IEEE Robotics and Automation Letters (RAL) paper presented at
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Incremental Triangle Mesh Generation with Mesh Refinement*

Jakub Niedzwiedzki[1], Piotr Lipinski[2], *Member, IEEE,* and Leszek Podsedkowski[1]

***Abstract*— This letter presents an incremental algorithm to generate triangle meshes from Light Detection and Ranging (LiDAR) point clouds with mesh refinement. The algorithm produces triangle mesh directly from LiDAR output without storing a dense point cloud to create a high-quality triangle mesh. In our algorithm, as the number of captured points increases during the LiDAR operation and robot movement, the new scan points from the LiDAR output refine or extend the existing triangle mesh. Such an approach is suitable for computationally-constrained systems like aerial vehicles, mobile robots, and smartphones, as it requires relatively limited resources. Our algorithm can reconstruct the topology of city-sized scenes maintaining a maximum triangle mesh error below 2 cm much faster than state-of-the-art triangle mesh generation algorithms that we demonstrate on publicly available data sets.**

I. INTRODUCTION

Researchers have studied algorithms for generating triangle meshes that represent the surrounding environment for decades, but the proliferation of point cloud acquisition hardware has recently sparked more attention toward them. Unfortunately, current mesh generation algorithms frequently encounter difficulties generating meshes for specific regions, particularly when constructing a triangle mesh from dense light detection and ranging (LiDAR) point cloud using incremental expansion methods. This process also requires significant computational resources [1].

When the scene is large, and the robotic system has limited resources, there is a trade-off between localization accuracy, mapping accuracy, the cost of map storage, and computation time. The use of triangle mesh for scene representation can vastly reduce the resources required for map storage and manipulation and maintain accuracy, which makes it a reasonable alternative for the dense point cloud [2], surfels [3], truncated signed distance function (TSDF) [4] or volumetric representation [5].

This article introduces a new incremental triangle mesh generation algorithm using Incremental Direct Triangle Mesh Mapping (IDTMM). The algorithm allows for incremental triangle mesh generation and existing mesh refinement based on a point cloud captured by a moving robot and computing hardware mounted on a robot. Triangle meshes in robotics have shown promising results when applied to autonomous mobile robots [6], path planning obstacle avoidance, navigation [8], virtual reality [7], disaster area [9], and more, especially when considering large, unexplored areas which may change over time, as it allows to reduce the size of the point cloud that represents the surrounding environment. This reduction is particularly crucial for robotic applications, where resource limitations are common. Furthermore, the utilization of a triangle mesh representation enables the convenient addition of texture to the map. Making the algorithm incremental gives an additional advantage, as it combines the benefits of reduced size with the ability to promptly add extra triangles to the triangle mesh without experiencing delays.

Point cloud acquisition is an essential element of real-time triangle mesh generation algorithms. Our algorithm can generate a triangle mesh acquired by any source, but we apply it to the point cloud obtained using LiDAR. As a result, our system performs well in both large-scale environments and in poor lighting conditions. LiDAR also ensures lower measurement errors at large distances than RGB-D-based algorithms [10], structure-from-motion algorithms, and photogrammetry-based algorithms (for comparison see [11]).

Our algorithm builds upon existing LiDAR odometry algorithms such as LOAM [12] and other work by Niedzwiedzki et al. [13], which have proved efficient. The main novelty of our algorithm lies in the functionality which allows extending the triangle mesh or updating it, depending on the localization of a newly acquired point by the LiDAR. The algorithm updates the triangle mesh if the newly acquired point lies over the triangle mesh or expands the triangle mesh if it is close to the triangle mesh border. Otherwise, the algorithm stores the point cloud until it can create a new mesh fragment. That allows for fast LiDAR Odometry and Mapping-based large-scale triangle mesh generation using limited resources, as the algorithm is fast and the number of stored point clouds is minimal.

The organization of this paper is as follows. Section II provides an overview of the most relevant related work on triangle mesh generation. In Section III, we describe our triangle mesh generation and refinement algorithm. Section IV presents the experimental setup for triangle mesh generation algorithms' comparison. The evaluation in Section V includes quantitative results of triangle mesh quality and speed of our algorithm.

II. RELATED WORK

Three-dimensional mesh generation is a challenging task that can be optimally solved using various methods depending on data acquisition, application, and purpose of the

*This work was supported by the NCBiR Agreement No. POIR.04.01.04-00-0040/17-00. and MNiSW Agreement No. DI2019 005149

[1] Jakub Niedzwiedzki and Leszek Podsedkowski are with the Institute of Machine Tools and Production Engineering, Lodz University of Technology, al. Politechniki 8, 93-590 Lodz, Poland jakub.niedzwiedzki@p.lodz.pl

[2] Piotr Lipinski is with the Institute of Information Technology, Lodz University of Technology, al. Politechniki 8, 93-90 Lodz, Poland piotr.lipinski@p.lodz.pl

Copyright ©2024 IEEE

triangle mesh [14]. Here we focus on large-scale, unstructured, and unexplored environments and noisy point clouds as we construct the triangle mesh incrementally utilizing data continuously acquired from the moving vehicle. Often, researchers use TSDF-based algorithms like C-blox [15], Voxblox [16], or VDBFusion [17] in such applications. Although TSDF is still a standard technique for 3D mapping in robotics [5], in this work, we aim to obtain a triangle mesh at each timestamp. Employing TSDF requires using the Marching Cubes to convert it to a triangle mesh, which is suboptimal because it consumes unnecessary resources.

The performance and complexity of the incremental triangle mesh generation algorithm strongly depend on the scale of the environment. For example the algorithm by Schops et al. [18] or Rosinol et al. [19] can reconstruct and render medium-scale environments (e.g., 300 square meters) in real-time at a resolution of 2-3 cm using dedicated hardware. However, they are not effective in large-scale environments. In contrast, the LiDAR odometry and mapping (LOAM) algorithm excels in handling large-scale environments but stores a complete point cloud representation instead of a triangle mesh. Conversely, the ImMesh [20] and CT-ICP [21] algorithms rely on voxel-based representation, which involves uniform space partitioning and subsequent erosion processes to merge triangles from neighboring voxels in InMesh algorithm, which is suboptimal in our approach. KISS-ICP [22] subsamples the point cloud to reduce the computation complexity. We follow the same simplified approach, but instead of subsampling, we create the triangle mesh directly from the points acquired by the LiDAR.

The Point Cloud Library [23] delivers several triangle mesh generation algorithms, such as GreedyProjcloudsn-Triangulation [24], MovingLeastSquares [25], Organized-FastMesh [26], Convex Hull, Concave Hull [27], Ear Clipping [28], GridProjection [29]. Still all these implementations are not incremental and fail in large-scale, rough terrain.

One of the most challenging tasks in incremental triangle mesh generation is triangle mesh expansion, namely merging existing triangle mesh with new triangles or new mesh fragments. Several approaches address this problem by leveraging structural regularity [30], point label information [31], segmentation [32], semantic information [33], stable region detection or object features in a point cloud [15]. Still, these methods are not applicable in unstructured and unexplored environments. In such a case, one can use overlapping areas as in algorithms by Marton et al. [31] and by Piazza et al. [1], but this requires unnecessary re-computation of existing mesh with minimal impact on the fidelity of the environment mapping. To avoid it, we add individual triangles or merge existing triangle mesh fragments without recalculating existing mesh. We also apply triangle mesh updates when the LiDAR acquires new points close to the existing mesh. This iterative approach improves the quality of the triangle mesh over time and prevents the generation of multiple layers of overlapping triangles seen in iterative closest point-based algorithms and Poisson-based algorithms , like PUMA [35]. These algorithms, however, are incremental and perform relatively well in the considered conditions, but they are resource-consuming, as they require existing triangle mesh recalculation during mesh expansion. In the next section, we introduce a much more efficient, incremental algorithm for triangle mesh generation from LiDAR scan point cloud - IDTMM - which is the main contribution of this paper.

III. APPROACH

In the first step of the algorithm we employ the Feature Point Extraction step originally introduced with the LOAM algorithm [12]. Next, we verify the relative location of the newly acquired point and the existing triangle mesh. Our algorithm acts differently depending on the relative location of the newly acquired scan point and the triangle mesh. In general, we consider the following four cases (see Fig. 1)

A.1) if the distance between the newly acquired point and the existing triangle mesh is above the threshold r, we store it for future use. We call such points 'accumulated points'; see Fig. 1a,

A.2) if the distance between the newly acquired point and the existing triangle mesh is above r and the number of accumulated points, marked as gray dots in Fig. 1b, at a distance closer than r from the newly acquired point is above $N = 10$ we trigger the generation of a new independent fragment of the triangle mesh; In the example shown in Fig. 1b N = 8 for clarity of the image. Section IIIA provides a detailed analysis of the selection of the N value,

B) if the distance between the newly acquired point and the existing triangle mesh is below r and the projection of the newly acquired point along the laser beam lies on the existing triangle mesh, as shown in Fig. 1c, we use the newly acquired point to update the existing triangle mesh,

C) if the distance between the newly acquired point and the existing triangle mesh is below r and the projection of the newly acquired point along the laser beam is not located on the existing triangle mesh, as shown in Fig. 1d, we use the newly acquired point to expand the existing mesh.

The following sections describe the detailed behavior of the algorithm.

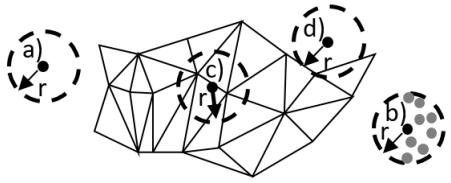

Fig. 1. Scan point relative location to the existing triangle mesh

A. Independent fragments of the triangle mesh generation

The algorithm builds a triangle mesh from newly acquired LiDAR scan points with no starting triangle mesh and no accumulated points. As a result, it always starts from case a)

on Fig. 1 described in the previous section and accumulates newly acquired points. The accumulation of new points continues until a newly acquired point has N points in the local neighborhood with radius r, i.e., a newly acquired point satisfies condition b) depicted in Fig. 1 from the previous section. This triggers the generation of the first local fragment of the triangle mesh, which we call a diamond. The diamond consists of two triangles that share one edge. We decided to use the diamond, not a single triangle, as it reflects the shape of the input local point cloud better than a single triangle and has a straightforward interpretation explained below. To find the diamond shape that reflects the shape of the corresponding accumulated input points, we find the centroid of N+1 accumulated points marked with a cross in Fig. 2 and calculate the covariance matrix of these N+1 points. Next, we use Singular Value Decomposition on the computed covariance matrix to find singular values and vectors of inertia of the accumulated point cloud. We then use the two most prominent vectors of inertia to find an approximation of directions of the most significant variance, see Fig. 2. Those directions are then treated as diamond diagonals. These diagonals cross in the centroid of the N+1 accumulated points. The lengths of diagonals are equal to the square root of the corresponding singular value. The algorithm removes all the points that it used to generate the diamond from the list. It removes other accumulated points not used for triangle generation after 3 seconds at most. This is because such scan points can represent outliers or moving objects. As a result, the number of scan points stored in memory during the algorithm operation is very low relative to the total number of scan points used in other incremental triangle mesh generation algorithms.

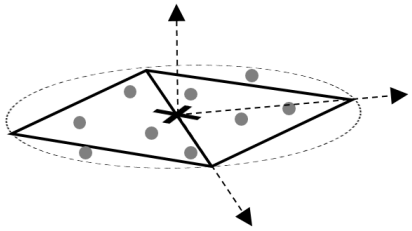

Fig. 2. Independent fragments of the triangle mesh generation - the diamond

B. Triangle Update

The IDTMM algorithm triggers the triangle update step process when the distance between the newly acquired point and the existing triangle mesh is below r, and the projection of the newly acquired point along the laser beam lies on the existing triangle mesh. This is true only when the laser beam crosses the triangle mesh. Considering only the triangle mesh vertices, which are closer than r, we have to analyze only a tiny fragment of the mesh to find the projection. We find the laser beam vector using the robot's location and the coordinates of the newly acquired point. When the newly acquired point meets the conditions mentioned above, we find the triangle closest to it; see Fig. 3. Next, we apply Extended Kalman Filter (EKF)-based algorithm to modify the vertices of this triangle. This modification influences the closest triangle and transforms all triangles that share the vertices and edges with the selected triangle. As a result, the update step modifies the fragment of the triangle mesh. The modification influences both the triangle vertices and their variance. The IDTMM algorithm performs this modification using an EKF-based algorithm based on the location of the newly acquired points covariance matrix, the location of the closest triangle vertices, and its variance.

The measurement model used expresses the distance measured by the scanner, projected onto the direction orthogonal to the selected triangle to the scan point:

$$z_t = n^T \left(P_C^G - P_S^G\right) = h\left(X_t, P_S^S, P_1, P_2, P_3\right) + s_{h_t}, \quad (1)$$

where X_t is robot localization in 6DOF, P_S^S is the scan point in the scanner frame, $P_S^G = \left[x_S^G, y_S^G, z_S^G\right]^T = f\left(X_t, P_S^S\right)$ is the scan point transformed into the global frame and n is the normal vector of the terrain plane calculated based on the nearest-triangle points, P_1, P_2, and P_3, and the center of the scanner, P_C^G. The parameter s_{h_t} represents the noise in the measurement model, including the scanner's measurement noise, the map noise, and the robot's estimated position noise.

The estimated measurement expresses the distance between the scanner center P_C^G and the scan point projected onto the terrain plane, $P_P^G = \left[x_P^G, y_P^G, z_P^G\right]^T$, as measured in the normal direction:

$$\hat{z}_t = n^T \left(P_C^G - P_P^G\right). \quad (2)$$

The innovation value used to compute the updated pose of the triangle is the difference between the measurement and its estimated value. It represents the distance from the scan point to the plane of the nearest triangle, as measured in the normal direction (see Figure 3):

$$i_t = z_t - \hat{z}_t = n^T \left(P_P^G - P_S^G\right) \quad (3)$$

Under the assumption that the scanner measurement, the estimated pose of the robot, and the estimated map parameters are independent, we can express the innovation covariance S by following the error propagation rule:

$$S_t = HPH^T + N_{scan} P_{scan} N_{scan}^T + N_{map} P_{map} N_{map}^T, \quad (4)$$

where H, N_{scan} and N_{map} are Jacobian matrices of the forms $H = \frac{\partial h}{\partial X_t}$, $N_{scan} = \frac{\partial h}{\partial P_S^S}$, and $N_{map} = \frac{\partial h}{\partial (P_1, P_2, P_3)}$, respectively.

P is a covariance matrix of a robot state, P_{scan} is the covariance of the scan point in the local frame, and P_{map} is the covariance of the triangle extracted from its' vertices (P_1, P_2, and P_3), respectively.

According to our research, it is better to use the same value of variance for all three dimensions to involve the measurement noise for each vertex in the process of triangle mesh update than to use the covariance matrix. If we define the triangle vertex variance as a 3x3 covariance matrix, in

the update phase of the EKF algorithm, the variance in the direction normal to the triangle surface tends to decrease over time, while the variance in two perpendicular directions tends to remain stable. That generates excessive modifications of triangle mesh vertices in the plane of the closest triangle. If we use the minimum variance value for all three dimensions, the triangle mesh updates more smoothly.

Based on the innovation covariance S, we calculate the Kalman gain as follows:

$$K_t = P_{map} N_{map}^T S_t^{-1}. \tag{5}$$

Finally, we calculate the vertex update vector length:

$$\Delta X_{t|t} = K_t i_t. \tag{6}$$

Then, we update each vertex along the direction of the triangle's normal vector as calculated with the EKF.

Since the innovation covariance matrix S has dimensions of 1×1, the execution of the EKF procedure is very fast (it is not necessary to perform a time-consuming matrix inversion operation), and it is possible to perform this correction procedure after every single laser scanner measurement.

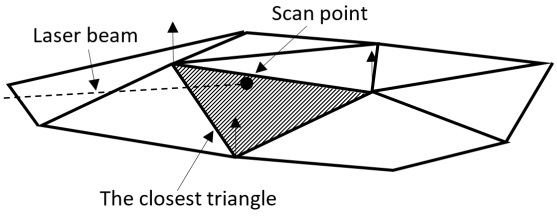

Fig. 3. Triangle update

C. Triangle Mesh Expansion

The IDTMM algorithm triggers the triangle mesh expansion step when the distance between the newly acquired point and the existing triangle mesh is below r, and the projection of the newly acquired point along the laser beam is not located on the existing triangle mesh. In contrast to the triangle update step, this is true only when the laser beam does not cross the triangle mesh. By analogy to the triangle update step, we consider only the triangle mesh vertices closer than r, so we have to analyze only a tiny fragment of the mesh to find the projection. We start the triangle mesh expansion when the newly acquired point meets the mentioned conditions. The expansion step is the most complex of all steps as it must consider several options for the relative localization of a newly acquired point to the triangle mesh. In general, the goal of the mesh expansion procedure is to add all possible triangles to the triangle mesh that take the newly acquired point as a vertex and do not break the integrity of the mesh. Triangle mesh expansion essentially has three sub-steps:

C.1) 2D projection,
C.2) triangle addition,
C.3) mesh merge.

which we detail below.

1) 2D projection: The procedure starts by finding planar projection on the plane perpendicular to the laser beam of all triangle mesh vertices at a distance of less than r from a newly acquired point A, see Fig. 4. The figure shows two triangle mesh fragments, both of which we extend and merge. We have deliberately chosen this example to cover all possible case variants of the introduced algorithm.

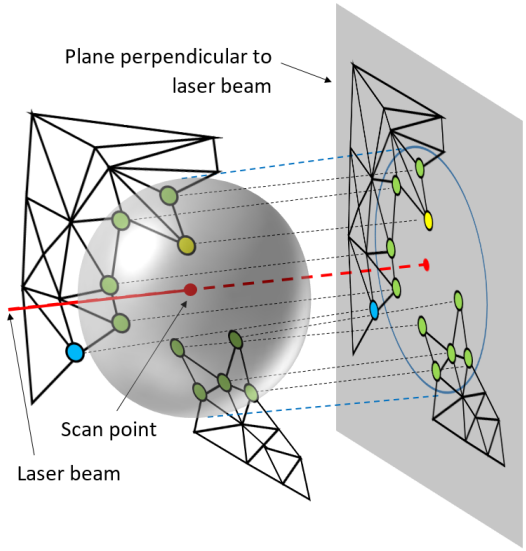

Fig. 4. Projection of triangle mesh fragments onto 2D plane normal to the laser beam

We perform further analysis on the projections of the vertices and the edges connecting them. Still, we add new triangles in 3D space and project them onto the plane perpendicular to the laser beam. From the projected vertices, we choose only 'open vertices', that is, the vertices containing exactly two edges linked with only one triangle. We call such edges 'open', as we can use them as an edge of a newly created triangle. As a result, our algorithm expands the triangle mesh with no risk of inconsistently splitting the mesh. We add the newly acquired point to the accumulated points list if there are no free edges with two end vertices closer to r from A. Otherwise, we find all open vertices closer to r from query point A; see Fig. 5. Next, we find the nearest free edge marked yellow and its closest end vertex marked yellow. We sort the remaining free vertices counterclockwise around the point A. We analyze these vertices according to this order in triangle generation and mesh merge procedures of triangle mesh expansion. Fig. 5 shows an exemplary scenario where we can potentially merge two independent triangle meshes.

2) Triangle addition: The procedure starts by analyzing the vertex at the other end of the counter-clockwise edge of vertex 0. This is vertex no. 2 in Fig. 5. If that point is next on our list of ordered points (which is not in our case) we add the triangle. Otherwise, we test if the next point on the list lies behind the edge by verifying the condition:

$$(\vec{V}_{lk} \times \vec{V}_{lm}) > 0, \tag{7}$$

where $\vec{V}_{lk}$ and $\vec{V}_{lm}$ are vectors ending in corresponding vertices k, l and m. In the example given in Fig. 5 k=1, l=0, m=2. If the condition (7) holds, we ignore this point and proceed to the next point on the sorted list. Next, we build the triangle $\Delta_{A\to 0\to 2}$ whose side is at the free edge 02, and A is its vertex. We check if the maximum edge length is below r:

$$max(|\vec{V}_{Ak}|, |\vec{V}_{Al}|, |\vec{V}_{kl}|) < r, \tag{8}$$

if the triangle is not too sharp:

$$min(\angle(\vec{V}_{Ak}, \vec{V}_{Al}), \angle(\vec{V}_{kl}, \vec{V}_{lA}), \angle(\vec{V}_{lA}\vec{V}_{Ak})) > \alpha_{min}, \tag{9}$$

and if the triangle is not flipped:

$$\angle(\vec{V}_{Ak}, \vec{V}_{Al}) < 180°, \tag{10}$$

If all the conditions mentioned above hold, we add the triangle to the triangle mesh. We repeat the procedure by analyzing the following free edges in counterclockwise order until we reach the last free edge, that is until the vertex is not connected by a free edge to the next vertex located closer than r from A. In the example shown in Fig. 5, there are three following free edges $E_{0\to 2}$, $E_{2\to 3}$, $E_{3\to 4}$, along with their corresponding triangles marked with red and blue vertex E which is not located within the r from A. The triangle generation stops at vertex E, as further mesh expansion in this direction is impossible. We stop the triangle addition procedure we move to the mesh merge procedure.

3) Mesh merge: The mesh merge procedure starts by verifying whether at least two separate mesh fragments are closer than r from A. We consider the mesh fragment to be separate if we cannot connect free edges using the triangle generation procedure. So, if the triangle generation procedure stops, and there are still free edges with end vertices located closer than r from A, the algorithm continues the triangle addition procedure for the following vertices (numbers 5 and 6 in our example in Fig. 5). The edge $E_{5\to 6}$ fulfills the condition given by (7), for k=5, l=6, m=6, so the triangle $\Delta_{A\to 5\to 6}$ is not created and we ignore the vertex 6. In the following step, we verify the conditions (8) - (10) for k=5, l=7 and, as they hold, we add the green triangle $\Delta_{A\to 5\to 7}$ to the second triangle mesh fragment, as shown in Fig. 5. Now we have two triangle mesh fragments connected by the single vertex A, which makes the meshes inconsistent, as we do not allow two mesh fragments to share only one vertex. Therefore, we try to add the blue triangle $\Delta_{A\to 4\to 5}$. We verify the conditions (8) - (10) for k=4, l=5. In the example from Fig. 5, the conditions given by (8) - (10) hold, so we add the blue triangle. If any of the conditions mentioned above do not hold, we add neither green nor blue triangles to the mesh, nor perform the mesh merge. Please note that the same mesh merge procedure applies when we cannot connect free edges using the triangle generation procedure in the same triangle mesh fragment. In such a case, the algorithm acts the same. Suppose during subsequent iterations, any of the conditions of the construction of a proper triangle do not hold, or further extension counterclockwise is not possible. In that case, we repeat both procedures in a clockwise direction for the remaining vertices and edges. The only difference in the clockwise direction is that the vector product in condition (7) is negative. The mesh expansion finishes when it is no longer possible to add any more triangles in the clockwise direction.

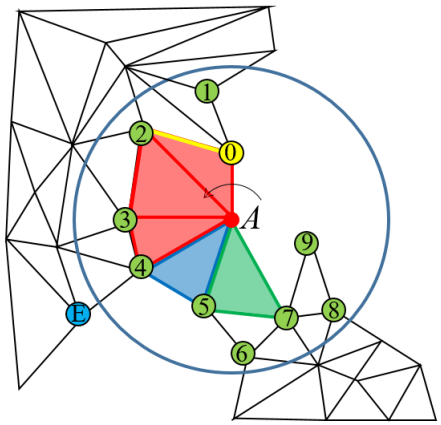

Fig. 5. The diagram illustrating the procedure for triangle addition and mesh merge

IV. RESULTS

Comparing the triangle mesh generation algorithms on a large scale has always been challenging, as precise reference values are difficult to obtain, especially when we capture data dynamically using moving LiDAR. Please note that in such a case, the reference location of the LiDAR moving along the path and the reference measurement point cloud suffer from significant measurement errors, making the comparison unreliable. Therefore, to compare the algorithms, we decided to simulate the LiDAR movement along the measurement path by selecting the measurement points from the static point cloud with no measurement errors mimicking the LiDAR installed on the robot capturing them. We added measurement noise reflecting the Velodyne VLP32 LiDAR angle measurement with sigma=0.3° for both vertical and horizontal angle and ray length measurement noise with sigma ranging from 1 to 4 cm. We used two data sets of different characteristics as the reference: publicly available synthetic data-set - *Synthcity*[1] and our data-set captured using FARO Focus X130 from the underground cave - *Cave*[2]. To conduct a comprehensive comparison, we also evaluated our algorithm using the *KITTI*[3] dataset, employing the captured point cloud as a reference. However, we opted not to employ the Mai City dataset utilized for the LOAM algorithm comparison, as we consider it too simplistic for our purposes.

We compared the method presented in Chapter III with the Point Cloud Library (PCL) [23] and PUMA [35], using Poisson surface reconstruction [34] [33] methods during our

[1][Online]. Available: www.synthcity.xyz
[2][Online]. Available: idtmm.pl
[3][Online]. Available: www.cvlibs.net/datasets/kitti/eval_odometry.php

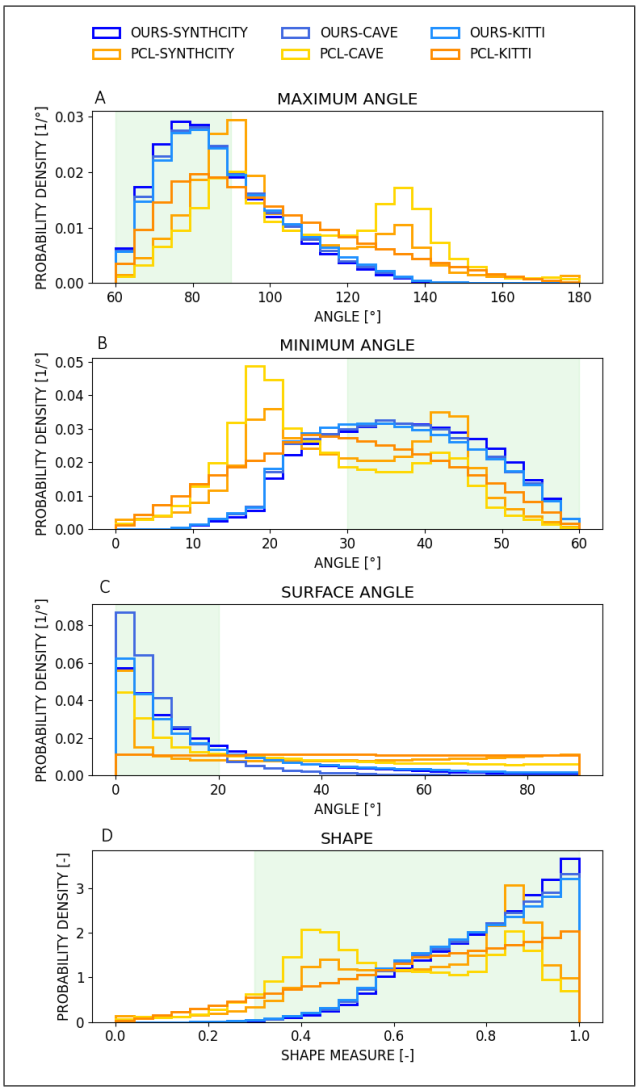

Fig. 6. The picture shows histograms of selected metrics of mesh quality assessment. Green areas on the plot show the acceptable range of values, the blue line shows results obtained by our method, while the orange line shows results obtained by the PCL method

research. The primary key to selecting the PCL method for comparison was its ability to create and expand the mesh while acquiring subsequent points incrementally. We have chosen the PUMA method as it is incremental, frequently used in many applications, and intensively improved.

To make the comparison as exhaustive as possible, we provided qualitative example presented on a film[4] and several different triangle mesh quality measures:

1) mean signed distance (MSD), mean unsigned distance (MUD), root mean square error (RMSE) between points on the triangle mesh obtained by the triangle mesh generation algorithm and the nearest point from the reference point cloud,
2) triangle density - number of triangles per square meter,
3) the histogram of maximum angle values of the triangles in triangle mesh,
4) the histogram of minimum angle values of the triangles in triangle mesh,
5) the histogram of surface angle - the angle between normal vectors of adjacent triangles,
6) the histogram of shape - for details, see [36].

Table I provides the numerical triangle mesh quality values, 1) - 2), for triangle meshes generated using PCL, PUMA, and our IDTMM algorithm. Fig. 6 depicts corresponding histogram-based triangle mesh quality measures, 3) - 6). Figure 7A shows a cross-section of the wall fragment from the Synthcity dataset. The black dashed line corresponds to the cross-section of the reference ideal wall shape, while red dots correspond to the LiDAR measurement data, which suffer from measurement noise with $\sigma = 4cm$. Fig. 7B and Fig. 7C show the triangle mesh cross-sections generated using PCL and PUMA algorithms, respectively.

We adjusted the meshes creation parameters for all algorithms so that the created meshes had similar sizes of triangles, i.e., the numbers of triangles per m^2 were close.

We benchmarked our algorithm utilizing an 8-core reference embedded device in a containerized environment. We believe such a setup is suitable for deployment on a mobile platform. Fig. 8 compares the computation time for PUMA, PCL, and IDTMM in such an environment.

TABLE I

THE TRIANGLE MESH QUALITY MEASURES FOR *Synthcity*, *Cave* AND *KITTI* DATASETS

Dataset	Method/noise	MSD [cm]	MUD [cm]	RMSE [cm]	Triangle count $[1/m^2]$
Synthcity	PCL 1cm	0.535	0.572	1.16	69
Synthcity	PUMA 1cm	0.104	**0.517**	**0.91**	**72**
Synthcity	Ours 1cm	**0.010**	0.740	1.56	81
Synthcity	PCL 3cm	0.731	0.782	1.33	73
Synthcity	PUMA 3cm	0.135	**0.593**	**1.00**	**71**
Synthcity	Ours 3cm	**0.020**	0.820	1.58	78
Synthcity	PCL 4cm	0.955	1.020	1.59	74
Synthcity	PUMA 4cm	0.137	**0.662**	**1.10**	**71**
Synthcity	Ours 4cm	**0.010**	0.870	1.62	77
Cave	PCL 1cm	0.125	**0.151**	**0.35**	115
Cave	PUMA 1cm	0.029	0.803	1.96	**63**
Cave	Ours 1cm	**0.009**	0.344	1.33	88
Cave	PCL 3cm	0.484	0.528	1.65	123
Cave	PUMA 3cm	0.021	0.802	1.98	**63**
Cave	Ours 3cm	**0.015**	**0.432**	**1.35**	86
Cave	PCL 4cm	0.714	0.777	2.12	121
Cave	PUMA 4cm	0.039	0.885	2.22	**62**
Cave	Ours 4cm	**0.038**	**0.487**	**1.18**	84
KITTI	PCL	*0.425*	*0.441*	*1.14*	139
KITTI	PUMA	4.050	4.37	7.81	136
KITTI	Ours	**0.879**	**1.03**	**1.45**	**55**

V. DISCUSSION

Our algorithm demonstrates a marked improvement in computation time when generating the triangle mesh incrementally, as demonstrated in Fig. 8. The triangle mesh generation in our algorithm initially exhibits slower performance than PUMA for the first 4s of generation. However,

[4][Online]. Available: idtmm.pl

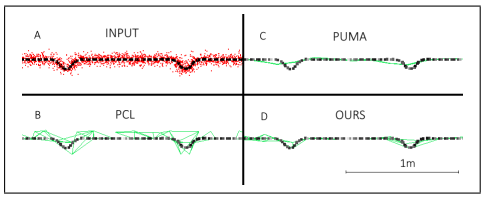

Fig. 7. The cross-section of part of the synthcity model. Red points represent the input point cloud with 4cm noise. Black points represent reference points, and green lines represent the wireframe of created mesh. A presents input noised point cloud, B presents mesh generated by the PCL method, C presents mesh generated by the PUMA method, while D presents mesh generated by our method

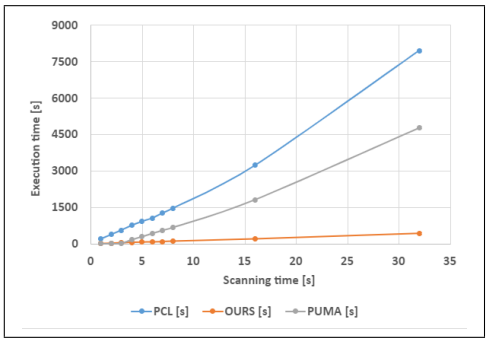

Fig. 8. Computation time comparison for PCL, PUMA and our algorithm.

once the algorithm finishes the initial triangle mesh fragment generation, it surpasses PUMA and outperforms PCL, particularly when the triangle mesh expands over time. We achieve these results by updating the orientation of individual triangles, incorporating new points to generate additional triangles, and merging with the existing mesh rather than recalculating the mesh fragment or entire triangle mesh. Our current implementation does not support parallel execution in any form nor we have optimized it for this use case, yet it is still 10 times faster than the PUMA method, which specifically takes advantage of CPU parallel paradigms.

We used the parameters presented in Fig. 6 and in Table I to evaluate and compare various aspects of the generated triangle point cloud characteristics.

First, let us analyze the results with simulated data sets: Synthcity and Cave. The MSD parameter describing bias for our method was the smallest and ranged from 0-0.4 mm. In terms of this parameter, the PCL method exhibited the poorest performance. On the other hand, the MUD and RMSE parameters describe the variance of reconstruction errors, i.e., the actual accuracy of the grid. From the table I, it is evident that the values of MUD and RMSE values exhibit notable variations with alterations in noise levels for the PCL method. In contrast, for the PUMA method and our proposed method, one can see that these parameters remain almost constant regardless of the level of noise. This implies that these methods perform well in filtering out measurement noise. In such a case, the noise is likely caused by two reasons: the complexity of the shape, which is impossible to reproduce using a triangle mesh with a minimal imposed triangle size or gross errors significantly affecting the triangle mesh quality. In the case of simulated data, it is only the first of these reasons. When analyzing the data from Table I, one can notice that for the Synthcity environment characterized by a large number of perfectly flat surfaces, all three methods produce very similar MUD and RMSE results, with the PUMA method providing the best results. This is the result of the strong smoothing characteristics of the PUMA method, which filters out all distortions and creates a maximally flat surface that is dominant in this data set, which is also apparent in Fig. 7. Fig. 7 also shows that our algorithm, thanks to the use of the Kalman filter, does not introduce unnecessary smoothing effects like PUMA or grid noise by voxelization like PCL. On the other hand, for the Cave environment, where all surfaces are uneven and characterized by a large shape variability, the PUMA method is the worst in terms of MSD, MUD, and RMSE. Indeed, the roughness of the cave environment is excessively smoothed. The data set under consideration demonstrates that the PCL method exhibits superior accuracy for low measurement noise owing to a significantly larger number of triangles in the output mesh, see Table I. When the measurement noise value reaches 3cm-4cm, which we believe more realistically reflects the natural environment scenario, the difference between the methods becomes insignificant. The KITTY experiment the lack of reference leads to a significant underestimation of the obtained results, especially in the case of the PCL algorithm. The reason behind this is that the PCL method generates vertices of triangles directly from the measurement points, and without the voxelization process, the outcome would have been zero.

When comparing the PUMA method and our proposed method, it is evident that our method has a several-fold advantage. One possible reason for this is that we involve all input data in the triangle mesh generation process and take into consideration the uncertainties of all input samples. This causes data collected at a considerable distance from the scanner subject to significant errors have minimal impact on the final result. Conversely, the PUMA method treats all points equally, leading to points with high uncertainties and significant errors substantially distorting the mesh.

The plots in Fig. 6 offer a comparison of the triangles in the triangle mesh. The green area represents the desired range of the triangles' characteristics. Histograms Fig. 6A and Fig. 6B and 6C depict the angle values of the triangles in the triangle mesh. We desire a regular triangle mesh without extreme angle values for its triangles. The histograms show that the triangle mesh generated using our algorithm is more regular than the triangle mesh generated using the PCL algorithm, as the number of triangles with high-angle values is smaller. Finally, the shape measure shown in fig. 6D quantifies the ratio between triangles in a triangle mesh and the equilateral triangle, which we believe is the ideal output triangle mesh [36]. A value of 1 indicates that the triangle is equilateral. The histogram in fig. 6D demonstrates that the proposed algorithm generates a triangle mesh predominantly consisting of equilateral triangles. Additionally, our algorithm produces a larger number of triangles that are similar in terms of shape

measure to the equilateral triangle compared to the PCL and PUMA algorithms.

VI. CONCLUSIONS

We propose an IDTMM algorithm that utilizes dense point clouds obtained from a LiDAR mounted on a mobile robot for unstructured and unexplored environments. The algorithm offers improved performance compared to state-of-the-art incremental algorithms, delivering increased speed while maintaining high accuracy and uniformity. Compared to PUMA and PCL algorithms, the IDTMM algorithm offers superior smoothness and accuracy in noisy environments. It generates triangle meshes directly from point clouds and refines the mesh using a Kalman-based update process, which distinguishes it from existing algorithms. As a potential future development, we plan to integrate it with a frame-to-mesh localization algorithm to create a simultaneous localization and mapping solution.

REFERENCES

[1] E. Piazza, A. Romanoni and M. Matteucci, "Real-Time CPU-Based Large-Scale Three-Dimensional Mesh Reconstruction", in *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1584-1591, July 2018, doi: 10.1109/LRA.2018.2800104.

[2] L. Linsen, "Point cloud representation". *Technical Report, Faculty of Computer Science, University of Karlsruhe*: Univ., Fak. für Informatik, Bibliothek. 2001.

[3] J. Behley and C. Stachniss, "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments", In *Proc. of Robotics: Science and Systems (RSS)*, 2018

[4] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images", in *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 303–312.ACM, 1996.

[5] E. Vespa, et al., "Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping," in *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1144-1151, April 2018, doi: 10.1109/LRA.2018.2792537.

[6] D. Clint et al., "Stochastic triangular mesh mapping: A terrain mapping technique for autonomous mobile robots", *Robotics and Autonomous Systems*, Volume 127, 2020, 103449, ISSN 0921-8890, doi: 10.1016/j.robot.2020.103449.

[7] C. Papachristos and K. Alexis, "Augmented reality-enhanced structural inspection using aerial robots", in *2016 IEEE International Symposium on Intelligent Control (ISIC)*, Buenos Aires, Argentina, 2016, pp. 1-6, doi: 10.1109/ISIC.2016.7579983.

[8] S. Weiss, et al. , "Intuitive 3D Maps for MAV Terrain Exploration and Obstacle Avoidance", *J Intell Robot Syst* 61, 473–493 (2011). doi: 10.1007/s10846-010-9491-y

[9] M. Salathe et al., "A multi-modal scanning system to digitize CBRNE emergency response scenes", in *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Sevilla, Spain, 2022, pp. 74-79, doi: 10.1109/SSRR56537.2022.10018826.

[10] A. Rosinol and L. Carlone, "Smooth Mesh Estimation from Depth Data using Non-Smooth Convex Optimization", in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6633-6640, 2021.

[11] L. Derek et al., "Comparison of Digital Photogrammetry and Laser Scanning", in: *Proc. International Society for Photogrammetr and Remote Sensing* 2002. p. 39-44.

[12] J. Zhang, and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time", *Robotics: Science and Systems*. 2014.

[13] J. Niedzwiedzki, et al., "Real-Time Parallel-Serial LiDAR-Based Localization Algorithm with Centimeter Accuracy for GPS-Denied Environments", *Sensors* 2020, 20, 7123. doi: 10.3390/s20247123.

[14] M. Berger et al., "A Survey of Surface Reconstruction from Point Clouds", *Computer Graphics Forum*, 2016, vol. 36, doi: 10.1111/cgf.12802.

[15] A. Millane, et al., "C-Blox: A scalable and consistent Tsdf-based dense mapping approach", in: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 995–1002, 2018, https://doi.org/10.1109/IROS.2018.8593427.

[16] H. Oleynikova et al., "Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1366-1373, 2017.

[17] I. Vizzo et al., "VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data", *Sensors*. 22. 1296, 2022, doi: 10.3390/s22031296.

[18] T. Schöps, et al., "3D Modeling on the Go: Interactive 3D Reconstruction of Large-Scale Scenes on Mobile Devices", in *2015 International Conference on 3D Vision*, 2015, pp. 291-299, doi: 10.1109/3DV.2015.40.

[19] A. Rosinol, et al., "Incremental Visual-Inertial 3D Mesh Generation with Structural Regularities", in *2019 International Conference on Robotics and Automation (ICRA)*, 8220-8226, 2019.

[20] L. Jiarong et al., ImMesh: An Immediate LiDAR Localization and Meshing Framework, arXiv preprint arXiv:2301.05206, 2023.

[21] P. Dellenbach et al., "CT-ICP: Real-time Elastic LiDAR Odometry with Loop Closure," 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 5580-5586, doi: 10.1109/ICRA46639.2022.9811849.

[22] I. Vizzo, et al. "KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way," in IEEE Robotics and Automation Letters, vol. 8, no. 2, pp. 1029-1036, Feb. 2023, doi: 10.1109/LRA.2023.3236571.

[23] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 1-4, doi: 10.1109/ICRA.2011.5980567.

[24] J. Liu, D. Bai and L. Chen, "3-D Point Cloud Registration Algorithm Based on Greedy Projection Triangulation", *Appl. Sci.* 2018, 8, 1776. doi: 10.3390/app8101776

[25] M. Alexa et al., "Computing and Rendering Point Set Surfaces. Visualization and Computer Graphics", *IEEE Transactions* on. 9, 3-15., 2003, doi: 10.1109/TVCG.2003.1175093.

[26] D. Holz and S. Behnke. "Fast Range Image Segmentation and Smoothing using Approximate Surface Reconstruction and Region Growing", in *Proceedings of the 12th International Conference on Intelligent Autonomous Systems (IAS)*, Jeju Island, Korea, June 26-29 2012, doi: 10.1007/978-3-642-33932-5_7.

[27] H. Edelsbrunner, D. Kirkpatrick and R. Seidel, "On the shape of a set of points in the plane", *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 551-559, July 1983, doi: 10.1109/TIT.1983.1056714.

[28] D. H. Eberly, "Triangulation by Ear Clipping", *Geometric Tools, LLC.*, www.geometrictools.com, 1998.

[29] R. Li et al., "Polygonizing extremal surfaces with manifold guarantees", in *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling (SPM '10). Association for Computing Machinery*, New York, NY, USA, 189–194, 2010, doi: 10.1145/1839778.1839808.

[30] A. Rosinol et al. "Incremental Visual-Inertial 3D Mesh Generation with Structural Regularities", in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8220-8226, doi: 10.1109/ICRA.2019.8794456.

[31] Z. C. Marton, R. B. Rusu and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," in *2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009, pp. 3218-3223, doi: 10.1109/ROBOT.2009.5152628.

[32] R. Dubé, et al. "SegMap: Segment-based mapping and localization using data-driven descriptors", *The International Journal of Robotics Research (IJRR)*, 2019, doi: 10.48550/arXiv.1804.09557.

[33] A. Rosinol et al., "Kimera: an OpenSource Library for Real-Time Metric-Semantic Localization and Mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020.

[34] M. Kazhdan, M. Bolitho, and H. Hoppe "Poisson Surface Reconstruction", in *Symposium on Geometry Processing*, 2006, doi: 10.2312/SGP/SGP06/061-070.

[35] I. Vizzo et al., "Poisson Surface Reconstruction for LiDAR Odometry and Mapping", in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China, 2021, pp. 5624-5630, doi: 10.1109/ICRA48506.2021.9562069.

[36] Knupp, P.M. (2001). "Algebraic Mesh Quality Metrics", *SIAM J. Sci. Comput.*, 23, 193-218.