# LiDAR-Integrated Coarse-to-Fine Optimization for Geometrically Consistent Triangle Splatting from Mobile Robots

Byoungkwon Yoon[1], Hojun Lee[1], Yuseop Sim[1], Hangyeom Lee[1], Dongjun Lee[2], and Martin Byung-Guk Jun[1]

***Abstract*— Recent advancements in 3D Gaussian Splatting (3DGS) have enabled efficient 3D reconstruction from object-centric images, but extracting reliable geometry remains challenging. Most 3DGS-based methods treat mesh extraction as a post-processing step, so high rendering quality does not necessarily imply high surface quality. Triangle Splatting (TS) addresses this limitation by using triangles as differentiable primitives, directly coupling image optimization with mesh reconstruction. However, under ego-centric mobile robot trajectories, which are often sparse and monotonic, TS can suffer from *lazy optimization*, where primitives overfit to limited training views instead of reconstructing the true scene geometry. To address this problem, we propose a LiDAR-guided coarse-to-fine optimization framework for geometrically consistent TS from mobile robot data. Our method initializes triangle primitives from LiDAR mesh-SLAM output by converting reconstructed surface meshes into decoupled and optimizable primitives. It then refines the reconstruction using LiDAR depth supervision and geometry-aware densification based on per-triangle depth and normal error variances. This design improves geometric consistency while preserving the coupled image-geometry representation of TS. Experiments on synthetic and real-world ego-centric datasets show that our method improves geometric accuracy while maintaining high photometric quality. We further demonstrate that the reconstructed meshes provide more reliable collision geometry than the TS baseline in a physics-based robot simulation. The code will be made publicly available at `https://purduelamm.github.io/lidar-ts-page/`.**

## I. Introduction

3D reconstruction is a fundamental task in computer vision and robotics for autonomous systems to interpret their surroundings through multi-view imagery or spatial sensors. Recently, the emergence of radiance field methods, such as Neural Radiance Fields (NeRF) [1] and 3D Gaussian Splatting (3DGS) [2], has opened new possibilities for spatial representation. Specifically, 3DGS utilizes explicit and differentiable primitives, and is increasingly used in robotics for navigation [3], simulation [4], and the creation of digital twins [5]. For these applications to be effective, extracting reliable geometric information from these primitives is essential [6].

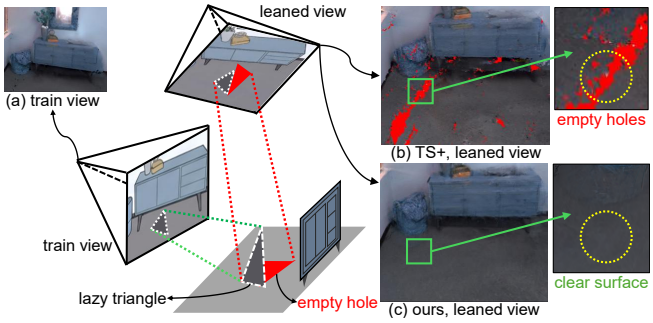


Fig. 1. Comparison of leaned views: Regions highlighted in red indicate holes/missing regions in the reconstruction. TS+ refers to the original Triangle Splatting+ method

A common approach to utilizing 3DGS as spatial information is to extract a mesh from the learned primitives [4], [6]. This step is crucial for compatibility with standard collision detection pipelines and physics engines, which inherently rely on explicit mesh geometry. Yet, extracting high-quality surfaces from Gaussians remains difficult due to their unstructured and unordered nature [7]. A critical limitation is the disconnection between rendering quality and geometric accuracy. Since Gaussian optimization and mesh extraction are typically decoupled, high-fidelity view synthesis does not guarantee a geometrically consistent 3D mesh. To bridge this gap, new triangle-based primitive splatting methods [8], [9], [10] have recently emerged, adopting the primitives for differentiable rendering. Utilizing the triangle, which is the fundamental unit of traditional meshes, these approaches naturally unify the image-based optimization process with the resulting geometric structure.

Despite the success of these methods to addressing the geometric inconsistency of standard 3DGS, their application to the autonomous mobile robots faces a distinct hurdle of non-object-centric data. While most differentiable splatting methods are evaluated on object-centric camera datasets [11], robotic vision data is often ego-centric, sparse, and outward-facing, following a monotonic trajectory. Consequently, the dense, multi-view coverage assumed in the original 3DGS formulation is rarely met in real-world robotics. Under such ill-posed conditions that frequently occur in practical robotics applications, primitives tend to overfit to the training images, creating a large discrepancy with the real-world geometry. This problem of *lazy optimization* in 3DGS [12] occurs since

*This research was supported in part by the Korea Institute of Machinery & Materials, Republic of Korea, under the project name of "Development of Core Technologies for Cognitive/Adaptive Digital Machine Tools toward Autonomous Operations (grant number: NK255A) and in part by the Technology Innovation Program (or Industrial Strategic Technology Development Program-Industry Technology Alchemist Project) (20025702, Development of smart manufacturing multiverse platform based on multi-sensory fusion avatar and interactive AI) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).

Corresponding author: Martin Byung-Guk Jun

[1]Byoungkwon Yoon, Hojun Lee, Yuseop Sim, Hangyeom Lee, and Martin Byung-Guk Jun are with School of Mechanical Engineering, Purdue University, West Lafayette, IN 47906, USA {yoon358, lee1764, sim64, lee3034, mbgjun}@purdue.edu

[2]Dongjun Lee is with the Department of Mechanical Engineering, IAMD and IOER, Seoul National University, Seoul, Republic of Korea, 08826 djlee@snu.ac.kr

primitives often overfit to specific training viewing angles by deforming to maximize screen-space coverage within a limited view frustum.

This lazy optimization problem also occurs for triangle primitives, while the resulting geometric issues are even more severe. In particular, an overfitted triangle primitive directly constitutes the mesh, resulting in a final representation with empty regions behind those lazy triangles. As highlighted in Fig. 1, comparing the training view (a) with a slightly offset "leaned" view (b) exposes the missing holes highlighted in red, due to the lazy triangles.

To address the issue of degenerate geometry in Triangle Splatting and fully utilize the joint optimization of mesh and image, we propose a novel coarse-to-fine approach with mesh-based LiDAR SLAM (mesh-SLAM) based primitive warmstart. Our contributions are summarized as follows:

- **LiDAR mesh-SLAM-guided primitive initialization**: We introduce a robust initialization strategy that transforms LiDAR mesh-SLAM reconstructions into decoupled, scaled, and optimizable triangle primitives. This initialization provides a reliable coarse geometry prior and reduces lazy optimization artifacts in ego-centric robot data.
- **LiDAR-guided optimization and densification**: We propose a geometric error variance based densification strategy that leverages the discrete nature of triangle primitives. By also incorporating a LiDAR depth error, we explicitly constrain the optimization to ensure geometric consistency alongside photometric quality.
- **Application to robotic simulation**: We validate the practical utility of our method by integrating the reconstructed meshes into a physics engine, thereby demonstrating accurate collision detection and interaction for autonomous robotic tasks.

The rest of the paper is organized as follows: Section II reviews existing differentiable primitive splatting and sensor-integrated methods. Section III explains the proposed method and implementation. Section IV presents experimental validation, including a demonstration in a robot simulation environment, and Section V concludes with limitations and future research topics.

## II. Related work

Our method lies at the intersection of sensor-aided Gaussian splatting and explicit surface reconstruction. As such, we review the foundational literature for both areas in the following.

### A. Gaussian Splatting with Additional Sensors

**Sensor-aided Gaussian initialization:** Standard 3DGS relies on Structure-from-Motion (SfM) point clouds, which are often sparse or noisy. Recent 3DGS-based SLAM methods bypass this limitation by leveraging sensor data for robust initialization. SplaTAM [13] utilizes input depth maps to unproject pixels into 3D space, directly initializing Gaussians with explicit spatial extents. Similarly, in feature-less or outdoor environments, approaches such as LetsGo [14] and LI-GS [15] utilize LiDAR point clouds for Gaussian initialization. LetsGo targets large-scale garage scenes, where COLMAP [16] reconstructions are often unreliable due to repetitive structures and weak visual features. To provide a more reliable initialization, LetsGo constructs a LiDAR-assisted level-of-detail octree map and initializes Gaussian primitives at multiple resolutions according to the corresponding spatial detail level. LI-GS further converts LiDAR point clouds into plane-constrained multimodal Gaussian Mixture Models, providing geometry-aware Gaussian initialization for large-scale reconstruction. For LiDAR-only mapping, Splat-LOAM [17] leverages spherical projection to encode LiDAR measurements into image-like representations, using them to guide the initialization and refinement of 2D Gaussian primitives. These methods initialize Gaussian primitives from LiDAR point clouds by estimating Gaussian attributes such as position, scale, orientation, or level-of-detail from local geometric cues. In contrast, our method leverages the direct correspondence between mesh-SLAM output and triangle primitives: reconstructed mesh faces can be converted into explicit triangle primitives with minimal geometric conversion. This reduces representation-specific assumptions and preserves the surface structure provided by the mesh-SLAM backend for subsequent optimization.

**Sensor-aided Gaussian optimization:** To prevent overfitting to RGB views and resolve geometric ambiguities, depth and normal priors are extensively used to regularize optimization. A standard approach involves a depth consistency loss, minimizing discrepancy between rendered depth maps and sensor measurements. For instance, SplaTAM [13] minimizes the error between the rendered depth and input depth maps to track camera pose and refine geometry. MM3DGS-SLAM [18] improves upon naive depth losses by using the Pearson correlation coefficient as a depth loss, which focuses on structural alignment rather than absolute value, thereby mitigating the impact of noise and scale ambiguity in depth estimates. GaussianLIC2 [19] utilizes LiDAR for SLAM-based incremental Gaussian map construction and employs a depth completion network to generate dense depth maps from sparse LiDAR and RGB inputs. By doing so, they provide supervision for unobserved regions within the sparse point cloud and effectively filter noise in the initial depth estimation.

In spite of the success of recent methods in regularizing 3DGS using sensor data, they primarily confine this supervision to 2D image-space losses. Consequently, the densification of primitives remains largely decoupled from the underlying geometry data from sensors. In the case of Gaussian primitive initialization, these approaches rely on LiDAR or depth maps to seed Gaussian centers, often employing heuristics to determine the initial scale and orientation of each primitive. However, this lack of structural priors inevitably introduces geometric uncertainty, which often requires extensive optimization to recover coherent surfaces or causes lazy optimization because of overfitting to photometric loss.

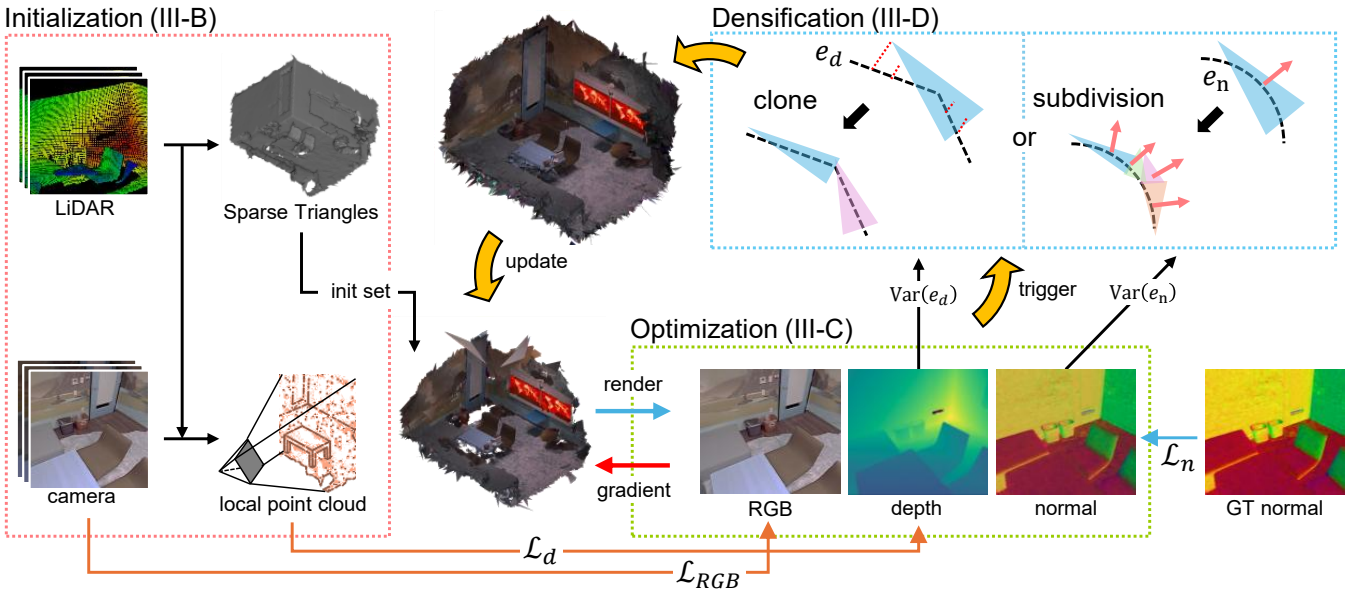


Fig. 2. Overview of proposed method. Detailed loss formulations are provided in Sec. III-C, and error terms $e_d, e_n$ are discussed in Sec. III-D.

### B. Surface Reconstruction from Gaussian Splatting

Maintaining an accurate surface from 3DGS is a significant challenge as the original representation contains millions of unorganized and overlapping ellipsoids that do not inherently conform to a continuous surface. 2DGS [20] addresses this by collapsing the 3D volume into oriented planar disks, providing intrinsic surface modeling and multi-view consistent geometry. However, 2DGS treats mesh extraction as a post-processing step, requiring the fusion of rendered depth maps into a Truncated Signed Distance Field (TSDF) to extract the final surface. Similarly, SuGaR [7] introduces a regularization term that encourages Gaussians to be flat and well-distributed along the scene's surface. Instead of standard marching cubes algorithm, they efficiently identify points on the Gaussian density level sets and use Poisson reconstruction to extract the mesh. MILo [21] proposes a mesh-in-the-loop approach where a mesh is differentiably extracted during each iteration using Gaussians as pivots for Delaunay triangulation, which enforces consistency between depth-based and Gaussian-based normals. Nevertheless, these methods still struggle to optimize 3DGS for geometry reconstruction and eventually overfit to the image, since the mesh is extracted from Gaussian primitives.

## III. Method

The key problem that this paper tackles is the reconstruction of 3D scenes with high photometric accuracy and geometric consistency, using triangle primitive warmstarting. To achieve the objective, our algorithm starts from the following three assumptions: 1) The input consists of posed and time-stamped RGB images with a LiDAR point cloud stream. 2) The LiDAR frame rate may be lower than the camera frame rate, while their temporal offset between them is manageable (e.g., 10 Hz LiDAR and 30 Hz RGB). 3) The sensor suite follows a trajectory that ensures robust mesh-SLAM tracking without failures. Since our method builds upon Triangle Splatting (TS) [8] and its recent extensions, TS+ [9], this section starts with a brief review of TS+, followed by our proposed coarse-to-fine refinement and optimization strategy. An overview of the proposed method is illustrated in Fig. 2.

### A. Preliminary: Triangle Splatting+

TS+ [9] is one variation of 3DGS that uses learnable triangle $\mathbf{T}_m$ as a primitive instead of 3D Gaussians. The scene is modeled with a set of vertices $\mathbf{v}_i \in \{\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^1\}$ and each $\mathbf{T}_m$ comprises a triplet of vertex indices $(i, j, k)$. Each $\mathbf{v}_i$ is parameterized with $x, y, z$ coordinates, vertex color $\mathbf{c}_i \in \mathbb{R}^3$, and opacity $o_i \in [0, 1]$. The opacity of $\mathbf{T}_m$ is the minimum value among the three vertex opacities and the color is calculated with barycentric coordinates $\mathbf{c}_{\mathbf{T}m}(\lambda_i, \lambda_j, \lambda_k) = \lambda_i \mathbf{c_i} + \lambda_j \mathbf{c_j} + \lambda_k \mathbf{c_k}$. Additionally, each $\mathbf{T}_m$ has a smoothness parameter, which determines color transition between outside and inside of the triangle.

### B. Coarse Step: Warmstarting Triangles with Mesh-SLAM

This sub-section details the coarse initialization of triangle primitives using the input LiDAR stream. Initialization of the original TS+ relies on 3D Delaunay triangulation [22] of sparse SfM points, yielding a complex volumetric mesh of semi-transparent tetrahedra that connect all SfM points. We presume this to be a primary factor of lazy optimization, as the resulting initial geometry poorly approximates the actual surface. In particular, when coupled with an ego-centric camera path, the lack of 360-degree viewing angles prevents the optimization process from effectively optimizing erroneous primitives within the volume. To go around this problem, a surface mesh generated by mesh-SLAM [23], [24] from a LiDAR point cloud is leveraged to initialize geometric primitives of a scene. We begin by converting the SLAM output into a set of independent primitives. Let the

mesh generated from SLAM with $N$ vertices and $M$ faces be $\mathcal{V}_{slam} = \{\mathbf{v}_1, \cdots, \mathbf{v}_N\}$, $\mathcal{F}_{slam} = \{f_1, \cdots, f_M\}$ and let the initialized set of triangles as $\mathcal{T}_{init} = \{\mathbf{T}_1, \cdots, \mathbf{T}_M\}$. First, connected mesh faces $\mathcal{F}_{slam}$ are decoupled into an initial set of triangles $\{\mathbf{T}_1, \cdots, \mathbf{T}_M\}$. Then each triangle $\mathbf{T}_i$ that is comprised of vertices $(v'_{i,1}, v'_{i,2}, v'_{i,3})$ is scaled as:

$$v'_{i,j} = \mathbf{g}_i + s \cdot (v_{i,j} - \mathbf{g}_i) \quad \text{for } j \in \{1, 2, 3\} \tag{1}$$

where $v_{i,j}$ represents the position of the $j$-th vertex of the $i$-th triangle in the mesh, $\mathbf{g}_i$ is its centroid, and $s$ is a scalar scaling factor ($s = 2$ in our experiments). Decoupling the mesh topology into individual primitives increases the degrees of freedom for vertex optimization by removing rigid connectivity constraints. Furthermore, scaling the triangles ensures sufficient spatial coverage to account for reconstruction gaps in the coarse SLAM output. Consequently, the proposed initialization yields a geometry-only mesh composed of coarse, large primitives without color.

### C. Optimization

After the coarse initialization, the parameters of the triangle primitives are optimized iteratively by minimizing the discrepancy between the rendered primitives and the ground truth image and geometry for better structural consistency. To incorporate geometric cues, methods such as 2D Gaussian Splatting (2DGS) [20] successfully employ normal consistency losses that align rendered normals with normals estimated from rendered depth gradients. However, this type of gradient-based normal refinement is less suitable for triangle primitives. Because triangles are inherently discrete and piecewise planar, their rendered depth maps exhibit constant normals within each face and sharp discontinuities at triangle boundaries, making depth-gradient-based normal refinement unstable. We instead use accumulated LiDAR points as direct depth supervision. For the $i$-th camera keyframe, we aggregate all LiDAR scans captured within the temporal interval $[(i-1), (i+1)]$ to create a LiDAR depth image that serves as ground truth depth. The resulting depth loss is as follows:

$$\mathcal{L}_d = L_1(D_R, D_L) \tag{2}$$

where $D_R$ is rendered depth and $D_L$ is LiDAR depth. The final objective function integrates our proposed geometric supervision with the standard photometric and regularization terms:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}} + \beta_1\mathcal{L}_o + \beta_2\mathcal{L}_n + \beta_3\mathcal{L}_d + \beta_4\mathcal{L}_{dist} \tag{3}$$

where the photometric $\mathcal{L}_1$ and $\mathcal{L}_{\text{D-SSIM}}$ losses, opacity regularization $\mathcal{L}_o$, and normal loss $\mathcal{L}_n$ are adopted from the original TS+ framework [9], while $\mathcal{L}_{dist}$ is the depth distortion loss [20].

### D. Fine Step: Densification via Geometric Error Variance

Every 500 iterations, we perform primitive densification. In 3DGS, densification is typically driven by the view-space positional gradient of the primitives [2]. Conversely, the original TS+ employs a probabilistic Markov Chain Monte Carlo (MCMC)-based approach [25], where candidate triangles are sampled based on their opacity. A common limitation of both strategies is their primary reliance on the photometric disparity between the rendered and ground truth images, which often fails to capture geometric details. Therefore, we propose a geometry-driven densification strategy that leverages the discrete nature of triangle primitives, instead of fully relying on photometric error.

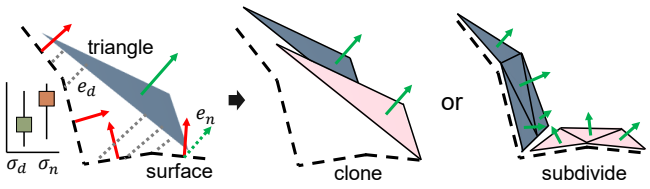


Fig. 3. Geometric error variance-based densification

By exploiting the explicit boundaries of triangle primitives, the exact set of rendered pixels per triangle is identified. We hypothesize that high variance in geometric error across these pixels serves as an indicator of structural underfitting. As illustrated in Fig. 3, consider a single flat triangle approximating a complex surface. While the primitive may minimize photometric error via texture interpolation, it cannot physically represent the true geometry, resulting in significant variance in the depth and normal error maps across its surface.

We integrate this insight into the MCMC-based method by sampling candidates based on geometric error variance rather than opacity. We define the depth variance $\sigma_d^2$ and normal variance $\sigma_n^2$ for a triangle $T$ as:

$$\sigma_d^2 = \mathbb{E}[(e_i^d)^2] - \mathbb{E}[e_i^d]^2 \tag{4}$$

$$e_i^d = d_i^{GT} - d_i^R \tag{5}$$

$$\sigma_n^2 = \mathbb{E}[(e_i^n)^2] - \mathbb{E}[e_i^n]^2 \tag{6}$$

$$e_i^n = (1 - \mathbf{n}_i^{GT} \cdot \mathbf{n}_i^R) \tag{7}$$

Here, $\mathbb{E}[\cdot]$ denotes the average over the pixels covered by the triangle. $d_i^{GT}$ and $\mathbf{n}_i^{GT}$ represent the ground truth depth and normal vector for the $i$-th pixel, while $d_i^R$ and $\mathbf{n}_i^R$ denote the rendered values, respectively. For the normal prior, we used a normal estimation model [26], followed by original TS+ method.

Furthermore, we decouple the densification operators to address the specific geometric failure modes. While the original TS+ relies on midpoint subdivision that splits a triangle into four interconnected primitives, we observe that this enforced vertex connectivity restricts the degrees of freedom required to model depth discontinuities. Therefore, we apply cloning (duplicating primitives to increase positional flexibility) when depth variance is high, and subdivision when normal variance is high. During the iterations, $\sigma_d^2$ and $\sigma_n^2$ for each triangle are continuously updated and utilized for sampling when densification is triggered.

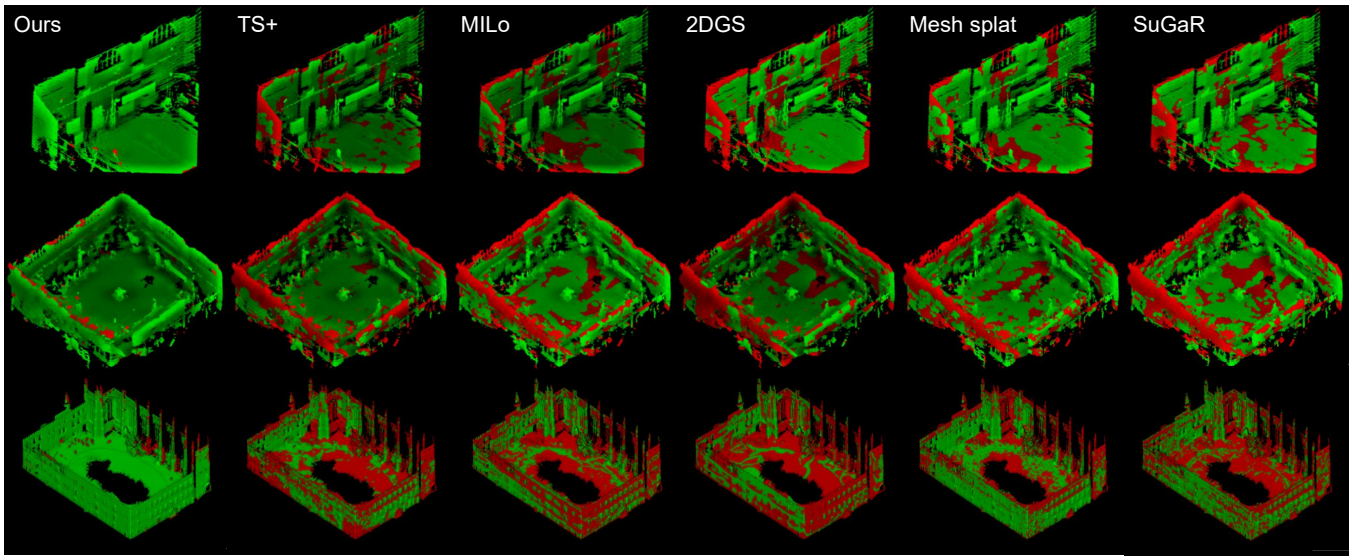


Fig. 4. F-score visualization over GT point-cloud. Top (fast) and middle (sq-2) rows of images are scenes from UTMM, while the bottom (q-easy) row shows those from NCD. Red regions indicate areas where the F-score exceeds the threshold.

TABLE I

QUANTITATIVE EVALUATION OF RENDERING QUALITY. **RED**, ORANGE, AND YELLOW INDICATE THE 1ST, 2ND, AND 3RD BEST

| Metric | Method | Replica | | | | | | | | UTMM | | | | | NCD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | of-0 | of-1 | of-2 | of-3 | of-4 | rm-0 | rm-1 | rm-2 | sq-1 | sq-2 | fast | slow-1 | slow-2 | q-easy |
| PSNR (dB) ↑ | Ours | **42.19** | **40.22** | **36.12** | **35.66** | 37.01 | 35.01 | 32.05 | 37.17 | 20.65 | **21.54** | **27.01** | 26.36 | 24.22 | 25.45 |
| | Triangle-splat | 35.99 | 35.44 | 29.95 | 27.50 | 32.08 | 23.72 | 27.33 | 30.08 | 18.95 | 19.01 | 23.57 | 24.88 | 22.87 | 21.49 |
| | MILo | 40.55 | 38.33 | 33.13 | 32.77 | 35.81 | 31.71 | 30.76 | 35.42 | 20.55 | 21.05 | 24.42 | 25.63 | 24.19 | 25.29 |
| | 2DGS | 41.08 | 39.13 | 36.02 | 35.59 | **37.14** | **35.41** | **32.47** | **37.49** | 20.48 | 21.09 | 26.44 | **27.29** | **25.38** | **26.47** |
| | Mesh-splat | 37.19 | 36.18 | 27.71 | 30.79 | 29.44 | 28.43 | 26.65 | 29.24 | 18.94 | 19.35 | 22.60 | 24.11 | 22.33 | 20.99 |
| | SuGaR | 40.97 | 38.45 | 34.66 | 35.05 | 36.92 | 34.08 | 30.58 | 36.94 | **20.89** | 21.52 | 26.07 | 27.27 | 25.05 | 25.64 |
| SSIM ↑ | Ours | **.981** | **.974** | **.961** | **.958** | **.960** | .955 | .935 | .970 | .646 | .670 | .803 | .813 | .772 | .777 |
| | Triangle-splat | .947 | .929 | .919 | .888 | .923 | .805 | .889 | .928 | .614 | .611 | .778 | .802 | .765 | .717 |
| | MILo | .976 | .965 | .947 | .943 | .953 | .935 | .929 | .964 | **.677** | **.697** | .813 | .835 | .809 | .822 |
| | 2DGS | .978 | .962 | .959 | .955 | .959 | **.958** | **.941** | **.971** | .668 | .678 | **.829** | **.853** | **.821** | **.832** |
| | Mesh-splat | .959 | .938 | .911 | .924 | .915 | .885 | .888 | .185 | .617 | .632 | .756 | .785 | .749 | .692 |
| | SuGaR | .977 | .963 | .953 | .952 | .956 | .947 | .923 | .967 | .674 | .690 | .820 | .846 | .810 | .814 |
| LPIPS ↓ | Ours | **.054** | **.083** | **.095** | **.086** | **.098** | **.085** | **.145** | **.086** | .357 | .358 | .258 | .249 | .290 | **.335** |
| | Triangle-splat | .143 | .195 | .189 | .206 | .182 | .291 | .229 | .175 | .410 | .426 | .306 | .289 | .328 | .418 |
| | MILo | .111 | .171 | .175 | .156 | .154 | .143 | .182 | .135 | .369 | .367 | .285 | .272 | .299 | .370 |
| | 2DGS | .095 | .171 | .145 | .127 | .134 | .110 | .148 | .120 | .359 | .363 | .250 | .226 | .265 | .351 |
| | Mesh-splat | .131 | .196 | .204 | .164 | .221 | .205 | .234 | .930 | .424 | .425 | .341 | .313 | .350 | .454 |
| | SuGaR | .078 | .146 | .131 | .108 | .114 | .104 | .154 | .106 | **.339** | **.341** | **.240** | **.213** | **.254** | .335 |

TABLE II

QUANTITATIVE EVALUATION OF GEOMETRY RECONSTRUCTION QUALITY

| Metric | Method | Replica | | | | | | | | UTMM | | | | | NCD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | of-0 | of-1 | of-2 | of-3 | of-4 | rm-0 | rm-1 | rm-2 | sq-1 | sq-2 | fast | slow-1 | slow-2 | q-easy |
| F-score ↑ | Ours | **.995** | **.972** | **.964** | **.959** | **.969** | **.967** | **.901** | **.953** | **.879** | **.879** | **.909** | **.905** | **.945** | **.875** |
| | Triangle-splat | .824 | .633 | .706 | .783 | .751 | .686 | .507 | .668 | .680 | .716 | .731 | .689 | .702 | .321 |
| | MILo | .938 | .885 | .893 | .912 | .905 | .883 | .695 | .892 | .678 | .670 | .678 | .701 | .669 | .390 |
| | 2DGS | .837 | .543 | .782 | .844 | .767 | .743 | .440 | .614 | .479 | .539 | .635 | .649 | .661 | .244 |
| | Mesh-splat | .813 | .734 | .637 | .724 | .644 | .669 | .552 | .626 | .675 | .677 | .734 | .631 | .694 | .468 |
| | SuGaR | .815 | .466 | .760 | .826 | .754 | .740 | .455 | .623 | .661 | .657 | .672 | .736 | .699 | .330 |
| Chamfer (m) ↓ | Ours | **.049** | **.080** | **.063** | **.069** | **.058** | **.067** | **.102** | **.073** | **.283** | **.227** | **.184** | **.216** | **.151** | **.312** |
| | Triangle-splat | .207 | .320 | .286 | .218 | .223 | .306 | .516 | .330 | .516 | .484 | .432 | .468 | .458 | 1.738 |
| | MILo | .088 | .129 | .115 | .108 | .103 | .125 | .263 | .112 | .501 | .513 | .470 | .457 | .491 | 1.328 |
| | 2DGS | .157 | .468 | .198 | .202 | .221 | .250 | .478 | .357 | .825 | .713 | .553 | .529 | .509 | 4.391 |
| | Mesh-splat | .229 | .264 | .371 | .263 | .378 | .319 | .399 | .352 | .525 | .524 | .422 | .555 | .468 | 1.146 |
| | SuGaR | .186 | .483 | .203 | .209 | .218 | .235 | .450 | .356 | .560 | .580 | .484 | .407 | .488 | 2.086 |

## IV. Experiments

This section presents a comprehensive evaluation of our method on both synthetic and real-world datasets under ego-centric trajectories. We also conduct a comprehensive comparison against state-of-the-art baselines and provide an ablation study to validate the effectiveness of our LiDAR-guided geometry and mesh-SLAM initialization.

### A. Experimental Details

**Baselines:** We compared our method with five state-of-the-art open-source methods that solve surface reconstruction and mesh extraction from differentiable primitives. To the best of our knowledge, there is currently no open-source, LiDAR-integrated Gaussian Splatting method with mesh extraction that can serve as a baseline. Among these five baseline methods, 2D Gaussian splatting (2DGS) [20] and SuGaR [7] focus on improving surface quality by using flat Gaussian primitives and mesh extraction methods. Meanwhile, Mesh-splatting [27] and MILo [21] focus on mesh extraction by either using meshes as primitives or applying a differentiable mesh extraction method from Gaussian primitives. We also compared with TS+ [9], upon which our method is built. All experiments were conducted on a workstation equipped with an Intel Core Ultra 9 285K CPU and an NVIDIA RTX 5080 GPU.

**Datasets:** For synthetic data, the Replica [28] dataset is used. Specifically, we select the challenging image sequences (office0-4, room0-2) provided by NICE-SLAM [29], which feature room-scale environments captured with egocentric, outward-facing camera trajectories. To simulate a robotic sensor setup, we generate solid-state LiDAR scans from the ground truth depth maps.

For real-world evaluation, we use the UTMM [18] (*square-1&2, fast-straight, slow-straight1&2* sequences) and Newer College dataset (NCD) [30] (*quad-easy* sequence). The UTMM dataset features RGB-LiDAR streams collected by a wheeled mobile robot following ego-centric, monotonic paths, but lacks a ground truth laser scan while NCD features a grayscale-image-LiDAR stream collected with a handheld device with ground truth laser scan geometry. We used every 10th frame for UTMM, 15th frame for replica, 30th frame for NCD as keyframes.

To isolate reconstruction quality from pose-estimation differences, all methods are evaluated using the same set of posed images. Camera poses are initialized from the mesh-SLAM trajectory for Replica and NCD, and from motion-capture data for UTMM, then refined with COLMAP bundle adjustment and fixed for all methods. For the image-based baselines, the sparse point cloud obtained from the same COLMAP reconstruction is used as the initial geometry. In contrast, our method uses the LiDAR mesh-SLAM output only for primitive initialization, while sharing the same optimized camera poses for training and evaluation.

**Metrics:** We evaluate performance across two dimensions:

- **Photometric Accuracy:** Following standard protocols [2], we evaluate PSNR, SSIM, and LPIPS on a test set consisting of every eighth frame of the trajectory.

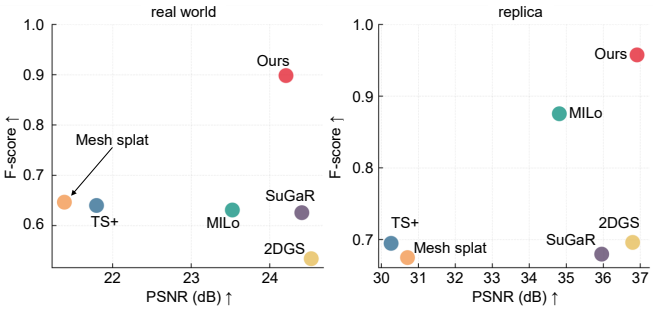


Fig. 5. Evaluation overview

- **Geometric Accuracy:** We assess the quality of the reconstructed surface using Chamfer distance ($L_1$) and F-score. Consistent with prior research [15], [31], [32], we set the F-score distance threshold to 20 cm for real world datasets, and 10 cm for synthetic datasets. For ground truth geometry, we use the provided mesh for Replica and laser scan point cloud for NCD. For the UTMM dataset, we generate a pseudo-ground truth by accumulating the LiDAR point clouds using the ground truth robot poses due to the lack of author provided ground truth geometry data. All ground truth point clouds are culled to the camera view frustum.

### B. Results and Discussion

Fig. 5 shows an overall performance comparison on the real world and Replica datasets against the baselines.

**Photometric Quality:** As detailed in Table I, our method demonstrates consistent high-fidelity rendering, ranking within the top three for PSNR across all evaluated sequences. While 2DGS [20] demonstrates strong image quality, the performance gap between our method and 2DGS is marginal. This shows that our method is able to accurately render the set of test camera poses even in non-object centric dataset.

**Geometric Accuracy:** In terms of geometric fidelity, our method outperforms all baselines by a substantial margin across every dataset and metric. Notably, while surface-regularized methods like 2DGS and SuGaR achieve high photometric scores, their geometric accuracy (F-score and Chamfer distance) remains suboptimal in comparison. This discrepancy aligns with our observation of lazy optimization, where models may minimize photometric loss by overfitting to training views with geometrically inconsistent primitives. Furthermore, while mesh-based methods (Mesh-Splatting and MILo) generally yield better geometric results than implicit methods, they still encounter difficulties in reconstructing accurate surfaces in ego-centric trajectories. In contrast, our method maintains a high F-score and a low Chamfer distance. Evaluation on UTMM is limited by the use of an accumulated LiDAR point cloud as pseudo-ground-truth geometry rather than an explicit ground-truth surface As such, we visualized the F-score result in Fig. 4. It shows the areas where the geometric error exceeds the F-score threshold. While baseline methods exhibit significant geometric degradation and inaccuracies, our approach maintains

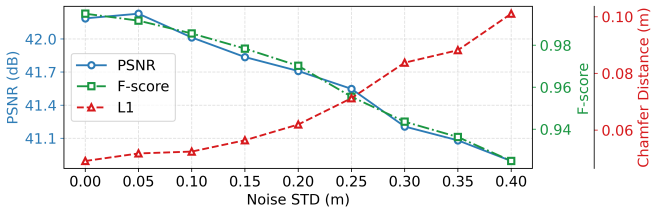


Fig. 6. Robustness evaluation results: "Noise" in the graph indicates the standard deviation of the noise.

high structural fidelity and precision. However, a limitation of our approach is its dependence on the quality of the geometric prior. In sequences such as *sq-1* and *q-easy*, which feature extensive glass surfaces, the LiDAR data is inherently unreliable due to transparency and specularity. Consequently, our method exhibits reduced geometric accuracy in these regions than on opaque surfaces.

### C. Robustness Evaluation and Ablation Study

**Robustness Evaluation:** Since our method relies on mesh-SLAM initialization, the quality of the SLAM result can directly affect the final reconstruction. To analyze this dependency, we simulate a mesh-SLAM failure by perturbing the initial mesh geometry. Using the *of-0* sequence, Gaussian noise is applied to every mesh vertex with increasing standard deviation, and both photometric and geometric quality are assessed. As shown in Fig. 6, both metrics degrade marginally up to a noise standard deviation of 10 cm, then decrease linearly up to 20 cm, and exhibit a steep decline beyond that threshold. Considering that the typical noise level of LiDAR sensors falls well within this range, the proposed method can be considered robust against variations in initial mesh quality.

**Ablation Study**: To show the effectiveness of our method, we conducted an ablation study on mesh warmstarting, and geometric error variance based densification using Replica *of-0* scene.

- **Mesh Init**: TS+ is initialized with ImMesh.
- **LiDAR Init**: The standard Delaunay triangulation is applied directly to the LiDAR point cloud instead of mesh warmstarting.
- **No clone**: No primitive cloning is applied during densification.
- **No subdiv**: No subdivision is applied during densification.

As summarized in Table III, geometric accuracy remained relatively stable for *mesh init*. However, the photometric quality suffered, evidenced by a 0.55 dB drop in PSNR. This demonstrates that our variance-based densification is significantly more effective for rendering quality. Next, *LiDAR init* led to a severe degradation across all metrics, causing a drop in PSNR of 5.17 dB and a significant loss of geometric fidelity. This stark contrast highlights the critical role of our mesh-based initialization and confirms that raw LiDAR alone is insufficient for high-quality scene representation. Additionally, the clone-only and subdivision-only baselines both demonstrated lower photometric quality.

TABLE III

ABLATION STUDY: REPLICA *of-0*

| | ΔPSNR (dB) ↑ | ΔChamfer (m) ↓ | ΔF-score ↑ |
|---|---|---|---|
| Full | **42.19** | **0.049** | **0.995** |
| Mesh Init | -0.55 | -0.003 | -0.024 |
| LiDAR Init | -5.17 | +0.267 | -0.362 |
| No clone | -0.50 | -0.004 | +0.002 |
| No subdiv | -2.72 | +0.051 | -0.004 |

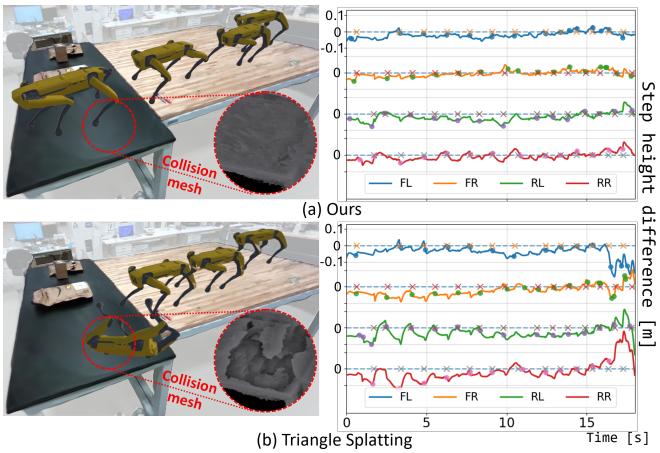


Fig. 7. Walking simulation using the extracted triangles as a collision mesh. Left: robot trajectory; right: step height differences. Crosses denote ground-truth foot contacts, and markers denote simulated contacts.

### D. Application

To demonstrate the practical utility of the reconstructed geometry, we evaluate the mesh as a collision layer in a robotic simulation. Photorealistic robot simulators require accurate meshes for contact physics [4], [6], [33]. Leveraging the decoupled nature of physics and rendering engines, we construct a hybrid simulation in Unity: 2DGS results are used for visual rendering and our reconstructed mesh forms the collision layer. We perform a walking simulation with a quadruped robot to compare the physical consistency of the reconstructed surfaces. As shown in Fig. 7, the baseline reconstruction contains holes caused by lazy triangle primitives, leading to unstable foot contacts and large deviations from the ground-truth contact height. In contrast, our method produces a more consistent collision surface, resulting in foot contact patterns that closely follow those obtained with the ground-truth collision mesh. The step height error plot further shows that our reconstructed mesh maintains substantially smaller deviations than the baseline, supporting more reliable locomotion simulation while preserving high-quality visual rendering.

## V. CONCLUSION AND DISCUSSION

In this work, we addressed the challenge of geometric degeneration in TS+ when applied to ego-centric mobile robot trajectories. We proposed a coarse-to-fine method that

integrates mesh-SLAM initialization with a novel LiDAR-based depth loss and densification strategy to maintain geometric consistency. Extensive evaluations on synthetic and real-world datasets demonstrate that our method significantly outperforms state-of-the-art baselines in geometric accuracy while maintaining competitive rendering quality. Despite these advancements, certain limitations remain. First, consistent with the original TS+ method, our reconstruction output consists of partially connected mesh patches rather than a globally watertight manifold. Consequently, further refinement is required to make these meshes suitable for downstream tasks such as path planning. Second, our approach relies on the fidelity of the initial mesh. In scenarios such as unconfined or large-scale environments, where the initial SLAM reconstruction may be sparse or noisy, the final reconstruction quality can be compromised. In future work, we aim to bridge these gaps by developing robust initialization strategies that account for input uncertainty and by extending our framework to support direct autonomous robot navigation.

## REFERENCES


[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering." *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139–1, 2023.

[3] T. Chen, O. Shorinwa, J. Bruno, A. Swann, J. Yu, W. Zeng, K. Nagami, P. Dames, and M. Schwager, "Splat-nav: Safe real-time robot navigation in gaussian splatting maps," *IEEE Trans. Robot.*, 2025.

[4] X. Li, J. Li, Z. Zhang, R. Zhang, F. Jia, T. Wang, H. Fan, K.-K. Tseng, and R. Wang, "Robogsim: A real2sim2real robotic gaussian splatting simulator," *arXiv preprint arXiv:2411.11839*, 2024.

[5] J. Guo, Y. Xin, G. Liu, K. Xu, L. Liu, and R. Hu, "Articulatedgs: Self-supervised digital twin modeling of articulated objects using 3d gaussian splatting," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2025, pp. 27 144–27 153.

[6] Y. Wu, L. Pan, W. Wu, G. Wang, Y. Miao, F. Xu, and H. Wang, "Rl-gsbridge: 3d gaussian splatting based real2sim2real method for robotic manipulation learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE, 2025, pp. 192–198.

[7] A. Guédon and V. Lepetit, "Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2024, pp. 5354–5363.

[8] J. Held, R. Vandeghen, A. Deliege, A. Hamdi, S. Giancola, A. Cioppa, A. Vedaldi, B. Ghanem, A. Tagliasacchi, and M. Van Droogenbroeck, "Triangle splatting for real-time radiance field rendering," *arXiv preprint arXiv:2505.19175*, 2025.

[9] J. Held, R. Vandeghen, S. Son, D. Rebain, M. Gadelha, Y. Zhou, M. C. Lin, M. Van Droogenbroeck, and A. Tagliasacchi, "Triangle splatting+: Differentiable rendering with opaque triangles," *arXiv preprint arXiv:2509.25122*, 2025.

[10] M. Wu, H. Dai, K. Yao, T. Tuytelaars, and J. Yu, "Bg-triangle: Bézier gaussian triangle for 3d vectorization and rendering," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2025, pp. 16 197–16 207.

[11] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022.

[12] S. Hwang, M.-J. Kim, T. Kang, J. Kang, and J. Choo, "Vegs: View extrapolation of urban scenes in 3d gaussian splatting using learned priors," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Springer, 2024, pp. 1–18.

[13] N. Keetha, J. Karhade, K. M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, and J. Luiten, "Splatam: Splat track & map 3d gaussians for dense rgb-d slam," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2024, pp. 21 357–21 366.

[14] J. Cui, J. Cao, F. Zhao, Z. He, Y. Chen, Y. Zhong, L. Xu, Y. Shi, Y. Zhang, and J. Yu, "Letsgo: Large-scale garage modeling and rendering via lidar-assisted gaussian primitives," *ACM Trans. Graph.*, vol. 43, no. 6, pp. 1–18, 2024.

[15] C. Jiang, R. Gao, K. Shao, Y. Wang, R. Xiong, and Y. Zhang, "Li-gs: Gaussian splatting with lidar incorporated for accurate large-scale reconstruction," *IEEE Robot. Autom. Lett.*, 2024.

[16] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[17] E. Giacomini, L. D. Giammarino, L. D. Rebotti, G. Grisetti, and M. R. Oswald, "Splat-loam: Gaussian splatting lidar odometry and mapping," 2025. [Online]. Available: https://arxiv.org/abs/2503.17491

[18] L. C. Sun, N. P. Bhatt, J. C. Liu, Z. Fan, Z. Wang, T. E. Humphreys, and U. Topcu, "Mm3dgs slam: Multi-modal 3d gaussian splatting for slam using vision, depth, and inertial measurements," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*. IEEE, 2024, pp. 10 159–10 166.

[19] X. Lang, J. Lv, K. Tang, L. Li, J. Huang, L. Liu, Y. Liu, and X. Zuo, "Gaussian-lic2: Lidar-inertial-camera gaussian splatting slam," *arXiv preprint arXiv:2507.04004*, 2025.

[20] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2d gaussian splatting for geometrically accurate radiance fields," in *Proc. ACM SIGGRAPH Conf. Papers*. Association for Computing Machinery, 2024.

[21] A. Guédon, D. Gomez, N. Maruani, B. Gong, G. Drettakis, and M. Ovsjanikov, "Milo: Mesh-in-the-loop gaussian splatting for detailed and efficient surface reconstruction," *ACM Trans. Graph.*, vol. 44, no. 6, pp. 1–15, 2025.

[22] Y. S. Elshakhs, K. M. Deliparaschos, T. Charalambous, G. Oliva, and A. Zolotas, "A comprehensive survey on delaunay triangulation: applications, algorithms, and implementations over cpus, gpus, and fpgas," *IEEE Access*, vol. 12, pp. 12 562–12 585, 2024.

[23] J. Ruan, B. Li, Y. Wang, and Y. Sun, "Slamesh: Real-time lidar simultaneous localization and meshing," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023, pp. 3546–3552.

[24] J. Lin, C. Yuan, Y. Cai, H. Li, Y. Ren, Y. Zou, X. Hong, and F. Zhang, "Immesh: An immediate lidar localization and meshing framework," *IEEE Trans. Robot.*, vol. 39, no. 6, pp. 4312–4331, 2023.

[25] S. Kheradmand, D. Rebain, G. Sharma, W. Sun, Y.-C. Tseng, H. Isack, A. Kar, A. Tagliasacchi, and K. M. Yi, "3d gaussian splatting as markov chain monte carlo," *Adv. Neural Inf. Process. Syst.*, vol. 37, pp. 80 965–80 986, 2024.

[26] M. Hu, W. Yin, C. Zhang, Z. Cai, X. Long, H. Chen, K. Wang, G. Yu, C. Shen, and S. Shen, "Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10 579–10 596, 2024.

[27] J. Held, S. Son, R. Vandeghen, D. Rebain, M. Gadelha, Y. Zhou, A. Cioppa, M. C. Lin, M. Van Droogenbroeck, and A. Tagliasacchi, "Meshsplatting: Differentiable rendering with opaque meshes," *arXiv preprint arXiv:2512.06818*, 2025.

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

[29] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022.

[30] L. Zhang, M. Camurri, D. Wisth, and M. Fallon, "Multi-camera lidar inertial extension to the newer college dataset," 2021.

[31] X. Zhong, Y. Pan, J. Behley, and C. Stachniss, "Shine-mapping: Large-scale 3d mapping using sparse hierarchical implicit neural representations," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023.

[32] J. Deng, Q. Wu, X. Chen, S. Xia, Z. Sun, G. Liu, W. Yu, and L. Pei, "Nerf-loam: Neural implicit representation for large-scale incremental lidar odometry and mapping," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2023, pp. 8218–8227.

[33] A. Escontrela, J. Kerr, A. Allshire, J. Frey, R. Duan, C. Sferrazza, and P. Abbeel, "Gaussgym: An open-source real-to-sim framework for learning locomotion from pixels," *arXiv preprint arXiv:2510.15352*, 2025.