Triangle Splatting SLAM

Nicholas Fry[1,2], Eric Dexheimer[2], Kirill Mazur[2], Paul H. J. Kelly[1,2], and Andrew J. Davison[2]

[1] Software Performance Optimisation Group, Imperial College London
[2] Department of Computing, Imperial College London
{nf20, e.dexheimer21, k.mazur21, p.kelly, a.davison}@imperial.ac.uk

Project page: https://nmjfry.github.io/triangle-splatting-slam/

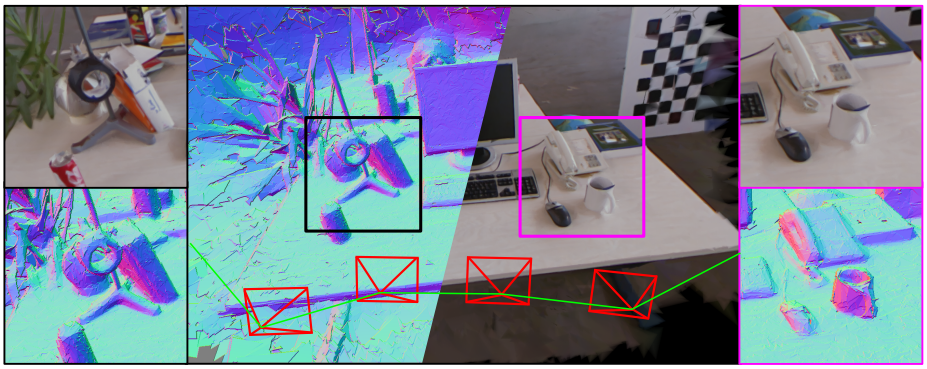

Fig. 1: Triangle Splatting SLAM uses triangles as the underlying scene representation to enable photo-realistic and high-fidelity geometry reconstruction, accurate camera pose estimation, and on-the-fly mesh generation.

Abstract. We present a dense RGB-D SLAM system using differentiable triangles as the 3D map representation. While 3D Gaussian Splatting has emerged as the leading method for novel-view synthesis, triangles remain the standard primitive for traditional rendering hardware, game engines, and downstream tasks requiring explicit geometry such as simulation, collision, and editing. Recent offline methods have demonstrated that an unstructured 'triangle soup' can be optimised into a photorealistic mesh via Delaunay triangulation across a set of posed images. Building upon this insight, we present the first dense SLAM system to employ Triangle Splatting to perform both tracking and mapping through online differentiable rendering of a triangle soup. The map can be converted into a connected mesh on-the-fly via restricted Delaunay triangulation, enabling new online capabilities such as mesh deformation and collision checking. On Replica and TUM-RGBD, our system outperforms baselines on 3D geometry, matches the camera-tracking accuracy, and enables online mesh-based scene editing.

1 Introduction

Embodied AI applications such as robotics can benefit strongly from explicit 3D scene representations built incrementally as a camera moves through the world. Visual SLAM has evolved from sparse landmark maps to dense photorealistic reconstructions, with differentiable rendering enabling joint optimisation of camera poses and scene geometry. More recently, strong learned priors have dramatically improved depth and camera pose estimation. However, there is still much debate about the right representation for 3D in such systems, and none suggested so far offers all of the ideal properties we might want. Ideally, such a representation would be photorealistic, geometrically precise for simulation and planning, compact in memory, fast to render, straightforward to update incrementally, and flexible enough to accommodate scene changes; yet no existing approach achieves all of these goals simultaneously.

For decades, triangle meshes have been the industry standard for 3D scene representation in games, films, and other digital media, with GPUs optimised specifically to render them at high performance. Beyond rendering speed, meshes represent surface geometry explicitly and adaptively, with high precision in detailed areas and efficiency in simpler structures. Furthermore, their built-in connectivity makes them straightforward to edit, deform, and warp.

However, the connectivity that makes meshes so valuable for downstream tasks proves to be a significant liability when it comes to incremental reconstruction. Maintaining topological consistency while continuously integrating new sensor measurements is notoriously difficult, and for this reason dense visual SLAM methods have usually operated with other representations such as voxel or Signed Distance Function (SDF) grids, implicit neural functions, or disconnected clouds of point-like primitives such as surfels or 3D Gaussians. A triangle mesh is then generated if desired, but only as a secondary representation not directly used by the SLAM algorithm [25, 29, 30].

In methods that use volumetric grids, meshing is achieved via the marching cubes algorithm [17]. The mesh must therefore be completely regenerated each time the map changes using a set of fixed 3D grid resolutions. Meshes produced indirectly from primitives such as 3D or 2D Gaussians [13, 21, 40] require ray-marching or density-based SDF extraction, meaning the depth estimates don't necessarily correspond to the final mesh and thus does not guarantee geometric consistency. In 3D point-based triangulation methods, such as Delaunay triangulation, existing points are connected to form tetrahedra with triangular faces. Given a set of estimated 3D points derived from depth measurements or a prior, Delaunay triangulation maintains the existing geometry rather than generating a new set of vertices. Consequently, the method is inherently adaptive to the resolution of the input data, preserves the precise positions of these estimates, and supports incremental updates as opposed to full mesh regeneration.

Recent offline methods have shown that the 3DGS rendering pipeline can be adapted to triangle primitives [7, 35], representing a scene as a soup of triangles with learnable per-vertex colour and opacity rather than a connected mesh. Subsequent works show that one can then recover a photorealistic mesh using

Delaunay triangulation of either this triangle soup, 3D Gaussian 'pivots' or implicit features, avoiding the need for intermediate fusion, post-processing or deep pre-trained priors [5, 7, 18].

In this paper, we present the first SLAM system to employ differentiable triangles [7, 8] as its sole map representation (see Fig. 1), which can operate with and without interleaved meshing. Rather than deferring to a secondary implicit structure, our system models the scene directly as a set of vertices and faces. Both tracking and mapping are driven by differentiable rendering, where the rendered pixel colours are computed via interpolation of vertex features across the projected faces. By optimising the geometry directly, we achieve a photo-realistic reconstruction that enables connected online mesh extraction through Delaunay triangulation.

In summary, our contributions are:

- A SLAM system using differentiable triangles as the underlying map representation.
- Novel strategies for achieving consistent online mapping from a stream of images, including camera pose optimisation, equilateral regularisation, and window-based densification.
- Competitive camera tracking quality on both Replica and TUM-RGBD datasets while achieving significantly higher 3D map quality over alternative differentiable rendering SLAM systems.
- Online scene editing capabilities unlocked by the triangle mesh representation.

2 Related Work

2.1 Representations for Differentiable Rendering

The landscape of representations for 3D modeling is vast and the choice of representation fundamentally dictates the capabilities of the downstream task. While triangles have been dominant in the field of computer graphics, the inverse problem of reconstructing geometry from sensor readings has not reached consensus on the ideal representation.

By making rendering differentiable, scene geometry and appearance can be optimised jointly from image observations alone. NeRF-based representations [1, 22, 23] are attractive for their high-fidelity view synthesis and ability to compress information. However, their rendering speed is limited by expensive ray-marching and MLP queries which are prohibitive in online systems. The implicit nature of these representations complicates geometric interpretation, editing, and integration with classical graphics and physics pipelines which hinders their adoption in intelligent embodied systems.

3D Gaussian Splatting (3DGS) [13] uses anisotropic Gaussian primitives rendered via differentiable alpha compositing to achieve real-time rendering with exceptional visual fidelity. While splatting methods significantly improve rendering speed compared to neural implicit models, most remain fundamentally

volumetric and do not directly yield structured surface representations. Consequently, their suitability for tasks requiring explicit geometry, such as physical simulation, lighting effects, collision detection, deformations or mesh-based editing, remains limited. Other methods, such as Gaussian Opacity Fields and 2D Gaussian Splatting [9, 40] use ray-based intersection to better approximate surface geometry but require post-processing and are still not natively compatible with game engines.

Recent works demonstrate that high-quality meshes can be recovered using differentiable rasterisation on the Delaunay triangulation of structure-from-motion points [5, 7, 18], enabling offline photorealistic mesh reconstruction from posed images. Mesh-In-the-Loop (MILo) [5] trains a set of 3D Gaussians before performing Delaunay triangulation on points sampled around the means of high-opacity Gaussians. A mesh is then extracted via Deep Marching Tetrahedra (DMTet) [31] on the Delaunay tetrahedra, and jointly supervised with the Gaussians to achieve a watertight, smooth, and photorealistic result. While the reconstruction quality is high, the process is slow, uses multiple representations, employs two triangulation strategies, and incurs significant memory cost, making it unsuitable for SLAM. Radiance Meshes [18] similarly achieve high-quality extraction by storing an Instant-NGP [23] within the Delaunay tetrahedra to model view-dependent appearance, but are also prohibitively slow to train. Mesh Splatting [7] takes a more lightweight approach, optimising an alpha-blended triangle soup before applying Restricted Delaunay triangulation to recover a mesh with corrected connectivity. In a post-Delaunay training phase, vertices are further optimised for photorealism, which degrades topological consistency and reintroduces self-intersections. Although the final mesh quality is lower than other methods, the significantly reduced computational cost makes it a compelling foundation for online reconstruction. Radiant Foam [4] also performs 3D Delaunay tetrehedralisation, however it takes an entirely different approach to rendering; performing volumetric ray tracing through Voronoi cells to enable ray-based effects. Its sensitivity to initialisation and incompatibility with other rasterisation pipelines limit its applicability.

2.2 Dense RGB-D SLAM

Sparse RGB-D SLAM achieves accurate pose estimation in real-time [24]. However, these methods lack a dense map which is required for augmented reality (AR) and robotics. It is possible to treat mapping as a problem conditioned on already known poses. For example, a voxel grid representing the truncated signed distance field (TSDF) integrates depth maps given the current pose estimate [2]. This was first shown for RGB-D sensors in KinectFusion, which also uses the mesh from marching cubes for tracking [25]. Storing a dense grid is limited by memory, resulting in a trade-off between efficiency and resolution. Kintinuous [36] addresses this by using a circular buffer for egocentric TSDF reconstruction, with portions exiting the grid converted into a mesh for large-scale reconstruction. However, decoupling the map into separate representations, one

active and the other inactive, means previously visited areas cannot be re-used for tracking or mapping.

Furthermore, sparse voxel structures such as voxel hashing [26] enable scalability to larger areas, but early integration of measurements and the lack of joint optimisation of the poses and map can result in accumulated error leading to map corruption. Point-based representations emerged as an alternative to handle larger scenes in real-time while also continuously fusing new measurements into the current map [12]. To handle drift, ElasticFusion [37] opts for a map-centric approach using surfels that is capable of deforming the entire map to correct for drift. BAD-SLAM [29] jointly optimises poses and surfels, which greatly improves upon the pose accuracy of dense methods that traditionally factorised the estimation. DROID-SLAM [34] achieves state-of-the-art robustness in both monocular and RGB-D scenarios by performing a learned dense bundle adjustment over depth maps. However, unstructured points are not suitable for interaction and bundle adjustment does not optimise for coherent geometry. While surfels provide local surface information, the map is not suitable for physics simulation and true interaction. SurfelMeshing [30] avoids the uniformity of voxel grids and the re-integration of all depths into a volume during loop closure by asynchronously meshing on top of updates to a surfel map. Ultimately, this still requires two representations, as the mesh doesn't inform the geometry of the surfels.

With the rise of differentiable rendering, forward models can be used to continuously update map parameters via back-propagation, which avoids the early integration of backward sensor models and the rigid updates imposed by previous data structures. Inspired by NeRF, iMAP [33] optimises a coordinate MLP via rendered RGB and depth within a parallel tracking and mapping framework inspired by PTAM [14]. NICE-SLAM [41] improves scalability by replacing the global MLP with hierarchical voxel feature grids decoded by MLPs. Differentiable point-based representations [28] avoid the need to choose such a discretisation a priori, allocating capacity adaptively where the scene requires it rather than committing to a fixed grid resolution.

Gaussian Splatting SLAM [11, 21, 38] brought photorealistic real-time mapping to dense SLAM, but the volumetric nature of 3D Gaussians means geometry must be extracted indirectly via density fields, requiring post-processing and failing to guarantee accurate surfaces. Methods using 2D Gaussians with normal supervision [20] move closer to an explicit surface representation, but like surfels, remain disconnected primitives that are not directly suitable for editing, simulation, or integration with standard graphics pipelines.

3 Method

We provide an overview of our method, including the triangle rendering definition and its differentiable form, the addition of pose optimisation, and the full SLAM system using a tracking and mapping framework. A system diagram is given in Fig. 2.

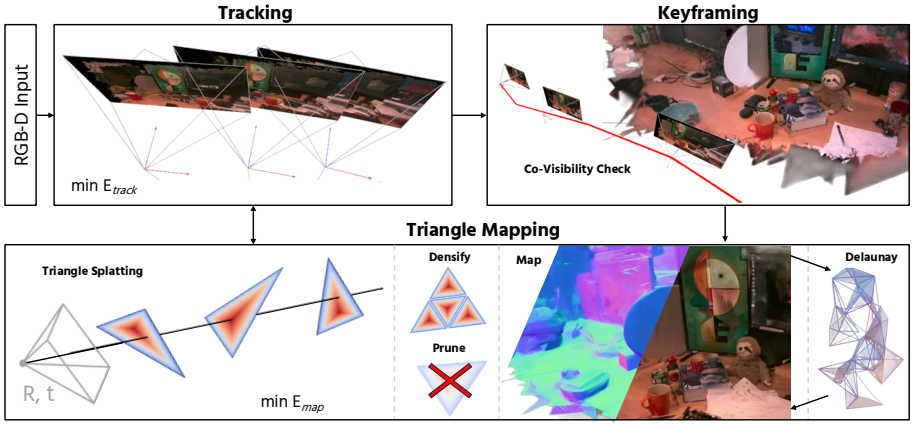

Fig. 2: SLAM System Overview. **Tracking (top left)**: RGB-D frames are processed to estimate camera poses. **Keyframing (top right)**: Frames with high co-visibility are selected as keyframes. **Mapping (bottom)**: A triangle-based map is continually optimised via triangle splatting, with adaptive densification and pruning. Triangles can be converted into a connected mesh through restricted Delaunay triangulation.

3.1 Mesh Splatting

The map representation for our SLAM system builds upon Mesh Splatting [7], where the scene is represented by a set of N vertices $\mathcal{V}$.

Each vertex $\mathbf{v}_i \in \mathcal{V}$, $i = 1, \ldots, N$ is parameterised as $\mathbf{v}_i = (x_i, y_i, z_i, \mathbf{c}_i, \mathbf{o}_i)$; the 3D vertex position being $(x_i, y_i, z_i) \in \mathbb{R}^3$ in world space, $\mathbf{c}_i \in \mathbb{R}^3$ denotes the colour and $\mathbf{o}_i \in [0, 1]$ the vertex opacity. A triangular face $\mathbf{F}_m$ is defined with three vertices $\mathbf{F}_m = \{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$. The topology is defined by a set of faces $\mathcal{F} = \{\mathbf{f}_1, \ldots, \mathbf{f}_M\}$, where each face $\mathbf{f}_m = (i_m, j_m, k_m)$ is an ordered triplet of vertex indices specifying the triangle. This separation of geometry and topology allows for dynamic re-indexing, enabling online Delaunay triangulation when a connected mesh is required.

The colour contribution to a pixel inside a triangle's 2D projection is obtained by interpolating the vertex colours using barycentric coordinates: $\mathbf{c}_{\mathbf{F}_m}(\lambda_i, \lambda_j, \lambda_k) = \lambda_i \mathbf{c}_i + \lambda_j \mathbf{c}_j + \lambda_k \mathbf{c}_k$ where $\lambda_i + \lambda_j + \lambda_k = 1$ and $\lambda_i, \lambda_j, \lambda_k \geq 0$. Note that this vertex-shared parameterisation differs from that of Triangle Splatting [8] where each face $\mathbf{F}_m$ has a single colour $\mathbf{c}_m$. The mesh parameterisation enables feature sharing across adjacent triangles, so that changes to connectivity do not also introduce discontinuities in the vertex attributes. In a departure from Mesh Splatting, which sets the face opacity to the minimum of its vertices ($\mathbf{o}_{\mathbf{F}_m} = \min(\mathbf{o}_i, \mathbf{o}_j, \mathbf{o}_k)$), we define the face opacity as the average of its constituent vertex opacities: $\mathbf{o}_{\mathbf{F}_m} = \frac{1}{3}(\mathbf{o}_i + \mathbf{o}_j + \mathbf{o}_k)$. The minimum operator is discontinuous and routes gradients solely to the vertex with the lowest opacity. By using the average, we ensure that all three point opacities receive gradients and are jointly optimised, which improves the map quality in our online system.

3.2 Differentiable Rasterisation

To ensure the rasterisation process is differentiable, Held et al. [8] introduce a 2D signed distance field ϕ of the triangle in image space. For each triangle $\mathbf{F}_m = \{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$, the 3D vertices are projected into image space via:

$$\mathbf{v}_I = \pi(\mathbf{T}_{CW}\mathbf{v}_W) \ , \tag{1}$$

where $\mathbf{v}_W \in \{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ denotes a vertex in world space (expressed in homogeneous coordinates for the transformation), $\mathbf{T}_{CW} \in \mathbf{SE}(3)$ is the camera pose, and π is the projection operator. The influence of a pixel $\mathbf{p}$ is then defined as a window function $I(\mathbf{p})$ based on the signed distance field (SDF) ϕ of the projected 2D face [8]. The SDF is given by the maximum of the linear edge functions:

$$\phi(\mathbf{p}) = \max_{i\in\{1,2,3\}} L_i(\mathbf{p}), \quad \text{where} \quad L_i(\mathbf{p}) = \mathbf{n}_i \cdot \mathbf{p} + d_i \ , \tag{2}$$

Here, $\mathbf{n}_i$ are the outward-facing unit normals of the 2D edges, and d_i are their corresponding offsets, computed directly from the projected vertices $\mathbf{v}_I^{(i)}$. We omit the details of their computation for simplicity. The window function is then defined relative to the face's incentre $\mathbf{s} \in \mathbb{R}^2$, the point where the SDF is minimised:

$$I(\mathbf{p}) = \text{ReLU}\left(\frac{\phi(\mathbf{p})}{\phi(\mathbf{s})}\right)^{\sigma} \ , \tag{3}$$

where $\sigma > 0$ is a parameter controlling the smoothness of the function. In Triangle Splatting the σ parameter is learnable per-triangle, whereas in Mesh Splatting it is a global constant shared across all triangles. The formulation ensures that the influence $I(\mathbf{p})$ is exactly 1 at the incentre of the face, decays to 0 at the face boundaries, and remains 0 everywhere outside the face. After projection the colour for each pixel $\mathbf{p}$ is accumulated in the following manner:

$$C(\mathbf{p}) = \sum_{n=1}^{N} \mathbf{c}_{\mathbf{F}_n} \mathbf{o}_{\mathbf{F}_n} I_n(\mathbf{p}) \left(\prod_{i=1}^{n-1} (1 - \mathbf{o}_{\mathbf{F}_i} I_i(\mathbf{p}))\right), \tag{4}$$

where the faces are sorted in depth order.

3.3 Analytic Camera Pose Jacobian

Camera pose optimisation typically requires ∼80 iterations of gradient descent to converge. Computing gradients via automatic differentiation incurs substantial overhead, so we instead derive analytical pose Jacobians and evaluate them directly in the backward pass of the CUDA kernel, which significantly reduces tracking runtime. We represent pose updates as elements of $\mathfrak{se}(3)$ [32] and derive the Jacobian of the rendering loss with respect to the pose. Applying the chain rule through projection and the world-to-camera transformation we get:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{T}_{CW}} = \frac{\partial \mathcal{L}}{\partial \mathbf{v}_I} \frac{\partial \mathbf{v}_I}{\partial \mathbf{v}_C} \frac{\mathcal{D}\mathbf{v}_C}{\mathcal{D}\mathbf{T}_{CW}}, \tag{5}$$

where $\mathbf{v}_C$ denotes a vertex in camera space. Taking partial derivatives on the manifold and using the same notation as [32] where $\tau \in \mathfrak{se}(3)$ we define:

$$\frac{\mathcal{D}f(\boldsymbol{T})}{\mathcal{D}\boldsymbol{T}} \triangleq \lim_{\tau \to 0} \frac{\mathrm{Log}(f(\mathrm{Exp}(\tau) \circ \boldsymbol{T}) \circ f(\boldsymbol{T})^{-1})}{\tau}, \tag{6}$$

And thus derive the following Jacobian of vertices with respect to the pose:

$$\frac{\mathcal{D}\mathbf{v}_C}{\mathcal{D}\mathbf{T}_{CW}} = \begin{bmatrix} \mathbf{I}_3 \ , & -[\mathbf{v}_C]_\times \end{bmatrix}, \tag{7}$$

where $\mathbf{v}_\times$ denotes the skew symmetric matrix of the 3D vector.

3.4 Tracking and Mapping

We follow a sequential single-processed tracking and mapping framework where the camera pose is optimised to convergence after which the mapping module performs joint optimisation of camera poses and geometry.

Tracking The tracking frontend estimates the camera pose of the latest frame with respect to the map. This is achieved by minimising a joint objective comprising both photometric and geometric error. Incorporating an optimised Structural Similarity (SSIM) loss for more stable training [19], we define E_{pho} as a weighted combination of the per-pixel ℓ_1 norm and the D-SSIM term between the image rendered from the triangles, $I(\mathcal{V}, \mathbf{T}_{CW})$, and the observed reference image $\bar{I}$:

$$E_{pho} = (1 - \lambda)\|I(\mathcal{V}, \mathbf{T}_{CW}) - \bar{I}\|_1 + \lambda \mathcal{L}_{\text{D-SSIM}}(I(\mathcal{V}, \mathbf{T}_{CW}), \bar{I}) \ . \tag{8}$$

We also add a supervised depth tracking term. The depth loss E_{dep} penalises the ℓ_1 difference between the rendered depth map $D(\mathcal{V}, \mathbf{T}_{CW})$ and the observed depth $\bar{D}$:

$$E_{dep} = \|D(\mathcal{V}, \mathbf{T}_{CW}) - \bar{D}\|_1 \ . \tag{9}$$

The final tracking loss E_{track} is formulated as the weighted sum of these terms, which is minimised to optimise the camera pose $\mathbf{T}_{CW} \in \mathbf{SE}(3)$:

$$E_{track} = E_{pho} + \lambda_{dep} E_{dep} \ . \tag{10}$$

Keyframing Keyframing is based on co-visibility checks, similar to MonoGS [21]. We keep track of triangles that were rendered from any keyframe during mapping, and a new keyframe is added based on the intersection-over-union of triangles with respect to the previous keyframe. We maintain a keyframe buffer which also stores a count of triangles that were visible when that keyframe was rendered. Once a keyframe has been registered, we initiate the mapping phase.

Mapping During mapping, we re-render from a selection of previous keyframes within the keyframe buffer and optimise both the keyframe poses and the map. We select 4 frames based on maximum covisibility, 2 random frames, and the most recently added keyframe. These are then replayed such that each keyframe is rendered for 30 iterations. The mapping loss E_{map} is formulated to jointly optimise for photometric reconstruction and geometry.

Photometric Supervision The photometric error E_{pho} penalises the difference between the rendered image $I(\mathcal{V}, \mathbf{T}_{CW})$ and the reference image $\bar{I}$. We use the same E_{pho} from the earlier tracking phase.

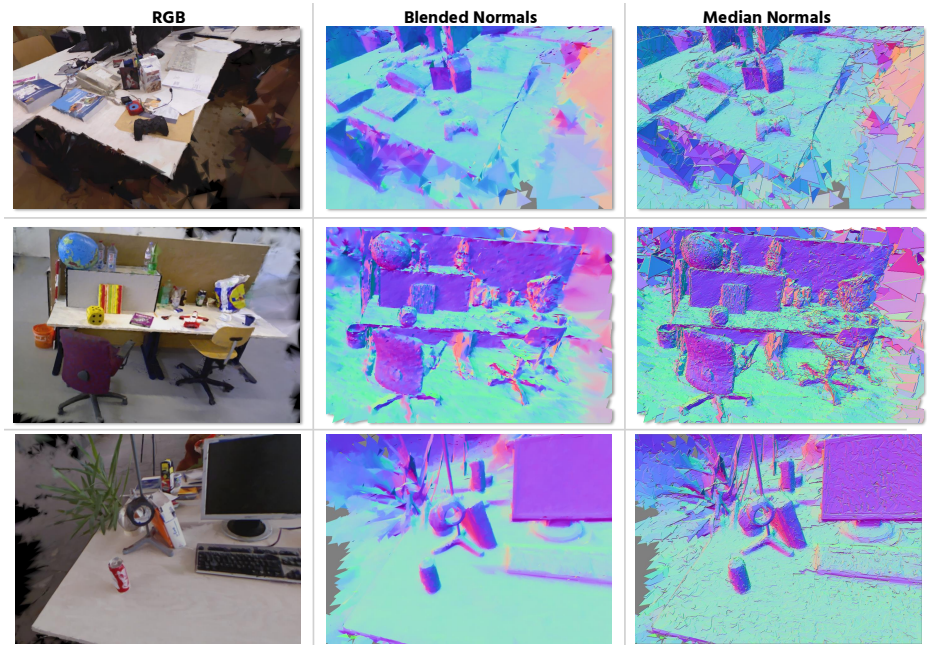

Fig. 3: Qualitative results of our method on the TUM RGB-D dataset. **(left)**: RGB rendering of the reconstructed map. **(middle)**: Average normal direction of all triangles intersecting each camera ray. **(right)**: Normal direction of the triangle with highest opacity along each ray. Overall, our method produces photorealistic results and accurate surface reconstructions.

Geometric Supervision Upon keyframing, we add new triangles to the scene by back-projecting RGB-D measurements and initialising an equilateral triangle around each point. A triangle is spawned with its points on a sphere of radius r_i, determined by the distance to its nearest neighbour, ensuring triangles are sized appropriately to the local point cloud density.

Following 4DTAM [20], we compute surface normal estimates directly from sensor depth measurements to initialise the triangle orientation. We compute

these normal maps by taking finite differences of points in the back-projected depth. The sensor normals $\bar{\mathbf{N}}$ are thus formulated:

$$\bar{\mathbf{N}} = \frac{\nabla_x \mathbf{p}_d \times \nabla_y \mathbf{p}_d}{\|\nabla_x \mathbf{p}_d \times \nabla_y \mathbf{p}_d\|} \ , \tag{11}$$

where $\mathbf{p}_d$ denotes the back-projected depth points. These normal maps are stored within the keyframes and used as direct supervision during mapping. The normal regularisation loss E_{norm} aligns the rendered geometry with the sensor data by penalising the cosine distance between the rendered normal map $N(\mathcal{V}, \mathbf{T}_{CW})$ returned by the rasteriser, and the pre-computed sensor normal map $\bar{\mathbf{N}}$:

$$E_{norm} = \sum_{\mathbf{p} \in \mathcal{P}} \left(1 - N(\mathcal{V}, \mathbf{T}_{CW})_{\mathbf{p}}^T \bar{\mathbf{N}}_{\mathbf{p}}\right) \ , \tag{12}$$

where $\mathcal{P}$ denotes the set of all valid pixels. Additionally, we apply a depth loss E_{dep} that penalises the ℓ_1 difference between the rendered depth map $D(\mathcal{V}, \mathbf{T}_{CW})$ and the reference depth $\bar{D}$, further encouraging geometrically accurate reconstructions:

$$E_{dep} = \|D(\mathcal{V}, \mathbf{T}_{CW}) - \bar{D}\|_1 \ . \tag{13}$$

We also introduce a regularisation term E_{equi} to prevent the formation of degenerate, elongated triangles. For each face $\mathbf{f}_m \in \mathcal{F}$, we calculate the cosine of its three internal angles $\theta_{m,1}, \theta_{m,2}, \theta_{m,3}$ via the dot product of its normalised edge vectors. The loss penalises the difference between these cosines and $\cos(60°) = 0.5$:

$$E_{equi} = \frac{1}{|\mathcal{F}|} \sum_{\mathbf{f}_m \in \mathcal{F}} \frac{1}{3} \sum_{n=1}^{3} \left(\cos \theta_{m,n} - 0.5\right)^2 \ . \tag{14}$$

The total mapping loss is the weighted sum of these visual, geometric, and regularisation objectives:

$$E_{map} = E_{pho} + \lambda_{dep} E_{dep} + \lambda_{norm} E_{norm} + \lambda_{equi} E_{equi} \ . \tag{15}$$

Pruning At every keyframe, unstable or redundant triangles are pruned from the map. This step is important to ensure the Delaunay triangulation does not contain geometrically inconsistent triangles, or those with low opacity in the blended render. A triangle is marked for removal if its mean vertex opacity falls below a threshold ϵ_o, or if its projected image-space area exceeds a maximum size ϵ_a, preventing large, under-resolved triangles from persisting in the map.

Densification Primitives are added also during the mapping phase using the refinement strategy proposed by Fang et al. [3] where triangles whose projected

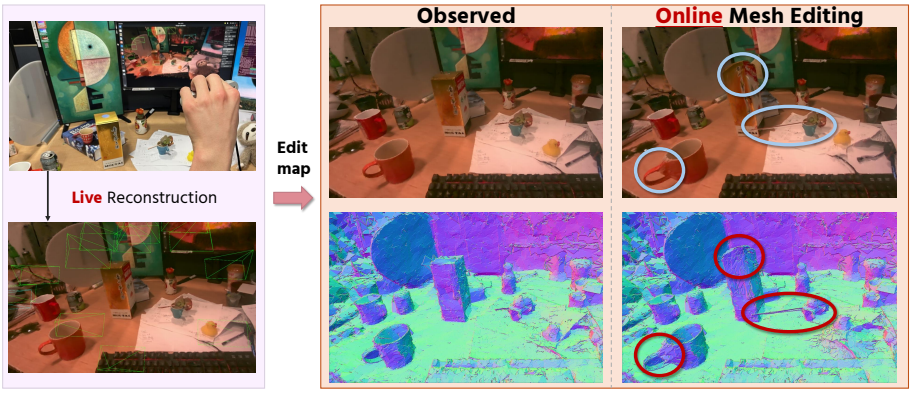

Fig. 4: Online mesh editing. The triangle mesh representation supports online editing after Delaunay triangulation. Edited regions are circled in both RGB and surface normal renderings. Notably, our method handles thin structures such as the chameleon's tongue and the mug's handle **(circled)**. It also maintains realistic appearance which differs from methods which use TSDF Fusion.

pixel coverage exceeds a blur threshold $\theta_{blur} \cdot H \cdot W$ are selected as split candidates, to reduce blurry regions. Only triangles that are visible in the current replay window are eligible for densification.

We perform triangle splitting using the strategy proposed in [16] where each selected primitive is subdivided into four smaller ones by introducing midpoint vertices along each edge. The positions, colours, and opacities of the three midpoint vertices are initialised by linear interpolation between their respective parent vertices. The original triangles are then removed and replaced by their four children. While Mesh Splatting [7] proposed sharing vertices between triangles, even during the triangle soup stage of training, we found that this constrains points during optimisation and leads to poor convergence.

4 Experiments

4.1 Datasets & Baseline

We benchmark our method using both TUM and Replica datasets following [21, 33]. We compare with similar map-centric RGB-D SLAM methods with explicit and implicit representations. For depth and Chamfer distance we compare with Gaussian Splatting SLAM (MonoGS) and our re-implementation of MonoGS using 2D Gaussian Splatting [9], which we label MonoGS-2D*. We also compare with the results presented in 4DTAM [20]. To extract a mesh with TSDF Fusion we follow the same procedure proposed in [9]. We also compare to a 'pruned' version of our Delaunay mesh, where we remove triangles that are occluded in all training views. More details of the setup are provided in the supplementary material. Qualitative examples of the rendering are shown in Fig. 1 and Fig. 3.

Table 1: ATE (cm) RGB-D camera tracking results on TUM.

Input	Loop Closure	Method	fr1/desk	fr2/xyz	fr3/office	Avg.
RGB-D	w/o	iMAP	4.90	2.00	5.80	4.23
		NICE-SLAM	4.26	6.19	3.87	4.77
		DI-Fusion	4.40	2.00	5.80	4.07
		Vox-Fusion	3.52	1.49	26.01	10.34
		Co-SLAM	2.40	1.70	2.40	2.17
		Point-SLAM	4.34	1.31	3.48	3.04
		MonoGS	**1.50**	1.44	**1.49**	**1.47**
		MonoGS-2D	<u>1.58</u>	<u>1.20</u>	<u>1.83</u>	<u>1.54</u>
		Ours	1.77	**1.12**	<u>1.83</u>	1.57
	w/	BAD-SLAM	1.70	1.10	1.70	1.50
		Kintinous	3.70	2.90	3.00	3.20
		ORB-SLAM2	**1.60**	**0.40**	**1.00**	**1.00**

4.2 Quantitative Results

We report Root Mean Squared Error (RMSE) for the Absolute Tracking Error (ATE) of keyframes in Tab. 1 and Tab. 2. L1 depth is shown in Tab. 3 and Chamfer Distance in Tab. 4 on the Replica dataset. In Tab. 4 we also provide the mesh generation times of the different methods. We achieve comparable ATE in both datasets and outperform the baselines on geometry.

While MonoGS-2D* achieves the most accurate pose estimation and mean L1 depth error, this does not result in more accurate geometry from a 3D perspective. The mean depth error reveals most frames with low error, but inaccurate depth maps exist in areas that are not viewed often. While this is not enough to skew the mean depth metrics, after performing TSDF fusion, the Chamfer distance degrades in these areas. Measuring the Chamfer distance also more directly measures the accuracy and utility of dense geometry. For many tasks, achieving a consistent and accurate 3D reconstruction is of greater importance than depth estimation, so we highlight the ability of our method to achieve the best upper bound using TSDF reconstruction, while also providing a more flexible and photorealistic reconstruction using Delaunay triangulation. In this way, our method is particularly well-suited for on-the-fly reconstruction where a mesh is required at any given moment, as evidenced by the live demonstrations in the supplementary video and illustrated in Fig. 4.

Fig. 5 shows a comparison between the meshes generated by MonoGS-2D* and those generated by our method using Delaunay and TSDF Fusion. While Delaunay meshing produces some irregular surfaces in unobserved regions, it more faithfully reconstructs the given input sequence as the surface is directly supervised during training. MonoGS-2D* notably produces artifacts in areas with few observations and depth outliers, which are not present in the TSDF meshes produced by our depth maps. This is reflected in Tab. 4 where our method achieves better Chamfer distance in all scenes for both Delaunay and TSDF Fusion of depth maps.

Table 2: ATE (cm) RGB-D camera tracking results on Replica.

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 [41]	0.97	1.31	1.07	0.88	1.00	1.06	1.10	1.13	1.07
Vox-Fusion [39]	1.37	4.70	1.47	8.48	2.04	2.58	1.11	2.94	3.09
ESLAM [10]	0.71	0.70	0.52	0.57	0.55	0.58	0.72	0.63	0.63
Point-SLAM [28]	0.61	0.41	0.37	0.38	0.48	0.54	0.69	0.72	0.53
MonoGS [21]	0.44	0.32	0.31	0.44	0.52	0.23	0.17	2.25	0.58
MonoGS (sp)	**0.33**	**0.22**	0.29	0.36	0.19	0.25	**0.12**	0.81	0.32
MonoGS-2D [20]	0.42	0.43	0.35	0.19	0.19	0.22	0.27	0.80	0.36
MonoGS-2D*	0.55	0.32	0.34	**0.15**	**0.10**	**0.09**	0.14	**0.22**	**0.24**
Ours	0.36	0.41	**0.25**	0.32	0.30	0.45	0.73	0.55	0.34

Fig. 5: Qualitative results on Replica. Comparison of three methods ordered by ascending reconstruction accuracy (left to right). Our method with TSDF fusion achieves the highest quality; our method with Delaunay triangulation offers the best accuracy-speed trade-off while also enabling novel capabilities.

Tab. 4 also highlights a trade-off between geometric accuracy and computational efficiency. Whilst TSDF fusion on our depth maps yields the lowest average Chamfer Distance (0.95 cm), it requires an average of 33.44 seconds to process a scene. In contrast, pruned Delaunay triangulation provides an effective middle ground. It achieves an average Chamfer Distance of 1.14 cm, remaining competitive with the TSDF upper bound and strictly outperforms MonoGS-2D* (1.36 cm). The mesh generation time of the pruned Delaunay method is less than half that of TSDF fusion (15.66 seconds versus 33.44 seconds). The unpruned Delaunay variant is faster (11.18 seconds) at the cost of slightly increased geometric error (1.55cm), due to the retention of extraneous faces in unobserved or poorly constrained regions. Ultimately, this demonstrates the flexibility of our representation: it provides the capacity for reconstruction via TSDF, and simultaneously enables on-the-fly mesh generation via Delaunay triangulation without sacrificing significant geometric accuracy. We provide further analysis of these extraction methods, including performance across varying TSDF voxel resolutions, in the supplementary material.

Table 3: Depth L1 error (cm) on the Replica dataset ↓.

Method	r0	r1	r2	o0	o1	o2	o3	o4	Avg.
MonoGS	3.00	3.47	4.66	3.10	6.08	6.15	4.77	4.94	4.52
MonoGS-2D*	1.44	0.78	**0.80**	**0.44**	0.36	0.59	**0.87**	**0.68**	0.74
Ours	**0.77**	**0.70**	0.91	0.48	**0.33**	**0.58**	0.95	0.74	**0.68**

Table 4: Quantitative evaluation and computational trade-offs on the Replica dataset. We report both reconstruction quality (Chamfer Distance in cm ↓, evaluated with 1M samples) and mesh generation time (wall clock in seconds ↓). For TSDF extraction, we render depth maps from MonoGS (sp), MonoGS-2D*, and our representation before performing fusion. We compare this against running Restricted Delaunay triangulation on the triangle soup. TSDF Fusion on our depth maps achieves the highest geometric accuracy, however the pruned Delaunay triangulation offers highly competitive quality at less than half the computational time. Best results are **bolded** and second best are underlined within their respective sections.

Method	r0	r1	r2	o0	o1	o2	o3	o4	Avg.
Reconstruction Quality: Chamfer Distance (cm) ↓									
MonoGS + TSDF	3.76	4.39	4.63	2.93	4.78	4.18	4.36	3.26	4.03
MonoGS-2D* + TSDF	1.93	1.16	1.54	1.56	0.70	1.49	1.37	1.15	1.36
Ours + TSDF	**1.00**	**0.77**	**1.01**	**0.66**	**0.56**	**1.12**	**1.69**	**0.78**	**0.95**
Ours + Delaunay	1.99	1.35	1.41	1.23	1.18	1.55	2.38	1.28	1.55
Ours + Delaunay (pruned)	1.16	1.01	1.09	0.77	0.69	1.37	2.00	1.02	1.14
Mesh Generation Time (seconds) ↓									
Ours + TSDF	47.40	25.10	32.20	25.60	13.50	33.90	50.20	39.60	33.44
Ours + Delaunay	**9.04**	**11.63**	**13.19**	**9.19**	**8.37**	**12.53**	**10.36**	**15.10**	**11.18**
Ours + Delaunay (pruned)	13.05	15.92	18.33	13.34	11.78	17.80	14.99	20.10	15.66

4.3 Runtime and Scaling Analysis

We provide a runtime breakdown and system scaling analysis with map size in Tab. 5. We report per-scene FPS, *average* per-frame latency, number of triangles and peak GPU usage with all components enabled across the TUM sequences and one Replica scene. The system runs live and interactively as shown in our video demonstration. The analysis shows that latency is generally correlated with map size, however sequences which keyframe less often can run much faster (e.g. fr2/xyz) as they perform fewer mapping iterations. We measure the map size using the model checkpoints, which indicate a low memory footprint.

4.4 Qualitative Capabilities

The primary advantage of differentiable triangles as a representation lies in its capacity to provide both realistic rendering and a connected mesh, ensuring its immediate applicability to downstream tasks. To highlight this, we demonstrate live reconstruction that directly facilitates interactive, mesh-based scene editing. Whilst Fig. 4 illustrates just one example of this capability, our supplementary video provides a comprehensive demonstration of the system in motion. This

Table 5: TUM and Replica latency and memory usage.

	fr1/desk	**fr2/xyz**	**fr3/office**	**office1**
FPS ↑	0.82	2.33	1.29	0.55
Latency (ms) ↓	1225	429	775	1809
Frontend (ms)	584	352	494	892
Backend (ms)	642	77	282	918
Triangles	29.0k	24.0k	34.4k	152.2k
Model size (MB) ↓	5.0	4.4	7.6	16.4
Peak GPU usage (GB)	0.47	0.47	0.53	1.25

combination of visual fidelity and explicitly connected geometry is highly valuable for augmented reality experiences and collaborative design. Crucially, our method can incrementally update the mesh topology during the mapping phase, enabling complex, connectivity-based operations, including structural deformations and physical collisions, without requiring a separate offline extraction step.

5 Conclusion and Future Work

We presented an RGB-D SLAM system using triangles as a unified representation for photorealistic rendering, camera localisation, and 3D mapping. Beyond accurate trajectory estimation and consistent geometry, our approach unlocks online mesh-based scene editing and potential for live interactive digital twins.

Several directions remain for future work. First, the topological quality of the reconstructed mesh could be improved by incorporating constraints that enforce manifoldness and prevent self-intersections during optimisation, producing meshes more suitable for applications such as rigging. Second, recent work on differentiable triangle soup optimisation [35] has demonstrated extreme map simplification while preserving visual fidelity; integrating such compression into our pipeline could substantially reduce the number of primitives without sacrificing reconstruction quality. To achieve hard real-time (30fps) we would add multiprocessing, conjugate gradients [27] and CUDA optimisations [6]. We prioritise a unified representation for tracking, mapping and meshing, however sparse bundle adjustment would also accelerate camera pose estimation. Finally, extending the system to monocular RGB input by tuning or using a strong prior [15] would broaden its applicability to more challenging sequences and outdoor reconstruction. Ultimately, this work demonstrates differentiable triangles can be used as a powerful, versatile representation for real-time spatial computing.

Acknowledgements

This research is funded by the EPSRC On-Sensor Computer Vision grant (EP/Y020499/1). We would like to thank Ignacio Alzugaray, Shinjeong Kim, Marwan Taher, Hidenobu Matsuki, Riku Murai and other members of the Robot Vision Group for insightful discussions.

References

1. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In: IEEE Conf. Comput. Vis. Pattern Recog. (2022)
2. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of SIGGRAPH (1996)
3. Fang, G., Wang, B.: Mini-Splatting: Representing scenes with a constrained number of Gaussians. In: Eur. Conf. Comput. Vis. (2024)
4. Govindarajan, S., Rebain, D., Yi, K.M., Tagliasacchi, A.: Radiant Foam: Real-time differentiable ray tracing. In: Int. Conf. Comput. Vis. (2025)
5. Guédon, A., Gomez, D., Maruani, N., Gong, B., Drettakis, G., Ovsjanikov, M.: MILo: Mesh-in-the-loop Gaussian splatting for detailed and efficient surface reconstruction. ACM Trans. Graph. (2025)
6. Hahlbohm, F., Franke, L., Eisemann, M., Magnor, M.: Faster-GS: Analyzing and improving Gaussian splatting optimization. In: IEEE Conf. Comput. Vis. Pattern Recog. (2026)
7. Held, J., Son, S., Vandeghen, R., Rebain, D., Gadelha, M., Zhou, Y., Cioppa, A., Lin, M.C., Van Droogenbroeck, M., Tagliasacchi, A.: MeshSplatting: Differentiable rendering with opaque meshes. In: IEEE Conf. Comput. Vis. Pattern Recog. (2026)
8. Held, J., Vandeghen, R., Deliege, A., Hamdi, A., Rebain, D., Giancola, S., Cioppa, A., Vedaldi, A., Ghanem, B., Tagliasacchi, A., et al.: Triangle splatting for real-time radiance field rendering. In: Int. Conf. 3D Vis. (2025)
9. Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D Gaussian splatting for geometrically accurate radiance fields. In: Proceedings of SIGGRAPH (2024)
10. Johari, M.M., Carta, C., Fleuret, F.: ESLAM: Efficient dense SLAM system based on hybrid representation of signed distance fields. In: IEEE Conf. Comput. Vis. Pattern Recog. (2023)
11. Keetha, N., Karhade, J., Jatavallabhula, K.M., Yang, G., Scherer, S., Ramanan, D., Luiten, J.: SplaTAM: Splat, track & map 3D Gaussians for dense RGB-D SLAM. In: IEEE Conf. Comput. Vis. Pattern Recog. (2024)
12. Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., Kolb, A.: Real-time 3D reconstruction in dynamic scenes using point-based fusion. In: Int. Conf. 3D Vis. (2013)
13. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3D Gaussian splatting for real-time radiance field rendering. ACM Trans. Graph. (2023)
14. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR) (2007)
15. Leroy, V., Cabon, Y., Revaud, J.: Grounding image matching in 3D with MASt3R. In: Eur. Conf. Comput. Vis. (2024)
16. Loop, C.T.: Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah (1987)
17. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3D surface construction algorithm. In: Proceedings of SIGGRAPH (1987)
18. Mai, A., Hedstrom, T., Kopanas, G., Kontkanen, J., Kuester, F., Barron, J.T.: Radiance meshes for volumetric reconstruction. In: IEEE Conf. Comput. Vis. Pattern Recog. (2026)
19. Mallick, S.S., Goel, R., Kerbl, B., Steinberger, M., Carrasco, F.V., De La Torre, F.: Taming 3DGS: High-quality radiance fields with limited resources. In: Proceedings of SIGGRAPH Asia (2024)

20. Matsuki, H., Bae, G., Davison, A.J.: 4DTAM: Non-rigid tracking and mapping via dynamic surface Gaussians. In: IEEE Conf. Comput. Vis. Pattern Recog. (2025)
21. Matsuki, H., Murai, R., Kelly, P.H., Davison, A.J.: Gaussian Splatting SLAM. In: IEEE Conf. Comput. Vis. Pattern Recog. (2024)
22. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. In: Eur. Conf. Comput. Vis. (2020)
23. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans. Graph. (2022)
24. Mur-Artal, R., Tardós, J.D.: ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. IEEE Transactions on Robotics (T-RO) (2017)
25. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon, A.: KinectFusion: Real-time dense surface mapping and tracking. In: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR) (2011)
26. Nießner, M., Zollhöfer, M., Izadi, S., Stamminger, M.: Real-time 3D reconstruction at scale using voxel hashing. ACM Trans. Graph. (2013)
27. Pehlivan, H., Boscolo Camiletto, A., Foo, L.G., Habermann, M., Theobalt, C.: Matrix-free second-order optimization of Gaussian splats with residual sampling. In: Int. Conf. 3D Vis. (2026)
28. Sandström, E., Li, Y., Van Gool, L., R. Oswald, M.: Point-SLAM: Dense neural point cloud-based SLAM. In: Int. Conf. Comput. Vis. (2023)
29. Schöps, T., Sattler, T., Pollefeys, M.: BAD SLAM: Bundle adjusted direct RGB-D SLAM. In: IEEE Conf. Comput. Vis. Pattern Recog. (2019)
30. Schöps, T., Sattler, T., Pollefeys, M.: SurfelMeshing: Online surfel-based mesh reconstruction. IEEE Trans. Pattern Anal. Mach. Intell. (2019)
31. Shen, T., Gao, J., Yin, K., Liu, M.Y., Fidler, S.: Deep Marching Tetrahedra: a hybrid representation for high-resolution 3D shape synthesis. In: Adv. Neural Inform. Process. Syst. (2021)
32. Sola, J., Deray, J., Atchuthan, D.: A micro lie theory for state estimation in robotics. arXiv preprint arXiv:1812.01537 (2018)
33. Sucar, E., Liu, S., Ortiz, J., Davison, A.J.: iMAP: Implicit mapping and positioning in real-time. In: Int. Conf. Comput. Vis. (2021)
34. Teed, Z., Deng, J.: DROID-SLAM: Deep visual SLAM for monocular, stereo, and RGB-D cameras. In: Adv. Neural Inform. Process. Syst. (2021)
35. Tojo, K., Bickel, B., Umetani, N.: DiffSoup: Direct differentiable rasterization of triangle soup for extreme radiance field simplification. In: IEEE Conf. Comput. Vis. Pattern Recog. (2026)
36. Whelan, T., Johannsson, H., Kaess, M., Leonard, J.J., McDonald, J.: Robust real-time visual odometry for dense RGB-D mapping. In: IEEE Int. Conf. Robot. Autom. (2013)
37. Whelan, T., Leutenegger, S., Salas-Moreno, R.F., Glocker, B., Davison, A.J.: ElasticFusion: Dense SLAM without a pose graph. In: Proceedings of Robotics: Science and Systems (RSS) (2015)
38. Yan, C., Qu, D., Xu, D., Zhao, B., Wang, Z., Wang, D., Li, X.: GS-SLAM: Dense visual SLAM with 3D Gaussian splatting. In: IEEE Conf. Comput. Vis. Pattern Recog. (2024)
39. Yang, X., Li, H., Zhai, H., Ming, Y., Liu, Y., Zhang, G.: 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)

40. Yu, Z., Sattler, T., Geiger, A.: Gaussian Opacity Fields: Efficient adaptive surface reconstruction in unbounded scenes. ACM Trans. Graph. (2024)
41. Zhu, Z., Peng, S., Larsson, V., Xu, W., Bao, H., Cui, Z., Oswald, M.R., Pollefeys, M.: NICE-SLAM: Neural implicit scalable encoding for SLAM. In: IEEE Conf. Comput. Vis. Pattern Recog. (2022)