_i^* are current parameter values, that is, values achieved after step $t-1$. Ω 's are importance weights for different parameters. In this way, we only need to maintain a small keyframe buffer but achieve better reconstruction quality.

To calculate importance weights, for each Gaussian represented by Eq. (1), we additionally add the following parameters:

$$\Delta_i := \{N_i^{\text{seen}}, \Sigma_i^s, \Sigma_i^c, \Sigma_i^d\} \quad (8)$$

where N_i^{seen} records the ‘‘seen times’’ of each Gaussian; Σ are the sum of gradients of $\mathcal{L}_{\text{mapping}}$ with respect to parameters, representing the sensitivity to changes. After training each frame, we update the importance weights by averaging the gradients of mapping loss with respect to the parameters. Taking the i th scale parameter s_i as an example, after training with N_i^{seen} frames, its importance weight Ω_i^s is:

$$\Omega_i^s = \frac{\Sigma_i^s}{N_i^{\text{seen}}} = \frac{1}{N_i^{\text{seen}}} \sum_j^{N_i^{\text{seen}}} \left(\frac{\partial \mathcal{L}_{\text{mapping}}}{\partial s_i} \right)_j \quad (9)$$

The final loss function used in the mapping is

$$\mathcal{L}'_{\text{mapping}} = \mathcal{L}_{\text{mapping}} + \mathcal{L}_{\text{reg}}. \quad (10)$$

B. Tracking

We conduct camera tracking in a *render-and-compare* way similar to that of *iNeRF* [3] for every input frame. The camera pose is parameterized by $T = (t, q)$, including the 3D translation t and the rotation quaternion q . When tracking the new frame, we initialize the camera pose assuming constant camera velocity. Since the color image can be affected by light, we decouple lightness from the image during tracking, similar to [23]. Specifically, let ρ denote the operation of converting the image from the RGB space to the LAB space, and we discard the light channel (L channel) when comparing the two images.

$$\mathcal{L}_{\text{tracking}} = \lambda_{\text{color}} |\rho(\hat{C}) - \rho(C)| + \lambda_{\text{depth}} |\hat{D} - D| \quad (11)$$

V. EXPERIMENTS

In this section, we demonstrate through quantitative and qualitative experiments that our method achieves better reconstruction results than the current state-of-the-art neural implicit SLAM methods and concurrent work using Gaussian splatting. Furthermore, the ablation studies of Sec. V-D show that our Gaussian densification strategy and regularization terms can effectively improve reconstruction quality.

A. Experiment Setup

Evaluation Metrics: We use ATE (Absolute Trajectory Error [cm]) to measure the tracking performance, which aligns the estimated trajectory and the ground truth trajectory and then computes the root mean square error. To assess the quality of color image rendering, we employ peak signal-to-noise ratio (PSNR [dB]), structural similarity index (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). PSNR evaluates pixel-wise RGB discrepancies, SSIM gauges structural resemblance, while LPIPS quantifies perceptual image patch similarity through neural network-based analysis. Since one can construct the mesh via *TSGF-Fusion* [24] with rendered depth along the estimated trajectory, we use metric Depth L1 [cm] to evaluate the geometric reconstruction

as in [8]. All metrics are calculated by rendering full-resolution images along the estimated trajectory after reading all frames.

Datasets: We conduct our experiments on two datasets: *Replica* [25] (a synthetic dataset) and *TUM-RGBD* [26] (a real-world dataset); the synthetic dataset provides noiseless RGB and depth images while real-world datasets provide noisy inputs.

Baselines: We evaluate our results against both state-of-the-art SLAM methods using neural implicit representation—*NICE-SLAM* [4], *ESLAM* [5], *Point-SLAM* [8]—and 3D Gaussian Splatting—*SplaTAM* [11] and *GS-SLAM* [12].

Implementation Details: We test all the methods on one V100 GPU with 24 GB memory. The hyper-parameters used in our method are as follows: $\tau_{\text{color}} = 0.6$, $\tau_{\text{depth}} = 0.1$, $\lambda_{\text{color}} = 0.5$, $\lambda_{\text{depth}} = 1.0$, $\lambda_{\text{SSIM}} = 0.2$.

B. Reconstruction Performance

Quantitative Results. Compared to neural RGBD SLAM and more recent Gaussian Splatting SLAM, our method shows better rendering performance across both datasets, on real-world and synthetic data, as shown in Tab. I and Tab. II. (It should be noted that since *SplaTAM* and *GS-SLAM* did not report rendering results on this dataset in their paper, we did not include them in the table.)

Our method achieves the **best** results in color image rendering, but the **second** results in depth rendering in the *Replica* dataset. To explain why our method does not consistently outperform *Point-SLAM* on depth rendering, one must note that, in neural RGBD SLAM, appearance and geometry are often represented and optimized separately. In *Point-SLAM*, even though the same point cloud is used as primitives, each point has a separate color feature descriptor and a geometric feature descriptor; also, two distinct neural networks are responsible for decoding color and depth. Hence, when optimizing the map parameters, color rendering and depth rendering do not affect each other. However, since our method uses Gaussian Splatting, the same Gaussian parameters $\{\mu, s, r, o\}$ are optimized for both color and depth rendering, which can impact the depth estimation.

Qualitative Results. Rendered images are shown in Figure 3. From the rendered results, *Point-SLAM* cannot recover fine details and *SplaTAM* generates floaters in the rendered image, which can be observed in the second and fourth row in Fig. 3. On the other hand, our method generates clean images with high fidelity.

C. Tracking Performance

On the *Replica* dataset, thanks to the improved reconstruction capabilities of our method shown in the previous section, we achieve the best tracking performance (see ATE RMSE in Tab. I). However, on *TUM-RGBD* (Tab. III), our method does not always perform the best. We hypothesise that, compared to methods using neural networks, our method is more sensitive to ‘‘noise’’, such as motion blur and exposure changes present in real data.

TABLE I: The SLAM performance on the Replica [25] dataset. The best results are highlighted by **first**, **second**, and **third**. \uparrow means larger is better while \downarrow means smaller is better. Our method achieves the best results in most metrics.

Method	Primitives	Metric	Room0	Room1	Room2	office0	office1	office2	office3	office4	Avg.
<i>NICE-SLAM</i> [4]	Neural + Voxels	PSNR [dB] \uparrow	22.12	22.47	24.52	29.07	30.34	19.66	22.23	24.94	24.42
		SSIM \uparrow	0.69	0.76	0.81	0.87	0.89	0.80	0.80	0.86	0.81
	Depth L1 [cm] \downarrow	LPIPS \downarrow	0.33	0.27	0.21	0.23	0.18	0.23	0.21	0.20	0.23
		ATE RMSE [cm] \downarrow	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.13	1.06
		PSNR [dB] \uparrow	1.81	1.44	2.04	1.39	1.76	8.33	4.99	2.01	2.97
<i>ESLAM</i> [5]	Neural + Feature Plane	PSNR [dB] \uparrow	25.25	25.31	28.09	30.33	27.04	27.99	29.27	29.15	27.80
		SSIM \uparrow	0.87	0.25	0.93	0.93	0.91	0.94	0.95	0.95	0.92
	Depth L1 [cm] \downarrow	LPIPS \downarrow	0.32	0.30	0.25	0.21	0.25	0.24	0.19	0.21	0.25
		ATE RMSE [cm] \downarrow	0.71	0.70	0.52	0.57	0.55	0.58	0.72	0.63	0.63
		PSNR [dB] \uparrow	0.97	1.07	1.28	0.86	1.26	1.71	1.43	1.06	1.18
<i>Point-SLAM</i> [8]	Neural + Point Cloud	PSNR [dB] \uparrow	32.40	34.08	35.50	38.26	39.16	33.99	33.48	33.49	35.17
		SSIM \uparrow	0.97	0.98	0.98	0.98	0.99	0.96	0.96	0.98	0.97
	Depth L1 [cm] \downarrow	LPIPS \downarrow	0.11	0.12	0.11	0.10	0.12	0.16	0.13	0.14	0.12
		ATE RMSE [cm] \downarrow	0.61	0.41	0.37	0.38	0.48	0.54	0.69	0.72	0.52
		PSNR [dB] \uparrow	0.53	0.22	0.46	0.30	0.57	0.49	0.51	0.46	0.44
<i>GS-SLAM</i> [12]	Parameterized Gaussians	PSNR [dB] \uparrow	31.56	32.86	32.59	38.70	41.17	32.36	32.03	32.92	34.27
		SSIM \uparrow	0.96	0.97	0.97	0.98	0.99	0.97	0.97	0.96	0.97
	Depth L1 [cm] \downarrow	LPIPS \downarrow	0.09	0.07	0.09	0.05	0.03	0.09	0.11	0.11	0.08
		ATE RMSE [cm] \downarrow	0.48	0.53	0.33	0.52	0.41	0.59	0.46	0.70	0.50
		PSNR [dB] \uparrow	1.31	0.82	1.26	0.81	0.96	1.41	1.53	1.08	1.16
<i>SplaTAM</i> [11]	Parameterized Gaussians	PSNR [dB] \uparrow	32.86	33.89	35.25	38.26	39.17	31.97	29.70	31.81	34.11
		SSIM \uparrow	0.98	0.97	0.98	0.98	0.97	0.97	0.95	0.95	0.97
	Depth L1 [cm] \downarrow	LPIPS \downarrow	0.07	0.10	0.08	0.09	0.09	0.10	0.12	0.15	0.10
		ATE RMSE [cm] \downarrow	0.31	0.40	0.29	0.47	0.27	0.29	0.32	0.55	0.36
		PSNR [dB] \uparrow	-	-	-	-	-	-	-	-	-
<i>Ours</i>	Parameterized Gaussians	PSNR [dB] \uparrow	33.06	35.74	37.21	41.12	41.11	33.56	33.21	34.48	36.19
		SSIM \uparrow	0.98	0.98	0.99	0.99	0.99	0.98	0.97	0.98	0.98
	Depth L1 [cm] \downarrow	LPIPS \downarrow	0.05	0.05	0.04	0.03	0.03	0.07	0.08	0.08	0.05
		ATE RMSE [cm] \downarrow	0.19	0.34	0.16	0.21	0.26	0.23	0.21	0.38	0.25
		PSNR [dB] \uparrow	0.39	0.34	0.33	0.29	0.26	0.67	0.93	0.97	0.52

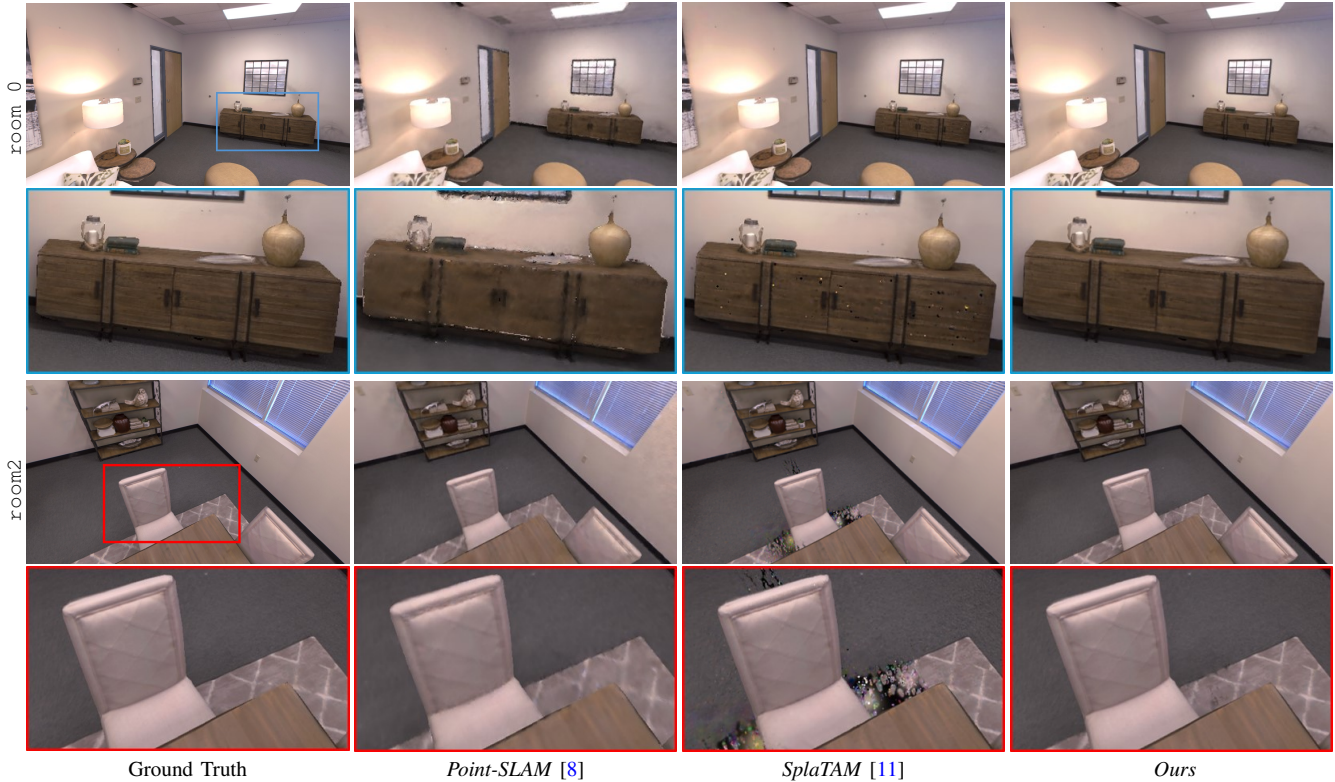


Fig. 3: The rendering results on the Replica dataset. The second and fourth rows are zoomed-in details of the colored squares. Compared to *Point-SLAM* [8], our method generates sharper results; compared to *SplaTAM* [11], our method does not have floaters.

TABLE II: The rendering performance on TUM-RGBD [26] dataset. The best results are highlighted by **first**, **second**, and **third**.

Method	Metrics	fr1/desk	fr1/desk2	fr1/room	fr3/off.
<i>NICE-SLAM</i> [4]	PSNR [dB] \uparrow	13.83	12.00	11.39	12.89
	SSIM \uparrow	0.57	0.51	0.37	0.55
	LPIPS \downarrow	0.48	0.52	0.63	0.50
<i>ESLAM</i> [5]	PSNR [dB] \uparrow	11.29	12.30	9.06	17.02
	SSIM \uparrow	0.67	0.63	0.93	0.46
	LPIPS \downarrow	0.36	0.42	0.19	0.65
<i>Point-SLAM</i> [8]	PSNR [dB] \uparrow	13.87	14.12	14.16	18.43
	SSIM \uparrow	0.63	0.59	0.65	0.75
	LPIPS \downarrow	0.54	0.57	0.55	0.45
<i>GS-SLAM</i>	-	-	-	-	-
<i>SplaTAM</i>	-	-	-	-	-
<i>Ours</i>	PSNR [dB] \uparrow	22.60	20.79	17.53	22.30
	SSIM \uparrow	0.91	0.85	0.72	0.89
	LPIPS \downarrow	0.15	0.22	0.32	0.16

TABLE III: The tracking performance on TUM-RGBD [26] dataset. The best results are highlighted by **first**, **second**, and **third**. The evaluation metric for the trajectory is the ATE [cm].

Method	fr1/desk	fr1/desk2	fr1/room	fr3/off.
<i>NICE-SLAM</i> [4]	4.26	4.99	34.49	3.87
<i>ESLAM</i> [5]	2.47	3.69	29.73	2.42
<i>Point-SLAM</i> [8]	4.34	4.54	30.92	3.48
<i>GS-SLAM</i> [12]	3.30	-	-	6.60
<i>SplaTAM</i> [11]	3.35	6.54	11.13	5.16
<i>Ours</i>	3.38	7.20	22.62	5.12

D. Ablation Study

In this section, we conduct ablation studies to demonstrate the effectiveness of the regularization terms and densification. For the regularization, we consistently render the first frame of Room0 in the Replica dataset at every step during the mapping. Figure 4 shows rendered results of frame0 after processing 350 frames. In the absence of regularization, the generated image often exhibits artifacts such as floaters at the edge. Conversely, our approach ensures a cleaner output, with reduced or eliminated artifacts.

We show the overall quantitative results on the Room0 in Replica dataset in Tab. IV. With regularization terms, we can achieve better reconstruction and tracking accuracy. Additionally, we test with and without color and depth rendering densification, and the result (see the second row in Tab. IV) reveals that the color and depth rendering densification can help improve rendering quality.

E. Runtime

Table V reports the runtime results on Room0 in Replica dataset running with a V100 GPU. For each iteration, *Point-*

TABLE IV: Quantitative results of ablation study. We conduct ablation studies on Room0 in Replica dataset. ‘‘Regularization’’ means whether to add regularization terms (Eq. (7)) during mapping; ‘‘Densification’’ means whether to densify regions based on color and depth rendering. The results show our proposed methods can improve reconstruction quality and tracking accuracy.

Regularization	Densification	PSNR [dB] \uparrow	SSIM \uparrow	LPIPS \downarrow	ATE [cm] \downarrow
\times	\checkmark	25.96	0.89	0.17	0.89
\checkmark	\times	25.58	0.85	0.21	10.02
\checkmark	\checkmark	35.74	0.98	0.05	0.34

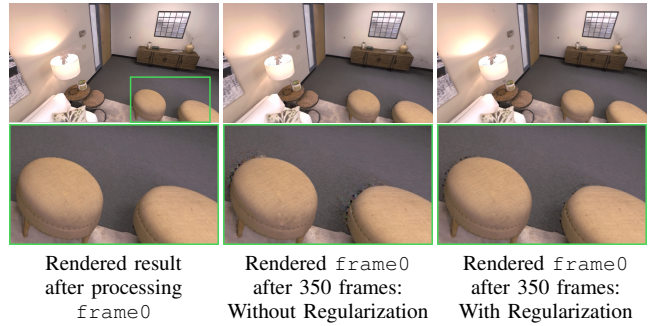


Fig. 4: The difference of without/with Regularization. The first column shows the rendered result after just mapping frame0. The second column illustrates the rendered image of frame0 after processing 350 frames without regularization, while the third column showcases the same after applying regularization. The second rows show the zoomed-in results in the green square, we can see that with regularization, the result can still maintain good quality, especially for edges.

TABLE V: Runtime results on Room0 in Replica dataset.

Method	Tracking [ms] / Iteration	Mapping [ms] / Iteration	Tracking [s] / Frame	Mapping [s] / Frame
<i>Point-SLAM</i> [8]	31.50	19.05	0.63	7.62
<i>SplaTAM</i> [11]	75.25	91.66	3.01	5.50
<i>Ours</i>	45.00	65.57	1.80	4.59

SLAM achieves the least time consumption because only a subset of pixels are used in the tracking and mapping (in *Point-SLAM*, 1000 pixels are used in the mapping and 200 pixels are used in the tracking¹). However, such a sparse sampling needs more iterations to converge, resulting in the longest time consumption for mapping; in addition, the sparse sampling decreases the rendering quality. *SplaTAM* and our method render full-resolution images, which requires slightly more time at each iteration, but converges faster. It should be noted that our method is faster than *SplaTAM* due to implementation differences, since we render color and depth in one single rasterization process, as opposed to two rasterization processes in *SplaTAM*.

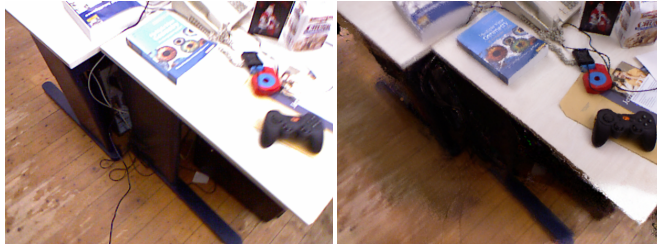
VI. SUMMARY AND FUTURE WORK

In this paper, we introduce a dense RGBD SLAM method based on the 3D Gaussian representation, which can render high-fidelity color and depth images. Our method uses rendering-based densification, which improves the quality of reobserved parts of the environment and allows the map to be extended. To alleviate the problem of ‘‘forgetting’’ (or overfitting) during mapping, we introduce a regularization loss in the optimization. Experiments show that our method consistently achieves higher-quality visual reconstruction than recent baselines, both those using neural representations and those using Gaussian representations. An interesting observation is that splatting-based methods, including ours, generally have lower tracking performance on real-world data (TUM-RGBD) compared to neural methods. However,

¹<https://github.com/eriksandroem/Point-SLAM>

our method still achieves the most high-fidelity visual reconstruction.

Despite the promising results we achieved in the experiments, our method has some limitations. As shown in Tab. III, our method does not always achieve good tracking results in real-world datasets due to motion blur and varying exposure. As shown in Fig. 5, though the rendered image is of higher quality since the mapping accumulates all previous frames, the blurry ground truth image makes tracking difficult. In addition, when evaluating the rendering quality, the fact that the ground truth image is blurry negatively affects the evaluation metric.



Ground truth image from TUM

The rendering image
PSNR: 17.16

Fig. 5: The ground truth image and the rendering image of TUM dataset. Our method maps by accumulating all previous frames, leading to more visually pleasing image quality. However, the ground truth image is poor in this case due to motion blur and varying exposure, which brings difficulty to tracking. Also, it negatively affects the image similarity evaluation metric: PSNR is only 17.16 in this example.

To improve tracking accuracy, it is popular to use bundle adjustment to optimize camera poses and the map simultaneously. However, we experimentally found that simply adopting bundle adjustment as in *BARF* [27] in our experiments will only make the result worse. Motivated by [28], we plan to investigate loop closure detection and optimize the trajectory with pose graph optimization in the future.

In addition to tracking accuracy, for future work, we will also investigate the following directions. First, how to improve efficiency to make it a “real-time” system. Second, motivated by [29, 30], we will investigate how to introduce high-level information, such as semantics, into the mapping.

REFERENCES

- [1] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. “ORB-SLAM: a versatile and accurate monocular SLAM system”. In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.
- [2] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. “Kinectfusion: Real-time dense surface mapping and tracking”. In: *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 2011, pp. 127–136.
- [3] Lin Yen-Chen, Pete Florence, Jonathan T Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. “IneRF: Inverting neural radiance fields for pose estimation”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1323–1330.
- [4] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. “Nice-slam: Neural implicit scalable encoding for slam”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12786–12796.
- [5] Mohammad Mahdi Johari, Camilla Carta, and François Fleuret. “ESLAM: Efficient Dense SLAM System Based on Hybrid Representation of Signed Distance Fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 17408–17419.
- [6] Hengyi Wang, Jingwen Wang, and Lourdes Agapito. “Co-SLAM: Joint Coordinate and Sparse Parametric Encodings for Neural Real-Time SLAM”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 13293–13302.
- [7] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. “iMAP: Implicit Mapping and Positioning in Real-Time”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 6229–6238.
- [8] Erik Sandström, Yue Li, Luc Van Gool, and Martin R Oswald. “Point-SLAM: Dense Neural Point Cloud-based SLAM”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 18433–18444.
- [9] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. “3D Gaussian Splatting for Real-Time Radiance Field Rendering”. In: *ACM Transactions on Graphics (ToG)* 42.4 (2023), pp. 1–14.
- [10] Johannes L Schonberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4104–4113.
- [11] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathan Luiten. “SplatAM: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM”. In: *arXiv preprint arXiv:2312.02126* (2023).
- [12] Chi Yan, Delin Qu, Dong Wang, Dan Xu, Zhigang Wang, Bin Zhao, and Xuelong Li. “GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting”. In: *arXiv preprint arXiv:2311.11700* (2023).
- [13] Jakob Engel, Vladlen Koltun, and Daniel Cremers. “Direct Sparse Odometry”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.3 (2017), pp. 611–625.
- [14] 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: *Communications of the ACM* 65.1 (2021), pp. 99–106.
- [15] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. “Gaussian Splatting SLAM”. In: *arXiv preprint arXiv:2312.06741* (2023).
- [16] Vladimir Yugay, Yue Li, Theo Gevers, and Martin R Oswald. “Gaussian-SLAM: Photo-realistic Dense SLAM with Gaussian Splatting”. In: *arXiv preprint arXiv:2312.10070* (2023).
- [17] Huajian Huang, Longwei Li, Hui Cheng, and Sai-Kit Yeung. “Photo-SLAM: Real-time Simultaneous Localization and Photorealistic Mapping for Monocular, Stereo, and RGB-D Cameras”. In: *arXiv preprint arXiv:2311.16728* (2023).
- [18] Guikun Chen and Wenguan Wang. “A Survey on 3D Gaussian Splatting”. In: *arXiv preprint arXiv:2401.03890* (2024).
- [19] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Transactions on Graphics (ToG)* 41.4 (2022), pp. 1–15.
- [20] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. “Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields”. In: *arXiv preprint arXiv:2304.06706* (2023).
- [21] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. “Plenoxels: Radiance Fields without Neural Networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5501–5510.
- [22] Michael McCloskey and Neal J Cohen. “Catastrophic interference in connectionist networks: The sequential learning problem”. In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.
- [23] Gu Wang, Fabian Manhardt, Jianzhun Shao, Xiangyang Ji, Nassir Navab, and Federico Tombari. “Self6D: Self-Supervised Monocular 6D Object Pose Estimation”. In: *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part 1* 16. Springer, 2020, pp. 108–125.
- [24] Brian Curless and Marc Levoy. “A volumetric method for building complex models from range images”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 303–312.
- [25] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. “The Replica Dataset: A Digital Replica of Indoor Spaces”. In: *arXiv preprint arXiv:1906.05797* (2019).
- [26] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. “A benchmark for the evaluation of RGB-D SLAM systems”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 573–580.
- [27] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. “BARF: Bundle-Adjusting Neural Radiance Fields”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5741–5751.
- [28] Lorenzo Liso, Erik Sandström, Vladimir Yugay, Luc Van Gool, and Martin R Oswald. “Loopy-SLAM: Dense Neural SLAM with Loop Closures”. In: *arXiv preprint arXiv:2402.09944* (2024).
- [29] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. “LERF: Language Embedded Radiance Fields”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 19729–19739.
- [30] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. “LangSplat: 3D Language Gaussian Splatting”. In: *arXiv preprint arXiv:2312.16084* (2023).