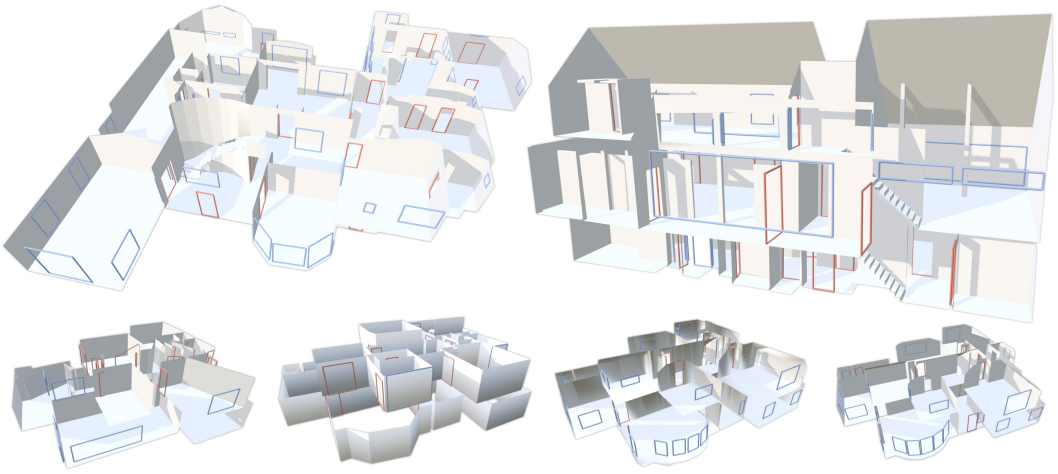
HouseLayout3D: A Benchmark and Training-Free Baseline for 3D Layout Estimation in the Wild

Valentin Bieri[1] **Marie-Julie Rakotosaona**[2] **Keisuke Tateno**[2]
Francis Engelmann[3] **Leonidas Guibas**[3]
[1]ETH Zurich [2]Google [3]Stanford University



RoomFormer [1] *SceneScript* [2] *MultiFloor3D* (Ours) *Ground truth*

Figure 1: *Top:* We present HOUSELAYOUT3D, a benchmark for 3D house layout estimation with greater scene diversity than prior datasets, including multi-floor buildings and detailed annotations for elements such as doors, windows, and staircases. *Bottom:* We further introduce MultiFloor3D, a training-free approach for 3D layout estimation that achieves state-of-the-art performance on both our benchmark and existing datasets.

Abstract

Most current 3D layout estimation models rely on synthetic datasets that feature simple single floor scenes. As a result, they struggle to generalize to complex multi floor buildings and are often forced into per floor processing, which loses global spatial context. We introduce HOUSELAYOUT3D, a real world benchmark that supports progress toward methods capable of handling large scale multi floor and architecturally intricate environments. We further propose MultiFloor3D, a baseline that builds on recent recent Foundation models for 3D scene understanding. Our method substantially outperforms state-of-the-art systems on both our benchmark and prior datasets, and it does so without any layout specific training. The HOUSELAYOUT3D dataset and evaluation tools are publicly available at https://houselayout3d.github.io .

1 Introduction

Estimating the layout of 3D environments is a key part of many vision and robotics systems [1, 2, 3, 4]. The objective is to capture a concise vectorized layout by abstracting a 3D scene into a set of polygons that represent structural elements such as walls, floors, ceilings, doors, windows, and staircases, while filtering out furniture and other occluders that commonly appear in real indoor scenes.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

Recent state-of-the-art models for layout prediction [1, 2, 5] are feed-forward deep-learning models trained on large-scale synthetic datasets [6, 2] and demonstrate impressive results even on real-world scenes. A key aspect of these models is that they are trained on synthetic data, which primarily consists of single rooms or small apartments. This is largely because such smaller scenes are easier to synthesize—they can be automatically generated at scale [3] or designed by professionals [6]. As a result, models trained on this data face significant limitations, struggling to generalize to large-scale buildings with substantially more rooms than a typical apartment and being entirely incapable of handling multi-level or multi-floor buildings. While it is possible to first divide large-scale buildings into individual floors and rooms and then process them separately, this approach discards valuable global context that can aid in local reasoning. For instance, detecting structural elements like staircases requires cross-floor reasoning, which is lost when floors are processed in isolation. Additionally, this method necessitates recombining individual room predictions to support building-level tasks such as path planning between rooms on different floors.

To advance research in 3D layout prediction for large-scale, multi-floor buildings, we introduce HOUSELAYOUT3D, a challenging benchmark dataset. Built upon real-world building scans from the Matterport3D [7] dataset, it captures expansive, architecturally complex spaces with up to five floors and forty rooms per floor, encompassing diverse room types, including partially open spaces that pose challenges for existing room-based approaches. We manually annotate all structural elements, including walls, floors, ceilings, staircases, as well as windows and doors, specifying the direction in which each door opens.

Inspired by the success of recent reconstruction and segmentation models, we propose a training-free approach called MultiFloor3D. Our goal is to demonstrate that by leveraging recent advances in 3D scene reconstruction, Gaussian Splatting models, and an innovative layout fitting technique, we can develop a simple yet effective method that outperforms existing approaches on the more challenging task of 3D layout estimation in multi-floor buildings. Our experiments on HOUSELAYOUT3D clearly highlight the limitations of current state-of-the-art methods in handling complex multi-floor buildings. In contrast, our approach generates more accurate and reasonable layouts, particularly for challenging multi-floor structures. We hope that these findings together with the benchmark dataset will inspire new research directions in multi-floor, large-scale 3D layout estimation.
In summary, our contributions are:

- We introduce HOUSELAYOUT3D, the first benchmark dataset for 3D layout estimation in large-scale, multi-floor buildings.
- We propose MultiFloor3D, a training-free baseline method that leverages recent reconstruction and segmentation techniques, achieving improved performance over current deep-learning models.
- Our extensive experiments clearly reveal the limitations of existing layout estimation methods, which we hope will drive further research in this direction.

2 Related Work

Manhattan Scene Layout Initial works on layout estimation impose Manhattan world assumptions on the output to then solve a constrained optimization problem based on detected walls (Scan2Bim [8]) or corners (DuLaNet [9], LayoutNet [10], FloorNet [11]). Notably, Ochmann et al. [12] allow angled walls by subdividing the 3D space into cells, ultimately determining the indoor space with an integer linear program.

2D Scene Layout Another line of work solves the problem from Birds-eye View (BEV): [13] uses shortest-path algorithms around the free space. Floor-SP [14] extends the concept with a room segmentation network. HovSG [15] combines 2D BEV point density maps with 2D object detection to build a scene graph of floors, rooms, and objects without predicting their geometry. This line of work is limited by its 2D predictions.

3D Scene Layout. Recent advances were made by end-to-end deep learning methods: SceneCAD [3] uses a graph neural network to infer a 3D layout and object bounding boxes. RoomFormer [1] trains a transformer to estimate a 2D floorplan enriched with semantics. SceneScript [2] proposes a *structured scene language* to predict 3D layout walls, windows, doors, and object bounding boxes from sparse point clouds. Importantly, available training data for end-to-end trainable methods is dominated by individual room scenes [3] or simple individual floors [6][2][16]. Moreover, the buildings are often unfurnished [16], synthetic [6][2], or limited to Manhattan layouts [17]. Another line of datasets annotates extracts of larger scenes in single 2D images or videos [4][18]. We find

Dataset	Real-world	Multi-room	Multi-floor	Full Scenes	Windows, Doors	Objects	Depth	3D Layouts
SceneCAD [3]	✓	(✓)	✗	✓	✓	✓	✓	✓
ASE [2]	✗	✓	✗	✓	✓	✓	✓	✓
Stru3D [6]	✗	✓	(✓)	✓	✓	✓	✓	✓
Zillow Indoor [16]	✓	✓	(✓)	✓	✗	✗	✗	✗
MP3D-Layout [18]	✓	✗	✗	✗	✓	✓	✓	✓
Zou et al [17]	✓	✗	✗	✗	✗	✗	✓	✓
CADEstate [4]	✓	✓	✗	✗	(✓)	✗	✗	✓
FloorNet [11]	✓	✓	✓	✓	(✓)	✗	✗	✗
HOUSELAYOUT3D (Ours)	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: **Dataset Comparisons** of existing dataset benchmarks for evaluating 3D layouts estimation.

that the limited availability of training data prevents end-to-end methods from generalizing beyond simple layouts.

3 The HOUSELAYOUT3D Dataset

We introduce a new dataset of hand-annotated CAD layouts derived from the Matterport3D [7] (MP3D) dataset (see Fig.2). Unlike previous works[3, 2], this is the first real-world benchmark dataset to provide CAD annotations for large-scale, multi-floor houses, encompassing numerous rooms, staircases, windows, and doors. Each structural element is annotated as a polygon in 3D space. Since our dataset is annotated on 3D meshes from MP3D [7], it inherits their per-vertex room ids and object instances.

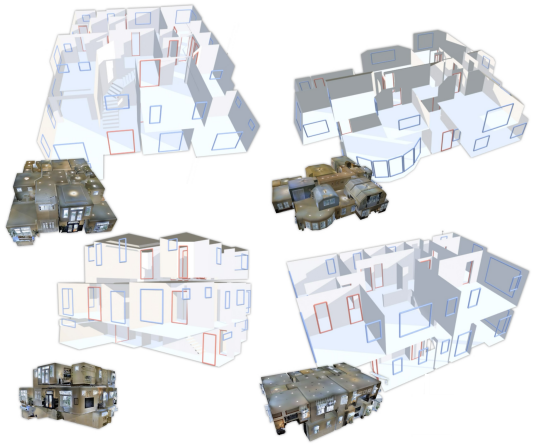


Figure 2: **Examples of our HOUSELAYOUT3D.** Our dataset includes multi-floor houses with annotations for walls, floors, ceilings and stairs, as well as windows *(blue)* and doors *(red)*. We also show the corresponding 3D meshes from MP3D [7].

Dataset Statistics. The dataset includes 16 buildings, 33 distinct levels, and 317 rooms, captured across more than 26,000 RGB-D frames. Its scale is comparable to the validation split of ScanNet [19]. In total, we annotated 292 doors, 379 windows, and 34 staircases. The lower number of doors compared to rooms is due to many spaces, such as hallways and dining areas, being connected by open passages or staircases rather than actual doors. Each building comprises between 1 and 5 levels and contains between 4 and 40 rooms. The annotation time varies depending on the building's size and the number of rooms, typically ranging from 4 to 10 hours per building. All annotations undergo visual verification by separate expert annotators. Table 1 compares properties across different datasets.

Annotation Tool and Labeling Details. To annotate the 3D scans, we use a free academic license of Scasa's PinPoint [20], a specialized software for building modeling from point clouds. It enables precise 3D geometry extraction even in occluded or incomplete areas through intuitive tools that automatically snap to edges and corners, streamlining the annotation process. In the 3D scans, doors are typically open, so we annotate both the current open position and the expected closed position, along with the opening direction. For doors that appear closed in the scans, we infer the opening direction by from the door hinge locations in the RGB images. For window annotations, we utilize the existing window object annotations from MP3D [7], projecting them onto the nearest annotated wall plane and fitting axis-aligned rectangles.

4 Method

Given N input RGB images of a scene, our goal is to produce a simple 3D layout consisting of polygons. Each polygon is assigned a label from a finite set of classes: walls, floors, ceilings, stairs, doors, and windows. The layout is organized into a scene graph with rooms as nodes and doors/stairs as edges, and each layout polygon is assigned to a room or an edge of the scene graph.

Figure 3 provides an overview of our approach, which consists of four stages. First, we compute a 3D mesh of the scene. In the second step, we extract the scene's main structural elements (floors, walls, ceilings) to form a *skeleton* of the layout. In the third step, we use geometric and semantic information to fit a layout *prototype* to the *skeleton*. Lastly, we parse the *prototype* into a scene graph, from which we extract the final layout.

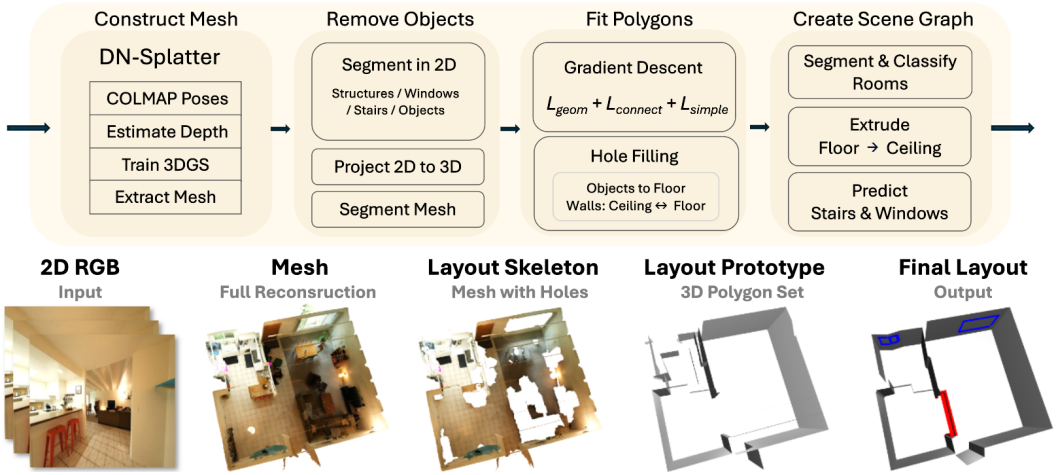


Figure 3: **Illustration of the MultiFloor3D model for 3D layout estimation.**

4.1 Generating a Mesh from RGB Images

Given a set of unposed 2D images, we follow DN-Splatter [21] to obtain a triangle mesh and 3D depth maps for every frame. DN-Splatter uses COLMAP [22] camera poses and a 2D depth model to train 3D Gaussian Splatting [23] (3DGS) reconstruction. DN-splatter then produces a Poisson Reconstruction [24] by sampling from the 3DGS rendered depth. In this work, we use the depth model Metric3d [25].

4.2 Extracting a Layout Skeleton from the Mesh

Once a mesh is generated, our next step is to use a pre-trained 2D segmentation model to extract a minimal, reliable geometry that serves as a basis ('*skeleton*') for the layout. This skeleton should consist exclusively of geometry that we want to include in the final layout. To distinguish such geometry, we define four semantic classes that we treat differently:

- **Structural Components** (*i.e.*walls, ceilings, and floors, but also large furniture such as closets): These are the main components of the layout skeleton. The structural components have accurate geometry that we wish to see represented in the final layout.
- **Geometrically inaccurate surfaces** (windows, mirrors): The 3D representation of windows and mirrors is often inaccurate due to noisy depth estimates. We do not wish to keep them in the layout skeleton.
- **Objects** Smaller furniture and objects such as tables or lamps are removed from the layout skeleton. Objects are later on used to complete unobserved areas of the layout.
- **Stairs** Are processed separately from the layout skeleton due to their complexity.

To construct the skeleton, we segment the 3D mesh into these classes. We run the segmentation model OneFormer [26] on the input images and map each output class [27] to one of the four semantic classes. To transfer OneFormer's segmentation to the mesh, we back-project $M = 5000$ randomly sampled pixels per image and their respective class to 3D. We collect class votes for each mesh vertex by assigning each back-projected point to the nearest mesh vertex. We further postprocess the obtained segmentation by clustering the mesh vertices into *superpoints*, following [28]'s preprocessing step. Each mesh vertex is then assigned to the most common class within its cluster. The result is a mesh segmented into our semantic classes. We create the *layout skeleton* by selecting only *structural components*, and extract *object* and *stair* meshes.

4.3 Fitting a Layout Prototype to the Skeleton

We observe significant artifacts in the layout skeletons, including holes and unobserved regions. For example, areas hidden behind furniture, or areas corresponding to windows are missing. In this stage, we use geometric and semantic information to correct the artifacts and infer a more complete *layout prototype*. To this end, we run an optimization that aims to improve the completeness of the obtained skeleton: We first initialize a collection of planar 3D polygons $\mathcal{P}$ from the layout skeleton. In particular, each segmented superpoint (see Sec. 4.2) of the skeleton is fitted to one or more planes. We then optimize the *vertex positions* and *plane equations* of the polygons using three main objectives:

- Reconstruct an accurate scene geometry with L_{geom}
- Produce a continuous and connected geometry with L_{connect}
- Produce a mesh with low vertex count with L_{simple}

During the optimization, we constrain the vertices of each polygon to be coplanar. We also allow and encourage polygons to share vertices. The initialization and the implementation of vertex constraints and shared vertices is detailed in the supplementary material.

Definitions. Given a polygon P consisting of edges E and a point $p \in \mathbb{R}$ we define the point-to-polygon distance $D_{pp}(P, p)$ as the minimal distance between p and any point on the surface of P. For $e \in E$ we define the point-to-edge distance $D_{pe}(p, e)$ as the minimal distance between p and any point on the line segment representing e.

Losses. We fit the polygon set $\mathcal{P}$ using gradient descent and three losses. The first loss L_{geom} encourages the polygons to reconstruct the original geometry and respect the *observed empty space*:

$$L_{\text{geom}} = L_{\text{prox}} + L_{\text{empty}} \tag{1}$$

L_{prox} penalizes the distance of each vertex $v \in V_{\text{skeleton}}$ of the Layout Skeleton to the closest polygon surface:

$$L_{\text{prox}} = \sum_{v \in V_{\text{skeleton}}} \min_{P \in \mathcal{P}} D_{pp}\big(v, P\big) \tag{2}$$

To prevent occluding the observed empty space (*i.e.*the space we believe to be empty based on the depth maps), we sample a set L of line segments using the input camera poses and computed depth maps. Each line segment extends from the camera pose to the back-projected depth. We then penalize line segment-polygon intersections as follows: If a line segment l intersects a polygon, the nearest polygon edge e^* should be moved closer to the intersection point p_{inter}.

$$L_{\text{empty}} = \sum_{l \in L} \sum_{\substack{P \in \mathcal{P} \\ l \cap P \neq \varnothing \\ D_{pe}(p_{inter}, e^*) \leq T_{\text{inter}}}} D_{pe}\big(p_{inter},\, e^*\big) \tag{3}$$

$$\text{where } p_{inter} = l \cap P \quad \text{and} \quad e^* = \operatorname*{argmin}_{e' \in \text{edges}(P)} D_{pe}\big(v, e'\big).$$

Note that we ignore intersections with $D_{pe}(p_{inter},\, e^*)$ greater than the threshold T_{inter} to avoid noise from intersections far from the polygon boundary.

The second loss L_{connect} prevents small gaps and encourages shared boundaries by making polygons attract vertices. Concretely, L_{connect} penalizes the distance from each polygon vertex to the closest surface of another polygon:

$$L_{\text{connect}} = \sum_{P \in \mathcal{P}} \sum_{v \in \text{vertices}(P)} \min_{P' \in \mathcal{P},\ P' \neq P} D_{pp}\big(v, P'\big) \tag{4}$$

As for L_{empty}, we ignore points with $D_{pp}\big(v, P'\big)$ greater than a threshold.

The third loss encourages simplicity and smooth polygon boundaries. L_{simple} penalizes the length of all edges that are not shared by at least two polygons. (*i.e.*not all edge vertices are shared). Intuitively, L_{simple} promotes shared edges (for instance, an edge between two walls) to represent the scene while edges that are not shared are shrunk until they are eliminated.

$$L_{\text{simple}} = \sum_{P \in \mathcal{P}} \sum_{e \in \text{edges}(P)} \mathbf{1}_{[\nexists\, P' \in \mathcal{P} \setminus \{P\}:\, e \subset P']} \|e\|_2 \tag{5}$$

Our final loss is given by $L = L_{\text{geom}} + L_{\text{connect}} + L_{\text{simple}}$.

Vertex Merging L_{simple} itself does not reduce the number of vertices or polygons in the polygon set. Instead, we periodically manually simplify P by (1) merging vertex pairs with distance below T_{merge}, (2) applying the RDP [29] algorithm with tolerance T_{merge} to the polygons individually, and (3) merging close polygons with similar normal. (Close in terms of minimal D_{pp} distance among the vertices.) For (3) we additionally verify that the merged polygon does not increase L_{prox} too strongly. Note that step (1) is the source of shared vertices between polygons.

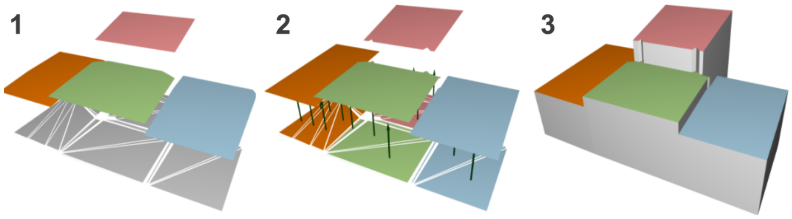


Figure 4: **Our proposed floor extrusion algorithm.** 1) Floor triangulation. 2) Triangles assigned to ceilings using midpoints. 3) Triangles extruded to ceiling planes.

Closing Holes in the Floor We observe that there is a floor under most objects in a room. We exploit this information by projecting objects to the floor, *i.e.*we project each triangle of the *object* mesh extracted in Sec 4.2 to the plane equation of the nearest floor-classified polygon whose centroid lies below the triangle. We recompute the floor polygon from the union of the original floor polygon and the projected triangle surfaces.

Closing Holes in Walls We extend walls to span from ceiling to floor. In particular, we identify polygon edges of wall-classified polygons whose normals face down. Then we count how many line segments in L (representing the observed empty space) intersect the area between each edge and the floor. If the number of intersections per cm^2 is below T_{extend}, we extend the edge to the floor. To ceilings and upwards-facing wall edges, we apply the same procedure. We call the output of this stage the *Layout Prototype*

4.4 Scene Graphs from a Layout Prototype

In this stage, we convert the prototype (a set of semantically labeled polygons) into the final layout. The final layout is organized as a scene graph, where the nodes (rooms) are connected by edges (doors, stairs). Every node is composed of a single floor, and a set of walls, ceiling, and window polygons. To achieve this, we first create 2D floorplans, which we later extrude into 3D space. The indirection via 2D is motivated by the fact that our 3D layout prototype neither gives us an understanding of indoor/outdoor space nor guarantees a closed or even connected layout.

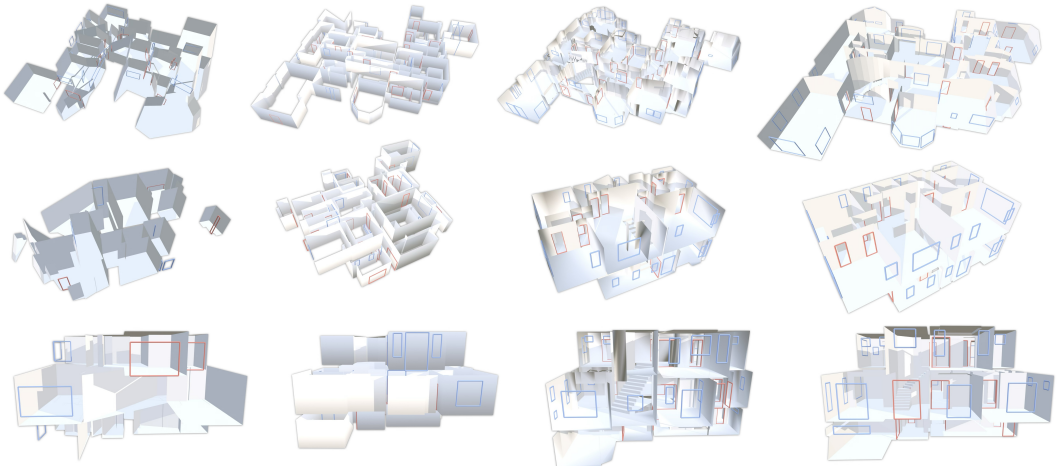
Creation of a Scene Graph of 2D Floorplans In this step we use the layout prototype and its semantics to (1) identify the different levels (floors) of the building, (2) create a 2D layout (floorplan) of each level, and (3) segment each level into rooms, extracting a per-level 2D scene graph from each floor and (4) detect stairs to connect the individual levels. In the following, we provide an outline of the applied algorithms, which are detailed in the appendix.

- To identify building floors, we use the floor-classified polygons of the layout prototype, merging close levels with similar heights.
- To create a 2D floorplan of each level, we merge each level's floor polygon(s) with suitable ceiling polygons — since ceilings are rarely occluded by objects and thus are more robustly represented in the layout prototype.
- To segment each level into rooms, we apply Hov-SG [15]'s room segmentation algorithm on each 2D floorplan (and the walls of the layout prototype). The segmentation outputs a scene graph with rooms as nodes, and *openings* as edges. We consider an opening edge a *door* if its width is below $1.5\,\text{m}$. Otherwise, we retain its edge but label it as *opening*. Furthermore, each room is associated with a room type (kitchen, office, ...).
- To identify stairs, we cluster connected components of the stair mesh extracted in Sec. 4.2. For each component, we add an edge to the scene graph between the rooms/floors it connects.

Back to 3D: Room Extrusion Sec. 4.4 describes how we use the layout prototype to generate a scene graph of rooms. In this section, we propose a simple algorithm inspired by layout annotation tools [20], that extrudes each node's 2D floorplan to the ceiling. For a single room, the extrusion algorithm creates a closed room shell using a 2D floorplan and a set of potential 3D ceiling polygons that at least partially cover the floorplan. Fig. 4 visualizes the extrusion process. Its core idea is to (1) triangulate the 2D floorplan, (2) assign each triangle to a ceiling polygon and (3) extrude each triangle to its ceiling. Specifically, we triangulate the room's 2D floorplan using a 2D Constrained Delaunay Triangulation [30] built from the boundary of the floorplan, the ceiling candidates' edges, and the projections of the pairwise intersection lines of the ceiling candidates' planes. For each triangle center, we cast a ray upward. If the ray hits a ceiling candidate, we assign the triangle to that ceiling's plane. Intuitively, this assignment partitions the floorplan by 'rendering' ceiling polygons on the floor. Triangles that do not hit a ceiling are assigned to the lowest ceiling plane reachable in

Method	Structures		Doors		Windows		Stairs		Depth		
	F1@0.5	Avg F1	F1@0.5	Avg F1	F1@0.5	Avg F1	F1@0.5	Avg F1	Δ_5	Δ_{10}	#Vertices
RoomFormer [1] (per floor)	$0.24_{\pm 0.06}$	$0.22_{\pm 0.06}$	$0.23_{\pm 0.10}$	$0.20_{\pm 0.09}$	$0.07_{\pm 0.06}$	$0.07_{\pm 0.04}$	–	–	$24.9_{\pm 11.5}$	$32.9_{\pm 14.9}$	764.9
RoomFormer [1] (per room)	$0.18_{\pm 0.14}$	$0.16_{\pm 0.12}$	$0.18_{\pm 0.14}$	$0.16_{\pm 0.12}$	$0.08_{\pm 0.08}$	$0.09_{\pm 0.07}$	–	–	$37.3_{\pm 10.4}$	$44.8_{\pm 10.7}$	1134.5
SceneScript [2] (per floor)	$0.28_{\pm 0.11}$	$0.26_{\pm 0.08}$	$0.23_{\pm 0.26}$	$0.20_{\pm 0.23}$	$0.16_{\pm 0.18}$	$0.15_{\pm 0.17}$	–	–	$22.5_{\pm 8.6}$	$33.8_{\pm 11.7}$	**677.1**
SceneScript [2] (per room)	$0.23_{\pm 0.12}$	$0.21_{\pm 0.11}$	$0.31_{\pm 0.26}$	$0.28_{\pm 0.23}$	$0.11_{\pm 0.11}$	$0.10_{\pm 0.09}$	–	–	$23.5_{\pm 7.2}$	$32.9_{\pm 6.7}$	1333.6
MultiFloor3D (Ours)	$\mathbf{0.40}_{\pm 0.10}$	$\mathbf{0.38}_{\pm 0.10}$	$\mathbf{0.55}_{\pm 0.16}$	$\mathbf{0.44}_{\pm 0.15}$	$\mathbf{0.43}_{\pm 0.29}$	$\mathbf{0.38}_{\pm 0.22}$	$\mathbf{0.42}_{\pm 0.48}$	$\mathbf{0.41}_{\pm 0.44}$	$\mathbf{61.1}_{\pm 9.2}$	$\mathbf{76.3}_{\pm 7.9}$	1957.0

Table 2: **Scores on HOUSELAYOUT3D.** Performance comparison with state-of-the-art layout estimation methods in terms of average and standard deviation across scenes. Structures include wall, floor and ceilings. MultiFloor3D is the only method predicting stairs.



RoomFormer [1] *SceneScript [2]* *MultiFloor3D (Ours)* *Ground-truth*

Figure 5: **Qualitative Results on HOUSELAYOUT3D.** We present layout estimation samples from our model alongside state-of-the-art methods. To enhance visualization, we apply back-face culling to the layout meshes, allowing a clear view inside the buildings. Since SceneScript represents walls as boxes, back-face culling is ineffective; instead, we remove the added floors and ceilings for better visibility.

the graph of unassigned triangles. (Lowest in terms of the triangle midpoint's projected z-coordinate on the target plane.) Lastly, we extrude each floor triangle to its assigned ceiling plane. That is, we produce ceiling and floor triangles on the ceiling and floor planes respectively, and add axis-aligned wall rectangles for triangle edges coinciding with a wall in the 2D floorplan. To ensure a closed room shell, we further add vertical rectangles along potential discontinuous edges in the extruded ceiling surface. To limit complexity, we only consider the 30 largest ceilings per room. Details on how we add doors and stairs after extruding are provided in the appendix.

Window Detection To detect windows, we back-project the 2D window segmentation of the input images obtained in Sec. 4.2 onto layout walls and cluster the result. Concretely, we create rays for window-classified pixels and intersect them with the walls of our 3D layout. We then filter outliers [31], split the points by wall instance, and run DBSCAN [32] for each wall to identify window clusters. To every cluster with at least $k = 10$ vertices, we fit an axis-aligned bounding rectangle. Finally, we predict a window for every rectangle with height and width greater than 30cm.

5 Experiments

In this section, we first introduce metrics to measure the performance of 3D room layout estimation, and then compare our approach to recent state-of-the-art methods on the proposed MultiFloor3D dataset (Sec. 5.1) as well as ScanNet++ [33] (Sec. 5.2). We then provide analysis experiments to understand the importance of the individual pipeline components (Sec. 5.3), and conclude with qualitative results and potential applications (Sec. 5.4).

Methods in Comparison. We compare our approach with two recent methods for scene layout estimation, namely RoomFormer [1] and the recent SceneScript [2]. Training these baselines on our multi-floor dataset is non-trivial – RoomFormer is designed for 2D floorplan prediction, while SceneScript is limited to 4-corner primitives. Instead, we evaluate them using their publicly available model weights on the full HOUSELAYOUT3D dataset. Both baselines are trained on large-scale synthetic datasets (∼100k samples), whereas our approach is training-free. Similar to [2], we extrude RoomFormer's 2D predictions to 3D. Finally, as neither baseline explicitly predicts ceilings or floors, we append ceiling and floor polygons to each predicted room to ensure a fair depth evaluation.

Method	#Vertices	Δ_5	Δ_{10}
DN-Splatter Mesh [21]	$354k$	84.1	92.6
RoomFormer [1]	**32.5**	36.8	48.9
SceneScript [2]	41.2	55.1	68.5
MultiFloor3D (Ours)	83.1	**67.8**	**84.7**

Table 3: **Scores on ScanNet++ [33].** Metrics evaluate depth accuracy as an approximation of layout estimation error. Scores are averaged over validation scenes.

Method	Avg F1	#Vertices	Sem.
Input Mesh + QSlim [35]	0.109	2000.0	✗
Layout Skeleton + QSlim [35]	0.223	2000.0	✗
Layout Prototype	0.373	2553.0	✗
MultiFloor3D (Ours)	**0.381**	**1957.1**	✓
(w/o prototype fitting)	0.214	2269.8	✓
(w/o room segmentation)	0.359	2442.2	(✓)

Table 4: **Ablation Study on HOUSELAYOUT3D.**

Layout Metrics. To assess the accuracy of the estimated layouts, we adopt the F1 score based on the *entity distance* d_E, following SceneScript [2]. This metric measures the alignment between ground truth entities E and predicted entities E'. For rectangular entities (*e.g.* doors and windows), d_E is computed as the maximum distance between corresponding corners of two rectangles of the same class: $d_E(E, E') = \max\left\{\|c_i - c'_{\pi(i)}\| : i = 1, \ldots, 4\right\}$ where $\pi(i)$ denotes the optimal corner permutation obtained via Hungarian matching. The F1 score @τ is then computed by applying a threshold τ to d_E as in [2].

For non-rectangular entities, we introduce a generalized entity distance d_H which allows comparison between entities with different numbers of corners. We define d_H as the Hausdorff distance between two polygon surfaces (*i.e.*entities) P, P' and their vertices V, V':

$$d_H(P, P') = \max\left\{\max_{v \in V} D_{pp}(v, P'),\ \max_{v' \in V'} D_{pp}(v', P)\right\} \tag{6}$$

for the point-to-polygon distance D_{pp} defined in Sec 4.3. We then use d_H analogously to d_E to compute the F1 score for walls, floors, and ceilings.

Depth Metrics. Following [17], we use input camera poses to render depth maps for the ground truth geometry D_{GT} and predict layouts D_{pred}. When explicit layout annotations are unavailable (*e.g.*, ScanNet++ [33]), depth consistency serves as a proxy for evaluating layout accuracy. Specifically, we compute the percentage of predicted pixel depths that fall within a threshold T cm of the GT depth:

$$\Delta_T = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{[|D_{\text{pred}}(i) - D_{GT}(i)| \leq T]} \tag{7}$$

This Δ_T metric was introduced [34] and is commonly used in monocular depth estimation [25].

5.1 Results on the HOUSELAYOUT3D Dataset

Table 2 shows the main results for the F1-based metrics across semantic classes, and depth metrics on our HOUSELAYOUT3D dataset. For this experiment, we use the camera poses, RGB-D images and mesh of MP3D [7]. As neither baseline is designed for multi-floor layout prediction, we apply them separately per floor (or per room) and then merge the per-floor (or per-room) predictions. We use the ground-truth MP3D level and room segmentation and report scores per-floor and per-room. Our MultiFloor3D does not have access to this privileged information.

MultiFloor3D significantly outperforms state-of-the-art layout estimation methods, despite not using ground-truth floor or room segmentation. While both baselines perform better on individual rooms than full floors, this gap is smaller for SceneScript, which favors compactness (*i.e.*, fewer vertices) at the cost of geometric fidelity.

5.2 Results on the Scannet++ Dataset

Table 3 shows additional results on the Scannet++ [33] *DSLR* validation split, consisting of 50 scenes captured with a monocular hand-held camera and COLMAP-generated image poses. As ScanNet++ does not provide ground truth layout annotation, we only report depth metrics as an approximation of the layout error. Since ScanNet++ scenes are populated with objects, we use ground truth semantic annotations to ignore those points during the evaluation, as well as points on windows which are typically not well reconstructed in the ground truth laser scan. As input for all methods, use the mesh provided by the Gaussian Splatting approach DN-Splatter [21] in the first stage of our method (Sec. 4.1). The results indicate that MultiFloor3D outperforms the baselines at the cost of compactness (larger number of vertices).

5.3 Analysis Experiments

Table 4 shows the contributions in terms of F1 score of each stage in our approach. Note that the outputs of the first and second stages (*mesh* from Sec. 4.1 and *layout skeleton* from Sec. 4.2) are triangle meshes, which we convert to polygon sets by first applying mesh simplification (QSlim [35]), and then greedily merging adjacent triangles whose normals differ by less than 20°. Performance drops significantly when either layout fitting or room segmentation is removed.

5.4 Qualitative Results and Applications

We show qualitative results of our approach in Fig. 5 and compare to RoomFormer [1] and SceneScript [2]. Both baselines methods struggle with large areas consisting of multiple rooms, RoomFormer even more than SceneScript. The baselines ere also inherently limitted to predicting rectangular primitives and cannot represent more complex shapes such as sloped ceilings *(top example)*. In Fig. 6 we visualizes qualitative results when removing loss objectives from the mesh fitting stage introduced in Sec. 4.3.

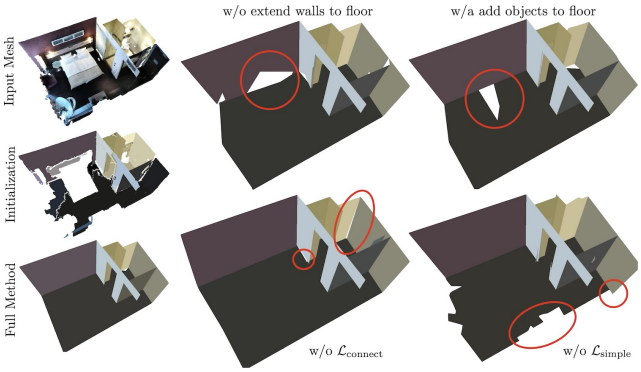


Figure 6: **Effect of Loss Terms.** *Left:* input and output of our approach. *Right:* result when ablating losses and components. Omitting object projection (top center) or wall extension (top right) produces holes in the layout. Without $\mathcal{L}_{\text{simple}}$, the polygon boundaries show dents. Without $\mathcal{L}_{\text{connect}}$, we observe gaps between polygons that otherwise share edges.

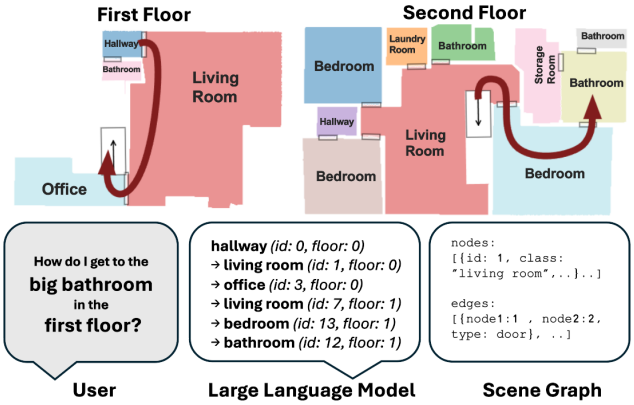


Figure 7: Navigation application based on 3D layouts and LLMs.

Down-stream Application. Next, we demonstrate a potential application of full-building 3D layouts. First, we obtain the 3D scene graph where nodes represent rooms, and edges are connections between rooms (doors, stairs, etc.) Then, we feed the scene graph in JSON format to an LLM, together with a user-prompt asking for directions. The LLM responds with turn-by-turn directions on how to reach the desired location. This concept is illustrated in Fig. 7.

6 Limitations

MultiFloor3D has a significantly longer runtime than the feed-forward baselines, taking one to two hours per HOUSELAYOUT3D scene on an NVIDIA GeForce RTX 4090, compared to one to two minutes for SceneScript [2] and RoomFormer [1]. Furthermore, MultiFloor3D occasionally struggles to remove outdoor elements perceived through large windows, which can introduce artifacts.

7 Conclusion and Discussion

We introduced HOUSELAYOUT3D, the first benchmark dataset for evaluating 3D layout estimation in large-scale, multi-floor buildings. Existing scene layout estimation methods are limited to single-floor buildings, and our experiments reveal their challenges in accurately parsing large-scale floors with multiple rooms—contrasted by our learning-free method, which already outperforms these baselines. Ideally, future research should develop learning-based approaches capable of handling multi-floor, multi-room buildings, rather than relying on heuristics, which, while effective, are significantly slower than feed-forward networks. In summary, we hope that our dataset and evaluation, which highlight the shortcomings of current methods, will drive further advancements in 3D layout estimation beyond single-room and single-level reconstruction.

References

[1] Yuanwen Yue, Theodora Kontogianni, Konrad Schindler, and Francis Engelmann. Connecting the Dots: Floorplan Reconstruction Using Two-Level Queries. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[2] Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, et al. Scenescript: Reconstructing scenes with an autoregressive structured language model. *European Conference on Computer Vision (ECCV)*, 2024.

[3] Armen Avetisyan, Tatiana Khanova, Christopher Choy, Denver Dash, Angela Dai, and Matthias Nießner. Scenecad: Predicting object alignments and layouts in rgb-d scans. In *The European Conference on Computer Vision (ECCV)*, August 2020.

[4] Denys Rozumnyi, Stefan Popov, Kevis-Kokitsi Maninis, Matthias Nießner, and Vittorio Ferrari. Estimating generic 3d room structures from 2d annotations, 2023.

[5] Jiacheng Chen, Ruizhi Deng, and Yasutaka Furukawa. Polydiffuse: Polygonal shape reconstruction via guided set diffusion models. *International Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[6] Jia Zheng, Junfei Zhang, Jing Li, Rui Tang, Shenghua Gao, and Zihan Zhou. Structured3d: A large photo-realistic dataset for structured 3d modeling. In *Proceedings of The European Conference on Computer Vision (ECCV)*, 2020.

[7] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[8] Srivathsan Murali, Pablo Speciale, Martin R. Oswald, and Marc Pollefeys. Indoor scan2bim: Building information models of house interiors. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6126–6133, 2017.

[9] Shang-Ta Yang, Fu-En Wang, Chi-Han Peng, Peter Wonka, Min Sun, and Hung-Kuo Chu. Dula-net: A dual-projection network for estimating room layouts from a single rgb panorama, 2019.

[10] Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. Layoutnet: Reconstructing the 3d room layout from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2051–2059, 2018.

[11] Chen Liu, Jiaye Wu, and Yasutaka Furukawa. Floornet: A unified framework for floorplan reconstruction from 3d scans, 2018.

[12] Sebastian Ochmann, Richard Vock, and Reinhard Klein. Automatic reconstruction of fully volumetric 3d building models from oriented point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 151:251–262, 2019.

[13] Ricardo Cabral and Yasutaka Furukawa. Piecewise planar and compact floorplan reconstruction from images. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 628–635, 2014.

[14] Jiacheng Chen, Chen Liu, Jiaye Wu, and Yasutaka Furukawa. Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path, 2019.

[15] Abdelrhman Werby, Chenguang Huang, Martin Büchner, Abhinav Valada, and Wolfram Burgard. Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation. *Robotics: Science and Systems*, 2024.

[16] Steve Cruz, Will Hutchcroft, Yuguang Li, Naji Khosravan, Ivaylo Boyadzhiev, and Sing Bing Kang. Zillow indoor dataset: Annotated floor plans with 360º panoramas and 3d room layouts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2133–2143, June 2021.

[17] Chuhang Zou, Jheng-Wei Su, Chi-Han Peng, Alex Colburn, Qi Shan, Peter Wonka, Hung-Kuo Chu, and Derek Hoiem. Manhattan room layout reconstruction from a single 360 image: A comparative study of state-of-the-art methods, 2020.

[18] VSIS Lab. Matterport3d-layout. https://github.com/vsislab/Matterport3D-Layout, 2020. GitHub repository. Accessed: 2025-03-02.

[19] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] SCASA. Pinpoint. https://scasa.eu/pinpoint/en/. Accessed: 2025-03-02.

[21] Matias Turkulainen, Xuqian Ren, Iaroslav Melekhov, Otto Seiskari, Esa Rahtu, and Juho Kannala. Dn-splatter: Depth and normal priors for gaussian splatting and meshing, 2024.

[22] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

[24] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. Goslar, DEU, 2006. Eurographics Association.

[25] Wei Yin, Chi Zhang, Hao Chen, Zhipeng Cai, Gang Yu, Kaixuan Wang, Xiaozhi Chen, and Chunhua Shen. Metric3d: Towards zero-shot metric 3d prediction from a single image, 2023.

[26] Jitesh Jain, Jiachen Li, MangTik Chiu, Ali Hassani, Nikita Orlov, and Humphrey Shi. OneFormer: One Transformer to Rule Universal Image Segmentation. 2023.

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[28] Damien Robert, Hugo Raguet, and Loic Landrieu. Scalable 3d panoptic segmentation as superpoint graph clustering. *Proceedings of the IEEE International Conference on 3D Vision*, 2024.

[29] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10:112–122, 1973.

[30] CGAL Team. *CGAL::Constrained_Delaunay_triangulation_2*. CGAL – Computational Geometry Algorithms Library, 2025. Accessed: 2025-03-07.

[31] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 93–104, New York, NY, USA, 2000. Association for Computing Machinery.

[32] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*, 1996.

[33] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes, 2023.

[34] Lubor Ladický, Jianbo Shi, and Marc Pollefeys. Pulling things out of perspective. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 89–96, 2014.

[35] 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*, SIGGRAPH '97, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co.