

# Fast 3D Sparse Topological Skeleton Graph Generation for Mobile Robot Global Planning

Xinyi Chen, Boyu Zhou, Jiarong Lin, Yichen Zhang, Fu Zhang and Shaojie Shen

**Abstract**—In recent years, mobile robots are becoming ambitious and deployed in large-scale scenarios. Serving as a high-level understanding of environments, a sparse skeleton graph is beneficial for more efficient global planning. Currently, existing solutions for skeleton graph generation suffer from several major limitations, including poor adaptiveness to different map representations, dependency on robot inspection trajectories and high computational overhead. In this paper, we propose an efficient and flexible algorithm generating a trajectory-independent 3D sparse topological skeleton graph capturing the spatial structure of the free space. In our method, an efficient ray sampling and validating mechanism are adopted to find distinctive free space regions, which contributes to skeleton graph vertices, with traversability between adjacent vertices as edges. A cycle formation scheme is also utilized to maintain skeleton graph compactness. Benchmark comparison with state-of-the-art works demonstrates that our approach generates sparse graphs in a substantially shorter time, giving high-quality global planning paths. Experiments conducted in real-world maps further validate the capability of our method in real-world scenarios. Our method will be made open source to benefit the community.

## I. INTRODUCTION

A sparse topological skeleton graph is a compact undirected graph structure capturing the spatial structure of the environment representing free space regions as vertices and their traversability as edges. By providing a high-level understanding and abstraction of the environment, the sparse topological skeleton graph allows highly efficient global path planning, which is a fundamental problem for mobile robots. Instead of finding paths directly on a map, which is computationally demanding especially in large-scale space, high-quality paths can be searched rapidly leveraging the skeleton graph.

Although a few methods have been developed to extract sparse 3D topological skeleton graphs, there are still some major limitations. First, many of them need to pre-proceed the mapping results into the specific map representation that the methods are tailored for. For example, [1] requires maintaining a Euclidean Signed Distance Field (ESDF) to build a Generalized Voronoi Diagram (GVD) and [2] needs a grids-based map to grow clusters by dilating to neighboring grids. These requirements make them less flexible to be applied elsewhere and take extra processing time. Second,

This work was supported by HKUST Postgraduate Studentship and HDJI Lab. X. Chen, B. Zhou, Y. Zhang and S. Shen are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, China. {xchencq, bzhouai, eeshaojie}@connect.ust.hk. J. Lin and F. Zhang are with the Department of Mechanical Engineering, The University of Hong Kong, Hong Kong SAR, China. {jiarong.lin, fuzhang}@hku.hk

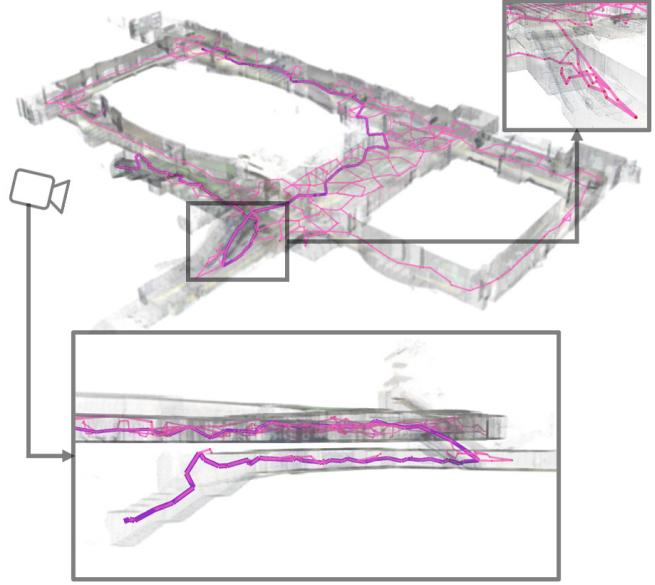


Fig. 1. A 3D sparse topological skeleton graph (pink) generated by the proposed method in a multi-floored real-world environment of size  $110 \times 60 \times 13 \text{ m}^3$  is demonstrated. A side-view from the left showing the multi-floor structure is placed at the bottom. Details of the skeleton graph passing an escalator are further enlarged in the top-right corner. Besides, a global planning A\* path searched on the skeleton graph is marked in purple. For more details, please check out the video.

methods like [2] typically follow a robot inspection trajectory to construct the skeleton graph. Generated in such a manner, the skeleton graph heavily depends on the trajectory, instead of reflecting the inherent structure of the environment. Lastly, current methods are known to be computationally expensive, which can consume unreasonable time for large-scale environments.

To address the aforementioned limitations, this work proposes an efficient and flexible approach to extract trajectory-independent sparse topological skeleton graphs from known environment maps. Our algorithm supports various types of map representation, as long as a collision checking interface is available. An efficient ray sampling and validating mechanism is adopted to iteratively extract distinctive free space regions, which contributes to vertices in the skeleton graph. The traversability between adjacent vertices corresponds to the skeleton graph edges. A cycle formation scheme is also utilized to maintain graph compactness.

We compare our approach with state-of-the-art works and evaluate the global planning performance of the skeleton graphs generated by different methods. Results show that the proposed method generates sparse skeleton graphs in a

TABLE I  
IMPORTANT ATTRIBUTES IN A FRONTIER

Name	Explanation
Facets	Component facets of the frontier
Normal	Average outward unit normals of facets
Center	A central position on one of its facets
Initial position	Position of a new node construction
Parent node	Node holding this frontier

TABLE II  
IMPORTANT ATTRIBUTES IN A VERTEX

Name	Explanation
Position	Location of the vertex
Type	Black / White
Detected polyhedron	The polyhedron that this vertex lies on
Projected position	Vertex projection on the unit sphere

substantially shorter time, capturing the spatial structure of free space precisely and giving high-quality global planning paths rapidly. Moreover, we conduct experiments on real-world maps containing significant noises. It demonstrates that our method is capable of handling noisy, complex and large-scale real-world environments, such as the multi-floored hall shown in Fig. 1.

In summary, the contributions of this work are:

- 1) An efficient and flexible algorithm that generates sparse trajectory-independent skeleton graph for mobile robots global planning, supporting various types of map representation as long as a collision checking interface is available.
- 2) Benchmark comparison shows that the efficiency and skeleton graph quality of the proposed method outperforms the state-of-the-art works. The methodology is also validated in complex and large-scale real-world maps. The source code of the proposed method will be made available to benefit the community.

## II. RELATED WORK

Skeleton graph generation in a given environment for mobile robot global planning is gaining increasing attention by the community in recent years. Generating skeleton graphs for 2D environments has been a well-studied topic. For the 2D case, most of the methods start with a Euclidean Signed Distance Field (ESDF) and build the skeleton making use of the Voronoi Diagram [3]–[5]. 3D skeleton graph generation is more difficult and still an open problem. Existing methods can be summarized as GVD-based and clustering-based methods.

### A. GVD-based Methods

It is natural to extend well-studied 2D methods into the 3D case, building Generalized Voronoi Diagrams (GVDs) for the skeleton graph generation. [6] developed a 3D-applicable algorithm to compute Voronoi diagram by combining Voronoi diagrams for 2D parallel slices, which are computed using graphics hardware. [7] combines [6] and probabilistic roadmaps [8] enabling path planning directly on the GVD graph. Moreover, several computer graphics literature studies on 3D shape skeletonization for object meshes [9] and point clouds [10]. On this topic, [11] presents a complete and detailed state-of-the-art report. However, these methods only consider regular small-scale objects, and are not directly applicable to large-scale maps containing substantial noises under mobile robot applications.

A recent GVD-based work [1] inspired by these works attempts to adapt these methods to complex and noisy environments for mobile robot navigation. They first generate a one-voxel-thick skeleton diagram from GVD and further extract a skeleton graph from the diagram. However, it still results in graphs containing numerous vertices and edges for noisy maps, affecting global planning efficiency. Besides, this

method requires an ESDF as input, whose maintenance is time-consuming, especially for large-scale environments.

### B. Clustering-based Methods

The key idea of clustering-based methods is to divide an environment into meaningful clusters, after which skeletons are generated by connecting neighboring clusters. One way is to generate keyframe or landmark clusters based on a similarity measure in visual SLAM [12]–[15]. But landmark clusters capture no information of free space, which is not desirable for safe global path planning. Besides, redundant clusters representing the same place may be created when this place is visited from different directions. Another kind of approaches is attaching local occupancy sub-maps along the metric SLAM map [16], where each sub-map form a cluster. These approaches only partially capture the topology of the space and can not be directly used for global planning.

A group of approaches that has advantages for motion planning is representing the environment as convex free-space clusters, as presented by some aerial robot trajectory planning literature [17]–[21]. Along an initial path, a series of convex free-space clusters are built in shapes of convex polyhedra [17]–[19], cubes [20] or spheres [21] and further used in trajectory optimization. However, these methods only grow clusters for regions around the initial path, which can not be directly applied to construct sparse topological graphs of the entire environment. The most relevant work to ours is Topomap [2], which grows clusters along the explorer trajectory in an occupancy map. Convexity and compactness of each cluster are ensured by exhaustively checking all voxels in every growing iteration, costing an unreasonable amount of time for growing large clusters.

A common downside of these works is that they are highly dependent on robot inspection trajectory, instead of reflecting the inherent topological structure of the free space. Centers of the clusters are often initialized along the given trajectory so the clusters only cover free space reached by it.

In contrast to these methods, we propose a novel approach that efficiently generates trajectory-independent sparse skeleton graph that reflects the topological structure of the environment and supports various types of map representation.

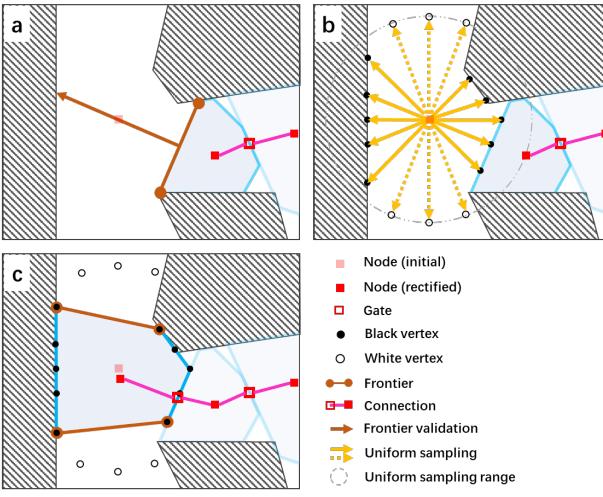


Fig. 2. An illustration of the skeleton generation algorithm workflow. It shows an example algorithm iteration including frontier validation and node expansion.

### III. PROBLEM STATEMENT

Given a bounded and known map  $\mathcal{M}$ , we aim to extract a 3D trajectory-independent sparse skeleton graph  $G$  reflecting the topological structure of the environment for mobile robot navigation. Hence, narrow and inaccessible regions are not expected to be included in the resulting graph. Serving as a high level understanding of the environment, the skeleton graph should be able to benefit the global planning for robots.

### IV. METHODOLOGY

#### A. Data Structures and Terminologies

To better explain the proposed approach, we would like to first introduce the data structures and some terminologies. A *node* represents a polyhedron covering a free space region and stores the polyhedron's facets and vertices. In 2D cases (Fig. 2), facets correspond to edges of the polygon. Nodes also hold *frontiers*, which are essentially special collections of adjacent facets. A frontier stands on the border between its polyhedron and the free space that is not yet reached by any nodes, guiding the skeleton growth. Important attributes stored in a frontier and vertex are listed in Table I and Table II respectively. Besides, a *gate* is built between two neighboring nodes and *connections* are established between them indicating traversability. In the resulting graph, both nodes and gates contribute to the vertices of the skeleton graph, whose edges are the connections.

#### B. Algorithm Overview

The algorithm workflow is described in Algorithm 1. At the beginning, the input map  $\mathcal{M}$  is set and an initial node is expanded at an arbitrary position within free space. During the initial node expansion, a few frontiers are identified and pushed into the pending frontiers First-In-First-Out(FIFO) list  $\mathcal{F}_{pndg}$ . Then, the algorithm iteratively grows the skeleton graph in a breadth-first manner by popping out frontier  $f$  from  $\mathcal{F}_{pndg}$  and, if possible, expanding a new node.

In each iteration,  $f$  is verified by performing raycasting from its center along its normal direction. Only when the

---

#### Algorithm 1 Skeleton Graph Generation

---

**Input:** Dense environment map  $\mathcal{M}$

**Output:** Sparse 3D skeleton graph  $G$

```

1:  $\mathcal{F}_{pndg}, \mathcal{N}, \mathcal{G}, \mathcal{C}, \mathcal{B} \leftarrow \emptyset$ 
2: initialize( $\mathcal{M}, \mathcal{F}_{pndg}, \mathcal{N}, \mathcal{B}$ )
3: while  $\mathcal{F}_{pndg}$  not empty do
4:    $f \leftarrow \mathcal{F}_{pndg}.\text{pop}()$ 
5:   verifyFrontier( $f$ )
6:   if  $f$  invalid then
7:     continue
8:   end if
9:    $n \leftarrow \text{new node}(f.\text{initial\_position})$ 
10:   $\mathcal{V}_{black}, \mathcal{V}_{white} \leftarrow \emptyset$ 
11:  generateVertices( $n, \mathcal{V}_{black}, \mathcal{V}_{white}, \mathcal{B}$ )
12:  if  $\mathcal{V}_{white} = \emptyset$  and  $n.\text{size} \leq \epsilon$  then
13:    continue
14:  end if
15:  cycleFormation( $n, \mathcal{V}_{black}$ )
16:   $P, \mathcal{F}_{new} \leftarrow \text{buildPolyAndFrontier}(n, \mathcal{V}_{black}, \mathcal{V}_{white})$ 
17:   $\mathcal{B}.\text{push}(\partial P)$ 
18:   $\mathcal{F}_{pndg}.\text{append}(\mathcal{F}_{new})$ 
19:  rectifyNodeCenter( $n, \mathcal{V}_{black}$ )
20:   $g \leftarrow \text{new gate}(f)$ 
21:   $\mathcal{C}.\text{push}(\text{buildConnection}(g, n))$ 
22:   $\mathcal{C}.\text{push}(\text{buildConnection}(g, f.\text{parent\_node}))$ 
23: end while
24: return  $G \leftarrow (\mathcal{N} \cup \mathcal{G}, \mathcal{C})$ 

```

---

ray doesn't detect occupied space or polyhedron within a threshold distance from the center of  $f$ , it will be marked valid. If  $f$  is valid, the mid-point of the ray segment (brown arrow in Fig. 2) is recorded in it. Then, a new node  $n$  is constructed taking this mid-point position as its initial position and a new gate  $g$  is created at the center of  $f$ . Next,  $n$  is expanded in line 9-19, which is the core of our algorithm (Section IV-C). During the node expansion, a polyhedron is grown while new frontiers are identified and pushed into  $\mathcal{F}_{pndg}$  (Section IV-D). Moreover, skeleton graph cycles are formed if necessary (Section IV-E). Upon expansion success, connections are built doubly-linked between  $g$  and  $n$ , as well as  $g$  and  $f$ 's parent node. The new nodes, gates and connections are collected in  $\mathcal{N}$ ,  $\mathcal{G}$  and  $\mathcal{C}$  respectively.

#### C. Node Expansion

Starting from this section, we present the core of our algorithm: node expansion (line 9-19 in Algorithm 1). To expand a node  $n$ , we first perform raycasts from  $n$ 's initial position along uniformly sampled directions up to a truncated distance, as in Fig. 2. For each single ray, if no environment obstacles or other polyhedra is detected within the truncated distance, a white vertex  $v_{white}$  is constructed at the truncated endpoint of the ray, indicating free space not yet reached by any node. Otherwise, a black vertex  $v_{black}$  is set at the first detected position on the ray segment. If a ray detects a polyhedron, it indicates that there exists a cycle in the skeleton graph to be closed and the detected polyhedron

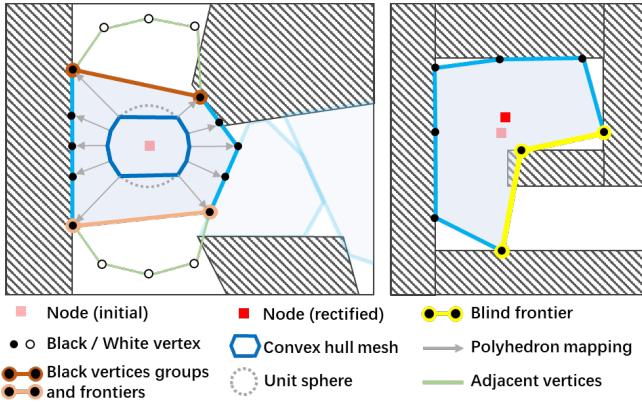


Fig. 3. The left image illustrates the polyhedron construction and the right image is an example of the blind frontiers.

is stored along with  $v_{black}$ . To avoid growing redundant branches in the skeleton graph, a node holding no white vertex with a size smaller than a threshold  $\epsilon$  is discarded, where the size of a node is computed as the mean distance from black vertices to the node center. After passing the size check, a *cycle formation* is performed (Section IV-E) by iterating over all black vertices to close potential cycles in the skeleton graph. Next, the polyhedron represented by  $n$  is constructed, based on which new frontiers are identified (Section IV-D) and pushed to  $\mathcal{F}_{pndg}$ . The boundary  $\partial P$  of the newly grown polyhedron  $P$  is recorded into  $\mathcal{B}$ , which will be used for raycasting for later nodes. Finally, the position of  $n$  is rectified as the average position of all  $v_{black}$ .

#### D. Polyhedron Construction and Frontier Identification

Given black and white vertices list  $\mathcal{V}_{black}$  and  $\mathcal{V}_{white}$ , we aim to construct the polyhedron  $P$  of the current node  $n$  and identify new frontiers  $\mathcal{F}_{new}$  on the polyhedron boundary  $\partial P$ . For all  $v_{black}$  in  $\mathcal{V}_{black}$ , we first compute the projected positions  $\hat{p}$  of  $v_{black}$  on the unit sphere centered at  $n$  as following:

$$\hat{p} = c + \frac{p - c}{\|p - c\|}$$

where  $p$  is  $v_{black}$  position and  $c$  is the initial position of  $n$ .

To generate the polyhedron for  $n$ , a convex hull mesh  $\hat{P}$  is computed on the projected positions  $\hat{p}$  of all  $v_{black}$ . Next, we map the facets of resulting mesh  $\hat{P}$  back to the black vertices  $\mathcal{V}_{black}$  to construct the polyhedron  $P$ , as in Fig. 3. Notice that the polyhedron constructed in this way is not necessarily convex and therefore is able to capture the free space more precisely.

To identify frontiers on  $\partial P$ , black vertices neighboring with adjacent white vertices are grouped as a set stored in  $\mathcal{S}_{group}$ . Other black vertices not neighboring with any white vertex will remain ungrouped. For a set  $s_i$  of grouped black vertices, the frontier  $f_i$  created by  $s_i$  consists of all the polyhedron facets whose vertices all belong to  $s_i$ . Then the new frontier  $f_i$  will be split if the angle difference of its component facets' normals exceeds a threshold. The split operation results in a few frontiers whose normals point towards different directions, guiding the skeleton graph to

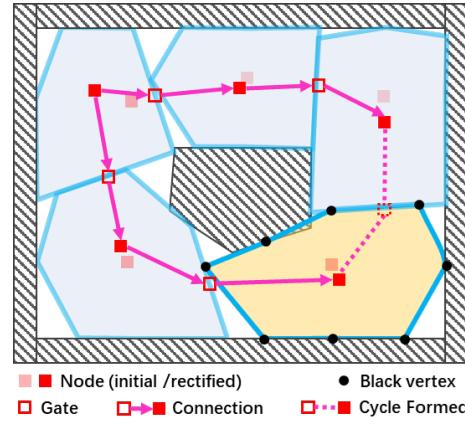


Fig. 4. An illustration of a cycle formation, which builds connections and gate between the newly grown polyhedron (yellow) and a previous polyhedron. The skeleton graph generation process is indicated in arrows along with the connection segments.

---

#### Algorithm 2 Polyhedron Construction/Frontier Identification

---

**Input:** Black vertices  $\mathcal{V}_{black}$ , white vertices  $\mathcal{V}_{white}$

**Output:** Polyhedron  $P$ , new frontiers  $\mathcal{F}_{new}$

```

1:  $\mathcal{S}_{group}, \mathcal{F}_{new} \leftarrow \emptyset$ 
2: for  $v_i \in \mathcal{V}_{black}$  do
3:    $\hat{p}_i \leftarrow \text{computeProjectedPosition}(v_i)$ 
4: end for
5:  $\hat{P} \leftarrow \text{convexHullMesh}(\{\hat{p}_i\})$ 
6:  $P \leftarrow \text{mapPolyhedron}(\hat{P})$ 
7:  $\mathcal{S}_{group} \leftarrow \text{groupBlackVertices}(\mathcal{V}_{black}, \mathcal{V}_{white})$ 
8: for  $s_i \in \mathcal{S}_{group}$  do
9:    $f_i \leftarrow \text{createFrontier}(s_i)$ 
10:   $\mathcal{F}_{new}.\text{append}(\text{splitFrontier}(f_i))$ 
11: end for
12:  $\mathcal{F}_{new}.\text{append}(\text{blindFrontiers}(\mathcal{V}_{black}))$ 
13:  $\text{sort}(\mathcal{F}_{new})$ 
14: return  $P, \mathcal{F}_{new}$ 

```

---

grow towards distinctive free space regions. Moreover, a special kind of frontier, called *blind frontier*, is identified to avoid missing possible passageway due to blind spot problems (Fig. 3). A blind frontier is formed by neighboring facets when their vertices have a large difference in distance to the node. For each frontier, we calculate its normal as the average outward unit normals of its facets. Also, the center of the frontier is computed as the projection of the average position of its facets centers along the normal to one of its facet. Finally, all the resulting frontiers  $\mathcal{F}_{new}$  will be sorted decreasingly by the number of facets they own, since beyond a larger frontier there is a higher chance of discovering uncovered free space.

#### E. Cycle Formation

A cycle is formed to ensure the compactness of the skeleton graph in the situation that the environment forms a loop, such as Fig. 4. To search cycles for the current node  $n_{cur}$ , we first find the polyhedron set  $\{P_i, i \in \mathcal{I}\}$  whose elements are the detected polyhedron stored in all

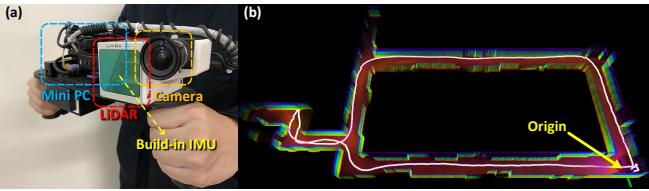


Fig. 5. (a) Our handheld device for data collection. (b) The map is reconstructed by R<sup>3</sup>LIVE, where the points are colored by their height and the white path is our traveling trajectory for sampling the data.

TABLE III  
QUALITY ASSESSMENT OF THE SKELETON GRAPH

Scene	Method	Generation Time (s)			#V	#E
		Avg	Std	Max		
Maze	[1]	69.35	0.278	69.78	68.79	7955 22015
	[2]	110.0	1.759	113.2	107.5	<b>123</b> <b>168</b>
	Ours	<b>2.126</b>	0.023	2.180	2.100	464 484
Machine Hall	[1]	1.029	1.081	0.977	0.417	531 1146
	[2]	2.570	0.021	2.613	2.542	<b>13</b> <b>13</b>
	Ours	<b>0.211</b>	0.003	0.219	0.208	69 77

black vertices in  $\mathcal{V}_{black}$ . Then, for each involved polyhedron  $P_i$  and their corresponding node  $n_i$  with  $i \in \mathcal{I}$ , we iterate through  $n_i$ 's frontiers and count the number of black vertices in  $\mathcal{V}_{black}$  that lies on it. The frontier  $f^*$  with the most black vertices count takes part in the cycle formation. A new gate  $g$  is established at the center of  $f^*$  and connections are set between  $g$  and  $n_{cur}$  as well as  $g$  and  $n_i$ . Then collision checks are performed on these two connections. If any of them fails the check, the cycle formation will be revoked.

## V. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our method in comparison with two representative state-of-the-art works. We also demonstrate our approach in real-world maps which contain significant noises.

### A. Implementation Details

Our implementation of the proposed algorithm uses point cloud as an input algorithm. Hence, collision checking is performed by the nearest neighbor search provided by the *Point Cloud Library (PCL)*<sup>1</sup>. The convex hull mesh computation used in Section IV-D is provided by *quickhull*<sup>2</sup>. All simulation experiments have been run in an Intel Core i7-4770K CPU at 3.5GHz, with 32 GB memory.

For real-world experiments, we collect the data with a handheld device shown in Fig. 5(a), which contains a mini PC *DJI Manifold 2C*<sup>3</sup>, a *FLIR* global shutter camera, and a *LiVOX AVIA*<sup>4</sup> LiDAR. And to build the precise, dense, 3D point cloud of the surrounding environment in real-time, we leverage a low-drift, LiDAR-Visual-Inertial tightly-coupled state estimator R<sup>3</sup>LIVE [22] for reconstructing the maps (Fig. 5(b)).

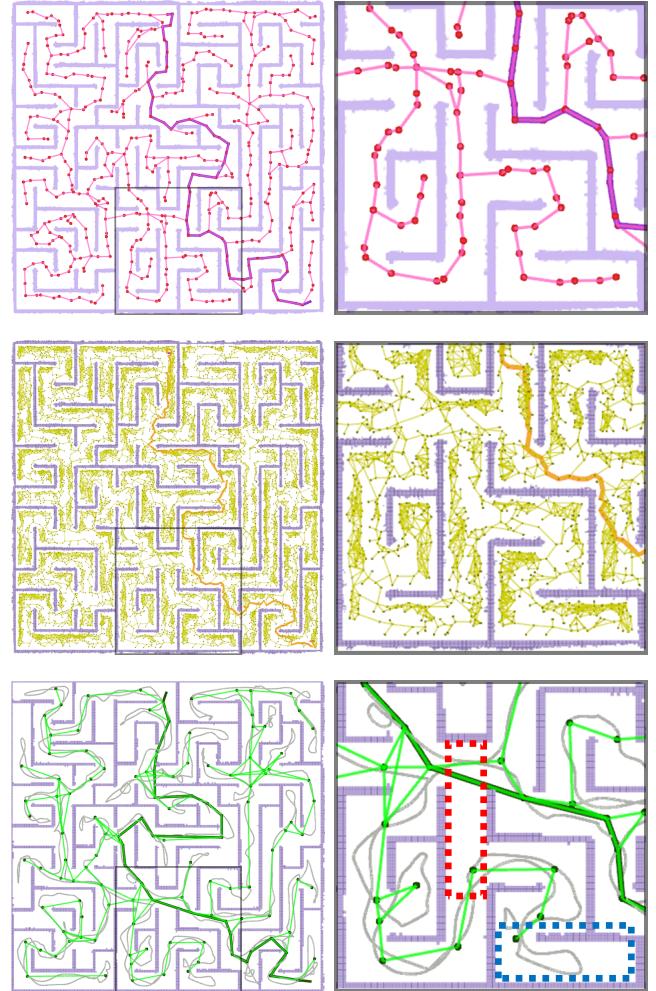


Fig. 6. Benchmark comparison results in the simulated maze scenario. From top to down, the works shown are proposed method, [1] and [2]. Pictures in left column are the full views, with details in the black rectangles are shown in the right column. The bold line segments highlighted are the global planning path utilizing the skeleton graphs. The gray curve in the result of [2] is the exploration trajectory used.

### B. Benchmark Analysis

In this section, we perform benchmark comparison with two state-of-the-art methods: a GVD-based method [1] and a clustering-based method [2]. Note that no open-source code is available for [2] so we use our implementation. For [1], we transit point cloud maps into ESDFs, which is the required input map representation. For [2], we perform an full coverage exploration utilizing [23] to obtain an exploration trajectory. Also, an occupancy map with a voxel size of 0.25m is built feeding as inputs together with the exploration trajectory. The three methods are tested in two scenarios, for each of which, all methods are run 10 times with statistics shown in Table III.

We first test the three methods in a  $60 \times 60 \times 2.5$  m<sup>3</sup> simulated large maze scenario shown in Fig. 6. The results indicate that our method generates a sparse graph that reflects the topological structure of free space more precisely. The skeleton graph generated by [1] is much denser, which has more than 17 times vertices and 45 times edges compared with the graph given by our method. Although the skeleton

<sup>1</sup><https://www.pointclouds.org/>

<sup>2</sup><https://github.com/akuukka/quickhull.git>

<sup>3</sup><https://www.dji.com/manifold-2>

<sup>4</sup><https://www.livoxtech.com/avia>

TABLE IV  
GLOBAL PLANNING PERFORMANCE

Scene	Method	A* Planning Time (ms)			Path(m)
		Avg	Std	Max	
Maze	Classical	N/A	N/A	N/A	N/A
	[1]	139.4	21.03	201.2	128.2
	[2]	<b>0.729</b>	0.066	0.818	0.596
Machine Hall	Ours	2.479	1.261	4.766	<b>117.3</b>
	Classical	5392	61.32	5460	5266
	[1]	4.789	1.188	8.227	4.089
Machine Hall	[2]	<b>0.055</b>	0.011	0.071	0.035
	Ours	0.434	0.133	0.712	0.265
					21.74

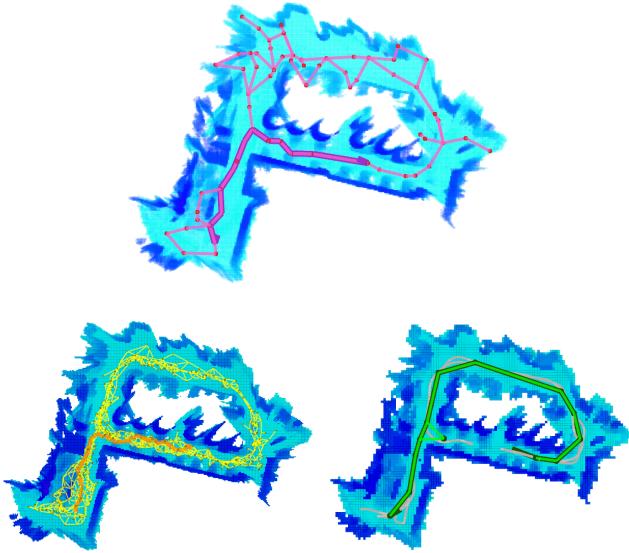


Fig. 7. Benchmark comparison results in the machine hall scenario. Our work is shown on top, with [1] and [2] on bottom left and right respectively. The bold line segments highlighted are the global planning path utilizing the skeleton graphs. The gray curve in the bottom-right picture is the exploration trajectory used in [2]. Note that the resulting skeleton graph from [2] fails to include some important edges.

graph produced by [2] is sparse, it fails to capture the true structure of the free space since it grows along the exploration trajectory. For example, in the bottom-right picture of Fig. 6, the skeleton graph in red dashed rectangle should have been connected and it rejects to grow into dead ends such as the one in the blue dashed rectangle. We also evaluate the three methods in a machine hall scenario, which is a  $25 \times 30 \times 2.5 \text{ m}^3$  open-source real-world dataset provided by [1]. As shown in Fig. 7, the experiments reveal that the three methods behave similarly as analyzed above and our method outperforms the others in skeleton graph quality. In both scenarios, our method achieves a much shorter skeleton graph generation time compared with [1] and [2]. Especially in large-scale environments such as the maze scenario, our approach is 30+ times faster than the other two methods.

Moreover, we evaluate of global planning performance utilizing the resulting skeleton graphs of the three methods by the A\* algorithm. To show the advantages of the skeleton graph, we also compare it with classical A\* algorithm, which performs a search on grids. The global planning paths are highlighted in bold line segments in Fig. 6 - 7 and statistics are shown in Table IV. Notice that the classical A\* algorithm

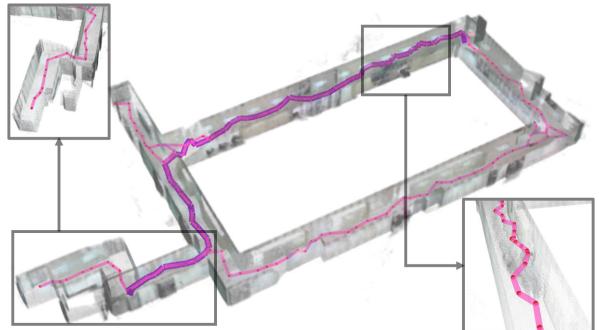


Fig. 8. This is an indoor corridor of size  $40 \times 30 \times 2 \text{ m}^3$ . Skeleton graph is shown in pink and the A\* global planning path is highlighted in purple. Pictures at the top-left and bottom-right corners show the details.

is not able to finish in 10 minutes in the large maze scenario. Global planning on the skeleton graphs generated by our method is at least 10 times faster compared with [1] and gives the shortest path among the three skeleton graphs. Although global planning on the skeleton graph given by [2] is fast, it fails to give the optimal path topology in tasks for both scenarios because of the incompleteness of the skeleton graph. Compared with classical global planning on grids, the A\* algorithm utilizing skeleton graphs rendered by our method is able to give competitive high-quality paths 20,000+ times faster in only a few milliseconds.

### C. Real-world Map Experiments

To further validate the feasibility of the proposed method, we conduct experiments on real-world data collected by us using the handheld device as mentioned in Sec. V-A. As in the enlarged picture at the bottom-right of Fig. 8, someone walked by and provided substantial noises to the map. The skeleton graph with 162 vertices and 167 edges is successfully generated in 0.654 seconds, demonstrating that our method is capable of handling noisy maps. An A\* global planning path with a length 58.69m is searched in 2.087 milliseconds. For the large-scale multi-floored hall scenario in Fig. 1, the skeleton graph with 438 vertices and 523 edges is generated in 2.856 seconds and the A\* global planning path with length 178.1m is searched in 3.788 milliseconds. The experiments further validate the capability of our algorithm in noisy, complex and large-scale maps.

## VI. CONCLUSIONS

In this paper, we proposed an efficient and flexible approach to generate trajectory-independent sparse topological skeleton graphs for mobile robot global planning. Our method adopts an efficient ray sampling and validating mechanism to iteratively extract polyhedra in free space. Frontiers are identified on the boundary of polyhedra, guiding the skeleton graph to grow towards distinctive free space regions. Compared with state-of-the-art works, our method generates sparse skeleton graph in a substantially shorter time, capturing the spatial structure of free space precisely and giving high-quality global planning paths rapidly. Experiments on real-world maps demonstrate our method is capable of handling noisy, complex and large-scale environments.

## REFERENCES

- [1] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Sparse 3d topological graphs for micro-aerial vehicle planning," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*. IEEE, 2018, pp. 1–9.
- [2] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.
- [3] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.
- [4] N. Kalra, D. Ferguson, and A. Stentz, "Incremental reconstruction of generalized voronoi diagrams on grids," *Robotics and Autonomous Systems*, vol. 57, no. 2, pp. 123–128, 2009.
- [5] M. Liu, F. Colas, L. Oth, and R. Siegwart, "Incremental topological segmentation for semi-structured environments using discretized gvg," *Autonomous Robots*, vol. 38, no. 2, pp. 143–160, 2015.
- [6] K. Hoff, T. Culver, J. Keyser, M. C. Lin, and D. Manocha, "Interactive motion planning using hardware-accelerated computation of generalized voronoi diagrams," *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, vol. 3, pp. 2931–2937, 2000.
- [7] M. Foskey, M. Garber, M. C. Lin, and D. Manocha, "A voronoi-based hybrid motion planner," *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, vol. 1, pp. 55–60, 2001.
- [8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [9] A. Tagliasacchi, I. Alhashim, M. Olson, and H. Zhang, "Mean curvature skeletons," in *Computer Graphics Forum*, vol. 31, no. 5. Wiley Online Library, 2012, pp. 1735–1744.
- [10] A. Tagliasacchi, H. Zhang, and D. Cohen-Or, "Curve skeleton extraction from incomplete point cloud," in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–9.
- [11] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea, "3d skeletons: A state-of-the-art report," in *Computer Graphics Forum*, vol. 35, no. 2. Wiley Online Library, 2016, pp. 573–597.
- [12] Z. Zivkovic, B. Bakker, and B. Kroese, "Hierarchical map building using visual landmarks and geometric constraints," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 2480–2485.
- [13] J.-L. Blanco, J. Gonzalez, and J.-A. Fernandez-Madrigal, "Consistent observation grouping for generating metric-topological maps that improves robot localization," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE, 2006, pp. 818–823.
- [14] F. Fraundorfer, C. Engels, and D. Nistér, "Topological mapping, localization and navigation using image collections," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee, 2007, pp. 3872–3877.
- [15] R. Vazquez-Martín, P. Nunez, A. Bandera, and F. Sandoval, "Spectral clustering for feature-based metric maps partitioning in a hybrid mapping framework," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 4175–4181.
- [16] K. Konolige, E. Marder-Eppstein, and B. Marthi, "Navigation in hybrid metric-topological maps," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3041–3047.
- [17] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, vol. 107, pp. 109–124.
- [18] S. Liu, M. Waterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters (RA-L)*, pp. 1688–1695, 2017.
- [19] X. Zhong, Y. Wu, D. Wang, Q. Wang, C. Xu, and F. Gao, "Generating large convex polytopes directly on point clouds," *arXiv preprint arXiv:2010.08744*, 2020.
- [20] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1476–1483.
- [21] F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2016, pp. 139–146.
- [22] J. Lin and F. Zhang, "R3live: A robust, real-time, rgb-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package," *arXiv preprint arXiv:2109.12400*, 2021.
- [23] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 779–786, 2021.