PlanarMesh: Building Compact 3D Meshes from LiDAR using Incremental Adaptive Resolution Reconstruction

Jiahao Wang[1], Nived Chebrolu[1], Yifu Tao[1], Lintong Zhang [2], Ayoung Kim[3], and Maurice Fallon[1]

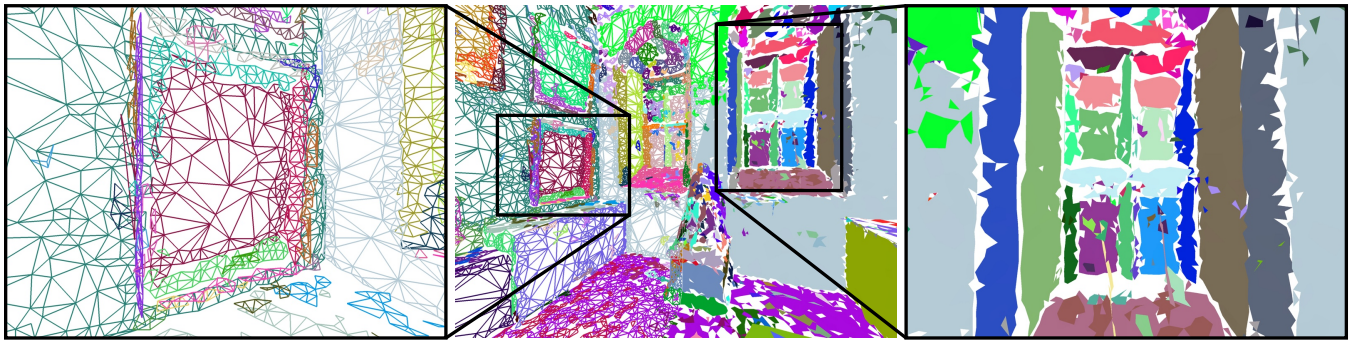

Fig. 1: PlanarMesh can reconstruct buildings and indoor spaces as a set of detailed, yet minimal, planar surfaces in real-time. It uses local curvature estimate to achieve adaptive resolution surface meshing. Left to right: [l] Plane reconstruction of a painting frame, [c] of a wider room, [r] and of inset window frame.

***Abstract*— Building an online 3D LiDAR mapping system that produces a detailed surface reconstruction while remaining computationally efficient is a challenging task. In this paper, we present PlanarMesh, a novel incremental, mesh-based LiDAR reconstruction system that adaptively adjusts mesh resolution to achieve compact, detailed reconstructions in real-time. It introduces a new representation, planar-mesh, which combines plane modeling and meshing to capture both large surfaces and detailed geometry. The planar-mesh can be incrementally updated considering both local surface curvature and free-space information from sensor measurements. We employ a multi-threaded architecture with a Bounding Volume Hierarchy (BVH) for efficient data storage and fast search operations, enabling real-time performance. Experimental results show that our method achieves reconstruction accuracy on par with, or exceeding, state-of-the-art techniques—including truncated signed distance functions, occupancy mapping, and voxel-based meshing—while producing smaller output file sizes (10 times smaller than raw input and more than 5 times smaller than mesh-based methods) and maintaining real-time performance (around 2 Hz for a 64-beam sensor).**

I. Introduction

Mobile 3D LiDAR scanners are becoming more affordable and reliable. They are now widely used in a variety of robotics and geospatial applications to model environments and buildings and provide accurate and reliable spatial data. However, the immense volume of raw point cloud data generated by these sensors poses a significant challenge when processing, storing, and carrying out real-time 3D reconstruction. An efficient reconstruction system is essential for converting raw LiDAR measurements into a compact and detailed representation that can be used for downstream tasks such as mission planning, localization, and inspection. In addition, when mapping online, sensor measurements arrive in a continuous stream. This requires the system to operate incrementally and maintain an up-to-date 3D model without imposing excessive computational overhead.

Triangle mesh is a popular 3D surface representation for its compactness and explicit surface connectivity, and there exist systems such as ImMesh [1], SLAMesh [2], and IDTMM [3] that can perform incremental mesh reconstruction. However, these methods create meshes with fixed resolution, which limits their performance: at low resolution, meshes are over-smoothed and cannot capture fine geometric details, whereas at high resolution, the increased number of mesh elements leads to larger file sizes, and smaller mesh elements that are more susceptible to over-fitting noise.

We introduce PlanarMesh, a novel surface reconstruction system that incrementally generates compact and detailed 3D reconstruction. PlanarMesh leverages the observation that planar structures (e.g. walls, ceilings, ground) are ubiquitous in built environment, and achieves adaptive resolution reconstruction by combining plane modeling with meshing. It uses planar models to represent flat surfaces, which provide an averaging effect to individual measurement noise, while employing variable-resolution meshing to capture fine details where needed. This adaptability is guided by local curvature information, which determines the appropriate resolution during mesh expansion. To efficiently update the planar-mesh, we propose Reverse Radius Search (RRS) for curvature-

[1]Oxford Robotics Institute, Univ. of Oxford, UK. { jiahaowang, yifu, nived, mfallon } @robots.ox.ac.uk
[2]NavLive Ltd, Oxford, UK. lzhang@navlive.ai
[3]Seoul National University, South Korea. ayoungk@snu.ac.kr
This project has been partly funded by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT)(No. RS-2024-00461409), NavLive Ltd (Wang), and Royal Society Univ. Research Fellowship (Fallon).

informed neighbor search and Face Intersection Search (FIS) to incorporate free-space information from LiDAR measurements, improving reconstruction accuracy. Our experimental results demonstrate that PlanarMesh achieves the highest compression rates, reducing mesh sizes by more than 5 times compared to existing mesh-based methods, while also maintaining comparable reconstruction quality.

In summary, our key contributions are as follows:

- **Adaptive Mesh Resolution:** We introduce PlanarMesh, which enables adaptive reconstruction by combining plane modeling with meshing.
- **Reduced Map Size Footprint:** Our adaptive meshing significantly reduces file sizes, achieving 10 times compression over raw point clouds and more than 5 times smaller outputs than existing mesh-based methods.
- **Incremental Mesh Reconstruction**: PlanarMesh updates existing planar-meshes as well as search trees incrementally, leveraging both local curvature and free-space information for an accurate reconstruction.
- **Real-Time Performance:** We achieve near LiDAR frame-rate reconstruction through a custom Bounding Volume Hierarchy (BVH) implementation, supporting efficient incremental updates and multi-threaded queries.

II. Related Works

We categorize the related work according to the 3D reconstruction approach taken so as to highlight the strengths and limitations of each, and to show how our work builds upon or differs from existing methods. These categories include point-based, surfel-based, mesh-based, occupancy-based, implicit surfaces, and global interpolation methods.

Point-based: Early work in 3D reconstruction began with point-based representations [4], where raw point cloud data formed the basis for subsequent processing. However, point clouds by themselves lack structural connectivity, which limits their utility in many applications.

Surfel-based: Surfels enhance point-based representations by incorporating local surface orientation and size properties, allowing for more descriptive scene reconstruction [5]. Systems like ElasticFusion [6] and Probabilistic Surfel Fusion [7] benefit from this discrete surface representation—ElasticFusion enables elastic deformation upon loop closure for greater flexibility, while Probabilistic Surfel Fusion employs surfel merging to improve robustness to noise. However, surfel-based methods suffer from a lack of explicit connectivity between surfels. Our system strikes a middle ground by using individual planar-mesh elements that can be integrated into future loop closure systems, fitting planes to points to reduce noise, and leveraging mesh structures to address the connectivity issues seen in surfel methods.

Meshes: In general, mesh-based methods can be divided into two subcategories: volumetric and surface-based [8]. Volumetric methods, such as Alpha Shapes [9] and the Crust Algorithm [10], [11], compute exhaustive triangulations—often enforcing Delaunay properties—and then selectively remove elements to extract the correct mesh geometry. Although effective for post-processing, these methods struggle with incremental updates, as repeated global triangulations are computationally intensive.

In contrast, surface-based methods (often related to region growing techniques [12], [13]) rely on local tangent plane approximations to sequentially build mesh surfaces from a set of points. Although they can locally expand, these approaches were originally designed for offline processing and lack robust mechanisms to seamlessly integrate new points. ImMesh [1] and SLAMesh [2] address these problems by creating mesh surfaces per voxel, and incrementally update the mesh inside each voxel. However, both approaches impose a maximum mesh size, leading to potential discontinuities between adjacent voxels. IDTMM [3], on the other hand, achieves incremental updates using Kalman filtering on mesh vertex positions not limited by voxel size. While our method shares the ability to operate without voxel size constraints like IDTMM, we further differentiate ourselves by supporting adaptive resolution, allowing efficient representation of both large planar regions and fine details.

Occupancy Maps: Occupancy mapping methods such as OctoMap [14], SuperEight [15], and D-Map [16] store occupancy probabilities. These methods inherently leverage free-space information from LiDAR, offering robustness in dynamic environments. SuperEight and Wavemap [17] use multi-resolution grids to achieve adaptive resolution, which helps to represent complex scenes with a smaller data footprint. However, occupancy mapping methods lack explicit surface representation, requiring post-processing techniques such as Marching Cubes [18] to extract surfaces, which can introduce geometric smoothing artifacts. Additionally, they struggle to accurately represent large planar surfaces when the plane is not aligned with the grid direction, often resulting in staircase-like artifacts due to voxel discretization.

Implicit Surfaces: In contrast to explicit reconstruction methods that produce surface elements directly, several approaches have investigated the use of implicit representations. The most well-known example involves the Signed Distance Function (SDF) method, which discretizes space into voxels that encode the distance to the nearest surface. Pioneered by Hoppe et al. [19] and popularized by systems like KinectFusion [20], SDF-based methods build implicit reconstructions that support real-time performance. More recent approaches, such as VoxBlox [21] and VDBFusion [22], extend these ideas to larger environments. Like occupancy-based methods, SDF representations require a post-processing step for surface extraction, making them less suitable for real-time applications requiring immediate surface access.

Global Interpolation: Instead of using input points to generate new geometry to represent a scene, global interpolation methods make direct estimates of the missing surface between the points. Poisson reconstruction [23] is a widely used method of this type, solving dense linear systems to recover smooth, high-quality surfaces from point clouds. Recent efforts, such as PUMA [24], modify Poisson reconstruction to improve efficiency, making it more suitable for real-time processing while still leveraging its surface estimation capabilities. However, due to its global implicit

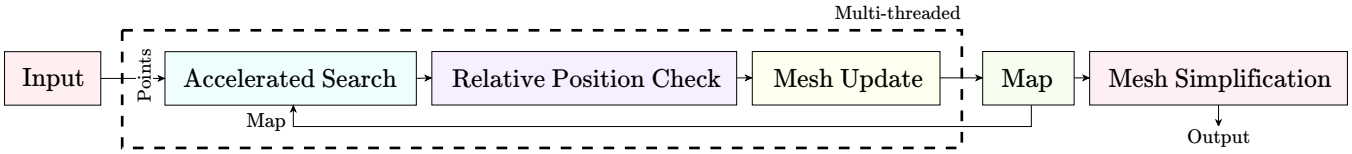

Fig. 2: System architecture overview. The system processes input points in parallel to reconstruct a map, which is then simplified before output. Key components include Accelerated Search (Sec. III-C), Relative Position Check (Sec. III-D), Mesh Update (Sec. III-E), and Mesh Simplification (Sec. III-F).

nature, Poisson reconstruction tends to hallucinate geometry in sparse regions, particularly along open sky boundaries. While fine-tuning parameters like reconstruction depth and output density can help, the method is highly sensitive to these choices and often requires different settings depending on the scale of the object being reconstructed.

III. Method

A. System Overview

The input to PlanarMesh is a stream of 3D point cloud scans from a mobile LiDAR sensor. We assume that an accurate pose for each scan has been estimated by a separate LiDAR odometry system (such as FastLIO [25] or VILENS [26]) which includes motion correction for its continuous scanning pattern. Each point from the point cloud is then processed by our system in a multi-threaded manner. Each point is queried against two separate search trees built from the point data—Face Intersection Search (FIS) and Reverse Radius Search (RRS)— to identify candidate planar-meshes for updates. A relative position check determines the appropriate mesh update operation. Before outputting a mesh map file, a mesh simplification step reduces file size by eliminating redundant triangulations. Fig. 2 provides a graphical overview of the system, and the algorithm's pseudocode is presented in Algo. 1 in Sec. III-E.

B. Planar-Mesh Representation

In our system, the surface of an object or building is modeled as a collection of **planar-meshes** $\mathcal{M}_\mathcal{P} : \{M_P^0, M_P^1, ..., M_P^N\}$ (see Fig. 4). Each planar-mesh M_P consists of two components:

1) **Plane** P, represents a surface patch, parameterized by its position $\mathbf{p}$ and a normal vector $\hat{\mathbf{n}}$, computed from LiDAR samples $\mathcal{L}_P \in P$.
2) **Mesh** M, a triangular mesh composed of vertices V, edges E and faces F, all of which lie on the associated plane P. Each $v \in V$ has an associated radius r that captures the local curvature.

We chose a planar representation for reconstruction because the built environment largely consists of planar patches, such as walls, ceilings, and floors. Most reconstruction approaches make prior assumptions [27]. A prior that favors smooth low-frequency surfaces cannot accurately model complex geometries, while a prior which seeks to model high-frequency surfaces can result in overfitting to the noise in the sensor measurements. To address this trade-off, our system is based on planes, which provide a simple yet robust surface approximation that minimizes sensitivity to noise, and uses a collection of planar-meshes (planes with boundaries) to represent complex geometries that can't be represented by a single plane. Although not implemented in this work, we believe such a representation is well-suited for downstream applications such as floor plan generation and 3D building information modeling (BIM). Moreover, this representation maintains a level of granularity comparable to point- or surfel-based methods, which allows for potential surface adjustments via elastic deformation in future work.

C. Accelerated Search

Given the input LiDAR point $\mathbf{l}_p$, sensor origin $\mathbf{o}$ and the corresponding ray $\hat{\mathbf{l}} = (\mathbf{l}_p - \mathbf{o})/\|\mathbf{l}_p - \mathbf{o}\|$), it is essential to make efficient queries of our existing planar-mesh model M_P to integrate this new information into our reconstruction. To achieve this, we construct two Bounded Volume Hierarchies (BVH) trees (see Fig. 3):

- Face Intersection Search Tree $\mathcal{T}_F$: Stores all the triangular faces $F \in M$ and returns a set of intersected faces F^* given the ray $\hat{\mathbf{l}}$. This result is used to integrate free-space-information into our reconstruction (see Sec. III-E.4).
- Reverse Radius Search Tree $\mathcal{T}_{V_\partial}$: Stores all boundary vertices $V_\partial \in M$. Given a query point $\mathbf{l}_p$, it returns the set of vertices V_∂^* that contain $\mathbf{l}_p$ within its associated radius r_∂. Since the radii r_∂ vary dynamically across V_∂, performing a reverse search using $\mathcal{T}_{V_\partial}$—which accounts for these varying radii—is more efficient than using a traditional KD-Tree with a large fixed radius followed by post-filtering. This result is used to guide our adaptive-resolution meshing (see Sec. III-E.5).

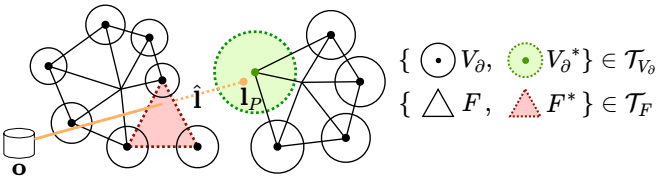

Fig. 3: The two BVHs used in our system. Given an input $(\hat{\mathbf{l}}, \mathbf{l}_p)$, the two BVHs retrieve intersected faces F^* and boundary vertices V_∂^* that contain $\mathbf{l}_p$ within their radius (a smaller sphere is used for display purposes).

Given the incremental update process of our system, the two BVH trees must be frequently updated to reflect changes in our mesh reconstruction. However, existing off-the-shelf BVH libraries, such as Embree [28] and OptiX [29], do not support incremental insertions and deletions, instead requiring a full search tree to be rebuild with each update. While individual rebuilds are computationally efficient, performing several thousand rebuilds per scan would introduce a substantial

overhead, making this approach impractical. Instead, we developed a custom accelerated BVH structure in accordance to Erin Catto's 2019 GDC talk[1]. Our implementation adopts the dynamic insertion strategy described in the talk, which helps avoid costly full tree rebuilds. Additionally, we implemented multi-threaded query support using fine-grained locking to fully parallelize the mesh-construction pipeline.

D. Relative Position Check

After querying the search trees, $\mathcal{T}_F$ and $\mathcal{T}_{V_\partial}$, with $\hat{\mathbf{l}}$ and $\mathbf{l}_p$, we obtain lists of F^* and V_∂^* whose associated planar-meshes $\mathbf{M_P}$ will be updated or modified using the $\mathbf{l}_p$ (see Sec. III-E). Prior to updating the planar-meshes $\mathbf{M_P}$, the relative position between $\mathbf{l}_p$ and each $M_P \in \mathbf{M_P}$ must be known to select the correct operation to be performed on M_P.

A lidar point $\mathbf{l}_p$ can be related to a planar-mesh M_P in three different ways (see Fig. 4):

1) $\mathbf{l}_p$ is observed in front of the M_P, denoted as $\mathbf{l}_p^{\text{f}}$ and M_P^{front}.
2) $\mathbf{l}_p$ is observed within a threshold σ_d from the M_P, denoted as $\mathbf{l}_p^{\text{w}}$ and M_P^{within} (typically because it lies on an underlying surface the plane $P \in M_P$ approximates).
3) $\mathbf{l}_p$ is observed behind the M_P, denoted as $\mathbf{l}_p^{\text{b}}$ and M_P^{behind}.

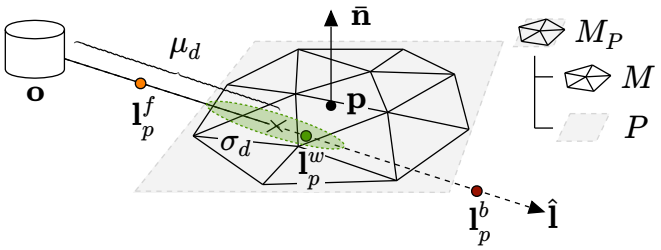

Fig. 4: Illustration of planar-mesh M_P. The relative position of $\mathbf{l}_p$ with respect to M_P is determined by comparing its location against the uncertainty threshold σ_d, visualized as an ellipse.

The position of $\mathbf{l}_p$ relative to M_P can be found by comparing the measured range $d = \|\mathbf{l}_p - \mathbf{o}\|_2$ to the expected range $\mu_d = (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{n}}/(\hat{\mathbf{n}}\cdot\hat{\mathbf{l}})$ using z-score $z = (d-\mu_d)/\sigma_d$, where $\sigma_d^2 = \sigma_{\mathbf{p}}^2+\sigma_{\mathbf{o}}^2+\sigma_{\hat{\mathbf{n}}}^2+\sigma_{\hat{\mathbf{l}}}^2+\sigma_{\text{noise}}^2$ with sensor noise σ_{noise} along the range axis. We compute each $\sigma_{\{\cdot\}}^2 = \mathbf{J}_{\{\cdot\}}\,\mathbf{\Sigma}_{\{\cdot\}}\,\mathbf{J}_{\{\cdot\}}^\top$, where $\mathbf{J}_{\{\cdot\}}$:

$$\mathbf{J}_{\mathbf{p}} = \frac{\partial \mu_d}{\partial \mathbf{p}} = \frac{\hat{\mathbf{n}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{l}}}, \qquad \mathbf{J}_{\hat{\mathbf{n}}} = \frac{\partial \mu_d}{\partial \hat{\mathbf{n}}} = \frac{\mathbf{l}_p-\mathbf{o}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{l}}} - \mu_d\frac{\hat{\mathbf{l}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{l}}},$$
$$\mathbf{J}_{\mathbf{o}} = \frac{\partial \mu_d}{\partial \mathbf{o}} = -\frac{\hat{\mathbf{n}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{l}}}, \qquad \mathbf{J}_{\hat{\mathbf{l}}} = \frac{\partial \mu_d}{\partial \hat{\mathbf{l}}} = -\mu_d\frac{\hat{\mathbf{n}}}{\hat{\mathbf{n}}\cdot\hat{\mathbf{l}}}.$$

We then perform a hypothesis test using a 95% confidence interval, corresponding to a critical value of 1.96:

$$\begin{aligned} H_0 &: \; z < -1.96, && \mathbf{l}_p^{\text{f}} \text{ observed in front of } M_P, \\ H_1 &: \; |z| \le 1.96, && \mathbf{l}_p^{\text{w}} \text{ observed within } M_P, \\ H_2 &: \; z > 1.96, && \mathbf{l}_p^{\text{b}} \text{ observed behind } M_P, \end{aligned}$$

to classify the relation into the three cases mentioned above.

[1]Erin Catto, GDC 2019 talk "Math for Game Developers: Dynamic Bounding Volume Hierarchies", https://gdcvault.com/play/1025909/Math-for-Game-Developers-Dynamic.

E. Mesh Update

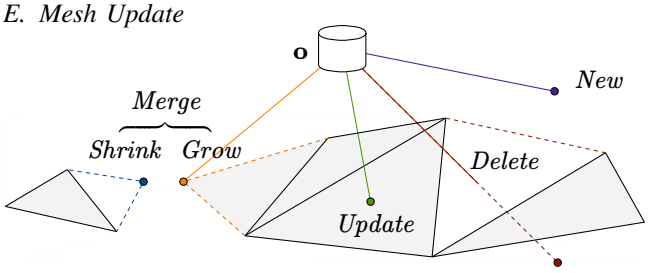

Fig. 5: Different mesh update operations: *Update (Sec. III-E.1)*, *Grow (Sec. III-E.2)*, *New (Sec. III-E.3)*, *Delete (Sec. III-E.4)*, *Shrink (Sec. III-E.5)*, *Merge (Sec. III-E.6)*.

Let $\mathbf{M_P}_{F^*}$, $\mathbf{M_P}_{V_\partial^*}$ denote the planar-meshes associated with F^*, V_∂^* returned from querying $\mathcal{T}_F$, $\mathcal{T}_{V_\partial}$ respectively. After the relative position check (Sec. III-D), we classify the planar-meshes into three groups for each F^*, V_∂^*: $\mathbf{M_P}_{\{\cdot\}}^{\text{front}}$, $\mathbf{M_P}_{\{\cdot\}}^{\text{within}}$, $\mathbf{M_P}_{\{\cdot\}}^{\text{behind}}$. Additionally, we define $\mathbf{M_P}_{\{\cdot\}}^{\text{seed}}$ to contain planar-mesh that does not have sufficient points for relative position check to be performed (more in Sec. III-E.3). The updates and modifications performed on M_P in these groups are illustrated in Fig. 5, with the corresponding algorithm provided in Algo. 1 and described below:

1) Update (Line 5 to 7): If $\mathbf{M_P}_{F^*}^{\text{within}} \neq \emptyset$, then we update the plane parameters $(\mathbf{p}, \hat{\mathbf{n}})$ of the largest planar-mesh $M_P^* \in \mathbf{M_P}_{F^*}^{\text{within}}$. This is achieved by performing Principle Component Analysis (PCA) on the covariance matrix Σ_P of $\mathcal{L}_P \in P$ and $\mathbf{l}_p$, and choosing the eigenvector corresponding to the smallest eigenvalue as the updated plane normal $\hat{\mathbf{n}}$.

Instead of re-computing Σ_P from scratch each time given $\mathcal{L}_P$ and $\mathbf{l}_p$, we incrementally update $\Sigma_P \leftarrow \tilde{\Sigma}_P$:

$$\tilde{\Sigma}_P = \frac{n_P\Sigma_{\mathbf{p}} + \Sigma_{\mathbf{l}_p} + n_P\Delta_{\mathbf{p}}\Delta_{\mathbf{p}}^\top + \Delta_{\mathbf{l}_p}\Delta_{\mathbf{l}_p}^\top}{1+n_P}, \tag{1}$$

where n_P is size of $\mathcal{L}_P$, $\Delta_{\{\cdot\}} = \{\cdot\} - \mu$ and $\mu = (\mathbf{l}_p + n_P \cdot \mathbf{p})/(1+n_P)$. We discuss the reason behind choosing the largest planar-mesh in Sec. III-E.6.

2) Grow (Line 8 to 13): If $\mathbf{M_P}_{V_\partial^*}^{\text{within}} \neq \emptyset$, then we expand the largest planar-mesh $M_P^* \in \mathbf{M_P}_{V_\partial^*}^{\text{within}}$, otherwise we expand the closest seed planar-mesh $M_P^* \in \mathbf{M_P}_{V_\partial^*}^{\text{seed}}$. The *Grow* step has lower priority over the *Update* step as an expansion creates mesh elements in newly observed space, which is less certain than updating an existing mesh we had previously observed. We expand M_P^* by iteratively creating edges $\tilde{E}$ between $\mathbf{l}_p$ and $V_\partial^* \in M_P^*$ provided that $\tilde{e} \in \tilde{E}$ does not intersect with $E \in M_P^*$. Additionally, we create new faces $\tilde{F}$ between two consecutive edges in $\tilde{E}$ if $\tilde{f} \in \tilde{F}$ does not contain an existing $v \in M_P^*$.

3) New (Line 14 to 16): We create new seed planar-meshes $\tilde{M}_P^{\text{seed}}$ if $\mathbf{l}_p$ is added to an empty region of the map. $\tilde{M}_P^{\text{seed}}$ is later updated or expanded as described in previous two steps. A planar-mesh is consider seed only if it has less than three points, or has a small mesh area, as it results in unstable plane fitting due to noisy LiDAR point $\mathbf{l}_p \in \mathcal{L}_P$.

4) Delete (Line 17 to 18): If the new point $\mathbf{l}_p$ is added to a planar-mesh M_P^* considered "within", i.e. $M_P^* \in \{\mathbf{M_P}_{F^*}^{\text{within}}, \mathbf{M_P}_{V_\partial^*}^{\text{within}}\}$, $\mathbf{l}_p$ is likely to be an inlier point and

Algorithm 1 Incremental PlanarMesh Reconstruction

Require: Map $\mathcal{M}_\mathcal{P}$, LiDAR Point $\mathbf{l}_p$, sensor origin $\mathbf{o}$,
Ensure: Updated Map $\tilde{\mathcal{M}}_\mathcal{P}$

▷ Accelerated Search

1: $F^*, \mathbf{M_P}_{F^*} \leftarrow \texttt{FIS}(\mathcal{T}_F, \hat{\mathbf{l}})$
2: $V^*_\partial, \mathbf{M_P}_{V^*_\partial} \leftarrow \texttt{RRS}(\mathcal{T}_{V_\partial}, \mathbf{l}_p)$

▷ Relative Position Check

3: $\mathbf{M_P}_{F^*}^{\text{within}}, \mathbf{M_P}_{F^*}^{\text{behind}} \leftarrow \{M_P \mid \texttt{RPC}(M_P, \mathbf{l}_p, \mathbf{o}),\ M_P \in \mathbf{M_P}_{F^*}\}$
4: $\mathbf{M_P}_{V^*_\partial}^{\text{within}}, \mathbf{M_P}_{V^*_\partial}^{\text{seed}} \leftarrow \{M_P \mid \texttt{RPC}(M_P, \mathbf{l}_p, \mathbf{o}),\ M_P \in \mathbf{M_P}_{V^*_\partial}\}$

▷ Mesh Update

▷ *Update (Sec. III-E.1)*

5: **if** $\mathbf{M_P}_{F^*}^{\text{within}} \neq \emptyset$ **then**
6: $\quad M^*_P \leftarrow \arg\max_{M_P \in \mathbf{M_P}_{F^*}^{\text{within}}} \texttt{Area}(M_P)$
7: $\quad \tilde{M}_P \leftarrow \texttt{UPDATE}(M^*_P, \mathbf{l}_p)$

▷ *Grow (Sec. III-E.2)*

8: **else if** $\mathbf{M_P}_{V^*_\partial}^{\text{within}} \neq \emptyset$ **then**
9: $\quad M^*_P \leftarrow \arg\max_{M_P \in \mathbf{M_P}_{V^*_\partial}^{\text{within}}} \texttt{Area}(M_P)$
10: $\quad \tilde{M}_P \leftarrow \texttt{GROW}(M^*_P, \mathbf{l}_p)$
11: **else if** $\mathbf{M_P}_{V^*_\partial}^{\text{seed}} \neq \emptyset$ **then**
12: $\quad M^*_P \leftarrow \arg\min_{M_P \in \mathbf{M_P}_{V^*_\partial}^{\text{seed}}} \texttt{Dist}(M_P, \mathbf{l}_p)$
13: $\quad \tilde{M}_P \leftarrow \texttt{GROW}(M^*_P, \mathbf{l}_p)$

▷ *New (Sec. III-E.3)*

14: **else**
15: $\quad \tilde{M}^{\text{seed}}_P \leftarrow \texttt{NewSeedPlanarMesh}(\mathbf{l}_p)$
16: **end if**

▷ *Delete (Sec. III-E.4)*

17: **if** $M^*_P \in \{\mathbf{M_P}_{F^*}^{\text{within}}, \mathbf{M_P}_{V^*_\partial}^{\text{within}}\}$ **then**
18: $\quad \tilde{\mathbf{M}}_{\mathbf{P}F^*}^{\text{behind}} \leftarrow \texttt{DELETE}(F^* \in \mathbf{M_P}_{F^*}^{\text{behind}})$

▷ *Shrink (Sec. III-E.5)*

19: $\quad \tilde{\mathbf{M}}_{\mathbf{P}V^*_\partial} \leftarrow \texttt{SHRINK}(V^*_\partial \in \{\mathbf{M_P}_{V^*_\partial} \setminus M^*_P\}, \mathbf{l}_p)$
20: **end if**
21:
22: $\tilde{\mathcal{M}}_\mathcal{P} \leftarrow \mathcal{M}_\mathcal{P} \cup \tilde{M}_P \cup \tilde{M}^{\text{seed}}_P \cup \tilde{\mathbf{M}}_{\mathbf{P}F^*}^{\text{behind}} \cup \tilde{\mathbf{M}}_{\mathbf{P}V^*_\partial}$

the corresponding ray $\hat{\mathbf{l}}$ can used to integrate free-space information. We remove all faces F whose planar-meshes have been "passed through" by $\mathbf{l}_p$ i.e. $F \in \mathbf{M_P}_{F^*}^{\text{behind}}$. An edge $e \in \mathbf{M_P}_{F^*}^{\text{behind}}$ with no connected faces is considered "dangling" and is deleted from $\mathbf{M_P}_{F^*}^{\text{behind}}$. This mechanism eliminates any faces and edges that erroneously span gaps between adjacent surface in the reconstructed map.

The fidelity of these "pass through" checks improves as more points are added to a plane, which in turn results in better plane estimates with lower $\Sigma_\mathbf{p}$ and $\Sigma_{\hat{\mathbf{n}}}$. This, in turn, reduces the range uncertainty σ_d used in the relative position check between a point $\mathbf{l}_p$ and a M_P, enabling more effective identification of "pass-through" points. In practice this allows subtle features to be captured more reliably, such as the slightly recessed window panes in a panel door (see Fig. 6 magenta box) or chamfered edges in walls.

5) Shrink (Line 19): For $M^*_P \in \{\mathbf{M_P}_{F^*}^{\text{within}}, \mathbf{M_P}_{V^*_\partial}^{\text{within}}\}$, we integrate $\mathbf{l}_p$ into the map to shrink nearby planar-meshes with high curvature and long edges, encouraging their replacement with smaller, more localized triangles.

To achieve this, we introduce the concept of the Reverse Radius Search (RRS), where the radius value is inspired by the medial axis distance—the shortest distance from a surface point to its medial axis, which correlates with local surface curvature [10]. In our approach, we define the radius r, as the shortest distance from a vertex $v \in M_P^{\text{current}}$ to the nearest vertex $v_{\text{nearest}} \in M_P^{\text{nearest}}$.

When $\mathbf{l}_p$ is added to M^*_P as a new vertex, it updates the radius of nearby boundary vertices $V^*_\partial \in \mathbf{M_P}^{\text{other}}$, where $\mathbf{M_P}^{\text{other}} \leftarrow \{\mathbf{M_P}_{V^*_\partial} \setminus M^*_P\}$. Any edges connected to V^*_∂ that exceed its updated radius are removed. A vertex $v \in V^*_\partial$ with no remaining edges is considered "dangling" and is deleted from $\mathbf{M_P}^{\text{other}}$, though it may later be reintroduced as a new LiDAR point. This process allows other LiDAR points that are closer to $\mathbf{M_P}^{\text{other}}$ to be incorporated with edges that conform to the local radius threshold. As a result, high-curvature regions (typically near object edges) are refined with smaller triangles, improving geometric accuracy. Conversely, in low-curvature regions, larger triangles facilitate rapid surface expansion with fewer sampled points, yielding a more compact yet efficient representation.

6) Merge: The final operation seeks to merge two planar-meshes $M_P^{\text{large}}, M_P^{\text{small}}$ which are close to each other and have similar orientations. When a new point $\mathbf{l}_p$ is integrated, we prioritize adding it to the larger mesh M_P^{large}. As M_P^{large} expands, the radius of $V_\partial \in M_P^{\text{small}}$ is reduced, triggering a removal of edges $E \in M_P^{\text{small}}$ in the smaller mesh (*Shrink*), and subsequent addition of the "dangling" vertices to the larger mesh M_P^{large} (*Grow*). This results in M_P^{large} "consuming" M_P^{small}, creating an spontaneous merge effect.

F. Mesh Simplification

Before finalizing the map $\mathcal{M}_\mathcal{P}$ and output it as a file, we simplify $M_P \in \mathcal{M}_\mathcal{P}$ as follows:

1) Collect all vertices V within the planar-mesh M_P.
2) Sort $v \in V$ in ascending order based on their radius r.
3) Traverse the sorted list, add a vertex v to V_{sampled} only if no previously sampled vertices $v_{\text{sampled}} \in V_{\text{sampled}}$ exist within its radius r.
4) Perform Delaunay triangulation on the sampled vertices V_{sampled} to generate a convex mesh M_{convex}.
5) Restore concavity by discarding $F_{\text{convex}} \in M_{\text{convex}}$ whose centroids do not fall within an original face $F \in M_P$.

IV. Experimental Results

A. Experimental Setup and Evaluation

We evaluate our reconstruction method using the Oxford Spires dataset [30] across diverse indoor and outdoor environments. The sequences used are listed in Tab. I. We compare our method to the following approaches: ImMesh [1] (voxel mesh-based), VDBFusion [22] (TSDF-based), and OctoMap [14] (occupancy-based). Although OctoMap is not a mesh-based reconstruction method, we include it as a reference point: it provides a theoretical upper bound on recall and offers insight into the compression capabilities of

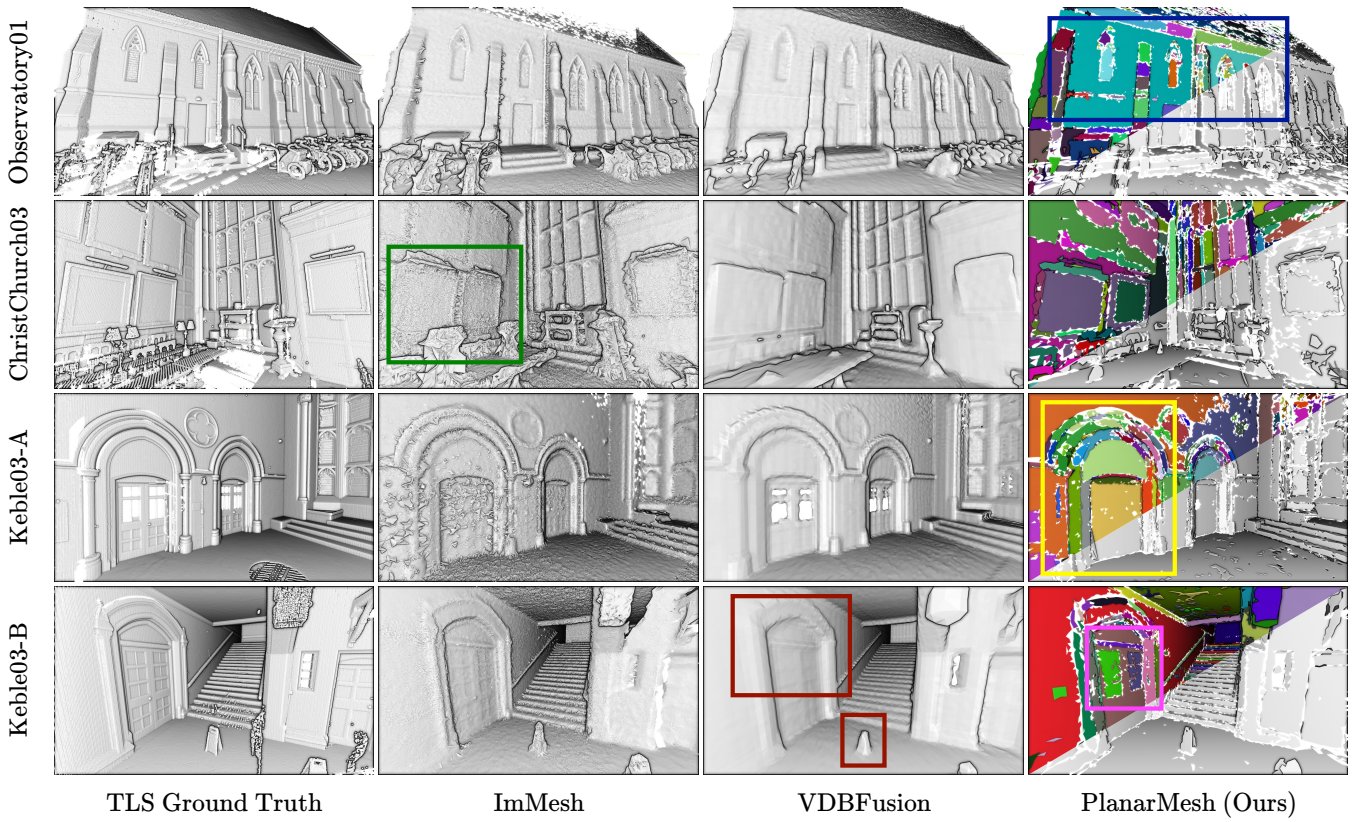

Fig. 6: Comparison between reconstructions from the different methods. Individual planar-meshes produced by PlanarMesh are colored. It can correctly detect subtle features such as the geometry around a door frame (yellow), recessed window panes in a paneled door (magenta) while fitting a single large plane to a uniform wall (blue). Examples of noisy surfaces from ImMesh (green) and overly-smoothed surfaces from VDBFusion (red) are also indicated.

mesh-based representations. All methods are configured for real-time mesh generation at approximately 1 Hz, achieved by setting the voxel size to 0.05 m for OctoMap and 0.1 m for both ImMesh and VDBFusion, running on a single CPU core. At normal walking speeds this corresponds to adding new scan every 0.5 -1 m traveled.

For each approach, we process individual LiDAR scans with ground truth poses to produce mesh reconstructions. The ground truth poses were obtained by registering each individual undistorted point cloud scans to the highly accurate TLS point cloud map of the test location.

We performed quantitative evaluation by converting reconstructed meshes into point clouds, uniformly sampling points to match the raw LiDAR scan count. To evaluate reconstruction quality, we report the mean and standard deviation of the distances to the ground truth, alongside precision, recall, and F-score at a 0.1 m threshold (Tab. I). We applied pre-filtering, as detailed in [31], to both the ground truth TLS scan and the reconstructed meshes, ensuring a fair comparison by excluding areas not captured by both scanning modalities. We also report the file size of the output mesh (in PLY binary format). All experiments were conducted on a 28-core Intel i7 CPU without GPU acceleration. While all baseline methods were run on a single core, our method requires all available CPU cores to achieve the aforedmentioned real-time performance.

B. Mesh Reconstruction Quality

In this section, we evaluate the reconstruction quality of our approach. Fig. 6 provides a visual comparison between the mesh reconstructions from the different methods. Both VDBFusion and Immesh use a fixed voxel size (0.1 m) for their reconstruction, whereas our approach adapts to scene curvature. This allows us to model complex door frames with multiple planes (yellow box) and uniform walls with single large plane (blue box). We also observe that VDBFusion tends to oversmooth shapes, which results in a loss of detail such as the door frame and the sign on the floor (red boxes), whereas our approach captures those details. On the other hand, ImMesh produces noisier meshes (green box) due to per-voxel plane fitting, resulting in rougher surfaces. While our mesh reconstruction retains these detailed features, it suffers from small holes and rugged edges in the mesh. This could be improved by scanning the scene more densely or by applying smoothing or plane intersection operations in processing, which we aim to incorporate in future work.

Tab. I presents quantitative evaluation of our approach. Our reconstruction has a mean mesh-to-point error in the range of 3-4 cm, outperforming the baselines. By grouping points that belong to the same plane, our planar-mesh representation effectively averages out point noise, leading to a better performance. We also obtain high precision (0.92 -0.97), recall (0.84 -0.96), and F1 scores (0.91 -0.96) for all the sequences

TABLE I: Evaluation of 3D Reconstruction Quality. OctoMap is placed as its own category as it is an occupancy-based method and therefore does not produce face or vertex counts.

Method	Per-Scan Time↓ (s)	File size↓ (MB)	Num of Faces↓	Num of Vertices↓	Mean↓ (m)	Std↓ (m)	Precision↑	Recall↑	F-Score↑
Christ Church 03 (∼307 m)									
VDBFusion	0.871	53.6	1,992,391	1,152,788	0.044	**0.077**	0.918	0.970	0.943
ImMesh	0.724	370.9	21,180,823	7,959,789	0.090	0.186	0.820	0.990	0.897
PlanarMesh (Ours)	**0.392**	10.1	**398,712**	**411,907**	**0.037**	0.081	**0.951**	0.964	0.957
OctoMap	0.432	**3.4**	N/A	N/A	0.040	0.083	0.943	**0.991**	**0.966**
Keble College 03 (∼108 m)									
VDBFusion	0.968	51.3	1,821,087	1,150,250	0.033	0.113	0.962	0.940	0.951
ImMesh	0.355	163.8	9,057,110	3,838,040	0.035	**0.064**	0.955	0.918	0.936
PlanarMesh (Ours)	0.416	**7.3**	**287,020**	**296,254**	**0.031**	0.134	**0.979**	0.894	0.935
OctoMap	**0.328**	24.3	N/A	N/A	0.040	0.159	0.964	**0.966**	**0.965**
Observatory 01 (∼324 m)									
VDBFusion	2.406	148.5	5,246,193	3,346,024	0.047	0.104	0.899	0.899	0.899
ImMesh	0.448	424.7	23,448,665	9,986,250	0.056	**0.089**	0.878	0.832	0.854
PlanarMesh (Ours)	**0.213**	**15.3**	**546,415**	**682,725**	**0.042**	0.114	**0.929**	0.847	0.886
OctoMap	0.659	67.8	N/A	N/A	0.055	0.150	0.896	**0.941**	**0.918**

which is either better or comparable to other methods. Our approach is intentionally biased toward higher precision and lower recall, as we do not retain infinitesimally small faces, which are unlikely to represent meaningful structural elements in our target application of building modeling.

C. File Footprint

We recognize that the built environment is dominated by planes, and our approach leverages this to achieve a significant file size reduction as seen in file and mesh size columns in Tab. I. Our method produces triangular meshes with 280 - 550 K faces, an order of magnitude lower than other methods (1.8 - 23 M). Compared to the accumulated LiDAR point cloud (∼100 MB), our reconstruction yields a file size of approximately 10 MB, representing a roughly tenfold compression.

Traditional mesh-based reconstructions often generate large files due to the storage of numerous small triangles, even in low-curvature regions. Whereas, our planar-mesh approach shares normals among faces within the same plane. By re-sampling vertices and re-triangulating on demand, we achieve significantly smaller file sizes.

D. Timing Analysis

This section validates the real-time capability of our approach and presents a breakdown of the computational time required by each module within our reconstruction pipeline. The processing time of each module is visualized as a stacked area plot (Fig. 7). On average, each scan is processed in around 0.4 sec, with peak processing times reaching 0.7 sec, resulting in an effective processing speed of about 2 Hz. This performance is on-par with other real-time reconstruction methods (Tab. I). Higher update rates could be achieved by reducing the number of points returned per scan from the LiDAR sensor while increasing the scan frequency—a viable strategy for applications where a higher refresh rate is required. The FIS and RRS data structures are implemented as binary search trees, the complexity of their search operations is therefore $\mathcal{O}(\log N)$, where N is the number of faces or boundary vertices, respectively.

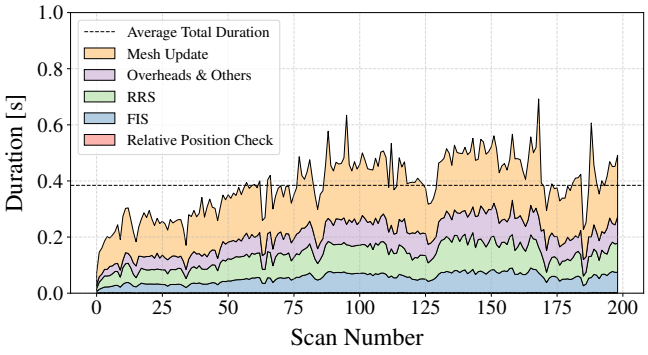

Fig. 7: Computation time for each module of PlanarMesh, visualized as a stacked area plot for the ChristChurch03 sequence.

E. Ablation Studies

TABLE II: Ablation Study on ChristChurch03 Dataset while varying the keep seed planar-meshes duration.

Num	Time↓ (s)	File size↓ (MB)	Mean↓ (m)	Std↓ (m)	Precision↑	Recall↑	F-Score↑
0	**0.295**	**5.9**	0.037	**0.075**	0.946	0.889	0.917
1	0.347	7.9	0.036	0.076	0.951	0.946	0.948
10	0.384	8.3	**0.036**	0.076	0.951	0.951	0.951
All	0.313	8.6	0.036	0.078	**0.951**	**0.953**	**0.952**

We present the results of an ablation study on the retention duration of seed planar-meshes. Seed planar-meshes typically contain few points or have small surface areas, often arising from outlier points. However, they can also result from limited observations of complex geometries that have not yet been sufficiently scanned. In our implementation, we keep the seed planar-meshes throughout the duration of mission and do not discard them. By increasing the duration seed planar-meshes are kept, we enhance the likelihood of identifying and developing complex geometry, thereby improving the

overall recall rate. However, this comes at the cost of longer processing times and larger file sizes, as demonstrated in Tab. II. By reducing the retention duration of the seed meshes from 10 scans to 1 scan, we are able to reduce the processing time from 0.384 to 0.295 sec and reduce the file size 8.6 to 5.9 MB. Notably, keeping all scans results in lower processing times by eliminating the overhead caused by the removal process, providing an extra benefit if file size is not prioritized. This trade-off can be achieved without any degradation in the precision.

V. Conclusions

In this work we introduced PlanarMesh, a novel real-time adaptive resolution mesh reconstruction system for 3D LiDAR data. By leveraging free-space information and introducing the Reverse Radius Field (RRF) for local curvature estimation, our method dynamically adjusts mesh resolution to capture both large-scale structures and fine detail efficiently. The system operates incrementally using multi-threaded BVH-based search operations, achieving real-time performance at approximately 2 Hz while significantly reducing memory footprint—producing mesh files up to 10 times smaller than raw input data. Our experimental evaluation demonstrates that PlanarMesh achieves accuracy on par with, or exceeding state-of-the-art methods, with output file size more than 5 times smaller, making it ideal for scalable 3D mapping in robotics and mobile scanning applications. At present, the reconstruction system uses continually more memory until it exceeds available memory at about 300 scans (about 300 m at 1 m/sec walking). In future work we aim to introduce a submapping mechanism to offload older mesh components and to achieve bounded memory usage. Additionally, we aim to incorporate loop closure for improved consistency in large-scale mapping and further explore mesh simplification and joining.

References

[1] 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. Robotics*, vol. 39, no. 6, pp. 4312–4331, Dec. 2023.

[2] J. Ruan, B. Li, Y. Wang, and Y. Sun, "SLAMesh: Real-time LiDAR Simultaneous Localization and Meshing," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2023, pp. 3546–3552.

[3] J. Niedzwiedzki, P. Lipinski, and L. Podsedkowski, "IDTMM: Incremental Direct Triangle Mesh Mapping," *IEEE Robotics and Automation Letters*, vol. 8, no. 9, pp. 5416–5423, Sep. 2023.

[4] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," in *Robotics: Science and Systems (RSS)*. Robotics: Science and Systems Foundation, Jul. 2014.

[5] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: surface elements as rendering primitives," in *SIGGRAPH*. ACM Press, 2000, pp. 335–342.

[6] T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Robotics: Science and Systems (RSS)*, Jul. 2015.

[7] C. Park, S. Kim, P. Moghadam, C. Fookes, and S. Sridharan, "Probabilistic Surfel Fusion for Dense LiDAR Mapping," in *IEEE/Intl. Conf. on Computer Vision Workshops (ICCV Workshops)*. IEEE, Oct. 2017, pp. 2418–2426.

[8] J.-D. Boissonnat, "Geometric structures for three-dimensional shape representation," *ACM Trans. on Graphics (TOG)*, vol. 3, no. 4, pp. 266–286, Oct. 1984.

[9] F. Bernardini and C. L. Bajaj, "Sampling and reconstructing manifolds using alpha-shapes," in *Canadian Conference on Computational Geometry (CCCG)*, 1997.

[10] N. Amenta, M. Bern, and M. Kamvysselis, "A new Voronoi-based surface reconstruction algorithm," in *SIGGRAPH*, 1998, pp. 415–421.

[11] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust," in *ACM Symposium on solid modeling and applications*, May 2001, pp. 249–266.

[12] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Trans. on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, Oct. 1999.

[13] R. Mencl, "A Graph–Based Approach to Surface Reconstruction," *Computer Graphics Forum*, vol. 14, no. 3, pp. 445–456, 1995.

[14] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013.

[15] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly, and S. Leutenegger, "Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1144–1151, Apr. 2018.

[16] Y. Cai, F. Kong, Y. Ren, F. Zhu, J. Lin, and F. Zhang, "Occupancy Grid Mapping Without Ray-Casting for High-Resolution LiDAR Sensors," *IEEE Trans. Robotics*, vol. 40, pp. 172–192, 2024.

[17] V. Reijgwart, C. Cadena, R. Siegwart, and L. Ott, "Efficient volumetric mapping of multi-scale environments using wavelet-based compression," *Robotics: Science and Systems (RSS)*, 2023-07.

[18] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH*, vol. 21, no. 4, pp. 163–169, 1987.

[19] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *SIGGRAPH*, vol. 26, no. 2, pp. 71–78, Jul. 1992.

[20] R. A. Newcombe, A. Fitzgibbon, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, and S. Hodges, "KinectFusion: Real-time dense surface mapping and tracking," in *IEEE/ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, Oct. 2011, pp. 127–136.

[21] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 1366–1373.

[22] I. Vizzo, T. Guadagnino, J. Behley, and C. Stachniss, "VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data," *Sensors*, vol. 22, no. 3, p. 1296, Jan. 2022.

[23] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," *Symposium on Geometry Processing*, 2006.

[24] I. Vizzo, X. Chen, N. Chebrolu, J. Behley, and C. Stachniss, "Poisson Surface Reconstruction for LiDAR Odometry and Mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2021, pp. 5624–5630.

[25] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast Direct LiDAR-Inertial Odometry," *IEEE Trans. Robotics*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.

[26] D. Wisth, M. Camurri, and M. Fallon, "VILENS: Visual, Inertial, Lidar, and Leg Odometry for All-Terrain Legged Robots," *IEEE Trans. Robotics*, vol. 39, no. 1, pp. 309–326, Feb. 2023.

[27] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A Survey of Surface Reconstruction from Point Clouds," *Computer Graphics Forum*, vol. 36, no. 1, pp. 301–329, Jan. 2017.

[28] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, "Embree: a kernel framework for efficient CPU ray tracing," *ACM Trans. on Graphics (TOG)*, vol. 33, no. 4, pp. 1–8, Jul. 2014.

[29] S. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: a general purpose ray tracing engine," *ACM Trans. on Graphics (TOG)*, vol. 29, no. 4, pp. 1–13, Jul. 2010.

[30] Y. Tao, M. Á. Muñoz-Bañón, L. Zhang, J. Wang, L. F. T. Fu, and M. Fallon, "The Oxford Spires dataset: Benchmarking large-scale LiDAR-visual localisation, reconstruction and radiance field methods," *arXiv preprint arXiv:2411.10546*, 2024.

[31] Y. Tao and M. Fallon, "Silvr: Scalable lidar-visual radiance field reconstruction with uncertainty quantification," *arXiv preprint arXiv:2502.02657*, 2025.