Hierarchical Space Partition for Surface Reconstruction

Minjie Tang
Independent Researcher
tangminjie.ai@gmail.com

Xiangfei Li*
Huazhong University of Science and Technology
lixiangfei@hust.edu.cn *

Abstract

Generating compact polygonal models from point clouds is a key problem in 3D vision and computer graphics. However, due to inherent limitations of LiDAR scanning (e.g. range constraints and occlusions), critical scene information is often missing, leading to degraded reconstruction accuracy. To address this, we propose a plane assembling strategy that effectively recovers missing details while maintaining model compactness. We classify all the planes extracted from the scene into three categories: highly visible, barely visible, and invisible. The invisible planes, which are recovered by scene structure analysis, indicate the missing details. The three types of planes correspond to the three growth priorities. Each plane grows according to the priority level, and the space is partitioned progressively, that is, the hierarchical partition. Subsequently, we generate a watertight polygonal mesh from the partition via a min-cut-based optimization. Finally, comparisons on public datasets show the effectiveness and superiority of our method against mainstream approaches.

1. Introduction

Surface reconstruction is a crucial topic in the fields of 3D vision and computer graphics. This problem can be formulated as follows: extracting semantic, structural, and topological information from dense, unordered, and redundant point cloud data (typically acquired via LiDAR scanning) or triangular meshes (often generated by Multi-View Stereo, MVS) to produce high-precision, watertight, and lightweight polygonal surface models [2]. Such models have significant applications in the domains of autonomous driving, robotics, augmented reality/virtual reality, digital twin, and building information modeling (BIM).

Plane assembly is a popular paradigm in contemporary surface reconstruction benchmarks. This approach partitions 3D space using detected planar shapes from input data and subsequently selects specific polygonal facets or polyhedral cells from the partitioned structure to construct simplified mesh models. The method demonstrates multiple advantages: 1) it is broadly applicable due to the prevalence of planar structures in man-made environments; 2) it exhibits some robustness to missing scan data; 3) it provides favorable geometric properties such as surface watertightness, orientability, and convexity of polyhedral elements. However, most existing methods adopt a homogeneous treatment for all detected planes during spatial partitioning, assigning uniform expansion factors [12, 25] or isotropic growth speed [2]. This method inevitably leads to considerable redundancy in spatial partitioning [35]. Besides, extending existing planar entities shows limited effectiveness in completing missing regions in scanning scenes [2].

To address these limitations, we propose a shape-assembling method that yields lighter yet meaningful 3D partitions and can accurately restore the local details of the incomplete scene. Our first contribution is to design a hierarchical space partition strategy based on a kinetic data structure [2, 16]. Unlike the previously described scheme, we divide planes into three different categories based on visibility (highly visible, barely visible, and invisible), corresponding to three levels. Planes at the current level will partition further based on the outcome only after planes at the immediately higher level have completed their spatial partition (i.e. hierarchical partition). At the same time, each plane is given a different growth speed according to its visibility. This hierarchical strategy produces lightweight and semantically meaningful 3D partitions, with substantially reduced algorithmic complexity compared to other methods. Our secondary contribution is to recover missing planes for models. In contrast to the saturated additions of *ghost planes* proposed in [5], we first use planar boundary segments and planes' intersection lines to detect singular segments, that is segments not contained by any intersection lines. The missing planes are then fitted through these segments. Experiments show that this strategy accurately recovers the missing details of the scanned model without introducing redundant structures. Finally, we evaluate our approach against state-of-the-art methods across datasets varying in complexity, size, and data collection.

*Corresponding author.

2. Related Work

Polygonal mesh reconstruction is a fundamental problem in computer vision and computer graphics. Prior studies can be grouped into three aspects: i) space partition and plane assembly, ii) scene decomposition and topology analysis, and iii) mesh simplification and geometry approximation.

Space Partition and Plane assembly. Such methods first utilize extracted geometric primitives (e.g. planes) to partition the 3D space. This space is typically the bounding box of the scene data, and it is divided into subregions such as polyhedral cells [3, 21, 24] or polygonal facets [11, 12, 25]. The primary limitation of these partitioning approaches is their $O(n^3)$ complexity, resulting in high computational/memory costs. The resulting massive polyhedral cells also create a complex optimization space for subsequent surface reconstruction. To address this issue, [5] proposed a two-level hierarchical scheme that first partitions space into super-cells and further subdivides only those intersected by planar primitives. This approach also introduced ghost primitives (i.e. vertical and orthogonal planes) to mitigate data incompleteness issues. More recently, [2] adopted a kinetic data structure, [7] generated a cell complex via adaptive space partitioning, and [35] introduced a compact plane arrangement to further avoid excessive subdivision. Following the space partitioning stage, the surface extraction is achieved either by classifying polyhedral cells as inside or outside the object [23, 27, 38], or by solving a constrained integer programming formulation [11, 12, 25] to select polygonal facets.

Scene decompose and Topology analysis. The second category of methods first leverages semantic or structural information to decompose the reconstruction problem into sub-problems focused on reconstructing individual elements. These reconstructed elements are then merged using their topological connections to produce the final model. Early approaches [6, 33, 37] for planar-faced architectures decomposed scenes into planar regions and reconstructed models by extracting vertices, edges, and facets from the regions' connectivity graph. Although the method is efficient, graph inaccuracies often lead to incomplete or erroneous results. Further advancing this approach, methods like those in [17, 19, 26, 40] decomposed the scene into semantic categories (e.g., ground, ceiling, columns, walls) and employed distinct reconstruction strategies for each category. This semantic decomposition is typically data-driven, potentially leading to semantic inaccuracies when applied to novel scenes. To circumvent this issue, [12] relied on predefined distance thresholds to classify scene clusters into structural and non-structural objects, which were then reconstructed separately. More recently, [29] introduced a novel inside/outside view analysis to identify 3D structures, then generated models with varying levels of detail.

Mesh simplification and Geometry approximation. Instead of directly generating compact polygonal models, extensive work in computer graphics has focused on simplifying dense meshes by reducing their facet counts [8, 22] or imposing geometric assumptions for the output surface [13]. [14] proposed iterative vertex-pair contraction using quadric error metrics; [9] performed face clustering guided by geometric proxies; and [32] developed edge-collapse decimation with planar proxy preservation. While these mesh reduction techniques can yield comparably compact models, they often result in polygonal structures with less regularity and structural coherence than direct generation methods.

3. Method

Taking a point cloud as input, our algorithm outputs a watertight and intersection-free polygonal mesh. Figure 1 illustrates the workflow of the proposed method. We first analyze the scene structure to extract geometric and topological information, which mainly includes planar primitives, intersection lines, the adjacency graph and boundary segments. Then we distinguish the highly/barely visible planes by the visibility. Meanwhile, in order to deal with the possible loss of scanning data, missing planes are recovered by fitting boundary segments of primitives. This type of planes is classified as invisible planes. Next, space partition is accomplished by assigning different levels and expansion speed to planes. Finally, a polygonal surface mesh is reconstructed by classifying polyhedral cells as interior or exterior via min-cut optimization.

3.1. Scene Structure Analysis

Planar primitives are initially extracted from the input point cloud using Random Sample Consensus (RANSAC)-based shape detection methods [28, 31, 34]. Subsequently, an α-shape algorithm [10] is employed to approximate the boundary of each detected primitive, as shown in Figure 1(b1). Then we project boundary points onto its plane, and calculate the 2D normal vector of each point using principal component analysis. Thereafter, 2D segments are fitted using region growing [28]. To mitigate missing data and noise, we adopt the regularization process of [17] to obtain cleaner and more accurate segments. The 2D segments and the corresponding 2D normal vectors are then back-projected into 3D space to obtain the final set of 3D segments, as shown in Figure 1(b3).

Using all primitives, an adjacency graph $\mathcal{G}$ is constructed by identifying the neighboring relationships between them, as illustrated in Figure 1(b2). Two planes are defined as adjacent if at least two inlier points, one from each primitive, are mutual neighbors in the k-nearest neighbor graph of the input points [11]. Moreover, for any pair of non-parallel primitives i and j, we compute their intersection line L_{ij}. The set of all such lines is denoted as $\mathcal{L}$, as in Figure 1(b4).

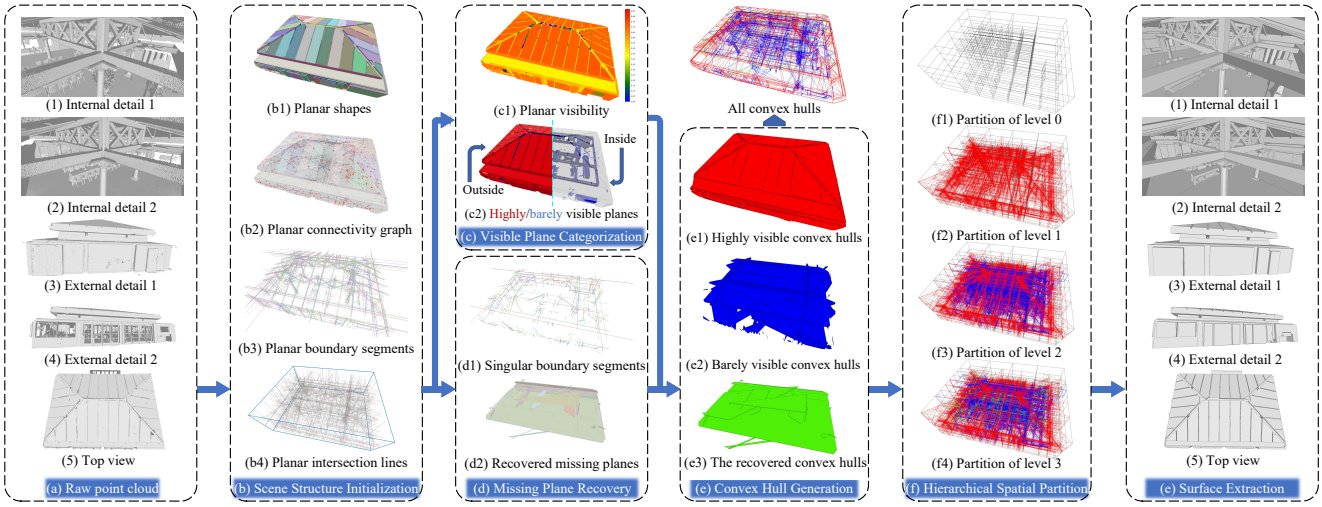
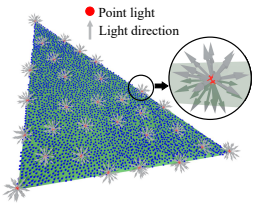


Figure 1. Overview. Our algorithm takes point cloud (a) as the input and first performs the scene structure analysis (b), which mainly includes: planes detection (b1), planar adjacency analysis (b2), boundary segments fitting (b3) and planar intersection lines calculation (b4). Subsequently visible planes categorization (c) is performed: including visible calculation (c1) and highly/barely visible labelling (c2). Meanwhile, we recover missing planes (d) through fitting the singular segments (i.e., the segments that are not contained by any intersection lines (d1)). After that, the 3D bounding box is hierarchically partitioned (f) using the different categories of primitives (i.e., convex hulls of highly visible planes (e1), barely visible planes (e2) and invisible (missing) planes (e3)). Finally, based on the inside/outside labeling of polyhedral cells, a concise, watertight polygonal mesh is generated (f).

3.2. Visible Plane Categorization

In this section, we model α-shapes as uniformly luminous objects to compute individual visibility ratios of primitives. Then, Markov random field (MRF) optimization method is used to allocate highly/barely visible labels for planes.

Planar visibility. For each planar primitive P_i, we use farthest point sampling [30] on its supporting points to generate keypoints, as point light sources. Then uniform sphere sampling [15] is used to generate lights (see inset). Thereafter, we compute the intersection of lights with all α-shapes using the ray-shooting method [1]. If not intersect, this light is visible, otherwise is invisible. The visibility ratio of planes is obtained by dividing the number of visible light by the total number of samples. If the visibility ratio is larger than 0.5, the plane is defined as original highly visible, otherwise, barely visible. The detailed pseudo-code is provided in the supplementary material.



The visibility ratio of all planes is shown in Figure 1(c1). It is basically consistent with spatial locations of planes, as is said, the closer a plane is to the external space, the greater the visibility ratio it got. However, some exceptions remain. For instance, the visibility ratio of the concave structure on the outside is less than 0.2 due to the obstruction of the surface patches in narrow spaces, as exemplified by the ceiling in Figure 2(a). On the other hand, due to the holes caused by the missing scans, the visibility ratio of some inside planes is higher than 0.5, as shown by the table in Figure 2(a). In order to improve this situation and make the label allocation of primitives more reasonable, we adopt an optimized scheme to assign visible labels to planes.

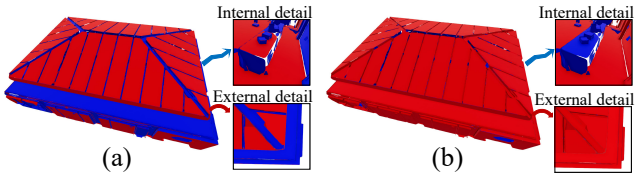


Figure 2. (a) Initial label assignment. (b) Optimized labels after energy minimization using MRF.

Planar Labeling. We have the following observation: the neighbors of highly visible planes are also likely to be highly visible, and vice versa. Based on this neighborhood correlation, the MRF energy formulation is established. Specifically, given the set of planes $\mathcal{P}$, the corresponding adjacency graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we aim to find a binary label x_i, where $x_i = 1$ indicates highly visible and $x_i = 0$ indicates barely visible, to each primitive p_i, whose visibility ratio is v_i, using a MRF formulation:

$$E(x) = \sum_{p_i \in \mathcal{P}} E_{\text{data}}(p_i, x_i) + w \sum_{(i,j) \in \mathcal{E}} E_{\text{smooth}}(x_i, x_j)$$

The data term tends to assign $x_i = 1$ to planes with high visibility ratios:

$$E_{\text{data}}(p_i, x_i) = v_i \cdot (1 - x_i) + (0.5 - v_i) \cdot x_i,$$

while the smoothness term aims to ensure the consistency of neighboring labels:

$$E_{\text{smooth}}(x_i, x_j) = |x_i - x_j|.$$

where w is the balance parameters. In our experiments, we typically set w to 0.5. $E(x)$ is minimized with graph cuts and alpha expansion [4].

The final results are demonstrated in Figure 2(b), where the visibility labels (highly/barely) of planes align consistently with both the visibility ratios and their spatial positions.

3.3. Missing Plane Recovery

A common characteristic of man-made objects is that each significant boundary line is typically formed by the intersection of two or more planes. Therefore, if a boundary segment is detected to be contained only by one single plane, then we consider this segment to be *singular*. Using this as a clue, we first identify all the singular segments and then recover the missing planes based on them.

Singular segments selection. For planar primitive p_i, let $S_i = \{s_{ij}\}$ be the set of boundary segments, and $L_i = \{l_{ik}\}$ be the set of intersection lines between p_i and its second-order adjacent planar primitives. Two planes p_i and p_k are called second-order adjacent if there exists a third plane p_t that is adjacent to both p_i and p_k. Segment $s_{ij} \in S_i$ is considered not singular if there exists $l_{ik} \in L_i$ such that: (i) The angle between the directions of l_{ik} and s_{ij} is less than r_a, (ii) The distance from centroid of s_{ij} to l_{ik} is less than r_d. In our approach, r_a is set to 10 degrees, and r_d is set to five times the average distance of the model. Furthermore, if s_{ij} is not singular, it should be contained by at least two planes, which in here are p_i and p_k.

Missing planes generation. A region-growing-liked method is used to fitting all planes from singular segments, as shown by the translucent planes in Figure 1(d). Specifically, we sort segments based on their lengths in descending order and select seed segments accordingly, starting from the longest. Each seed segment determines an initial plane p_{ini} based on its centroid and normal vector. Then a neighborhood search is performed around the seed segment, considering all segments within the second-order adjacent planes of the given segment's plane. A neighboring segment is incorporated into the region of the seed segment if two conditions are satisfied: (i) The angle between its normal vector and that of the initial plane p_{ini} is less than a threshold r_a, (ii) The average distance of its supporting points to the initial plane p_{ini} is below r_d. Then, the plane p_{ini} is updated based on its region. This process is repeated until neighborhood searches for all segments have been completed. Planes containing at least two segments are selected as the final set of recovered planes. A detailed comparison between results with and without recovered planes is shown in Figure 3. Comprehensive pseudo-code is in the supplementary material.

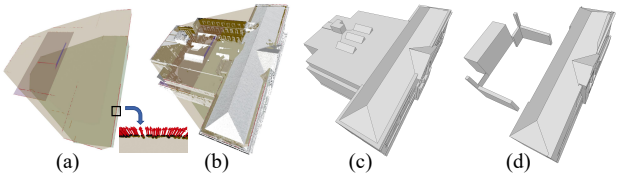


Figure 3. (a) the recovered missing planes, black points are support points of boundary segments, and the red arrows are their normal vectors reprojected from 2D into 3D. (b) recovered planes and point cloud model. (c) the output model with recovered planes. (d) the output model without recovered planes.

3.4. Convex Hulls Generation

Visible planar convex hulls generation. As said before, if a segment is not singular, it should be contained by the intersection line of two planes. Therefore, the supporting points of this segment will be incorporated into the calculation of the convex hulls in corresponding planes. In particular, for the primitive p_i, we combine its original supporting points and the newly detected segments' supporting points to calculate the final expanded convex hull, as an effective compensation for missing data.

Recovered planar convex hulls generation. The computation of missing planar convex hulls is relatively straightforward, as it simply computes the convex hulls of corresponding boundary segments' supporting points.

3.5. Hierarchical Space Partition

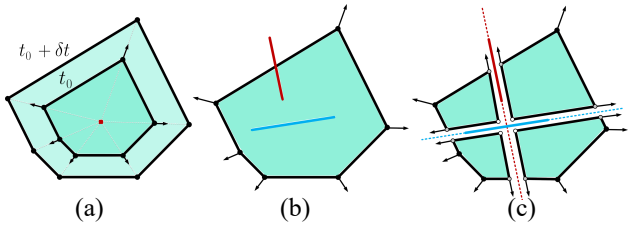


Figure 4. (a) The polygon grows by uniform scaling. (b) The intersection of the blue polygon with two others is illustrated by red and green line-segments. (c) The shape is decomposed into non-intersecting polygons by cutting along the intersection lines (shown as red and green dashed lines). Original vertices (black) retain their positions, while new vertices are either fixed at junctions (white, frozen vertices) or allowed to slide along intersection lines (gray, sliding vertices). The same as KSR [2].

Background on kinetic framework The 3D kinetic framework originally developed by [2], which enables convex polygons to expand at consistent velocities (conformal transformation, see Figure 4(a)), until collisions occur between each other, thereby partitioning the 3D space into polyhedral cells. This computational paradigm was first rigorously defined in the field of Computational Geometry

by [16]. Within this framework, convex polygons have coordinates that evolve as continuous functions of time. The core mechanism involves maintaining spatial relationships among these primitives as they expand dynamically. When two or more primitives come into collisions (referred to as an *event* in kinetic data structures), corrective geometric operations are executed to restore system validity, which is decomposing primitives into intersection-free polygons with cuts along the intersection lines, as show in Figure 4(b-c). Algorithmically, the framework relies on a *priority queue* to efficiently manage the temporal sequence of events, processing collisions iteratively until all primitives reach stationary states [2].

Hierarchical strategy. Based on kinetic framework, a hierarchical space partition strategy is introduced. This strategy shares the same objective as other partitioning strategies: minimizing unnecessary splits [5, 29, 35]. However, unlike approaches that prioritize planes based on size or other attributes, we employ visibility ratios to categorize primitives into distinct levels (i.e., level 1: highly visible planes; level 2: barely visible planes; level 3: invisible planes (missing planes)). Then, the lower the level, the first to grow, and the space is partitioned progressively. This insight is based on our observation that primitives exhibiting higher visibility ratios—typically located near the periphery—should be prioritized and expanded faster during the growth process, thereby forming boundaries for inner planes with lower visibility ratio.

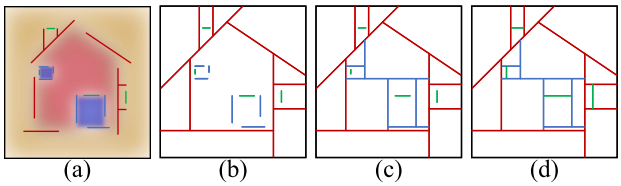


Figure 5. Different colors represent different visible hierarchical regions (a), red line-segments (i.e., highly visible planes) grows first (b), then blue grows (barely visible planes) (c), and finally green grows (invisible planes) (d).

Specifically, as shown in Figure 5, let the bounding box be the level 0, and the corresponding space be space-0. The highly visible planes first perform space partition based on space-0 to generate the space-1. Barely visible planes then grow to generate space-2 based on space-1. And finally the missing planes expand to subpart the space-2 to generate space-3. Namely, the (i+1)-th space is generated by planes of (i+1) level to subpart the (i)-th space (i = 0, 1, 2). The facets generated in the previous level are equivalent to the bounding box of current level. And the space-3 is what we sought. Also as shown in Figure 1(f). See supplementary material for detailed pseudo-code. Moreover, unlike [2, 25], which handle all planes uniformly, we introduce a unique expand speed for each primitive and a more natural condition for convex hulls to stop growing.

Expanding speed. The growth of each plane gives a speed positively correlated with the visibility ratio; that is, the faster the primitive with a higher visible rate will grow in the same level. Specifically, for each level, the visibility ratios of its planes are normalized to a unified range of [0.5, 1], where the minimum value corresponds to 0.5 and the maximum value corresponds to 1, and this defines their growth speed.

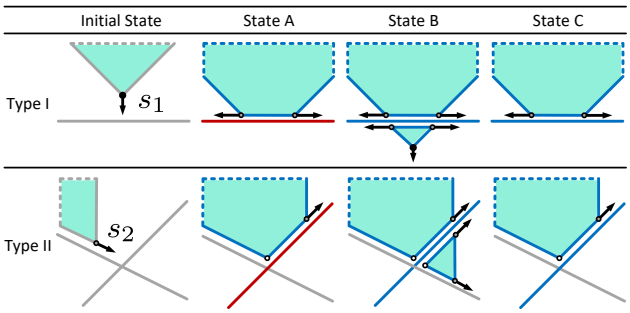


Figure 6. Different polygon collision types and growth states. s_1 and s_2 represent different speeds. The red lines indicate highly visible polygons, the blue lines indicate barely visible polygons, and the gray lines denote polygons that may be any type. The point colors follow the same definition as in Figure 4.

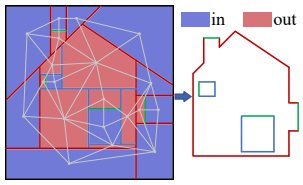
State Transition. We treat the vertices of each polygon as a state machine to update the growth process. The initial state is when all the polygons are divided into non-intersecting sub-polygons before they begin to grow, as shown in Figure 4(c). Different state transitions are triggered based on vertices and different types of plane encounters (*events*): if a vertex meets the plane of the previous level (*events A*) or the adjacent plane of the current level (*events C*), the direction of propagation of it will be modified to follow the intersection line with the intersecting plane. Specifically, for states A and C in type I in Figure 6, we split the vertex into two sliding vertices diverging along the intersection line. For type II states A and C, we reorient the sliding vertex propagation to align with the target polygon's intersection. Meanwhile, a frozen vertex is established at the convergence point. In addition to the above events, collisions also occur between non-neighboring primitives at the same level (*events B*). In this case, it is necessary to introduce a newly generated polygon into the kinetic data structure. This polygon extends the originating primitive across the shared boundary defined by the intersecting polygon (see state B in Figure 6). The initialization of this polygon differs depending on the configuration: in state B of type I, it consists of two sliding vertices and one original vertex; whereas in state B of type II, it is composed of two sliding vertices along with one fixed vertex. Finally, we present the terminal state: a sliding vertex reaches the intersection line, at a moment when another sliding vertex—guided by the contacted polygon—is already in place. At the inter-

section point of the corresponding lines, a frozen vertex is introduced, effectively halting the local propagation of both sliding vertices.

Accordingly, we use the boundary and neighborhood relationship to constrain growth. Thanks to the expansion of visible primitives and the addition of missing planes, which ensures the effective completion of the missing data, so only a few growth events are needed to achieve an effective spatial division, ensuring detail recovery, completing missing structures, and reducing unnecessary partitions.

3.6. Surface Extraction

We extract the surface by performing a min-cut on the polyhedral partition, assigning inside-outside labels to the cells. The resulting surface corresponds to the interface facets between labeled regions (see inset). Owing to the validity of the kinetic embedding, the output is guaranteed to be watertight and free of self-intersections, similar to [2, 5, 23, 40].



Given a partitioned polyhedral set $\mathcal{C} = \{c_i\}$, we construct its dual graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as follows: (i) For each polyhedron $c_i \in \mathcal{C}$, create a vertex $v_i \in \mathcal{V}$ positioned at c_i's centroid. (ii) For each pair (c_i, c_j) sharing a facet f_{ij}, add an edge $e_{ij} \in \mathcal{E}$ connecting v_i and v_j, as shown in inset.

To assign binary labels $x_i \in \{0, 1\}$ to each node v_i, where $x_i = 0$ denotes that c_i lies outside and $x_i = 1$ denotes that it lies inside, we define the following energy function U:

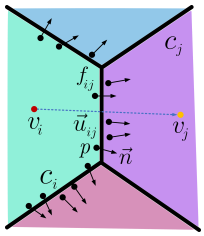
$$U(\mathbf{x}) = P(\mathbf{x}) + \lambda M(\mathbf{x}) \tag{2}$$

where $\lambda \in [0, 1]$ is a balance parameter, which is set to 0.5 in our methods. The optimal output surface that minimizes U is found by a max-flow algorithm [4].

Following the convention used in smooth surface reconstruction [20], normals are assumed to point outward. Accordingly, the point supporting term $P(\mathbf{x})$ evaluates the consistency between the assigned labels and the orientations of inlier normals:

$$P(\mathbf{x}) = \frac{\pi \cdot r^2}{\mathcal{A}} \sum_{v_i \in \mathcal{V}} \sum_{e_{ij} \in \mathcal{E}} \sum_{p_k \in I_{ij}} \mathbf{1}_{\{(2x_i - 1) \cdot \vec{n}_k \cdot \vec{u}_{ij} < 0\}} \tag{3}$$

where $\mathcal{A}$ is a normalization factor defined as the sum of the areas of all facets of the partition, I_{ij} is the set of inlier points associated with common facet f_{ij} between c_i and c_j. $\mathbf{1}_{\{\circ\}}$ is the characteristic function, $\vec{n}_k$ is the normal vector of inlier point p_k, and $\vec{u}_{ij}$ is the vector from v_i to v_j, as shown in the inset. r is the average spacing of input point cloud model. In this example, we prefer assigning label *in* to polyhedron c_i (i.e., x_i=1) and label *out* to polyhedron c_j (i.e., x_j=0). We use the centroid line $\vec{u}_{ij}$ of the two polyhedra c_i and c_j to determine the points on their common surface f_{ij}. This local neighborhood context makes our voting function more robust to inaccuracies in normal orientations, since normals only need to point toward the correct half-space separating the facet.



The regularization term $M(\mathbf{x})$ penalizes surface complexity through area minimization, where reduced area correlates with simplified geometry. Furthermore, facets with abundant supporting points exhibit higher retention likelihood during optimization:

$$M(\mathbf{x}) = \frac{1}{\mathcal{A}} \sum_{e_{ij} \in \mathcal{E}} (a_{ij} - \pi \cdot r^2 \cdot |I_{ij}|) \cdot \mathbf{1}_{\{x_i \neq x_j\}} \tag{4}$$

where a_{ij} represents the area of facet f_{ij}, and $|I_{ij}|$ is the number of inlier points associated with facets f_{ij}.

4. Results

Our algorithm is implemented in C++ using the CGAL [36] library. We evaluate the effectiveness of our method for mesh reconstruction under a standard experimental setup with an Intel i5-10300H CPU and an NVIDIA GTX 1650 GPU. The evaluation includes benchmark datasets, quantitative metrics, and both objective and subjective comparisons with contemporary reconstruction methods. Due to space constraints, parameter sensitivity analysis, ablation studies, and generalization experiments are provided in the supplementary material.

Dataset and Metrics. For benchmarking purposes, test point clouds from the Assembly Dataset [39], the KSR-42 Dataset [2], and the ScanNet++ V2 Dataset [41] are employed, as they exhibit varying degrees of reconstructed geometric fidelity. Inspired by [18] and guided by the data scale in [13], we select 200 CAD models (*CAD-200*) and 100 real scanned architectural models (*Arch-100*) from these datasets—that exhibit diverse geometric characteristics, to evaluate the reconstruction performance. We assess our method in terms of correctness, conciseness, and computational performance. Correctness is quantified using Mean Hausdorff Error (*MHE*) and Root Mean Squared Error (*RMSE*), reflecting how well the reconstructed polygonal planes align with the input points. Conciseness is evaluated based on the number of vertices ($\mathrm{P}^{\mathrm{Avg.}}$) and facets ($\mathrm{F}^{\mathrm{Avg.}}$) in the reconstructed mesh, as well as a composite metric ($\mathrm{RH}^{\mathrm{Avg.}}$) that combines the Hausdorff distance and a simplification ratio. The simplification ratio is calculated as the number of vertices in the reconstructed mesh divided by the number of points in the input point cloud, following the definition in [18]. Computational efficiency is assessed via runtime and peak memory usage during reconstruction.

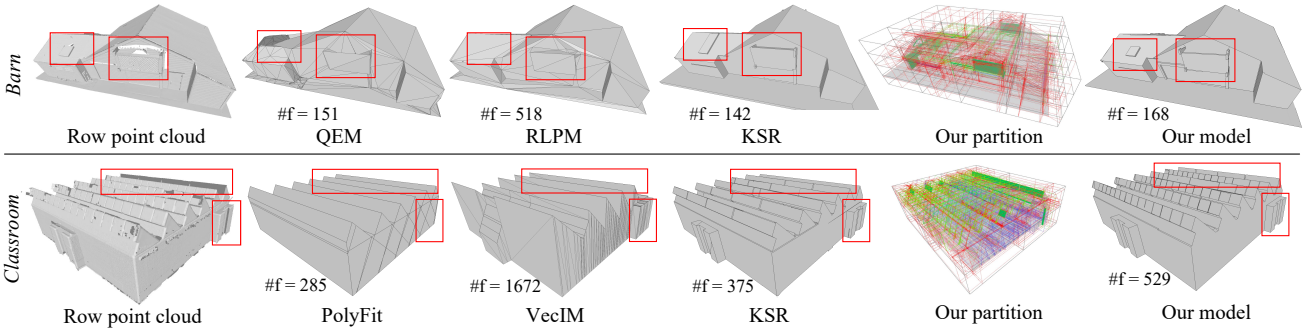


Figure 7. Reconstruction results from different methods on the urban scene *Barn* and the indoor scene *Classroom*. #f refers to the number of output facets. Our method achieves the most accurate reconstructions, with local details highlighted in red boxes. In the subfigures labeled "Our partition", green planes indicate the recovered planar regions.

Comparative analysis. Based on the above dataset and evaluation criteria, we compare our method against several representative approaches, including QEM [14], PolyFit [25], KSR [2], Robust Low-Poly Meshing (RLPM) [8], and VecIM [17], which reflect current mainstream strategies. Since QEM and RLPM are incapable of directly processing raw point clouds, we first apply the Poisson surface reconstruction technique [20] to generate triangular meshes for these methods. VecIM primarily focuses on indoor scene reconstruction and requires accurate semantic information for guidance, so we only test this method on indoor scenes. Excessive primitives make PolyFit's mixed-integer optimization costly; thus, keeping their number below 100 ensures feasible vectorization, though some local details may be lost.

		Correctness		*Conciseness*			*Time*
		MHE ↓	RMSE ↓	$P^{Avg.}$ ↓	$F^{Avg.}$ ↓	$RH^{Avg.}$ ↓	T(s) ↓
Arch-100	QEM [14]	2.702	0.082	602	811	0.0137	**76**
	Polyfit [25]	2.165	0.059	516	723	0.0061	1253
	KSR [2]	1.926	0.037	**260**	**372**	0.0057	265
	VecIM [17]	2.962	0.096	712	1215	0.0129	152
	RLPM [8]	2.298	0.078	565	761	0.0102	351
	Ours	**1.066**	**0.025**	315	501	**0.0036**	327
CAD-200	PolyFit [25]	0.159	0.015	371	559	0.0023	867
	KSR [2]	0.127	0.016	**292**	**314**	0.0015	**182**
	RLPM [8]	0.236	0.037	421	926	0.0025	291
	Ours	**0.109**	**0.009**	265	331	**0.0012**	205

Table 1. Comparison of correctness, conciseness, and runtime.

Correctness. As representative examples, the reconstruction results on the *Barn* and *Classroom* (C.Room) scenes are shown in Figure 7. As shown in the figures, our algorithm's missing planes recovery technique enables the faithful reconstruction of fine details, such as the window sill and skylights in the *Barn* scene. Similarly, the ceiling and the lower wall in the *Classroom* scene are also accurately recovered. The green planes in Figure 7 indicate the recovered planes, corresponding to regions that are missing or unscanned during the LiDAR acquisition. These regions significantly affect the overall reconstruction accuracy. While planar expansion helps compensate for some of these deficiencies, it remains an imperfect solution [2, 25]. A quantitative comparison of reconstruction accuracy is presented in Figure 8 and Table 1. Figure 8 shows that our algorithm successfully reconstructs the internal beam structure of the *Meetingroom* (M.Room), which is missing in the point cloud due to occlusion, leading to a significantly reduced reconstruction error. Table 1 confirms that our method attains superior reconstruction accuracy compared to other algorithms.

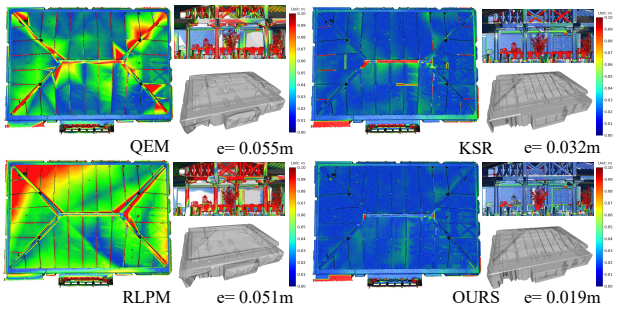


Figure 8. Output models and distance maps from various methods on the Meetingroom. e: RMSE from input points to output model.

Conciseness. In terms of conciseness, as shown in Table 1, our method reconstructs the model with slightly more vertices and facets due to an increased number of planes (i.e., the recovered planes). However, it achieves the highest overall compactness and the best trade-off between facets count and reconstruction accuracy. Further, in order to prove the superiority of our hierarchical spatial partition, we made a more detailed comparison on the *CAD-200*. Since the *CAD-200* point cloud is sampled from complete CAD meshes, no planes are missing, so all plane assembly schemes yield the same number of primitives. In Figures 9 and 10, it can be seen that our algorithm achieves the most compact spatial division, owing to a hierarchical strategy and the scheme of correlating the growth speed of convex

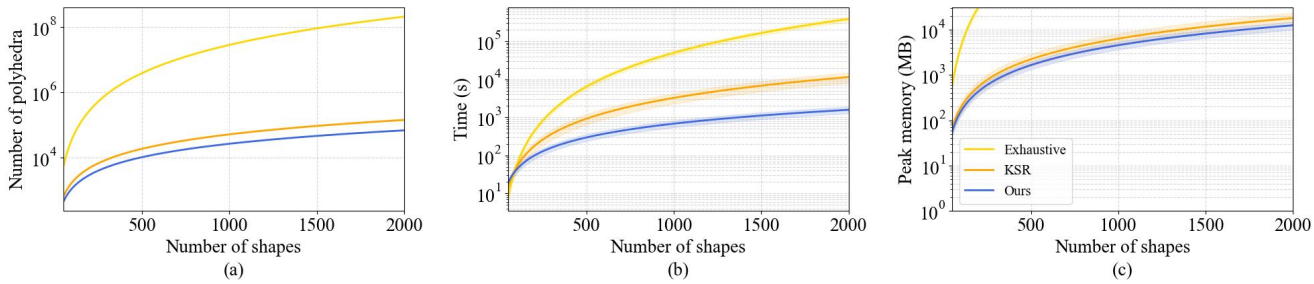


Figure 9. Performances of partitioning and surface extraction. The transparent band around each curve indicates the minimal and maximal values measured on various models.

hulls with the visibility rate that avoids meaningless partitions. Moreover, by expanding plane polygons along contained line-segments (Section 3) and stopping growth based on neighborhood cues, our method enables each plane to partition the required polyhedral cells with minimal extension, achieving both compactness and accuracy. As shown in Figure 9(a), our method produces the fewest polyhedral cells and demonstrates the slowest growth in cell count across different planar motifs.

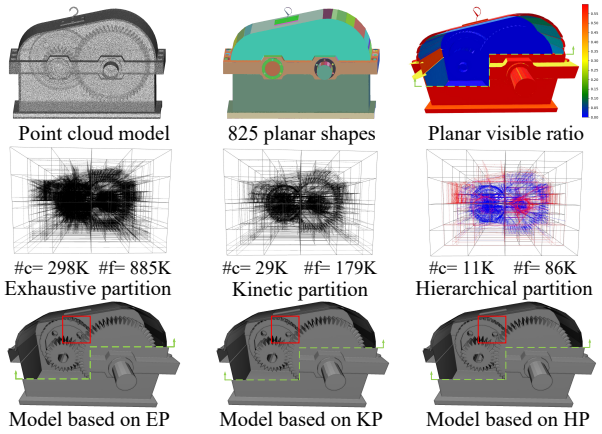


Figure 10. Exhaustive, kinetic, and hierarchical partitions on the $Gearbox$. #f and #c denote the numbers of facets and polyhedra, respectively. Our hierarchical partitions (HP) are more compact and yield more accurate reconstructions than exhaustive (EP) and kinetic partitions (KP), as highlighted in red. The model is sliced along the green dashed line for internal visualization.

Efficiency. Our method is slightly slower than KSR but significantly faster than PolyFit, as shown in Table 1. For the plane classification module, thanks to our use of a bounding volume hierarchy [1] and a fixed number of sampled keypoints per plane, the overall algorithm complexity is slightly above $O(n)$ but significantly lower than $O(n^2)$, where n is the number of planes. The complexity of the missing plane recovery module is similar to that of the plane classification module, since only a few number of segments ultimately pass the screening. The most time-consuming part of our algorithm is the hierarchical spatial partitioning module, as shown in the runtime breakdown in Table 2. Moreover, due to our neighborhood constraint strategy and planar hierarchical growth strategy, our memory peak and partition time are the lowest, as shown in Figure 9.

	M.Room	Barn	C.Room	Gearbox
# input points	3.07M	2.01M	3.98M	1.85M
# highly visible planes	872	157	175	209
# barely visible planes	815	38	80	616
# recovered planes	35	19	38	no
# output facets	1929	168	529	2010
Categorization (sec)	91	12	21	76
Recovery (sec)	79	10	35	no
Partitioning (sec)	219	67	123	192
Extraction (sec)	71	25	57	75
Memory peak (MB)	675	152	335	596

Table 2. Performance statistics of different scenes.

5. Conclusion

We propose a novel algorithm that reconstructs compact and watertight polygonal meshes from raw point clouds. The main contribution of this work is the introduction of hierarchical space partitioning strategy, where the space are progressively partitioned by prioritizing the growth of planes according to their visibility ratio into a low number of polyhedra. This idea not only improves correctness over existing methods by recovering missing scene details, but also enables the generation of more concise polygonal meshes from a more compact and semantically meaningful partition of polyhedra. We conducted extensive comparative experiments on diverse datasets against state-of-the-art methods, demonstrating the accuracy and effectiveness of our approach. In future work, we intend to incorporate additional available information, such as multi-view stereo image data, to capture richer spatial structures. We also plan to investigate how the proposed hierarchical space partition strategy can be applied to the repair of CAD models.

Acknowledgments. This work was supported by the National Natural Science Foundation of China under Grant Nos. 52575574, U24A20130 and 52188102.

References

[1] Pierre Alliez, Stéphane Tayeb, and Camille Wormser. 2D and 3D fast intersection and distance computation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. 3, 8

[2] Jean-Philippe Bauchet and Florent Lafarge. Kinetic shape reconstruction. *ACM Trans. Graph.*, 39(5), 2020. 1, 2, 4, 5, 6, 7

[3] Alexandre Boulch, Martin de La Gorce, and Renaud Marlet. Piecewise-planar 3d reconstruction with edge and corner regularization. In *Computer Graphics Forum*, pages 55–64. Wiley Online Library, 2014. 2

[4] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004. 4, 6

[5] Anne-Laure Chauve, Patrick Labatut, and Jean-Philippe Pons. Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 1261–1268. IEEE, 2010. 1, 2, 5, 6

[6] Jie Chen and Baoquan Chen. Architectural modeling from sparsely scanned range data. *International Journal of Computer Vision*, 78:223–236, 2008. 2

[7] Zhaiyu Chen, Hugo Ledoux, Seyran Khademi, and Liangliang Nan. Reconstructing compact building models from point clouds using deep implicit fields. *ISPRS Journal of Photogrammetry and Remote Sensing*, 194:58–73, 2022. 2

[8] Zhen Chen, Zherong Pan, Kui Wu, Etienne Vouga, and Xifeng Gao. Robust low-poly meshing for general 3d models. *ACM Transactions on Graphics (TOG)*, 42(4):1–20, 2023. 2, 7

[9] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *ACM SIGGRAPH 2004 Papers*, pages 905–914. 2004. 2

[10] Tran Kai Frank Da. 2D alpha shapes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. 2

[11] Hao Fang and Florent Lafarge. Connect-and-slice: an hybrid approach for reconstructing 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13490–13498, 2020. 2

[12] Hao Fang, Cihui Pan, and Hui Huang. Structure-aware indoor scene reconstruction via two levels of abstraction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 178: 155–170, 2021. 1, 2, 4

[13] Xifeng Gao, Kui Wu, and Zherong Pan. Low-poly mesh generation for building models. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022. 2, 6

[14] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997. 2, 7

[15] Álvaro González. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *Mathematical geosciences*, 42:49–64, 2010. 3

[16] Leonidas Guibas. Kinetic data structures. In *Handbook of Data Structures and Applications*, pages 377–388. Chapman and Hall/CRC, 2018. 1, 5

[17] Jiali Han, Mengqi Rong, Hanqing Jiang, Hongmin Liu, and Shuhan Shen. Vectorized indoor surface reconstruction from 3d point cloud with multistep 2d optimization. *ISPRS Journal of Photogrammetry and Remote Sensing*, 177:57–74, 2021. 2, 7, 4

[18] Xin He, Chenlei Lv, Pengdi Huang, and Hui Huang. Windpoly: Polygonal mesh reconstruction via winding numbers. In *European Conference on Computer Vision*, pages 294–311. Springer, 2024. 6

[19] Jingwei Huang, Shanshan Zhang, Bo Duan, Yanfeng Zhang, Xiaoyang Guo, Mingwei Sun, and Li Yi. Arrangementnet: learning scene arrangements for vectorized indoor scene modeling. *ACM Transactions on Graphics (TOG)*, 42(4):1–15, 2023. 2

[20] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013. 6, 7

[21] Pierre-Alain Langlois, Alexandre Boulch, and Renaud Marlet. Surface reconstruction from 3d line segments. In *2019 International Conference on 3D Vision (3DV)*, pages 553–563. IEEE, 2019. 2

[22] Minglei Li and Liangliang Nan. Feature-preserving 3d mesh simplification for urban buildings. *ISPRS Journal of Photogrammetry and Remote Sensing*, 173:135–150, 2021. 2

[23] Minglei Li, Peter Wonka, and Liangliang Nan. Manhattan-world urban reconstruction from point clouds. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 54–69. Springer, 2016. 2, 6

[24] Claudio Mura, Oliver Mattausch, and Renato Pajarola. Piecewise-planar reconstruction of multi-room interiors with arbitrary wall arrangements. In *Computer graphics forum*, pages 179–188. Wiley Online Library, 2016. 2

[25] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. pages 2372–2380, 2017. 1, 2, 5, 7

[26] Sebastian Ochmann, Richard Vock, Raoul Wessel, and Reinhard Klein. Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics*, 54:94–103, 2016. 2, 4

[27] Sven Oesau, Florent Lafarge, and Pierre Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. *ISPRS journal of photogrammetry and remote sensing*, 90:68–82, 2014. 2

[28] Sven Oesau, Yannick Verdie, Clément Jamin, Pierre Alliez, Florent Lafarge, Simon Giraudot, Thien Hoang, and Dmitry Anisimov. Shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. 2

[29] Shanshan Pan, Runze Zhang, Yilin Liu, Minglun Gong, and Hui Huang. Building lod representation for 3d urban scenes. *ISPRS Journal of Photogrammetry and Remote Sensing*, 226:16–32, 2025. 2, 5

[30] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets

in a metric space. In *Advances in neural information processing systems*, 2017. 3

[31] Tahir Rabbani, Frank Van Den Heuvel, and George Vosselmann. Segmentation of point clouds using smoothness constraint. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(5):248–253, 2006. 2

[32] David Salinas, Florent Lafarge, and Pierre Alliez. Structure-aware mesh decimation. In *Computer Graphics Forum*, pages 211–227. Wiley Online Library, 2015. 2

[33] Falko Schindler, Wolfgang Wörstner, and Jan-Michael Frahm. Classification and reconstruction of surfaces from point clouds of man-made objects. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 257–263. IEEE, 2011. 2

[34] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, pages 214–226. Wiley Online Library, 2007. 2

[35] Raphael Sulzer and Florent Lafarge. Concise plane arrangements for low-poly surface and volume modelling. In *European Conference on Computer Vision*, pages 357–373. Springer, 2024. 1, 2, 5

[36] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. 6

[37] Marc Van Kreveld, Thijs Van Lankveld, and Remco C Veltkamp. On the shape of a set of points and lines in the plane. In *Computer Graphics Forum*, pages 1553–1562. Wiley Online Library, 2011. 2

[38] Yannick Verdie, Florent Lafarge, and Pierre Alliez. Lod generation for urban scenes. *ACM Transactions on Graphics*, 34 (3):30, 2015. 2

[39] Karl DD Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G Lambourne, Armando Solar-Lezama, et al. Joinable: Learning bottom-up assembly of parametric cad joints. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15849–15860, 2022. 6

[40] Xiaojun Xiang, Hanqing Jiang, Yihao Yu, Donghui Shen, Jianan Zhen, Hujun Bao, Xiaowei Zhou, and Guofeng Zhang. Efficient high-quality vectorized modeling of large-scale scenes. *International Journal of Computer Vision*, 132 (10):4564–4588, 2024. 2, 6, 4

[41] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12–22, 2023. 6