

# SLAMesh: Real-time LiDAR Simultaneous Localization and Meshing

Jianyuan Ruan<sup>1</sup>, Bo Li<sup>2</sup>, Yibo Wang<sup>1</sup>, and Yuxiang Sun<sup>1,\*</sup>

**Abstract**—Most current LiDAR simultaneous localization and mapping (SLAM) systems build maps in point clouds, which are sparse when zoomed in, even though they seem dense to human eyes. Dense maps are essential for robotic applications, such as map-based navigation. Due to the low memory cost, mesh has become an attractive dense model for mapping in recent years. However, existing methods usually produce mesh maps by using an offline post-processing step to generate mesh maps. This two-step pipeline does not allow these methods to use the built mesh maps online and to enable localization and meshing to benefit each other. To solve this problem, we propose the first CPU-only real-time LiDAR SLAM system that can simultaneously build a mesh map and perform localization against the mesh map. A novel and direct meshing strategy with Gaussian process reconstruction realizes the fast building, registration, and updating of mesh maps. We perform experiments on several public datasets. The results show that our SLAM system can run at around 40Hz. The localization and meshing accuracy also outperforms the state-of-the-art methods, including the TSDF map and Poisson reconstruction. Our code and video demos are available at: <https://github.com/lab-sun/SLAMesh>.

## I. INTRODUCTION

Simultaneous localization and mapping (SLAM) aims to estimate sensor poses and reconstruct traversed environments at the same time. 3-D LiDAR SLAM has attracted significant attention in the robotics research community in recent years. Usually, LiDAR SLAM produces a point-cloud map, which is a basic representation for 3-D environment models [1]. With such map type, registration and fusion of new range data can be straightforwardly operated [2]. Unfortunately, a point-cloud map requires extra data structures, such as KD-tree, to speed up the query of the unordered points. The memory cost also grows significantly with the increased scale or density. Moreover, although point-cloud map seems dense to human eyes, it would still be sparse when zoomed in. Due to the sparsity, it usually requires post-processing operations before the use for navigation, such as voxelization, to produce dense voxel maps.

In this work, we consider a dense map as a kind of representation that models 3-D objects in terms of surfaces or volumes. Under this definition, there exist several types of dense maps. For instance, occupancy grid maps [5] and truncated signed distance function (TSDF) maps [6] divide

This work was supported by the HK PolyU Start-up Fund under Grant P0034801, which is awarded to Dr. Yuxiang Sun.

<sup>1</sup>The Hong Kong Polytechnic University, Hung Hom, Hong Kong (e-mail: [jianyuan.ruan@connect.polyu.hk](mailto:jianyuan.ruan@connect.polyu.hk); [me-yibo.wang@connect.polyu.hk](mailto:me-yibo.wang@connect.polyu.hk); [yx.sun@polyu.edu.hk](mailto:yx.sun@polyu.edu.hk), [sun.yuxiang@outlook.com](mailto:sun.yuxiang@outlook.com)).

<sup>2</sup>Zhejiang University of Science and Technology, Hangzhou, China (e-mail: [120052@zust.edu.cn](mailto:120052@zust.edu.cn)).

\*Corresponding author: Yuxiang Sun.

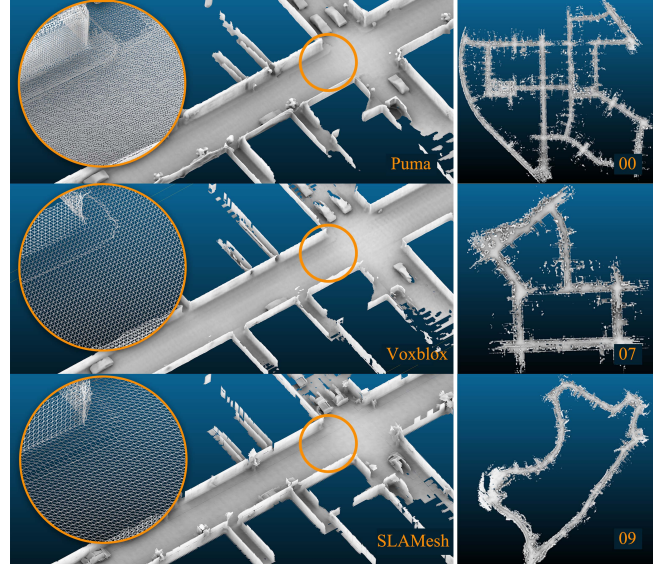


Fig. 1. Left: Mesh maps produced by SLAMesh, Puma [3] and Voxblox [4] on the Mai City dataset [3]. A zoomed view in each map highlights the connections of the mesh edges. The mesh map of Puma is watertight but with relatively complex multi-layers. The mesh maps produced by Voxblox and our SLAMesh are of high orderliness, and the vertices are evenly distributed. Right: Mesh maps built by our SLAMesh on the KITTI sequences 00, 07, and 09 demonstrate the accuracy and scalability of SLAMesh in large-scale environments.

the 3-D space into voxels, making them applicable for robot navigation [4]. However, the discretization introduces the scalability problem and limits the localization accuracy to the grid size. The normal distribution transform (NDT) [7][8] and surfel maps [9][10][11] can also be seen as dense maps. They use parameterized representations to describe local structures. Parameterization ensures memory efficiency but limits the ability to describe fine details unless a high resolution is used. Besides, gaps exist around the NDT and surfel map elements.

Mesh is a 3-D solid surface comprised of faces, edges, and vertices. In the field of 3-D modeling, the triangular mesh has become the dominating representation because it is simple and can approximate most complex 3-D structures. Unlike grid maps, mesh maps alleviate the discretization problem and can model smooth surfaces for robotics applications. Mesh can also depict manifold structures compared to surfel and NDT maps. In addition, mesh map is memory efficient, scalable, and retains the topological information [12][13].

However, real-time 3-D LiDAR mesh mapping is still hard to realize now, especially in large-scale environments. The main difficulty is that mesh building and updating are usually time-consuming. A widely-adopted solution to this problem

is first focusing on point-cloud mapping and then performing meshing separately using the point-cloud maps. This two-step solution could not enable localization and meshing to benefit each other [14].

To address this issue, this work designs a simultaneous localization and meshing system named SLAMesh. The system can run in real time with only a CPU. In SLAMesh, we make the vertices uniformly distributed, enabling mesh building, registration, and updating more efficiently. Fig. 1 shows such a mesh map constructed by the Gaussian process (GP) [15] reconstruction. Our previous works [16][17] have shown the benefits of utilizing GP in SLAM, especially when the range data is sparse. However, they produce point-cloud maps without exporting the surfaces model. In addition, they cannot achieve real-time performance when using sensors with denser point clouds, like 64-beam LiDAR. This work exploits GP for mesh map building and is an upgraded version upon GP-SLAM+ [17]. The main contributions of this work are summarized as follows:

- We propose a novel meshing strategy based on GP reconstruction and vertices connection, which allows fast building, query, and updating of mesh maps.
- We design a point-to-mesh registration method. Together with constraints combination and multi-thread implementation, we ensure the efficiency and accuracy of localization and mapping with the mesh map.
- We develop a dense, real-time, and scalable open-source LiDAR SLAM system upon mesh map and demonstrate the advantages through experiments.

## II. RELATED WORK

Indoor dense mapping is more investigated than the outdoor case. A widely-adopted idea to build mesh is fusing depth data into a TSDF map and then extracting the implicit mesh on demand through the Marching cubes algorithm [18] like Voxelblox [4] and Kinectfusion [19]. On the contrary, some methods produce mesh maps explicitly. Ryde *et al.* [20] voxelized point clouds and regressed a plane inside each voxel to form polygons. Piazza *et al.* [21] used the Delaunay triangulation [22] on the sparse feature points to build and update the mesh. Schreiberhuber *et al.* [23] connected the depth point in the camera image and then stitched the local mesh patches together. Our work explicitly produces mesh maps based on GP. Different from [23], our SLAMesh tackles unordered point clouds and builds mesh maps in the world frame rather than merging mesh patches in different image frames.

Recently, reliable 3-D LiDARs have enabled outdoor mapping. The outdoor environments are more challenging than indoor ones because the scenarios are several magnitudes larger, vehicle movements are fast, and the point cloud is sparse in remote areas. Suma [9] and Suma++ [10] provide accurate odometry, but the surfel map is relatively cluttered. LiTAMIN2 [8] shows the efficiency advantage of NDT registration with a map consisting of ellipsoids. Based on the TSDF map, Kühner *et al.* [24] modified the ray-tracking model for LiDAR and included an offline refinement process.

Roldão *et al.* [25] combined explicit and implicit meshing approaches for better accuracy and mesh density. Those methods [24][25] mainly focus on mapping function and can not provide real-time odometry. Puma [3] and our SLAMesh both uses mesh maps. However, Puma utilizes ray-casting for registration and needs to rebuild the mesh map repetitively via the Poisson reconstruction [26] rather than updating it. Lin *et al.* [27] also investigated online meshing in LiDAR SLAM. They use a separate SLAM system and a direct connecting meshing strategy upon point-cloud maps. Our method reconstructs unordered points to integrate the mesh map into the whole system.

The ray-tracking model in the above TSDF-based methods neglects the fact that local surfaces are continuous. Conversely, GP builds a continuous model accompanied by uncertainty. This property attracts people to use GP for recovering occupancy field [28] and TSDF field [29], which are continuous naturally. Different from [28][29], we use GP to predict the Euclidean coordinate of surfaces. Moreover, our system can also apply to large-scale environments.

## III. THE PROPOSED APPROACH

### A. Approach Overview

The motivation of this work is to build, utilize, and maintain a mesh map in a LiDAR SLAM system. Fig. 2 illustrates a general overview. The system mainly consists of three components: meshing, registration, and mesh management. Firstly, each new LiDAR scan is transformed into the world frame  $\{W\}$  as  $S^{raw}$  using the initial guess from a constant-velocity model. The following operations are also performed in  $\{W\}$ . Then, points are assigned into voxel cells. GP reconstructs the local surface inside each cell and obtains vertices  $v_i$ , which are connected to form a mesh. In the registration component, a point-to-mesh registration is designed to align the reconstructed current scan with the built mesh map  $\mathcal{M}$ . Finally, the mesh map is updated iteratively.

### B. Meshing Strategy

As aforementioned, building and updating mesh is time-consuming. To tackle this problem, we adopt a reconstructing and connection strategy to facilitate the following pipeline so that the whole system can run in real-time. As shown in Fig. 2, GP recovers the local surface from noisy and sparse point clouds inside voxels. Then, the vertices are interpolation results of the surface. Two coordinates of the 3-D vertices are evenly located (named as *locations*), and the other one (named as *predictions*) has a continuous value domain. The *locations* serve as the indices to enable fast query in constant time. The value domain of *predictions* is continuous to avoid discretization-induced accuracy loss.

Here the GP process is described (a more detailed description of GP can be found in [16][17]). The bold lowercase letters represent vectors, and the uppercase letters represent matrices. The input to GP is a subset of the raw point cloud  $S_k^{raw} = \{\mathbf{p}_i = (x_i, y_i, z_i, \sigma_{in}^2), i = 1, \dots, n_i\}$  containing  $n_i$  points, where  $k$  represents the  $k$ -th cell  $\mathcal{C}_k$  in the current scan,  $\sigma_{in}^2$  is the isotropic variance of input

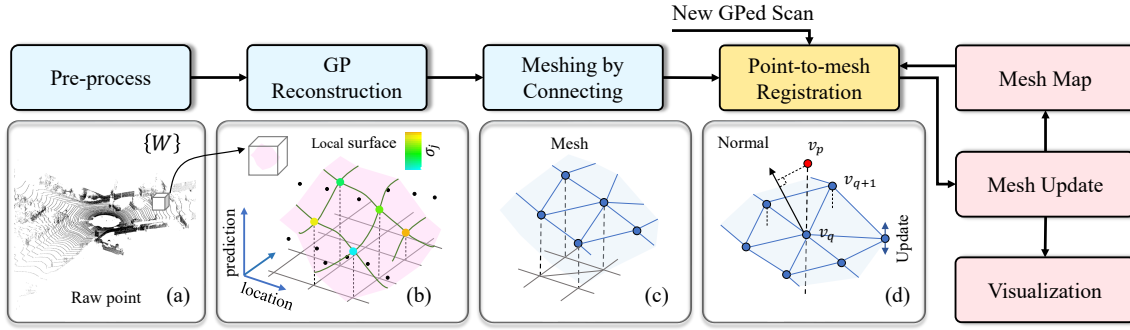


Fig. 2. The general diagram of our SLAMesh. The system has three parts: meshing, registration, and mesh management. (a) The raw point cloud is transformed into the world frame  $\{W\}$ , down-sampled, and assigned into voxel cells; (b) GP models the local surface (light purple) of each cell to get uniformly distributed vertices. The color of vertices indicates their uncertainty (warmer means higher), which increases with its distance to the raw points; (c) We build mesh (light blue) by connecting adjacent vertices directly; (d) Point-to-mesh registration aligns the reconstructed current scan (red points) to the mesh map. Fast matching is feasible based on the *locations* of vertices. The normal is smoothed from the surrounding mesh. GPed scan means scan after the GP reconstruction. Finally, we only need to adjust the 1-D *predictions* of vertices and their uncertainty to maintain the mesh map.

noise. The output is a *layer* containing  $n_j$  vertices  $\mathcal{L}_k^z = \{v_j = (x_j, y_j, z_j, \sigma_j^2), j = 1, \dots, n_j\}$  with uncertainty  $\sigma_j^2$ . The superscript  $z$  means that GP predicts the coordinate  $z$  as  $\tilde{f} = \{z_j, j = 1, \dots, n_j\}$ , and it is omitted when the coordinate can be any one in  $x, y$ , or  $z$ . We denote the *locations* of input and output point set as  $i$  and  $j$ . Given the input observation  $f$ , the *predictions*  $\tilde{f}$  also follow the Gaussian distribution. The expectation of  $\tilde{f}$  is:

$$\tilde{f} = k_{ij}^T (\sigma_{in}^2 I + K_{ii})^{-1} f. \quad (1)$$

The uncertainty of *predictions* is its variance:

$$\sigma_j^2 = k_{jj} - k_{ij}^T (\sigma_{in}^2 I + K_{ii})^{-1} k_{ij}, \quad (2)$$

where  $k_{jj}$ ,  $k_{ij}$ , and  $K_{ii}$  represent different combinations of the kernel function of *locations*. For instance,

$$k(i, j) = \exp(-\kappa |i - j|), \quad (3)$$

where  $k(i, j)$  is a scale value, and  $\kappa$  is a constant ( $\kappa = 1$  in our algorithm). This exponential kernel function can represent a local smooth surface in a 2-D manifold. Thus, one cell can contains more mesh *layers* with different GP functions (up to 3 if all coordinates are interpolated) when modeling a complicated structure [17].

A vertex with uncertainty  $\sigma_j^2$  under a certain threshold  $\sigma_{match}^2$  is accurate enough or valid. Triangle mesh faces are built by connecting adjacent or diagonal vertices in the 2-D space of *locations*. A mesh face is valid if all vertices are valid. Similar to the Delaunay triangulation, which establishes edges in a 2-D space, our method can prevent sliver mesh faces.

### C. Point-to-Mesh Registration

As aforementioned, we perform localization and meshing simultaneously so that meshing can benefit localization. To this end, one intuitive idea is to treat vertices as points or extract points from faces and then employ traditional point-cloud registration for pose estimation. However, this idea neglects the normal information of the mesh faces. Puma [3] indicates that a point-to-mesh error could improve accuracy. Unlike the ray-casting data association in Puma, our SLAMesh establishes correspondences based on *locations*.

For a vertex  $v_p$  in the reconstructed current scan  $\mathcal{S}^{gp}$ , we first query the sub-mesh *layer* located in the same or adjacent cells (see Fig. 3), and then find the vertex  $v_q$  sharing the same *locations* (see Fig. 2(d)). Data association is established between  $v_p$  and the valid faces that contain  $v_q$ . Finding the  $n_q$  neighboring vertices along the edges in a mesh is quick. The normals of those faces that contain  $v_q$  are averaged to get a smoothed normal  $\bar{n}$  in case the surface is rugged:

$$\bar{n}_q = \frac{\sum_{q=1}^{n_q-2} (v_q - v_{q+1}) \times (v_q - v_{q+2})}{\sum_{q=1}^{n_q-2} \|(v_q - v_{q+1}) \times (v_q - v_{q+2})\|}, \quad (4)$$

where  $\|\cdot\|$  is the 2-norm. We denote the number of correspondences in one cell as  $n_q$  and the number of overlapped cells as  $K$ . The point  $v'_p$  and the point-to-mesh residual after applying the transformation  $T \in SE3$  including rotation  $R$  and translation  $t$  on  $v_p$  are:

$$\begin{bmatrix} v'_p \\ 1 \end{bmatrix} = T \begin{bmatrix} v_p \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_p \\ 1 \end{bmatrix}, \quad (5)$$

$$e_p = \bar{n}_q^T (v'_p - v_q), \quad (6)$$

where the superscript  $T$  means matrix transpose. The optimal relative transformation  $T$  is computed in the optimization problem where point-to-mesh residuals in all  $K$  overlapped *layers*  $\mathcal{L}_k$  are joined:

$$T = \underset{T}{\operatorname{argmin}} \sum_{k=1}^K \sum_{v_p \in \mathcal{L}_k, p=1}^{n_p} e_p. \quad (7)$$

We use the Levenberg-Marquardt algorithm in Ceres solver<sup>1</sup> to solve this non-linear least square problem. The analytical Jacobin matrix is derived to accelerate the solving process:

$$J = \frac{\partial e_p}{\partial T} = \frac{\partial e_p}{\partial v'_p} \frac{\partial v'_p}{\partial T}, \quad (8)$$

$$\frac{\partial e_p}{\partial v'_p} = \bar{n}_p^T, \quad \frac{\partial v'_p}{\partial t} = I, \quad \frac{\partial v'_p}{\partial R} = - (R v_p + t)_{\times}, \quad (9)$$

where the symbol  $(\cdot)_{\times}$  represents the corresponding skew-symmetric matrix of a vector. The mesh may contain many

<sup>1</sup><http://ceres-solver.org/>

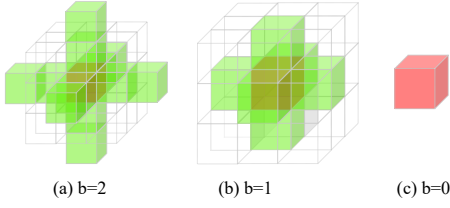


Fig. 3. The data association can be established across cells alongside the predicted coordinates (green cells). The query length  $b$  can be decreased as registration tends to converge. From (a) to (c),  $b$  is decreased from 2 to 0.

faces so the optimization problem can be huge. We combine the residuals inside each *layer* to be one by averaging before optimization to speed up the optimization process.

Storing data in voxel cells induces discontinuities of data association on the border. If we only perform registration on overlapped cells, when the vehicle moves fast, the convergence basins of registration would be too narrow. Thus, we allow cross-cell overlapping by querying cells alongside the axis of *predictions* for each *layer* as shown in Fig. 3. The length of the query is decreased when the registration tends to converge.

#### D. Mesh Management and Multithreading

Mesh maps are more challenging to maintain than point-cloud maps and grid-based maps. Points and grids can be independent of their neighborhoods. On the contrary, mesh retains the topological structure with vertices connected to each other. The map update process should maintain existing connections and avoid sliver triangles. Typical solutions include re-meshing with Delaunay triangulation [21], maintaining an implicit field similar to Voxelbox [4], or repeating the meshing process like Puma [3]. SLAM requires a fast update of mesh. In our SLAMesh, the *locations* of vertices are fixed, so only the 1-D *predictions* need to be updated. Given the previous  $t$  data below the threshold  $\sigma_{update}^2$ , the new *prediction* can be solved by the iterative least square:

$$\tilde{f}_{t_k} = \sum_{t=1}^{t_k} (\tilde{f}_t \sigma_t^2) / \sum_{t=1}^{t_k} (\sigma_t^2), \text{ if } \sigma_t^2 < \sigma_{update}^2. \quad (10)$$

The cells are stored in a hash map, which ensures a linear complexity for inserting, deleting, and querying. Also, this structure is flexible because it can grow incrementally. An advantage of our cell-based map shared with VGICP [30] is the compatibility for parallel processing. Since each cell is independent in these steps, we can accelerate the GP reconstruction and mesh publishing processes with multithreading.

### IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

#### A. Experimental Setup

We compare SLAMesh with several state-of-the-art SLAM systems or meshing tools. Algorithms are run on a PC with a 3.6 GHz 8-core Intel i7-11700KF CPU. In SLAMesh, the number of *predictions* in each cell  $n_j = 6^2$ . The cell size is 1.5m so that the resolution of *locations* is identical to the voxel size in Voxelbox when evaluating mesh. In other tests, the cell size is 1.6m. The variance of the input noise is  $\sigma_{in}^2 = 0.02$ . The thresholds  $\sigma_{match}^2 = 0.5$  and  $\sigma_{update}^2 = 1.0$ . These

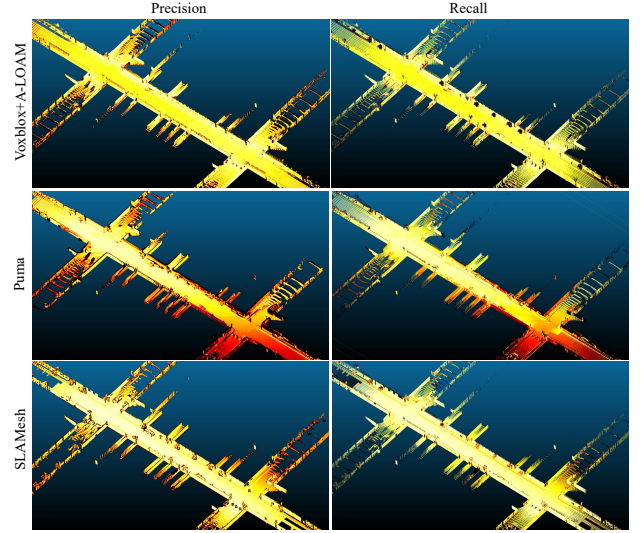


Fig. 4. Mesh maps compared with the ground-truth model on the Mai City dataset [3]. The left and right columns illustrate Precision and Recall values, respectively. Darker colors indicate worse performance and vice versa. Voxelbox recovers good results but fails to build thin structures like trees. Puma presents less accuracy on the edges of structures or areas where the pose drifts. Our SLAMesh presents both good Recall and Precision.

parameters are tuned manually. 8 threads are allocated in multithreading. We use the Mesh Tools [12] for visualization. Other methods are run in their default configurations.

#### B. Meshing Evaluation

The first experiment evaluates the meshing quality in terms of accuracy and completeness. We use the Mai City dataset [3] built on the CARLA simulation environment with ground truth of maps. In this dataset, the sensor is a simulated 64-beam Velodyne HDL-64E LiDAR. The vehicle drives for 99m and produces 100 frames of LiDAR scans. The ground truth of the environment is a dense point cloud scanned by a high-resolution sensor. We evaluate the combination of Voxelbox with A-LOAM<sup>2</sup>, Puma, and SLAMesh. Because this work aims to build a SLAM system rather than an offline mapping tool, and the ground truth pose is usually unavailable in real environments, we use the poses estimated in each pipeline or an extra odometry system (i.e., using A-LOAM for Voxelbox).

Fig. 1 presents the mesh map produced by each method. All maps reconstruct the main structures well. In the zoomed views, we can observe the characteristic of each strategy. The vertices in SLAMesh and Voxelbox are evenly organized due to the reconstruction or voxelization. The mesh of Puma is water-tight with Poisson reconstruction but is relatively complex with multi-layer phenomenons. We believe that this is because SLAMesh and Voxelbox both iteratively fuse observation into surfaces, while Puma accumulates several scans and then conducts a meshing upon them.

The quantitative evaluation uses the standard point-cloud-based metrics: Precision, Recall, and F1-score [31]. We densely and uniformly sample the mesh maps to form a point

<sup>2</sup><https://github.com/HKUST-Aerial-Robotics/A-LOAM>



TABLE I: QUANTITATIVE COMPARATIVE RESULTS (%) OF MESHING.  
THE BOLD FONT INDICATES BEST RESULTS.

Metric	Voxblox[4]+A-LOAM	Puma [3]	SLAMesh (Ours)
Precision	69.50	43.00	<b>74.96</b>
Recall	85.91	62.90	<b>86.09</b>
F1-score	76.83	51.08	<b>80.14</b>

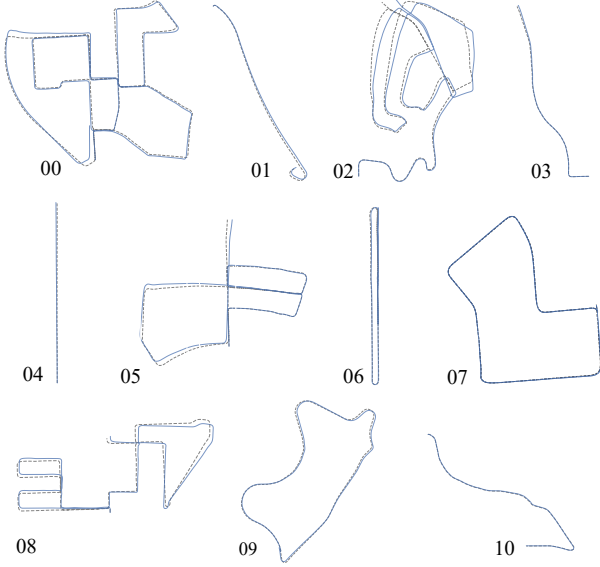


Fig. 5. Trajectories estimated by our SLAMesh on the KITTI 00 ~ 10 sequences (solid blue line) compared with the ground truth (dash gray line). Most trajectories are well consistent with the ground truth even without loop closures.

cloud, and then compare it with the ground-truth point cloud. The quantitative results are displayed in Table. I, where the distance threshold  $d$  is set to 0.3m. The performance of our SLAMesh is best, followed by Voxblox+A-LOAM and Puma. Fig. 4 visualizes the Precision and Recall results. The values decrease from lighter to darker. Voxblox erases tiny or thin objects like trees (the black areas in the Recall sub-figures) with its ray-tracing method. Puma accumulates errors along the trajectory. The water-tight assumption makes the top edge of walls slightly warped. Our SLAMesh can recover both large and small structures with high accuracy.

### C. Odometry Evaluation

We also quantitatively evaluate our pose estimation performance in the widely-used KITTI [32] odometry benchmark. The KITTI dataset provides point clouds from a Velodyne HDL-64E LiDAR and ground-truth poses. The results on the sequences 00 ~ 10 (including urban, country, and highway environments) are shown in Fig. 5. Our estimated trajectories present high consistency compared with ground truth.

Several state-of-the-art methods with different types of maps are tested. Suma [9] is a surfel-based method, and Suma++ [10] enhances the odometry accuracy with dynamic object removal. LiTAMIN2 [8] maintains a voxelized NDT map. A-LOAM is an implementation of LOAM [1]. Puma [3] is also a mesh-based method. Tab. II shows the quantitative evaluation results with the standard relative translation error and rotation error [32]. The results of these

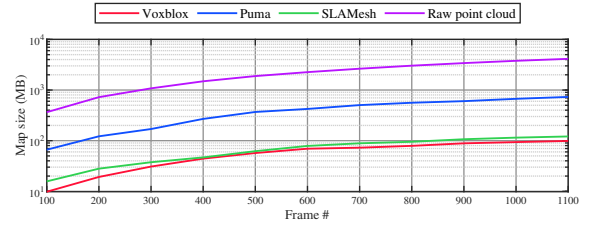


Fig. 6. Memory consumption of maps with the growing number of frames on the KITTI sequence 07. The raw point cloud can occupy several GB of disk space. The three mesh-based methods have lower memory costs. However, the mesh maps produced by Puma are more complex and heavier than those produced by Voxblox and SLAMesh.

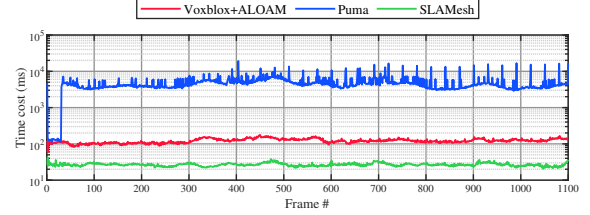


Fig. 7. Time cost per frame on the KITTI sequence 07. SLAMesh can run at 40Hz. The Voxblox+A-LOAM pipeline [4] achieves approximate real-time performance, while Puma cannot.

methods are directly imported from their published papers. Our SLAMesh achieves superior performance with 0.676% average translation error and 0.291deg/100m rotation error, which outperforms those point cloud, NDT, surfel, and mesh-based methods. Notice that we can achieve this without any loop closure. Also, as reported in [9][10], Suma needs GPU, and the surfel map is relatively cluttered.

Fig. 1 also displays the online mesh maps built by SLAMesh in the sequences 00, 07, and 09. The maps exhibit fine alignments when the vehicle travels back to the start. The maximum travel length of these sequences is 5.07km, which demonstrates the scalability of SLAMesh. An interesting observation is that when we allow the current scan to register with revisited areas, the map and the whole trajectory show better consistency, but when evaluating the relative error of odometry in KITTI, disabling this implicit loop closure enables higher accuracy. We believe this phenomenon is common in methods using cell-organized maps like [30].

### D. Memory and Computational Efficiency

The map size from each method in the KITTI sequence 07 is reported in Fig. 6. The raw point-cloud map proliferates to several GBs, which would be too heavy for online tasks. Three mesh-based methods demonstrate their superiority in memory efficiency. Voxblox and SLAMesh consume fewer memory resources than Puma. We conjecture the reason is that Puma does not iteratively fuse point clouds.

Fig. 7 displays the time cost per frame on the KITTI sequence 07. The time cost of SLAMesh does not increase with the scale of the scenario due to the cell-organized map. Note that the LiDAR runs at 10Hz in this dataset. However, Puma costs 4.7s per frame on average. For the first 30 frames, Puma uses iterative closest point (ICP) rather than mesh. The A-LOAM+Voxblox pipeline costs 129ms per frame (7.8Hz),

TABLE II: QUANTITATIVE RESULTS OF ODOMETRY ACCURACY ON THE KITTI DATASET. THE FIRST AND THE SECOND ROW ARE THE RELATIVE TRANSLATION ERROR (%) AND ROTATION ERROR (DEG/100M), RESPECTIVELY. THE RED BOLD AND THE BLUE FONTS INDICATE THE BEST AND THE SECOND BEST RESULTS, RESPECTIVELY. COMB. MEANS CONSTRAINTS COMBINATION, AND P2MESH MEANS POINT-TO-MESH REGISTRATION.

Map type	Method	00	01	02	03	04	05	06	07	08	09	10	Mean
Point Cloud	LOAM [1]	0.78	1.43	0.92	0.86	0.71	0.57	0.65	0.63	1.12	0.77	0.79	0.839
	A-LOAM	0.97 0.74	2.75 0.96	4.91 2.70	1.22 1.06	1.35 0.67	0.63 0.51	0.61 0.59	0.48 0.66	1.17 0.76	1.11 0.75	1.58 0.82	1.525 0.928
Surfel	Suma [9]	0.70 0.30	1.70 0.30	1.10 0.40	0.70 0.50	0.40 0.30	0.50 <b>0.20</b>	0.40 0.20	0.40 <b>0.30</b>	1.00 0.40	0.50 0.30	0.70 0.30	0.736 0.318
	Suma++ [10]	<b>0.64</b> <b>0.22</b>	1.60 0.46	1.00 0.37	0.67 0.46	<b>0.37</b> 0.26	<b>0.40</b> <b>0.20</b>	0.46 0.21	<b>0.34</b> 0.19	1.10 0.35	<b>0.47</b> <b>0.23</b>	<b>0.66</b> <b>0.28</b>	<b>0.701</b> <b>0.294</b>
NDT	Litamin2 [8]	0.70 0.28	2.10 0.46	0.98 0.32	0.96 0.48	1.05 0.52	0.45 0.25	0.59 0.34	0.44 0.32	0.95 0.29	0.69 0.40	0.80 0.47	0.883 0.375
Mesh	Puma [3]	1.46 0.68	3.38 1.00	1.86 0.72	1.60 1.10	1.63 0.92	1.20 0.61	0.88 0.42	0.72 0.55	1.44 0.61	1.51 0.66	1.38 0.84	1.551 0.737
	Full	0.77 0.35	<b>1.25</b> 0.30	<b>0.77</b> 0.30	0.64 0.43	0.50 0.13	0.52 0.30	0.53 0.22	0.36 0.23	0.87 0.27	0.57 0.25	<b>0.65</b> 0.42	<b>0.676</b> <b>0.291</b>
	SLAMesh (Ours)	0.96 0.41	1.70 0.41	1.04 0.39	0.81 0.55	0.67 0.30	0.51 0.29	0.64 0.30	0.46 0.28	<b>0.86</b> 0.30	0.74 0.32	1.02 0.44	0.857 0.363
	w/o Comb.	0.70 0.43	<b>1.33</b> <b>0.29</b>	<b>0.86</b> 0.38	<b>0.59</b> <b>0.42</b>	0.69 0.36	0.49 0.34	<b>0.37</b> <b>0.13</b>	0.43 0.40	1.14 0.42	0.77 0.29	0.93 0.57	0.756 0.366
	w/o P2Mesh	0.70 0.43	<b>1.33</b> <b>0.29</b>	<b>0.86</b> 0.38	<b>0.59</b> <b>0.42</b>	0.69 0.36	0.49 0.34	<b>0.37</b> <b>0.13</b>	0.43 0.40	1.14 0.42	0.77 0.29	0.93 0.57	0.756 0.366

which is approximately real-time. Our SLAMesh can run at 40Hz and surpasses other methods.

The time costs of each module are further analyzed in Fig. 8. They are generally categorized into three groups. The first part, pre-process, includes down-sampling and GP reconstruction in SLAMesh, feature extraction in A-LOAM, and normal computation in Puma. The second part is registration, including data association and optimization. The last part is map maintenance, including mesh map update and publishing. In Puma, the Poisson reconstruction is the main burden. In SLAMesh and A-LOAM, registration both iterates twice. The main computational cost of SLAMesh is the pre-process (account for 63%), while for A-LOAM, the registration and map update cost 87% time. This observation shows our different strategies for using reconstruction to structure data in advance. Voxblox is an efficient meshing tool due to its fast-integrate strategy. The max range of the sensor is cut into 50m rather than 100m, which also benefits its efficiency. However, the pipeline needs to maintain two maps, a point cloud map with KD-tree and a TSDF map, which costs more computation and memory resources.

#### E. Ablation Study

We perform an ablation study to investigate the contribution of each technique in our system. The parameter in SLAMesh is the same for all the variants. Tab. II shows the results. If SLAMesh disables the point-to-mesh error metric, the translation and rotation errors increase. This demonstrates that the pose estimation could be more accurate given more geometric information, like the normal of surfaces. The constraints combination also reduces the errors. The reason is that individual point-to-mesh error metric may not be robust enough in unstructured areas, and the constraints combination could suppress the influence of outliers.

Constraints combination and multithreading also play essential roles in efficiency improvement. In Fig. 8, the multithreading reduces the processing time of SLAMesh from 57.1ms to 23.8ms per scan. In detail, multithreading

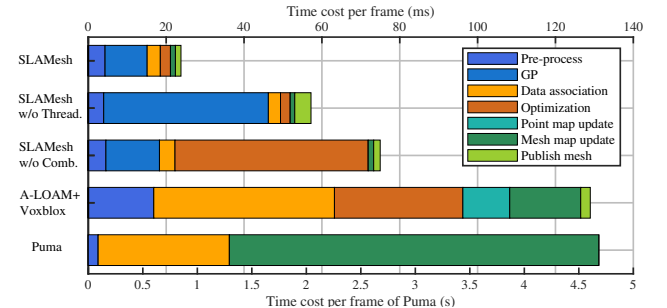


Fig. 8. The time cost breakdown for each module in different methods. The main burdens in SLAMesh, Puma, and A-LOAM are GP reconstruction, mesh building, and registration, respectively. The Voxblox+A-LOAM pipeline needs to maintain two maps. The constraints combination (Comb.) and multithreading (Thread.) notably accelerate the GP-reconstruction and optimization modules in our SLAMesh. Note that Puma uses the bottom x-axis, and the other methods use the top x-axis. The figure is best viewed in color.

makes the GP reconstruction three times faster and the mesh publishing 2.5 times faster. We note that Puma and Voxblox also leverage the multi-cores of the CPU in practice, which guarantees a fair comparison. The constraints combination saves about 95% time cost in optimization as the average number of constraints is reduced from 9,000 to 540.

We conduct extensive tests with a set of parameters and find that our method has a wide tolerance for parameters. As the cell size grows to 3m, the frame rate of SLAMesh can increase to about 80Hz with 0.88% relative translation error. The reason is that we use an un-parameterized model inside each voxel rather than a simple element.

#### V. CONCLUSIONS

We proposed a novel real-time localization and meshing system based on the reconstruction and connection strategy. Uniformly populated mesh vertices are the main property of the map, which lays a good foundation for mesh building, data association, and mesh update. Experiments showed that SLAMesh presents high odometry accuracy, efficiency, and mesh quality compared to state-of-the-art methods.

## REFERENCES

- [1] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, 2017.
- [2] Paul J Besl and Neil D McKay. Method for registration of 3-D shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.
- [3] Ignacio Vizzo, Xieyuanli Chen, Nived Chebrolu, Jens Behley, and Cyrill Stachniss. Poisson surface reconstruction for lidar odometry and mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.
- [4] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373. IEEE, 2017.
- [5] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.
- [6] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.
- [7] Martin Magnusson, Achim Lilienthal, and Tom Duckett. Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827, 2007.
- [8] Masashi Yokozuka, Kenji Koide, Shuji Oishi, and Atsuhiko Banno. Litamin2: Ultra light lidar-based slam using geometric approximation applied with kl-divergence. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11619–11625. IEEE, 2021.
- [9] Jens Behley and Cyrill Stachniss. Efficient surfel-based slam using 3d laser range data in urban environments. In *Robotics: Science and Systems*, volume 2018, 2018.
- [10] Xieyuanli Chen, Andres Milioto, Emanuele Palazzolo, Philippe Giguere, Jens Behley, and Cyrill Stachniss. Suma++: Efficient lidar-based semantic slam. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4530–4537. IEEE, 2019.
- [11] Chanoh Park, Peyman Moghadam, Jason L. Williams, Soohwan Kim, Sridha Sridharan, and Clinton Fookes. Elasticity meets continuous-time: Map-centric dense 3d lidar slam. *IEEE Transactions on Robotics*, 38(2):978–997, 2022.
- [12] Sebastian Pütz, Thomas Wiemann, and Joachim Hertzberg. The mesh tools package—introducing annotated 3d triangle maps in ros. *Robotics and Autonomous Systems*, 138:103688, 2021.
- [13] Xieyuanli Chen, Ignacio Vizzo, Thomas Läbe, Jens Behley, and Cyrill Stachniss. Range image-based lidar localization for autonomous vehicles. *arXiv preprint arXiv:2105.12121*, 2021.
- [14] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.
- [15] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.
- [16] Bo Li, Yingqiang Wang, Yu Zhang, Wenjie Zhao, Jianyuan Ruan, and Ping Li. GP-SLAM: laser-based SLAM approach based on regionalized Gaussian process map reconstruction. *Autonomous Robots*, 44(6):947–967, 2020.
- [17] Jianyuan Ruan, Bo Li, Yingqiang Wang, and Zhou Fang. Gp-slam+: real-time 3d lidar slam based on improved regionalized gaussian process map reconstruction. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5171–5178. IEEE, 2020.
- [18] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
- [19] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. IEEE, 2011.
- [20] Julian Ryde, Vikas Dhiman, and Robert Platt. Voxel planes: Rapid visualization and meshification of point cloud ensembles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3731–3737. IEEE, 2013.
- [21] Enrico Piazza, Andrea Romanoni, and Matteo Matteucci. Real-time cpu-based large-scale three-dimensional mesh reconstruction. *IEEE Robotics and Automation Letters*, 3(3):1584–1591, 2018.
- [22] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.
- [23] Simon Schreiberhuber, Johann Prankl, Timothy Patten, and Markus Vincze. Scalablefusion: High-resolution mesh-based real-time 3d reconstruction. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 140–146. IEEE, 2019.
- [24] Tilman Kühner and Julius Kümmerle. Large-scale volumetric scene reconstruction using lidar. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6261–6267. IEEE, 2020.
- [25] Luis Roldão, Raoul de Charette, and Anne Verroust-Blondet. 3d surface reconstruction from voxel-based lidar data. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2681–2686. IEEE, 2019.
- [26] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, page 0, 2006.
- [27] Jiarong Lin, Chongjiang Yuan, Yixi Cai, Haotian Li, Yuying Zou, Xiaoping Hong, and Fu Zhang. Immesh: An immediate lidar localization and meshing framework. *arXiv preprint arXiv:2301.05206*, 2023.
- [28] Soohwan Kim and Jonghyuk Kim. *GPmap: A Unified Framework for Robotic Mapping Based on Sparse Gaussian Processes*, pages 319–332. Springer International Publishing, Cham, 2015.
- [29] Boram Lee, Clark Zhang, Zonghao Huang, and Daniel D Lee. Online continuous mapping using gaussian process implicit surfaces. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6884–6890. IEEE, 2019.
- [30] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Voxelized gicp for fast and accurate 3d point cloud registration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11054–11059. IEEE, 2021.
- [31] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.
- [32] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.