Gaussian Splatting SLAM

Hidenobu Matsuki[1*] Riku Murai[2*] Paul H. J. Kelly[2] Andrew J. Davison[1]

[1]Dyson Robotics Laboratory, Imperial College London
[2]Software Performance Optimisation Group, Imperial College London

Website: https://rmurai.co.uk/projects/GaussianSplattingSLAM/
Video: https://youtu.be/x604ghp9R_Q/

Figure 1. From a single monocular camera, we reconstruct a high fidelity *3D scene* live at 3fps. For every incoming RGB frame, 3D Gaussians are incrementally formed and optimised together with the camera poses. We show both the rasterised Gaussians (left) and Gaussians shaded to highlight the geometry (right). Notice the details and the complex material properties (e.g. transparency) captured. Thin structures such as wires are accurately represented by numerous small, elongated Gaussians, and transparent objects are effectively represented by placing the Gaussians along the rim. Our system significantly advances the fidelity a live monocular SLAM system can capture.

Abstract

We present the first application of 3D Gaussian Splatting to incremental 3D reconstruction using a single moving monocular or RGB-D camera. Our Simultaneous Localisation and Mapping (SLAM) method, which runs live at 3fps, utilises Gaussians as the only 3D representation, unifying the required representation for accurate, efficient tracking, mapping, and high-quality rendering.

Several innovations are required to continuously reconstruct 3D scenes with high fidelity from a live camera. First, to move beyond the original 3DGS algorithm, which requires accurate poses from an offline Structure from Motion (SfM) system, we formulate camera tracking for 3DGS using direct optimisation against the 3D Gaussians, and show that this enables fast and robust tracking with a wide basin of convergence. Second, by utilising the explicit nature of the Gaussians, we introduce geometric verification and regularisation to handle the ambiguities occurring in incremental 3D dense reconstruction. Finally, we introduce a full SLAM system which not only achieves state-of-the-art results in novel view synthesis and trajectory estimation, but also reconstruction of tiny and even transparent objects.

1. Introduction

A long-term goal of online reconstruction with a single moving camera is near-photorealistic fidelity, which will surely allow new levels of performance in many areas of Spatial AI and robotics as well as opening up a whole range of new applications. While we increasingly see the benefit of applying powerful pre-trained priors to 3D reconstruction, a key avenue for progress is still the invention and development of core 3D representations with advantageous properties. While many "layered" SLAM systems exist which combine multiple representations, the most in-

*Authors contributed equally to this work.

teresting advances are when a new unified dense representation can be used for all aspects of a system's operation: local representation of detail, large-scale geometric mapping and also camera tracking by direct alignment.

In this paper we present the first online visual SLAM system based solely on the 3D Gaussian Splatting (3DGS) representation [10] recently making a big impact in offline scene reconstruction. In 3DGS a scene is represented by a large number of Gaussian blobs with orientation, elongation, colour and opacity. Other previous world/map-centric scene representations used for visual SLAM include occupancy or Signed Distance Function (SDF) voxel grids [23]; meshes [28]; point or surfel clouds [9, 29]; and recently neural fields [33]. Each of these has disadvantages: grids use significant memory and have bounded resolution, and even if octrees or hashing allow more efficiency they cannot be flexibly warped for large corrections [25, 37]; meshes require difficult, irregular topology to fuse new information; surfel clouds are discontinuous and difficult to fuse and optimise; and neural fields require expensive per-pixel raycasting to render. We show that 3DGS has none of these weaknesses. As a SLAM representation, it is most similar to point and surfel clouds, and inherits their efficiency, locality and ability to be easily warped or modified. However, it also represents geometry in a smooth, continuously differentiable way: a dense cloud of Gaussians merge together and jointly define a continuous volumetric function. And crucially, the design of modern graphics cards means that a large number of Gaussians can be efficiently rendered via "splatting" rasterisation, up to 200fps at 1080p. This rapid, differentiable rendering is integral to the tracking and map optimisation loops in our system.

The 3DGS representation has up until now only been used in offline systems for 3D reconstruction with known camera poses, and we present several innovations to enable online SLAM. We first derive the analytic Jacobian of camera pose with respect to a 3D Gaussians map, and show that this can be seamlessly integrated into the existing differentiable rasterisation pipeline to enable camera poses to be optimised alongside scene geometry. Second, we introduce a novel Gaussian shape regularisation to ensure geometric consistency, which we have found is important for incremental reconstruction. Third, we propose a novel Gaussian resource allocation and pruning method to keep the geometry clean and enable accurate camera tracking. Our experimental results demonstrate photorealistic online local scene reconstruction, as well as state-of-the-art camera trajectory estimation and mapping for larger scenes compared to other rendering-based SLAM methods. We further show the uniqueness of the Gaussian-based SLAM method such as an extremely large camera pose convergence basin, which can also be useful for map-based camera localisation. Our method works with only monocular input, one of the most challenging scenarios in SLAM, but we show that it can also incorporate depth measurements when available.

In summary, our contributions are as follows:

- The first near real-time SLAM system which works with a 3DGS as the only underlying scene representation.
- Novel techniques within the SLAM framework, including the analytic Jacobian for camera pose estimation, Gaussian shape regularisation and geometric verification.
- Extensive evaluations on a variety of datasets both for monocular and RGB-D settings, demonstrating competitive performance, particularly in real-world scenarios.

2. Related Work

Dense SLAM: Dense visual SLAM focuses on reconstructing detailed 3D maps, unlike sparse SLAM methods which excel in pose estimation [4, 5, 21] but typically yield maps useful mainly for localisation. In contrast, dense SLAM creates interactive maps beneficial for broader applications, including AR and robotics. Dense SLAM methods are generally divided into two primary categories: Frame-centric and Map-centric. **Frame-centric SLAM** minimises photometric error across consecutive frames, jointly estimating per-frame depth and frame-to-frame camera motion. Frame-centric approaches [1, 36] are efficient, as individual frames host local rather than global geometry (e.g. depth maps), and are attractive for long-session SLAM, but if a dense global map is needed, it must be constructed on demand by assembling all of these parts which are not necessarily fully consistent. In contrast, **Map-centric SLAM** uses a unified 3D representation across the SLAM pipeline, enabling a compact and streamlined system. Compared to purely local frame-to-frame tracking, a map-centric approach leverages global information by tracking against the reconstructed 3D consistent map. Classical map-centric approaches often use voxel grids [2, 23, 26, 40] or points [9, 29, 41] as the underlying 3D representation.

While voxels enable a fast look-up of features in 3D, the representation is expensive, and the fixed voxel resolution and distribution are problematic when the spatial characteristics of the environment are not known in advance. On the other hand, a point-based map representation, such as surfel clouds, enables adaptive changes in resolution and spatial distribution by dynamic allocation of point primitives in the 3D space. Such flexibility benefits online applications such as SLAM with deformation-based loop closure [29, 41]. However, optimising the representation to capture high fidelity is challenging due to the lack of correlation among the primitives.

Recently, in addition to classical graphic primitives, neural network-based map representations are a promising alternative. iMAP [33] demonstrated the interesting properties of neural representation, such as sensible hole filling of unobserved geometry. Many recent approaches combine

the classical and neural representations to capture finer details [8, 27, 46, 47]; however, the large amount of computation required for neural rendering makes the live operation of such systems challenging.

Differentiable Rendering: The classical method for creating a 3D representation was to unproject 2D observations into 3D space and to fuse them via weighted averaging [16, 23]. Such an averaging scheme suffers from over-smooth representation and lacks the expressiveness to capture high-quality details.

To capture a scene with photo-realistic quality, differentiable volumetric rendering [24] has recently been popularised with Neural Radiance Fields (NeRF) [17]. Using a single Multi-Layer Perceptron (MLP) as a scene representation, NeRF performs volume rendering by marching along pixel rays, querying the MLP for opacity and colour. Since volume rendering is naturally differentiable, the MLP representation is optimised to minimise the rendering loss using multiview information to achieve high-quality novel view synthesis. The main weakness of NeRF is its training speed. Recent developments have introduced explicit volume structures such as multi-resolution voxel grids [6, 14, 34] or hash functions [19] to improve performance. Interestingly, we can infer from these projects that the main contributor to high-quality novel view synthesis is not the neural network but rather differentiable volumetric rendering, and that it is possible to avoid the use of an MLP and yet achieve comparable rendering quality to NeRF [6]. However, even in these systems, per-pixel ray marching remains a significant bottleneck for rendering speed. This issue is particularly critical in SLAM, where immediate interaction with the map is essential for tracking. In contrast to NeRF, 3DGS performs differentiable rasterisation. Similar to regular graphics rasterisations, by iterating over the primitives to be rasterised rather than marching along rays, 3DGS leverages the natural sparsity of a 3D scene and achieves a representation which is expressive to capture high-fidelity 3D scenes while offering significantly faster rendering. Several works have applied 3D Gaussians and differentiable rendering to static scene capture [11, 38], and in particular more recent works utilise 3DGS and demonstrate superior results in vision tasks such as dynamic scene capture [15, 42, 44] and 3D generation [35, 45].

Our method adopts a Map-centric approach, utilising 3D Gaussians as the only SLAM representation. Similar to surfel-based SLAM, we dynamically allocate the 3D Gaussians, enabling us to model an arbitrary spatial distribution in the scene. Unlike other methods such as ElasticFusion [41] and PointFusion [9], however, by using differentiable rasterisation, our SLAM system can capture high-fidelity scene details and represent challenging object properties by direct optimisation against information from every pixel.

3. Method

3.1. Gaussian Splatting

Our SLAM representation is 3DGS, mapping the scene with a set of anisotropic Gaussians $\mathcal{G}$. Each Gaussian $\mathcal{G}^i$ contains optical properties: colour c^i and opacity α^i. For continuous 3D representation, the mean $\boldsymbol{\mu}_W^i$ and covariance $\boldsymbol{\Sigma}_W^i$, defined in the world coordinate, represent the Gaussian's position and its ellipsoidal shape. For simplicity and speed, in our work we omit the spherical harmonics representing view-dependent radiance. Since 3DGS uses volume rendering, explicit extraction of the surface is not required. Instead, by splatting and blending $\mathcal{N}$ Gaussians, a pixel colour $\mathcal{C}_p$ is synthesised:

$$\mathcal{C}_p = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \,. \tag{1}$$

3DGS performs rasterisation, iterating over the Gaussians rather than marching along the camera rays, and hence, free spaces are ignored during rendering. During rasterisation, the contributions of α are decayed via a Gaussian function, based on the 2D Gaussian formed by splatting a 3D Gaussian. The 3D Gaussians $\mathcal{N}(\boldsymbol{\mu}_W, \boldsymbol{\Sigma}_W)$ in world coordinates are related to the 2D Gaussians $\mathcal{N}(\boldsymbol{\mu}_I, \boldsymbol{\Sigma}_I)$ on the image plane through a projective transformation:

$$\boldsymbol{\mu}_I = \pi(\boldsymbol{T}_{CW} \cdot \boldsymbol{\mu}_W) \,, \boldsymbol{\Sigma}_I = \mathbf{J}\mathbf{W}\boldsymbol{\Sigma}_W\mathbf{W}^T\mathbf{J}^T \,, \tag{2}$$

where π is the projection operation and $\boldsymbol{T}_{CW} \in \boldsymbol{SE}(3)$ is the camera pose of the viewpoint. $\mathbf{J}$ is the Jacobian of the linear approximation of the projective transformation and $\mathbf{W}$ is the rotational component of $\boldsymbol{T}_{CW}$. This formulation enables the 3D Gaussians to be differentiable and the blending operation provides gradient flow to the Gaussians. Using first-order gradient descent [12], Gaussians gradually refines both their optic and geometric parameters to represent the captured scene with high fidelity.

3.2. Camera Pose Optimisation

To achieve accurate tracking, we typically require at least 50 iterations of gradient descent per frame. This requirement emphasises the necessity of a representation with computationally efficient view synthesis and gradient computation, making the choice of 3D representation a crucial part of designing a SLAM system.

In order to avoid the overhead of automatic differentiation, 3DGS implements rasterisation with CUDA with derivatives for all parameters calculated explicitly. Since rasterisation is performance critical, we similarly derive the camera Jacobians explicitly.

To the best of our knowledge, we provide the first analytical Jacobian of $\boldsymbol{SE}(3)$ camera pose with respect to the

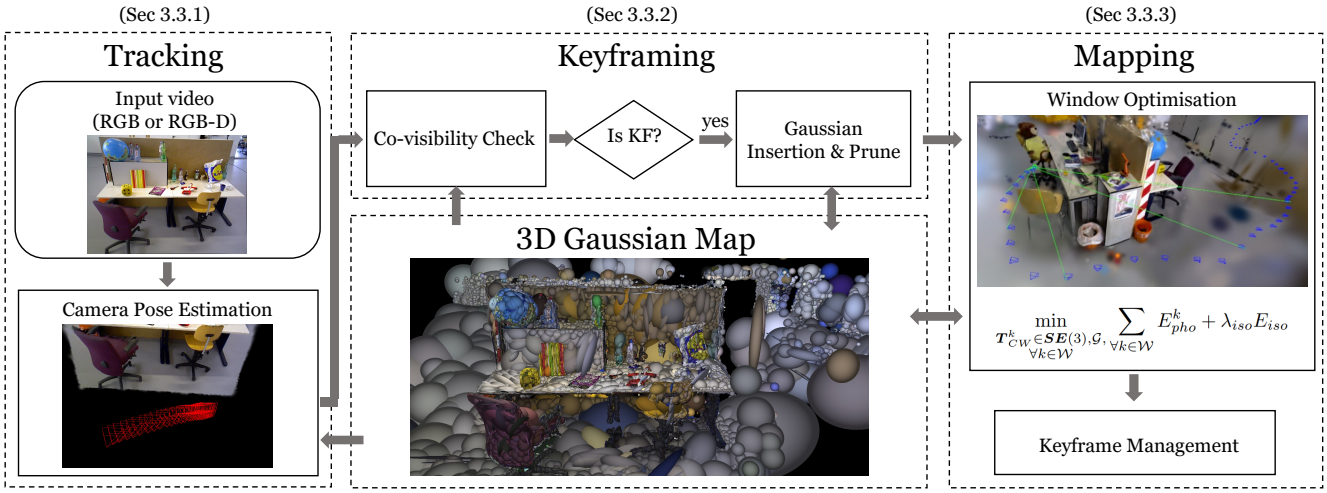

Figure 2. **SLAM System Overview:** Our SLAM system uses 3D Gaussians as the only representation, unifying all components of SLAM, including tracking, mapping, keyframe management, and novel view synthesis.

3D Gaussians used in EWA splatting [48] and 3DGS. This opens up new applications of 3DGS beyond SLAM.

We use Lie algebra to derive the minimal Jacobians, ensuring that the dimensionality of the Jacobians matches the degrees of freedom, eliminating any redundant computations. The terms of Eq. (2) are differentiable with respect to the camera pose $\boldsymbol{T}_{CW}$; using the chain rule:

$$\frac{\partial \boldsymbol{\mu}_I}{\partial \boldsymbol{T}_{CW}} = \frac{\partial \boldsymbol{\mu}_I}{\partial \boldsymbol{\mu}_C} \frac{\mathcal{D}\boldsymbol{\mu}_C}{\mathcal{D}\boldsymbol{T}_{CW}} , \tag{3}$$

$$\frac{\partial \boldsymbol{\Sigma}_I}{\partial \boldsymbol{T}_{CW}} = \frac{\partial \boldsymbol{\Sigma}_I}{\partial \mathbf{J}} \frac{\partial \mathbf{J}}{\partial \boldsymbol{\mu}_C} \frac{\mathcal{D}\boldsymbol{\mu}_C}{\mathcal{D}\boldsymbol{T}_{CW}} + \frac{\partial \boldsymbol{\Sigma}_I}{\partial \mathbf{W}} \frac{\mathcal{D}\mathbf{W}}{\mathcal{D}\boldsymbol{T}_{CW}} . \tag{4}$$

We take the derivatives on the manifold to derive minimal parameterisation. Borrowing the notation from [30], let $\boldsymbol{T} \in \boldsymbol{SE}(3)$ and $\tau \in \mathfrak{se}(3)$. We define the partial derivative on the manifold as:

$$\frac{\mathcal{D} f(\boldsymbol{T})}{\mathcal{D}\boldsymbol{T}} \triangleq \lim_{\tau \to 0} \frac{\mathrm{Log}(f(\mathrm{Exp}(\tau) \circ \boldsymbol{T}) \circ f(\boldsymbol{T})^{-1})}{\tau} , \tag{5}$$

where $\circ$ is a group composition, and Exp, Log are the exponential and logarithmic mappings between Lie algebra and Lie Group. With this, we derive the following:

$$\frac{\mathcal{D}\boldsymbol{\mu}_C}{\mathcal{D}\boldsymbol{T}_{CW}} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{\mu}_C^{\times} \end{bmatrix} , \frac{\mathcal{D}\mathbf{W}}{\mathcal{D}\boldsymbol{T}_{CW}} = \begin{bmatrix} \mathbf{0} & -\mathbf{W}_{:,1}^{\times} \\ \mathbf{0} & -\mathbf{W}_{:,2}^{\times} \\ \mathbf{0} & -\mathbf{W}_{:,3}^{\times} \end{bmatrix} , \tag{6}$$

where $^{\times}$ denotes the skew symmetric matrix of a 3D vector, and $\mathbf{W}_{:,i}$ refers to the ith column of the matrix.

3.3. SLAM

In this section, we present details of full SLAM framework. The overview of the system is summarised in Fig. 2. Please refer to the supplementary material for the further parameter details.

3.3.1 Tracking

In tracking only the current camera pose is optimised, without updates to the map representation. In the monocular case, we minimise the following photometric residual:

$$E_{pho} = \left\| I(\mathcal{G}, \boldsymbol{T}_{CW}) - \bar{I} \right\|_1 , \tag{7}$$

where $I(\mathcal{G}, \boldsymbol{T}_{CW})$ renders the Gaussians $\mathcal{G}$ from $\boldsymbol{T}_{CW}$, and $\bar{I}$ is an observed image.

We further optimise affine brightness parameters for varying exposure. When depth observations are available, we define the geometric residual as:

$$E_{geo} = \left\| D(\mathcal{G}, \boldsymbol{T}_{CW}) - \bar{D} \right\|_1 , \tag{8}$$

where $D(\mathcal{G}, \boldsymbol{T}_{CW})$ is depth rasterisation and $\bar{D}$ is the observed depth. Rather than simply using the depth measurements to initialise the Gaussians, we minimise both photometric and geometric residuals: $\lambda_{pho} E_{pho} + (1 - \lambda_{pho}) E_{geo}$, where λ_{pho} is a hyperparameter.

As in Eq. (1), per-pixel depth is rasterised by alpha-blending:

$$\mathcal{D}_p = \sum_{i \in \mathcal{N}} z_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) , \tag{9}$$

where z_i is the distance to the mean $\boldsymbol{\mu}_W$ of Gaussian i along the camera ray. We derive analytical Jacobians for the camera pose optimisation in a similar manner to Eq. (3), (4).

3.3.2 Keyframing

Since using all the images from a video stream to jointly optimise the Gaussians and camera poses online is infeasible, we maintain a small window $\mathcal{W}_k$ consisting of carefully

selected keyframes based on inter-frame covisibility. Ideal keyframe management will select non-redundant keyframes observing the same area, spanning a wide baseline to provide better multiview constraints.

Selection and Management Every tracked frame is checked for keyframe registration based on our simple yet effective criteria. We measure the covisibility by measuring the intersection over union of the observed Gaussians between the current frame i and the last keyframe j. If the covisibility drops below a threshold, or if the relative translation t_{ij} is large with respect to the median depth, frame i is registered as a keyframe. For efficiency, we maintain only a small number of keyframes in the current window $\mathcal{W}_k$ following the keyframe management heuristics of DSO [4]. The main difference is that a keyframe is removed from the current window if the overlap coefficient with the latest keyframe drops below a threshold. The parameters are detailed in supplementary. 7.1.2.

Gaussian Covisibility An accurate estimate of covisibility simplifies keyframe selection and management. 3DGS respects visibility ordering since the 3D Gaussians are sorted along the camera ray. This property is desirable for covisibility estimation as occlusions are handled by design. A Gaussian is marked to be visible from a view if used in the rasterisation and if the ray's accumulated α has not yet reached 0.5. This enables our estimated covisibility to handle occlusions without requiring additional heuristics.

Gaussian Insertion and Pruning At every keyframe, new Gaussians are inserted into the scene to capture newly visible scene elements and to refine the fine details. When depth measurements are available, Gaussian means $\boldsymbol{\mu}_W$ are initialised by back-projecting the depth. In the monocular case, we render the depth at the current frame. For pixels with depth estimates, $\boldsymbol{\mu}_W$ are initialised around those depths with low variance; for pixels without the depth estimates, we initialise $\boldsymbol{\mu}_W$ around the median depth of the rendered image with high variance (parameters are in supplementary. 7.1.2).

In the monocular case, the positions of many newly inserted Gaussians are incorrect. While the majority will quickly vanish during optimisation as they violate multiview consistency, we further prune the excess Gaussians by checking the visibility amongst the current window $\mathcal{W}_k$. If the Gaussians inserted within the last 3 keyframes are unobserved by at least 3 other frames, we prune them out as they are geometrically unstable.

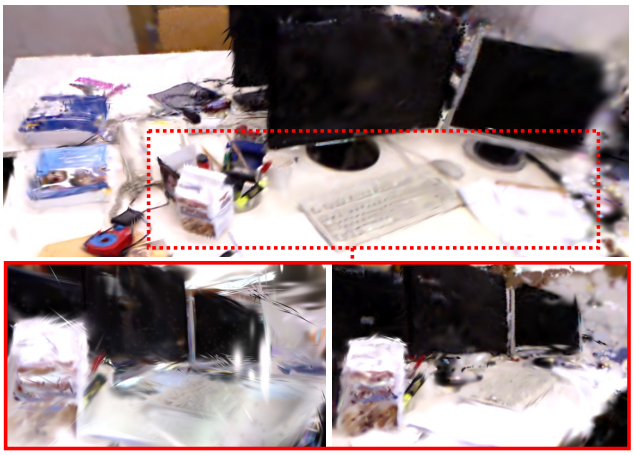

Figure 3. **Effect of isotropic regularisation**: **Top:** Rendering close to a training view (looking at the keyboard). **Bottom:** Rendering 3D Gaussians far from the training views (view from a side of the keyboard) without (left) and with (right) the isotropic loss. When the photometric constraints are insufficient, the Gaussians tend to elongate along the viewing direction, creating artefacts in the novel views, and affecting the camera tracking.

3.3.3 Mapping

The purpose of mapping is to maintain a coherent 3D structure and to optimise the newly inserted Gaussians. During mapping, the keyframes in $\mathcal{W}_k$ are used to reconstruct currently visible regions. Additionally, two random past keyframes $\mathcal{W}_r$ are selected per iteration to avoid forgetting the global map. Rasterisation of 3DGS imposes no constraint on the Gaussians along the viewing ray direction, even with a depth observation. This is not a problem when sufficient carefully selected viewpoints are provided (e.g. in the novel view synthesis case); however, in continuous SLAM this causes many artefacts, making tracking challenging. We therefore introduce an isotropic regularisation:

$$E_{iso} = \sum_{i=1}^{|\mathcal{G}|} \|\mathbf{s}_i - \tilde{\mathbf{s}}_i \cdot \mathbf{1}\|_1 \tag{10}$$

to penalise the scaling parameters $\mathbf{s}_i$ (i.e. stretch of the ellipsoid) by its difference to the mean $\tilde{\mathbf{s}}_i$. As shown in Fig 3, this encourages sphericality, and avoids the problem of Gaussians which are highly elongated along the viewing direction creating artefacts. Let the union of the keyframes in the current window and the randomly selected one be $\mathcal{W} = \mathcal{W}_k \cup \mathcal{W}_r$. For mapping, we solve the following problem:

$$\min_{\substack{\boldsymbol{T}^k_{CW} \in \boldsymbol{SE}(3), \mathcal{G}, \\ \forall k \in \mathcal{W}}} \sum_{\forall k \in \mathcal{W}} E^k_{pho} + \lambda_{iso} E_{iso} \, . \tag{11}$$

If depth observations are available, as in tracking, geometric residuals Eq. (8) are added to the optimisation problem.

4. Evaluation

We conduct a comprehensive evaluation of our system across a range of both real and synthetic datasets. Additionally, we perform an ablation study to justify our design choices. Finally, we present qualitative results of our system operating live using a monocular camera, illustrating its practicality and high fidelity reconstruction.

4.1. Experimental Setup

Datasets For our quantitative analysis, we evaluate our method on the TUM RGB-D dataset [32] (3 sequences) and the Replica dataset [31] (8 sequences), following the evaluation in [33]. For qualitative results, we use self-captured real-world sequences recorded by Intel Realsense d455. Since the Replica dataset is designed for RGB-D SLAM evaluation, it contains challenging purely rotational camera motions. We hence use the Replica dataset for RGB-D evaluation only. The TUM RGB-D dataset is used for both monocular and RGB-D evaluation.

Implementation Details We run our SLAM on a desktop with Intel Core i9 12900K 3.50GHz and a single NVIDIA GeForce RTX 4090. We present results from our multi-process implementation aimed at real-time applications. For a fair comparison with other methods on Replica, we additionally report result for single-process implementation which performs more mapping iterations. As with 3DGS, time-critical rasterisation and gradient computation are implemented using CUDA. The rest of the SLAM pipeline is developed with PyTorch. Details of hyperparameters are provided in the supplementary material.

Metrics For camera tracking accuracy, we report the Root Mean Square Error (RMSE) of the Absolute Trajectory Error (ATE) of the keyframes. To evaluate map quality, we report standard photometric rendering quality metrics (PSNR, SSIM and LPIPS) following the evaluation protocol used in [27]. To evaluate the map quality, on every fifth frame, rendering metrics are computed. However, for fairness, we exclude the keyframes (training views). We report the average across three runs for all our evaluations. In the tables, the best result is in bold, and the second best is underlined.

Baseline Methods We primarily benchmark our SLAM method against other approaches that, like ours, do not have explicit loop closure. In monocular settings, we compare with state-of-the-art classical and learning-based direct visual odometry (VO) methods. Specifically, we compare DSO [4], DepthCov [3], and DROID-SLAM [36] in VO configurations. These methods are selected based on their public reporting of results on the benchmark (TUM dataset) or the availability of their source code for getting the benchmark result. Since one of our focuses is the online scale estimation under monocular scale ambiguity, the method which uses ground truth poses for the system initialisation such as [13] is not considered for the comparison. In the RGB-D case, we compare against neural-implicit SLAM methods [7, 8, 27, 33, 39, 43, 46] which are also map-centric, rendering-based and do not perform loop closure.

4.2. Quantitative Evaluation

Camera Tracking Accuracy Table 1 shows the tracking results on the TUM RGB-D dataset. In the monocular setting, our method surpasses other baselines without requiring any deep priors. Furthermore, our performance is comparable to systems which perform explicit loop closure. This clearly highlights that there still remains potential for enhancing the tracking of monocular SLAM by exploring fundamental SLAM representations.

Our RGB-D method shows better performance than any other baseline method. Notably, our system surpasses ORB-SLAM in the fr1 sequences, bridging the gap between Map-centric SLAM and the state-of-the-art sparse frame-centric methods. Table 2 reports results on the synthetic Replica dataset. Our single-process implementation shows competitive performance and achieves the best result in 4 out of 8 sequences. Our multi-process implementation which performs fewer mapping iterations still performs comparably. In contrast to other methods, our system demonstrates higher performance on real-world data, as our system flexibly handles real sensor noise by direct optimisation of the Gaussian positions against information from every pixel.

Novel View Rendering Table 5 summarises the novel view rendering performance of our method with RGB-D input. We consistently show the best performance across most sequences and is least second best. Our rendering FPS is hundreds of times faster than other methods, offering a significant advantage for applications which require real-time map interaction. While Point-SLAM is competitive, that method focuses on view synthesis rather than novel-view synthesis. Their view synthesis is conditional on the availability of depth due to the depth-guided ray-sampling, making novel-view synthesis challenging. On the other hand, our rasterisation-based approach does not require depth guidance and achieves efficient, high-quality, novel view synthesis. Fig. 4 provides a qualitative comparison of the rendering of ours and Point-SLAM (with depth guidance).

Ablative Analysis In Table 3, we perform ablation to confirm our design choices. Isotropic regularisation and geometric residual improve the tracking of monocular and RGB-D SLAM respectively, as they aid in constraining the

Input	Loop-closure	Method	fr1/desk	fr2/xyz	fr3/office	Avg.
Monocular	w/o	DSO [4]	22.4	**1.10**	9.50	11.0
		DROID-VO [36]	5.20	10.7	7.30	7.73
		DepthCov [3]	5.60	1.20	68.8	25.2
		Ours	**4.15**	4.79	**4.39**	**4.44**
	w/	DROID-SLAM [36]	**1.80**	**0.50**	2.80	1.70
		ORB-SLAM2 [20]	2.00	0.60	**2.30**	**1.60**
RGB-D	w/o	iMAP [33]	4.90	2.00	5.80	4.23
		NICE-SLAM [46]	4.26	6.19	6.87	5.77
		DI-Fusion [7]	4.40	2.00	5.80	4.07
		Vox-Fusion [43]	3.52	1.49	26.01	10.34
		ESLAM [8]	2.47	**1.11**	2.42	2.00
		Co-SLAM [39]	2.40	1.70	2.40	2.17
		Point-SLAM [27]	4.34	1.31	3.48	3.04
		Ours	**1.52**	1.58	**1.65**	**1.58**
	w/	BAD-SLAM [29]	1.70	1.10	1.70	1.50
		Kintinous [40]	3.70	2.90	3.00	3.20
		ORB-SLAM2 [20]	**1.60**	**0.40**	**1.00**	**1.00**

Table 1. **Camera tracking result on TUM for monocular and RGB-D.** ATE RMSE in cm is reported. We divide systems into with and without explicit loop closures. In both monocular and RGB-D cases, we achieve state-of-the-art performance. In particular, in the monocular case, not only do we outperform systems which use deep prior, but we achieve comparable performance with many of the RGB-D systems.

Method	r0	r1	r2	o0	o1	o2	o3	o4	Avg.
iMAP [33]	3.12	2.54	2.31	1.69	1.03	3.99	4.05	1.93	2.58
NICE-SLAM	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.13	1.07
Vox-Fusion [43]	1.37	4.70	1.47	8.48	2.04	2.58	1.11	2.94	3.09
ESLAM [8]	0.71	0.70	0.52	0.57	0.55	0.58	0.72	**0.63**	0.63
Point-SLAM [27]	0.61	0.41	0.37	**0.38**	**0.48**	0.54	0.69	0.72	**0.53**
Ours	**0.47**	0.43	0.31	0.70	0.57	0.31	0.31	3.2	0.79
Ours*	0.76	**0.37**	**0.23**	0.66	0.72	**0.30**	**0.19**	1.46	0.58

Table 2. **Camera tracking result on Replica for RGB-D SLAM.** ATE RMSE in cm is reported. We achieve best performance across most sequences. Here, Ours is our multi-process implementation and Ours* is the single-process implementation which performs more mapping iterations.

Input	Method	fr1/desk	fr2/xyz	fr3/office	Avg.
Mono	w/o E_{iso}	4.54	4.87	5.1	4.84
	w/o kf selection	48.5	4.36	8.70	20.5
	Ours	**4.15**	**4.79**	**4.39**	**4.44**
RGB-D	w/o E_{geo}	1.66	1.51	2.45	1.87
	w/o kf selection	1.93	1.46	4.07	2.49
	Ours	**1.52**	**1.58**	**1.65**	**1.58**

Table 3. **Ablation Study on TUM RGB-D dataset.** We analyse the usefulness of isotropic regularisation, geometric residual, and keyframe selection to our SLAM system.

Memory Usage				
iMAP [33]	NICE-SLAM [46]	Co-SLAM [39]	Ours (Mono)	Ours (RGB-D)
0.2M	101.6M	1.6M	2.6MB	3.97MB

Table 4. **Memory Analysis on TUM RGB-D dataset.** We compare the size of our Gaussian map to other methods. Baseline numbers are taken from [39].

geometry when photometric signals are weak. For both cases, keyframe selection significantly improves systems performance, as it automatically chooses suitable keyframes based on our occlusion-aware keyframe selection and management. We further compare the memory usage of different 3D representations in Table 4. MLP-based iMAP is clearly more memory efficient, but it struggles to express high-fidelity 3D scenes due to the limited capacity of small MLP. Compared with a voxel grid of features used in NICE-SLAM, our method uses significantly less memory.

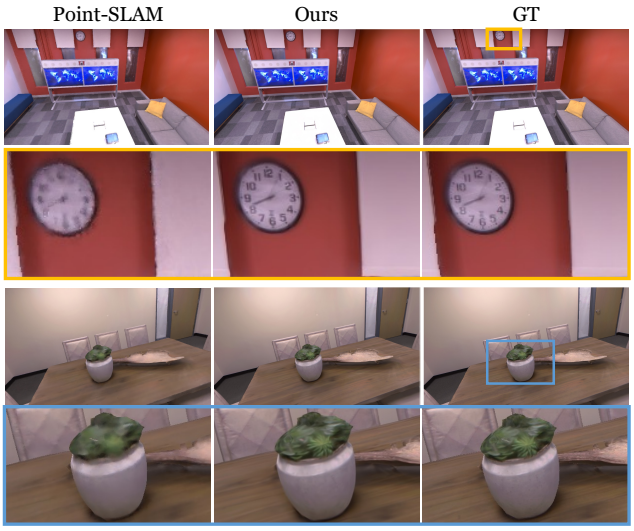

Figure 4. **Rendering examples on Replica.** Due to the stochastic nature of ray sampling, Point-SLAM struggle with rendering fine details.

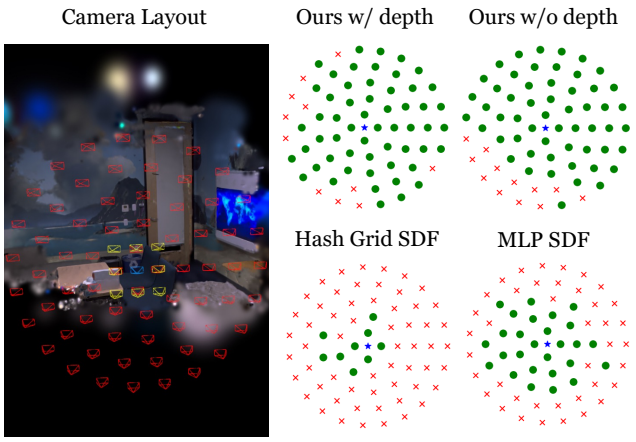

Figure 5. **Convergence basin analysis**: **Left:** 3D Gaussian reconstructed using the training views (Yellow) and visualisation of the test poses (Red). We measure the convergence basin of the target pose (Blue) by performing localisation from the test poses. **Right:** Visualisation of convergence basin of our method (top, with and without depth for training) and other representations (bottom). The green circle marks successful convergence, and the red cross marks failure.

Convergence Basin Analysis In our SLAM experiments, we discovered that 3D Gaussian maps have a notably large convergence basin for camera localisation. To investigate further, we conducted a convergence funnel analysis, an

Method	Metric	room0	room1	room2	office0	office1	office2	office3	office4	Avg.	Rendering FPS
NICE-SLAM [46]	PSNR[dB] ↑	22.12	22.47	24.52	29.07	30.34	19.66	22.23	24.94	24.42	0.54
	SSIM ↑	0.689	0.757	0.814	0.874	0.886	0.797	0.801	0.856	0.809	
	LPIPS↓	0.33	0.271	0.208	0.229	0.181	0.235	0.209	0.198	0.233	
Vox-Fusion [43]	PSNR[dB] ↑	22.39	22.36	23.92	27.79	29.83	20.33	23.47	25.21	24.41	2.17
	SSIM ↑	0.683	0.751	0.798	0.857	0.876	0.794	0.803	0.847	0.801	
	LPIPS↓	0.303	0.269	0.234	0.241	0.184	0.243	0.213	0.199	0.236	
Point-SLAM [27]	PSNR[dB] ↑	32.40	34.08	35.5	38.26	39.16	33.99	33.48	33.49	35.17	1.33
	SSIM ↑	**0.974**	**0.977**	**0.982**	**0.983**	**0.986**	0.96	0.960	**0.979**	**0.975**	
	LPIPS↓	0.113	0.116	0.111	0.1	0.118	0.156	0.132	0.142	0.124	
Ours	PSNR[dB] ↑	**34.83**	**36.43**	**37.49**	**39.95**	**42.09**	**36.24**	**36.7**	**36.07**	**37.50**	**769**
	SSIM ↑	0.954	0.959	0.965	0.971	0.977	**0.964**	**0.963**	0.957	0.960	
	LPIPS↓	**0.068**	**0.076**	**0.075**	**0.072**	**0.055**	**0.078**	**0.065**	**0.099**	**0.070**	

Table 5. **Rendering performance comparison of RGB-D SLAM methods on Replica.** Our method outperforms most of the rendering metrics compared to existing methods. Note that Point-SLAM uses sensor depth (ground-truth depth in Replica) to guide sampling along rays, which limits the rendering performance to existing views. The numbers for the baselines are taken from [27].

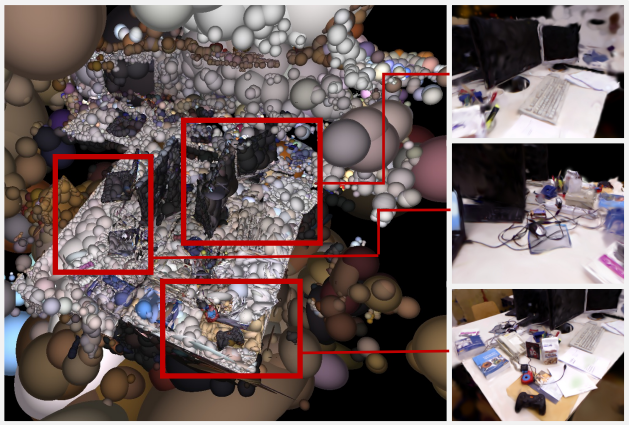

Figure 6. **Monocular SLAM result on fr1/desk sequence:** We show the reconstructed 3D Gaussian maps (Left) and novel view synthesis result (Right).

Method	seq1	seq2	seq3	Avg.
Neural SDF (Hash Grid)	0.13	0.15	0.16	0.14
Neural SDF (MLP)	0.40	0.38	0.22	0.33
Ours w/o depth	0.82	0.91	**0.65**	0.79
Ours w/ depth	**0.83**	**1.0**	**0.65**	**0.82**

Table 6. **Camera convergence analysis.** We report the ratio of successful camera convergence for the different sequences, across different differentiable 3D representations.

evaluation methodology proposed in [18] and used in [22]. Here, we train a 3D representation (e.g. 3DGS) using 9 fixed views arranged in a square. We set the viewpoint in the middle of the square to be the target view. As shown in Fig 5, we uniformly sample a position, creating a funnel. From the sampled position, given the RGB image of the target view, we perform camera pose optimisation for 1000 iterations. The optimisation is successful if it converges to within 1cm of the target view within the fixed iterations. We compare our Gaussian approach with Co-SLAM [39]'s network (Hash Grid SDF) and iMAP's [33] network with

Figure 7. **Self-captured Scenes:** Challenging scenes and objects, for example transparent glasses and crinkled texture of salad are captured by our monocular SLAM running live.

Co-SLAM's SDF loss for further geometric accuracy (MLP Neural SDF). We render the training views using a synthetic Replica dataset and create three sequences for testing (seq1, seq2 and seq3). The width of the square formed by the training view is 0.5m, and the test cameras are distributed with radii ranging from 0.2m to 1.2m, covering a larger area than the training view. When training the map, the three methods— Ours w/depth, Hash Grid SDF, and MLP SDF—use RGB-D images, whereas Ours w/o depth utilises only colour images. Fig. 5 shows the qualitative results and Table 6 reports the success rate. For both with and without depth for training, our method shows better convergence. Unlike hashing and positional encoding which can lead to signal conflict, anisotropic Gaussians form a smooth gradient in 3D space, increasing the convergence basin. Further experimental details are available in supplementary 8.3.

4.3. Qualitative Results

We report both the 3D reconstruction of the SLAM dataset and self-captured sequences. In Fig. 6, we visualise the monocular SLAM reconstruction of fr1/desk. The placements of the Gaussians are geometrically sensible and are 3D coherent, and our rendering from the different viewpoints highlights the quality of our systems' novel view synthesis. In Fig. 7, we self-capture challenging scenes for monocular SLAM. By not explicitly modelling a surface, our system naturally handle transparent objects which are challenging for many other SLAM systems.

5. Conclusion

We have proposed the first SLAM method using 3D Gaussians as a SLAM representation. Via efficient volume rendering, our system significantly advances the fidelity and diversity of object materials a live SLAM system can capture. Our system achieves state-of-the-art performance across benchmarks for both monocular and RGB-D cases. Interesting directions for future research are the integration of loop closure for handling large-scale scenes and extraction of geometry such as surface normal as Gaussians do not explicitly represent surface.

6. Acknowledgement

Research presented in this paper has been supported by Dyson Technology Ltd. We are very grateful to Eric Dexheimer, Kirill Mazur, Xin Kong, Marwan Taher, Ignacio Alzugaray, Gwangbin Bae, Aalok Patwardhan, and members of the Dyson Robotics Lab for their advice and insightful discussions.

Gaussian Splatting SLAM

Supplementary Material

7. Implementation Details

7.1. System Details and Hyperparameters

7.1.1 Tracking and Mapping (Sec. 3.3.1 and 3.3.3)

Learning Rates We use the Adam optimiser for both camera poses and Gaussian parameters optimisation. For camera poses, we used 0.003 for rotation and 0.001 for translation. For 3D Gaussians, we used the default learning parameters of the original Gaussian Splatting implementation [10], apart from in monocular setting where we increase the learning rate of the positions of the Gaussians $\boldsymbol{\mu}_W$ by a factor of 10.

Iteration numbers 100 tracking iterations are performed per frame for across all experiments. However, we terminate the iterations early if the magnitude of the pose update becomes less than 10^{-4}. For mapping, 150 iterations are used for the single-process implementation.

Loss Weights Given a depth observation, for tracking we minimise both photometric Eq. (7) and geometric residual Eq. (8) as:

$$\min_{\boldsymbol{T}_{CW} \in \boldsymbol{SE}(3)} \lambda_{pho} E_{pho} + (1 - \lambda_{pho}) E_{geo} \,, \tag{12}$$

and similarly, for mapping we modify Eq. (11) to:

$$\min_{\substack{\boldsymbol{T}^k_{CW} \in \boldsymbol{SE}(3), \mathcal{G}, \\ \forall k \in \mathcal{W}}} \sum_{\forall k \in \mathcal{W}} (\lambda_{pho} E^k_{pho} + (1 - \lambda_{pho}) E^k_{geo}) + \lambda_{iso} E_{iso} \,. \tag{13}$$

We set $\lambda_{pho} = 0.9$ for all RGB-D experiments, and $\lambda_{iso} = 10$ for both monocular and RGB-D experiments.

7.1.2 Keyframing (Sec. 3.3.2)

Gaussian Covisibility Check (Sec. 3.3.2) As described in Sec. 3.3.2, keyframe selection is based on the covisibility of the Gaussians. Between two keyframes i, j, we define the covisibility using Intersection of Union (IOU) and Overlap Coefficient (OC):

$$IOU_{cov}(i,j) = \frac{|\mathcal{G}^v_i \cap \mathcal{G}^v_j|}{|\mathcal{G}^v_i \cup \mathcal{G}^v_j|} \,, \tag{14}$$

$$OC_{cov}(i,j) = \frac{|\mathcal{G}^v_i \cap \mathcal{G}^v_j|}{\min(|\mathcal{G}^v_i|, |\mathcal{G}^v_j|)} \,, \tag{15}$$

where $\mathcal{G}^v_i$ is the Gaussians visible in keyframe i, based on visibility check described in Section 3.3.2, Gaussian Covisibility. A keyframe i is added to the keyframe window $\mathcal{W}_k$ if given last keyframe j, $IOU_{cov}(i,j) < kf_{cov}$ or if the relative translation $t_{ij} > kf_m \hat{D}_i$, where $\hat{D}_i$ is the median depth of frame i. For Replica $kf_{cov} = 0.95, kf_m = 0.04$ and for TUM $kf_{cov} = 0.90, kf_m = 0.08$. We remove the registered keyframe j in $\mathcal{W}_k$ if the $OC_{cov}(i,j) < kf_c$, where keyframe i is the latest added keyframe. For both Replica and TUM, we set the cutoff to $kf_c = 0.3$. We set the size of keyframe window to be for Replica, $|\mathcal{W}_k| = 10$, and for TUM, $|\mathcal{W}_k| = 8$.

Gaussian Insertion and Pruning (Sec. 3.3.2) As we optimise the positions of Gaussians and prune geometrically unstable Gaussians, we do not require any strong prior such as depth observation for Gaussian initialisation. When **inserting** new Gaussians in a monocular setting, we randomly sample the Gaussians position $\boldsymbol{\mu}_W$ using rendered depth D. Since the estimated depth may sometimes be incorrect, we account for this by initialising the Gaussians with some variance. For a pixel p where the rendered depth $\mathcal{D}_p$ exists, we sample the depth from $\mathcal{N}(\mathcal{D}_p, 0.2\sigma_D)$. Otherwise, for unobserved regions, we initialise the Gaussians by sampling from $\mathcal{N}(\hat{D}, 0.5\sigma_D)$, where $\hat{D}$ is the median of D. For **pruning**, as described in Section 3.3.2, we perform visibility-based pruning, where if new Gaussians inserted within the last 3 keyframes are not observed by at least 3 other frames, they are pruned. We only perform visibility-based pruning once the keyframe window $\mathcal{W}_k$ is full. Additionally, we prune all Gaussians with opacity of less than 0.7.

8. Evaluation details

8.1. Camera Tracking Accuracy (Table 1 and Table 2)

8.1.1 Evaluation Metric

We measured the keyframe absolute trajectory error (ATE) RMSE. For monocular evaluation, we perform scale alignment between the estimated scale-free and ground-truth trajectories. For RGB-D evaluation, we only align the estimated trajectory and ground truth without scale adjustment.

8.1.2 Baseline Results

Table 1 Numbers for monocular DROID-SLAM [36] and ORB-SLAM [20] is taken from [13]. We have lo-

Method	Rendering FPS ↑	Rendering time per image [s] ↓
NICE-SLAM [46]	0.54	1.85
Vox-Fusion [43]	2.17	0.46
Point-SLAM [27]	1.33	0.75
Ours	**769**	**0.0013**

Table 7. **Further dettail of Rendering FPS and Rendering Time comparison based on Table 5**

.

cally run DSO [4], DepthCov [3] and DROID-VO [36] – which is DROID-SLAM without loop closure and global bundle adjustment. For the RGB-D case, numbers for NICE-SLAM [46], DI-Fusion [7], Vox-Fusion [43], Point-SLAM [27] are taken from Point-SLAM [27], and all the other baselines: iMAP [33], ESLAM [8], Co-SLAM [39], BAD-SLAM [29], Kintinous [40], ORB-SLAM [20]

Table 2 and 5 We took the numbers from Point-SLAM [27] paper.

Table 4 The numbers are from Co-SLAM [39] paper.

8.2. Rendering FPS (Table 5)

In Table 5, we reported the photometric quality metrics (PSNR, SSIM and LPIPS) and rendering fps of our methods. We demonstrated that our rendering fps (769) is much higher than other existing methods (VoxFusion is the second best with 2.17fps). Here we describe the detail of how we measured the fps. The rendering time refers to the duration necessary for full-resolution rendering (1200×680 for the Replica sequence). For each method, we perform 100 renderings and report the average time taken per rendering. The reported rendering fps is found by taking 1 and dividing it by the average rendering time. We summarise the numbers in Table 7. Note that the "rendering fps" means the fps just for the forward rendering, which differs from the end-to-end system fps reported in Table 8 and 9.

8.3. The convergence basin analysis (Table 6 and Fig 5)

8.3.1 The detail of the benchmark Dataset

For convergence basin analysis, we create three datasets by rendering the synthetic Replica dataset. In addition to the qualitative visualisation in Figure 5, we report more detailed camera pose distributions in Figure 8. Figure 8 shows the camera view frustums of the test (red), training (yellow) and target (blue) views. As we mentioned in the main paper, we set the training view in the shape of a square with a width of 0.5m and test views are distributed with radii ranging from 0.2m to 1.2m, covering a larger area than the training views.

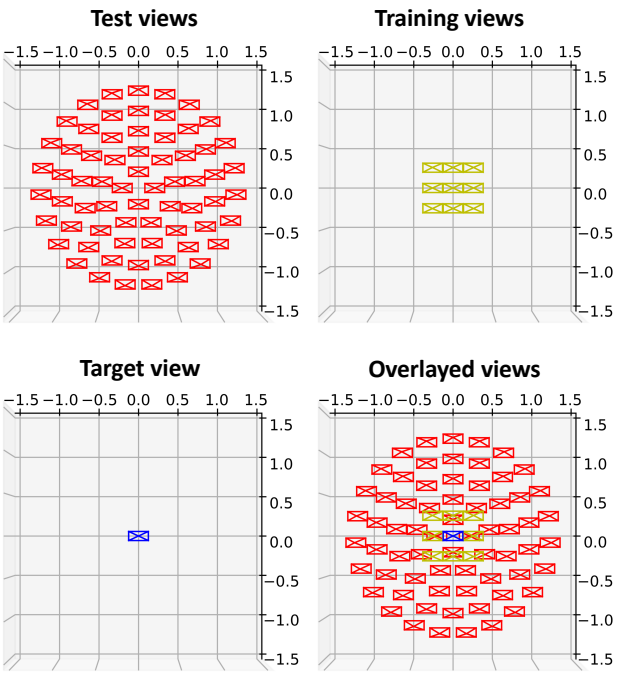

Figure 8. **2D Visualisation of the camera pose distributions used for convergence basin analysis in Figure 5.**

We only apply displacements to the camera translation but not to the rotation. For each sequence, we use a total of 67 test views.

8.3.2 Training setup

For each method, the 3D representation is trained for 30000 iterations using the training views. Here, we detail the training setup of each of the methods:

Ours We evaluated our method under two settings: "w/ depth" and "w/o depth", where we train the initial 3D Gaussian map $\mathcal{G}_{init}$ with and without depth supervision. In the "w/o depth" setting, the 3D Gaussians' positions are randomly initialised, and we minimise the monocular mapping cost Eq. (11) for the 3D Gaussian training, but keeping the camera poses fixed. Specifically, let $k \in \mathbb{N}$ be a number of training views and 3D Gaussians $\mathcal{G}$, we find $\mathcal{G}_{init}$ by:

$$\mathcal{G}_{init} = \arg\min_{\mathcal{G}} \sum_{\forall k \in \mathcal{W}} E^k_{pho} + \lambda_{iso} E_{iso} \,. \tag{16}$$

Note that training views' camera poses $\boldsymbol{T}^k_{CW}$ are fixed during the optimisation.

In the "w/ depth" setting, we train the Gaussian map by minimising the same cost function as our RGB-D SLAM system:

$$\mathcal{G}_{init} = \arg\min_{\mathcal{G}} \sum_{\forall k \in \mathcal{W}} (\lambda_{pho} E^k_{pho} + (1 - \lambda_{pho}) E^k_{geo}) + \lambda_{iso} E_{iso} \,, \quad (17)$$

where we use $\lambda_{pho} = 0.9$ and $\lambda_{iso} = 10$ for all the experiments

Baseline Methods For Hash Grid SDF, we trained the same network architecture as Co-SLAM [39]. For MLP SDF, we trained the network of iMAP [33]. For both baselines, we supervised networks with the same loss functions as Co-SLAM, which are colour rendering loss L_{rgb}, depth rendering loss L_{depth}, SDF loss L_{fs}, free-space loss L_{fs}, and smoothness loss L_{smooth}. Please refer to the original Co-SLAM paper for the exact formulation (equation (6) - (9)). All the training hyperparameters (e.g. learning rate of the network, number of sampling points, loss weight) are the same as Co-SLAM's default configuration of the Replica dataset. While Co-SLAM stores training view information by downsampling the colour and depth images, we store the full pixel information because the number of training views is small.

8.3.3 Testing Setup

For testing, we localise the camera pose by minimising only the photometric error against the ground-truth colour image of the target view.

Ours Let the camera pose $\boldsymbol{T}_{CW} \in \boldsymbol{SE}(3)$ and initial 3D Gaussians $\mathcal{G}_{init}$, the localised camera pose $\boldsymbol{T}^{est}_{CW}$ is found by:

$$\boldsymbol{T}^{est}_{CW} = \arg\min_{\boldsymbol{T}_{CW}} \left\| I(\mathcal{G}_{init}, \boldsymbol{T}_{CW}) - \bar{I}_{target} \right\|_1 . \quad (18)$$

Note that $\mathcal{G}_{init}$ is fixed during the optimisation. We initialise $\boldsymbol{T}_{CW}$ at one of the test view's positions, and optimisation is performed for 1000 iterations. We perform this localisation process for all the test views and measure the success rate. Camera localisation is successful if the estimated pose converges to within 1cm of the target view within the 1000 iterations.

Baseline Methods For the baseline methods, the camera localisation is performed by minimising colour volume rendering loss L_{rgb}, while all the other trainable network parameters are fixed. The learning rates of the pose optimiser are also the same as Co-SLAM's default configuration of Replica dataset.

Method	Total Time [s]	FPS	Avg. Map. Iter.
Monocular	798.9	3.2	88.1
RGB-D	986.7	2.5	81.0

Table 8. **Performance Analysis using fr3/office.** Both monocular and RGB-D implementation uses multiprocessing. We report **the total execution time of our system**, FPS computed by dividing the total number of processed frames with the total time, and average mapping iteration per added keyframe.

Method	Total Time [s]	FPS	Avg. Map. Iter.
RGB-D	1002.7	2.0	27.5
RGB-D*	1878.1	1.1	150

Table 9. **Performance Analysis using replica/office2.** RGB-D uses multi-process implementation and RGB-D* is the single-process implementation. We report **the total execution time of our system**, FPS computed by dividing the total number of processed frames with the total time, and average mapping iteration per added keyframe.

Input	Method	fr1/desk	fr2/xyz	fr3/office	Avg.
Mono	w/o pruning	77.4	12.0	129.0	72.9
	Ours	**4.15**	**4.79**	**4.39**	**4.44**

Table 10. **Pruning Ablation Study on TUM RGB-D dataset (Monocular Input).** Numbers are camera tracking error (ATE RMSE) in cm.

9. Further Ablation Analysis (Table 3)

9.1. Pruning Ablation (Monocular input)

In Table 9.1, we report the ablation study of our proposed Gaussian pruning, which prunes randomly initialised 3D Gaussians effectively in monocular SLAM setting. As the result shows, Gaussian pruning plays a significant role in enhancing camera tracking performance. This improvement is primarily because, without pruning, randomly initialised Gaussians persist in the 3D space, potentially leading to incorrect initial geometry for other views.

9.2. Isotropic Loss Ablation (RGB-D input)

Table 11 and 12 report the ablation study of the effect of isotropic loss E_{iso} for RGB-D input. In TUM, as Table 11 shows, isotropic regularisation does not improve the performance but only shows a marginal difference. However, for Replica, as summarised in Table 12, isotropic loss significantly improves camera tracking performance. Even with the depth measurement, since rasterisation does not consider the elongation along the viewing axis. Isotropic regularisation is required to prevent the Gaussians from over-stretching, especially for textureless regions, which are common in Replica.

Input	Method	fr1/desk	fr2/xyz	fr3/office	Avg.
RGB-D	w/o E_{iso}	1.60	**1.54**	**1.53**	**1.56**
	Ours	**1.52**	1.58	1.65	1.58

Table 11. **Isotropic Loss Ablation Study on TUM RGB-D dataset (RGB-D input).** Numbers are camera tracking error (ATE RMSE) in cm.

Method	r0	r1	r2	o0	o1	o2	o3	o4	Avg.
w/o E_{iso}	0.69	0.53	0.39	4.30	2.01	1.24	0.32	3.58	1.63
Ours	**0.47**	**0.43**	**0.31**	**0.70**	**0.57**	**0.31**	**0.31**	**3.20**	**0.79**

Table 12. **Isotropic Loss Ablation Study on Replica dataset (RGB-D input).** Numbers are camera tracking error (ATE RMSE) in cm.

9.3. Memory Consumption and Frame Rate (Table. 4)

9.3.1 Memory Analysis

In memory consumption analysis, for Table. 4, we measure the final size of the created Gaussians. The memory footprint of our system is lower than the original Gaussian Splatting, which uses approximately 300-700MB for the standard novel view synthesis benchmark dataset [10], as we only maintain well-constrained Gaussians via pruning and do not store the spherical harmonics.

9.3.2 Timing Analysis

To analyse the processing time of our monocular/RGB-D SLAM system, we measure the total time required to process all frames in the TUM-RGBD fr3/office dataset. This approach assesses the performance of our system as a whole, rather than isolating individual components. By adopting this approach, we gain a more realistic understanding of the system's true performance which better reflects the real-world operating conditions, as it avoids the assumption of an idealised, sequential interleaving of the tracking and mapping processes. As shown in Table 9, our system operates at 3.2 FPS with monocular and 2.5 FPS with depth. The FPS is found by dividing the number of processed frames by the total time. We conducted a similar analysis with the Replica dataset office2. Here, we compare the RGB-D method with and without multiprocessing. As expected, single-process implementation takes longer as it performs more mapping iterations.

10. Camera Pose Jacobian

Use of 3D Gaussian as a primitive and performing camera pose optimisation is discussed in [11]; however, the method assumes a smaller number of Gaussians and is based on ray-intersection not splatting; hence, is not applicable to 3DGS. While many applications of 3DGS exist, for example, dynamic tracking and 4D scene representation [15, 42], they assume offline application and require accurate camera position. In contrast, we perform camera pose optimisation by deriving the minimal analytical Jacobians, and for completeness, we provide the derivation of the Jacobians presented in Eq. (6).

$$\frac{\mathcal{D}\boldsymbol{\mu}_C}{\mathcal{D}\boldsymbol{T}_{CW}} = \lim_{\tau\to 0} \frac{\mathrm{Exp}(\tau)\cdot\boldsymbol{\mu}_C - \boldsymbol{\mu}_C}{\tau} \tag{19}$$

$$= \lim_{\tau\to 0} \frac{(\boldsymbol{I}+\tau^\wedge)\cdot\boldsymbol{\mu}_C - \boldsymbol{\mu}_C}{\tau} \tag{20}$$

$$= \lim_{\tau\to 0} \frac{\tau^\wedge\cdot\boldsymbol{\mu}_C}{\tau} \tag{21}$$

$$= \lim_{\tau\to 0} \frac{\theta^\times\boldsymbol{\mu}_C + \rho}{\tau} \tag{22}$$

$$= \lim_{\tau\to 0} \frac{-\boldsymbol{\mu}_C^\times\theta + \rho}{\tau} \tag{23}$$

$$= \begin{bmatrix}\boldsymbol{I} & -\boldsymbol{\mu}_C^\times\end{bmatrix} \tag{24}$$

where $\boldsymbol{T}\cdot\mathbf{x}$ is the group action of $\boldsymbol{T}\in\boldsymbol{SE}(3)$ on $\mathbf{x}\in\mathbb{R}^3$.

Simiarly, we compute the Jacobian with respect to $\mathbf{W}$. Since the translational component is not involved, we only consider the rotational part $\boldsymbol{R}_{CW}$ of $\boldsymbol{T}_{CW}$.

$$\frac{\mathcal{D}\mathbf{W}}{\mathcal{D}\boldsymbol{R}_{CW}} = \lim_{\theta\to 0} \frac{\mathrm{Exp}(\theta)\circ\mathbf{W} - \mathbf{W}}{\theta} \tag{25}$$

$$= \lim_{\theta\to 0} \frac{(\boldsymbol{I}+\theta^\wedge)\circ\mathbf{W} - \mathbf{W}}{\theta} \tag{26}$$

$$= \lim_{\theta\to 0} \frac{\theta^\wedge}{\theta}\circ\mathbf{W} \tag{27}$$

$$= \lim_{\theta\to 0} \frac{\theta^\times}{\theta}\circ\mathbf{W} \tag{28}$$

Since skew symmetric matrix is:

$$\theta^\times = \begin{bmatrix} 0 & -\theta_z & \theta_y \\ \theta_z & 0 & -\theta_x \\ -\theta_y & \theta_x & 0 \end{bmatrix} \tag{29}$$

The partial derivative of one of the component (e.g. θ_x) is:

$$\frac{\partial\theta^\times}{\partial\theta_x} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \mathbf{e}_1^\times \tag{30}$$

where $\mathbf{e}_1 = [1,0,0]^\top, \mathbf{e}_2 = [0,1,0]^\top, \mathbf{e}_3 = [0,0,1]^\top$.

$$\frac{\partial\mathbf{W}}{\partial\theta_x} = \mathbf{e}_1^\times\mathbf{W} = \begin{bmatrix} \mathbf{0}_{1\times 3} \\ -\mathbf{W}_{3,:} \\ \mathbf{W}_{2,:} \end{bmatrix} \tag{31}$$

$$\frac{\partial\mathbf{W}}{\partial\theta_y} = \mathbf{e}_2^\times\mathbf{W} = \begin{bmatrix} \mathbf{W}_{3,:} \\ \mathbf{0}_{1\times 3} \\ -\mathbf{W}_{1,:} \end{bmatrix} \tag{32}$$

$$\frac{\partial\mathbf{W}}{\partial\theta_z} = \mathbf{e}_3^\times\mathbf{W} = \begin{bmatrix} -\mathbf{W}_{2,:} \\ \mathbf{W}_{1,:} \\ \mathbf{0}_{1\times 3} \end{bmatrix} \tag{33}$$

where $\mathbf{W}_{i,:}$ refers to the ith row of the matrix. After column-wise vectorisation of Eq. (31), (32), (33), and stacking horizontally we get:

$$\frac{\mathcal{D}\mathbf{W}}{\mathcal{D}\boldsymbol{R}_{CW}} = \begin{bmatrix} -\mathbf{W}_{:,1}^{\times} \\ -\mathbf{W}_{:,2}^{\times} \\ -\mathbf{W}_{:,3}^{\times} \end{bmatrix} , \tag{34}$$

where $\mathbf{W}_{:,i}$ refers to the ith column of the matrix. Since translational part is all zeros, with this we get Eq. (6).

11. Additional Qualitative Results

We urge readers to view our supplementary video for convincing qualitative results. In Fig. 9 - Fig. 16, we further show additional qualitative results. We visually compare other state-of-the-art SLAM methods using differentiable rendering (Point-SLAM [27] and ESLAM [8]).

12. Limitation of this work

Although our novel Gaussian Splatting SLAM shows competitive performance on experimental results, the method also has several limitations.

- Currently, the proposed method is tested only on room-scale scenes. For larger real-world scenes, the trajectory drift is inevitable. This could be addressed by integrating a loop closure module into our existing pipeline.
- Although we achieve interactive live operation, hard real-time operation on the benchmark dataset (30 fps on TUM sequences) is not achieved in this work. To improve speed, exploring a second-order optimiser would be an interesting direction.

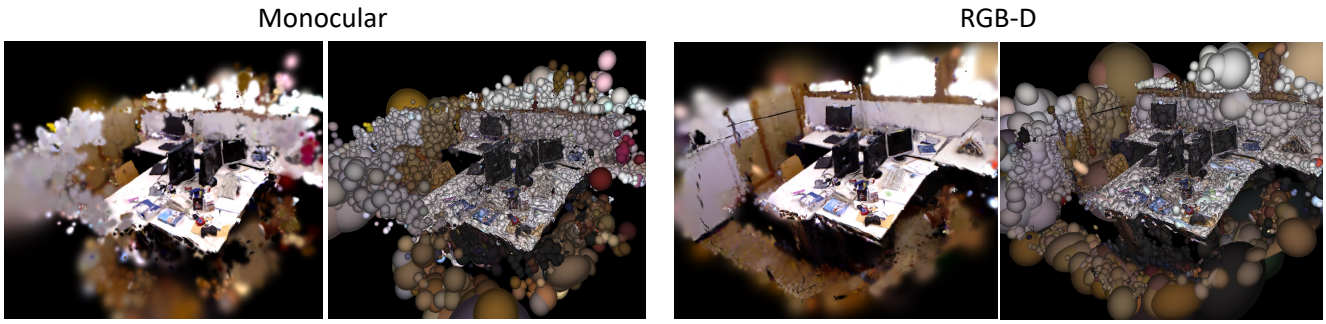

Figure 9. **Novel view rendering and Gaussian visualizations on TUM fr1/desk**

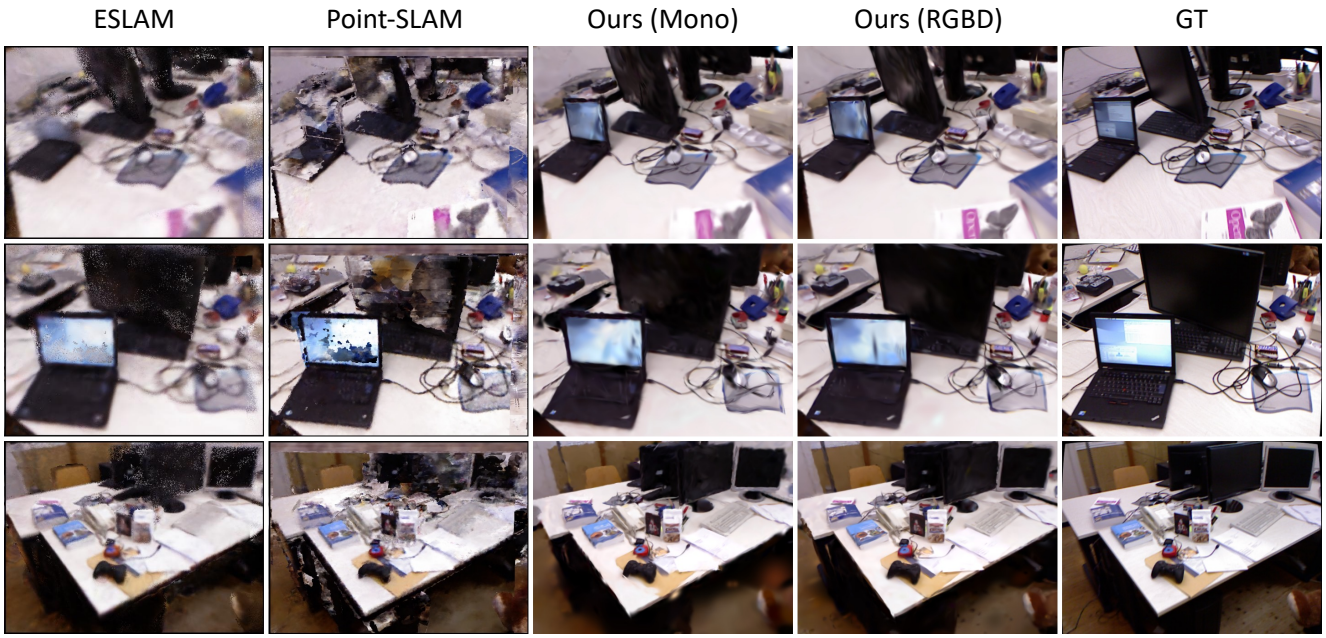

Figure 10. **Rendering comparison on TUM fr1/desk**

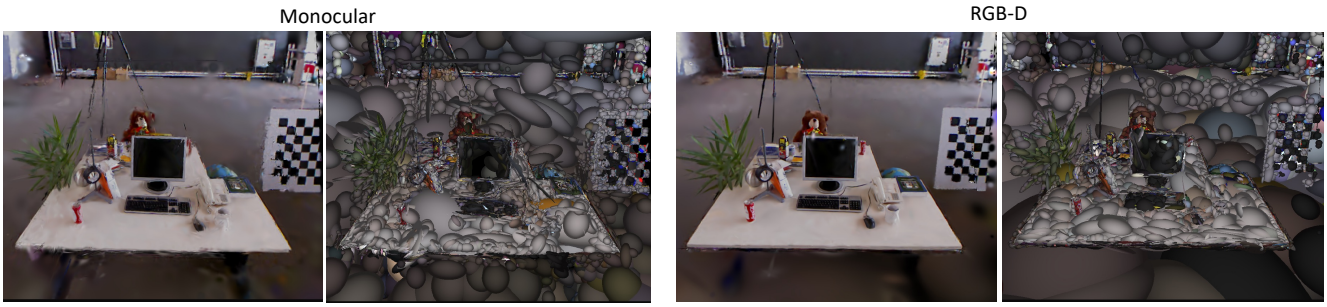

Figure 11. **Novel view rendering and Gaussian visualizations on TUM fr2/xyz**

Figure 12. **Rendering comparison on TUM fr2/xyz**

Figure 13. **Novel view rendering and Gaussian visualizations on TUM fr3/office**

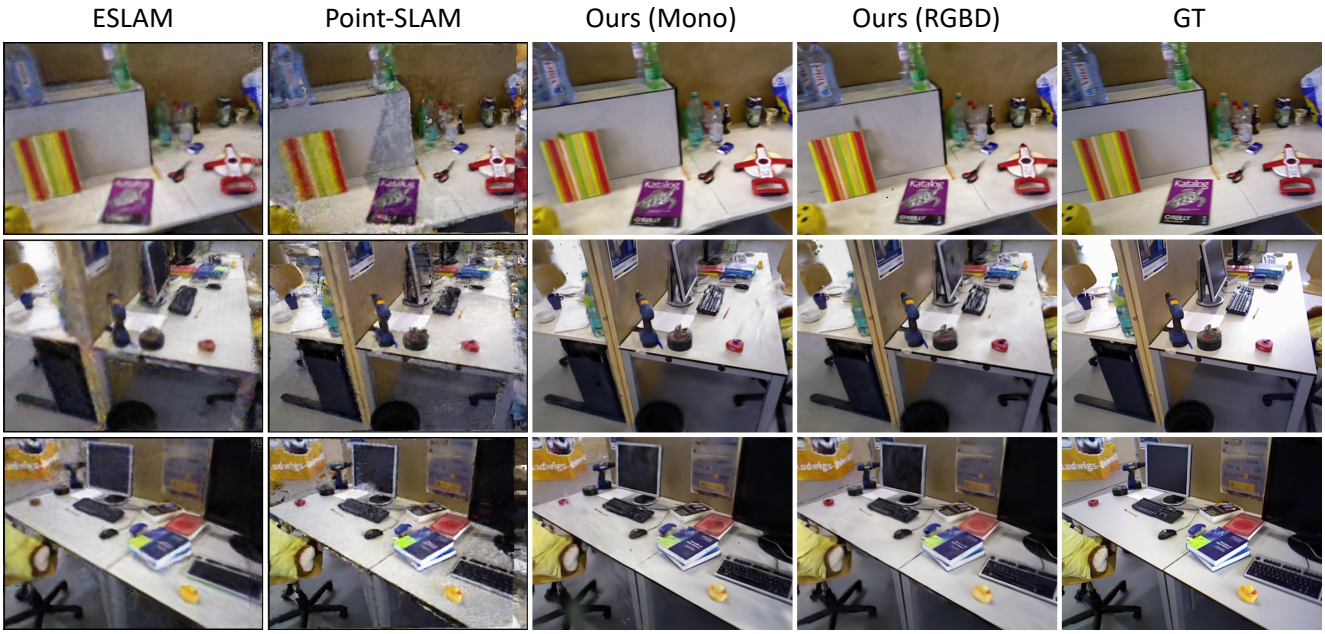

Figure 14. **Rendering comparison on TUM fr3/office**

Figure 15. **Novel view rendering and Gaussian visualizations on Replica**

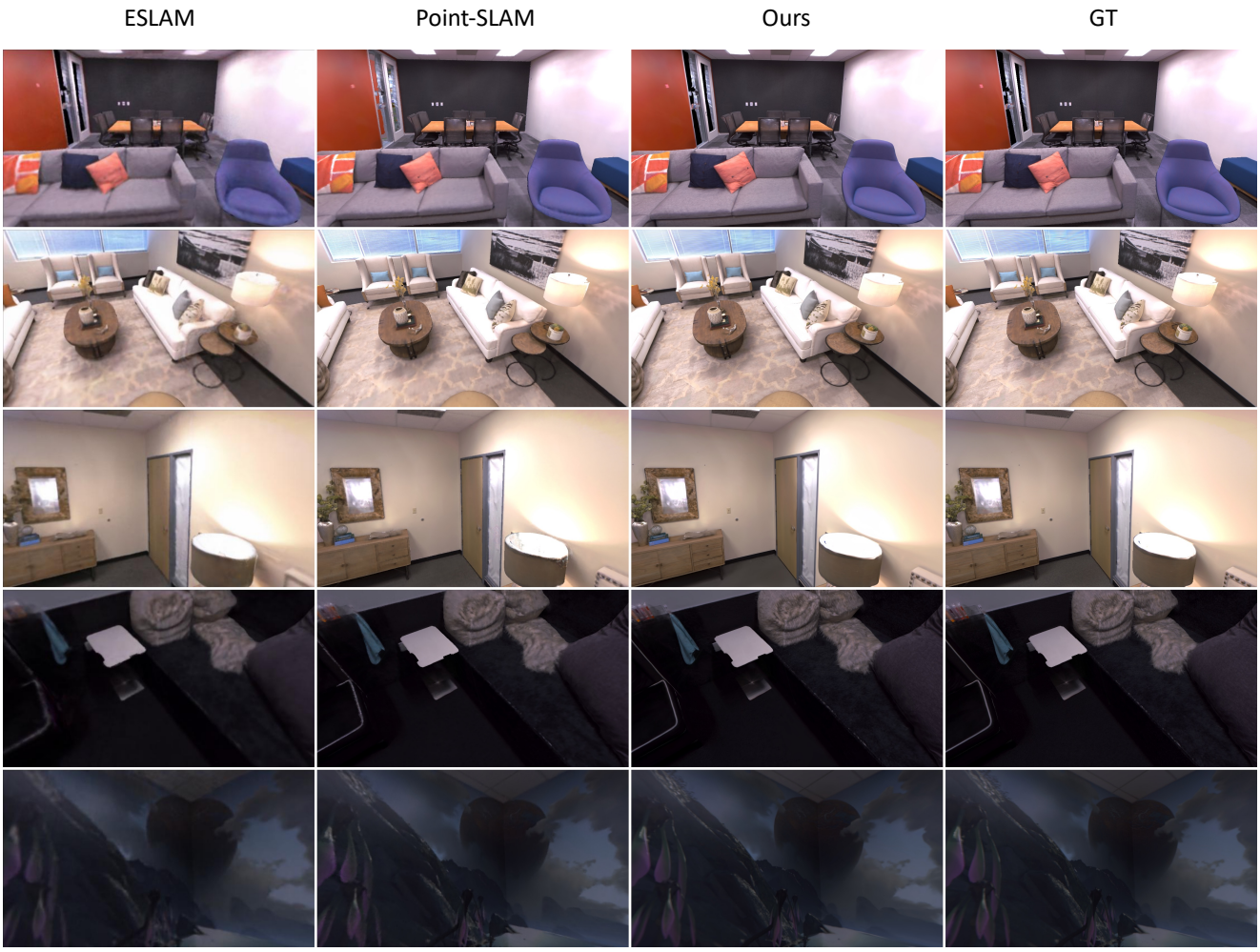

Figure 16. **Rendering comparison on Replica**

References

[1] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison. Deepfactors: Real-time probabilistic dense monocular SLAM. *IEEE Robotics and Automation Letters (RAL)*, 5(2): 721–728, 2020.

[2] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration. *ACM Transactions on Graphics (TOG)*, 36(3):24:1–24:18, 2017.

[3] Eric Dexheimer and Andrew J. Davison. Learning a Depth Covariance Function. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[4] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2017.

[5] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[6] 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 Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[7] Jiahui Huang, Shi-Sheng Huang, Haoxuan Song, and Shi-Min Hu. Di-fusion: Online implicit 3d reconstruction with deep priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[8] M. M. Johari, C. Carta, and F. Fleuret. ESLAM: Efficient dense slam system based on hybrid representation of signed distance fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[9] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)*, 2013.

[10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 2023.

[11] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[13] Heng Li, Xiaodong Gu, Weihao Yuan, Luwei Yang, Zilong Dong, and Ping Tan. Dense rgb slam with neural implicit maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[14] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020.

[15] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. *3DV*, 2024.

[16] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger. SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[17] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[18] N. J. Mitra, N. Gelfand, H. Pottmann, and L. J. Guibas. Registration of Point Cloud Data from a Geometric Optimization Perspective. In *Proceedings of the Symposium on Geometry Processing*, 2004.

[19] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 2022.

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

[21] R. Mur-Artal, J. M. M Montiel, and J. D. Tardós. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163, 2015.

[22] R. A. Newcombe. *Dense Visual SLAM*. PhD thesis, Imperial College London, 2012.

[23] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[24] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[25] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proceedings of SIGGRAPH*, 2013.

[26] Victor Adrian Prisacariu, Olaf Kähler, Ming-Ming Cheng, Carl Yuheng Ren, Julien P. C. Valentin, Philip H. S. Torr, Ian D. Reid, and David W. Murray. A framework for the volumetric integration of depth images. *CoRR*, abs/1410.0925, 2014.

[27] Erik Sandström, Yue Li, Luc Van Gool, and Martin R. Oswald. Point-slam: Dense neural point cloud-based slam. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023.

[28] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Surfelmeshing: Online surfel-based mesh reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2020.

[29] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[30] J. Solà, J. Deray, and D. Atchuthan. A micro Lie theory for state estimation in robotics. *arXiv:1812.01537*, 2018.

[31] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei 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 M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

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

[33] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison. iMAP: Implicit mapping and positioning in real-time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[34] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[35] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *arXiv preprint arXiv:2309.16653*, 2023.

[36] Zachary Teed and Jia Deng. DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras. In *Neural Information Processing Systems (NIPS)*, 2021.

[37] Emanuele Vespa, Nikolay Nikolov, Marius Grimm, Luigi Nardi, Paul HJ Kelly, and Stefan Leutenegger. Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping. *IEEE Robotics and Automation Letters (RAL)*, 2018.

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

[39] Hengyi Wang, Jingwen Wang, and Lourdes Agapito. Co-slam: Joint coordinate and sparse parametric encodings for neural real-time slam. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[40] T. Whelan, M. Kaess, H. Johannsson, M. F. Fallon, J. J. Leonard, and J. B. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *International Journal of Robotics Research (IJRR)*, 34(4-5):598–626, 2015.

[41] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015.

[42] 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 preprint arXiv:2310.08528*, 2023.

[43] 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. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2022.

[44] Zeyu Yang, Hongye Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. *arXiv preprint arXiv 2310.10642*, 2023.

[45] 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:2310.08529*, 2023.

[46] 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 Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[47] Zihan Zhu, Songyou Peng, Viktor Larsson, Zhaopeng Cui, Martin R Oswald, Andreas Geiger, and Marc Pollefeys. Nicer-slam: Neural implicit scene encoding for rgb slam. *arXiv preprint arXiv:2302.03594*, 2023.

[48] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.