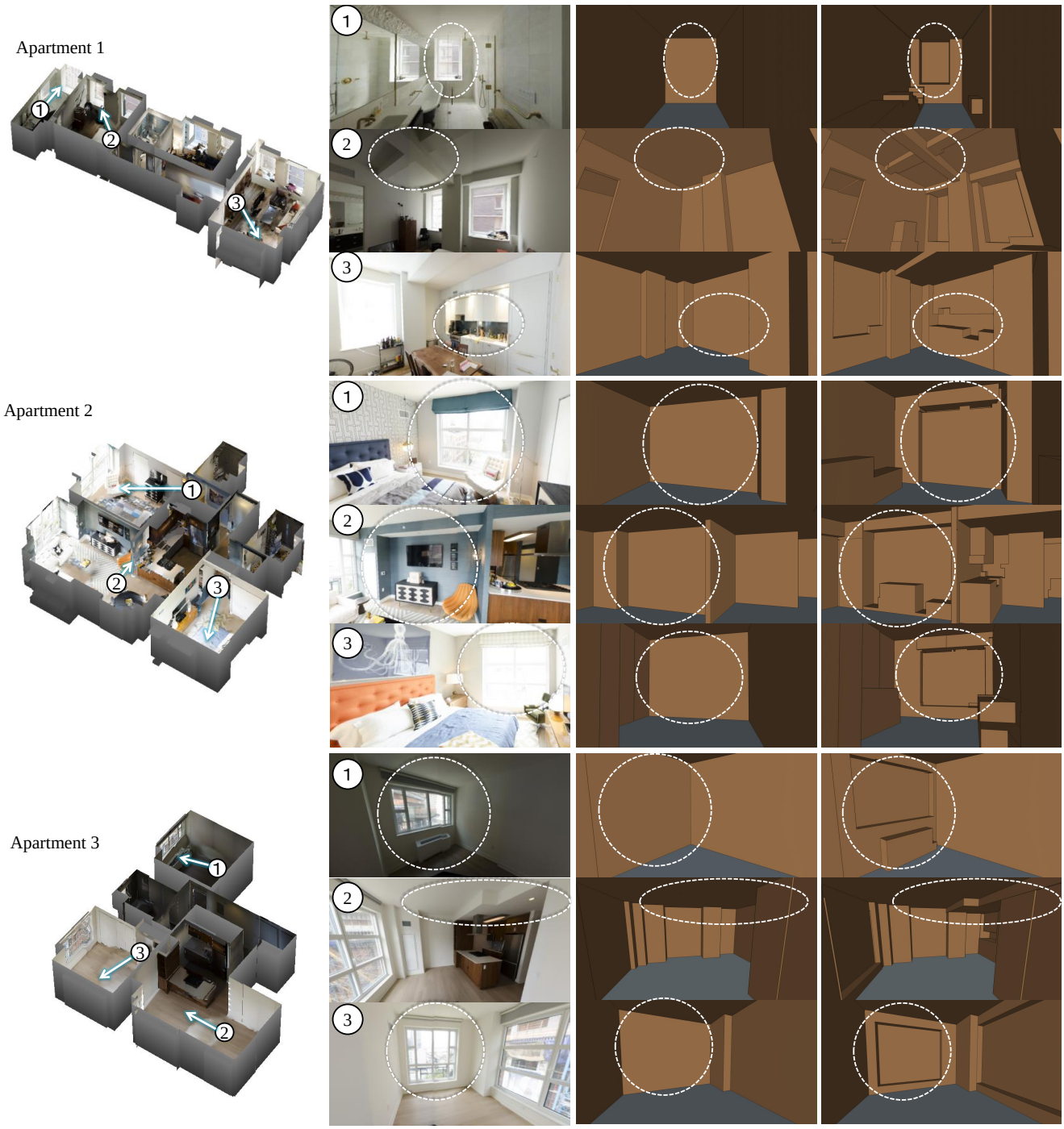
Structured Indoor Modeling
Supplementary Material

Satoshi Ikehata Hang Yan Yasutaka Furukawa
Washington University in St. Louis

The document provides additional experimental results, algorithmic details, and theorem proofs, which were not included in the main paper due to the space limitation.

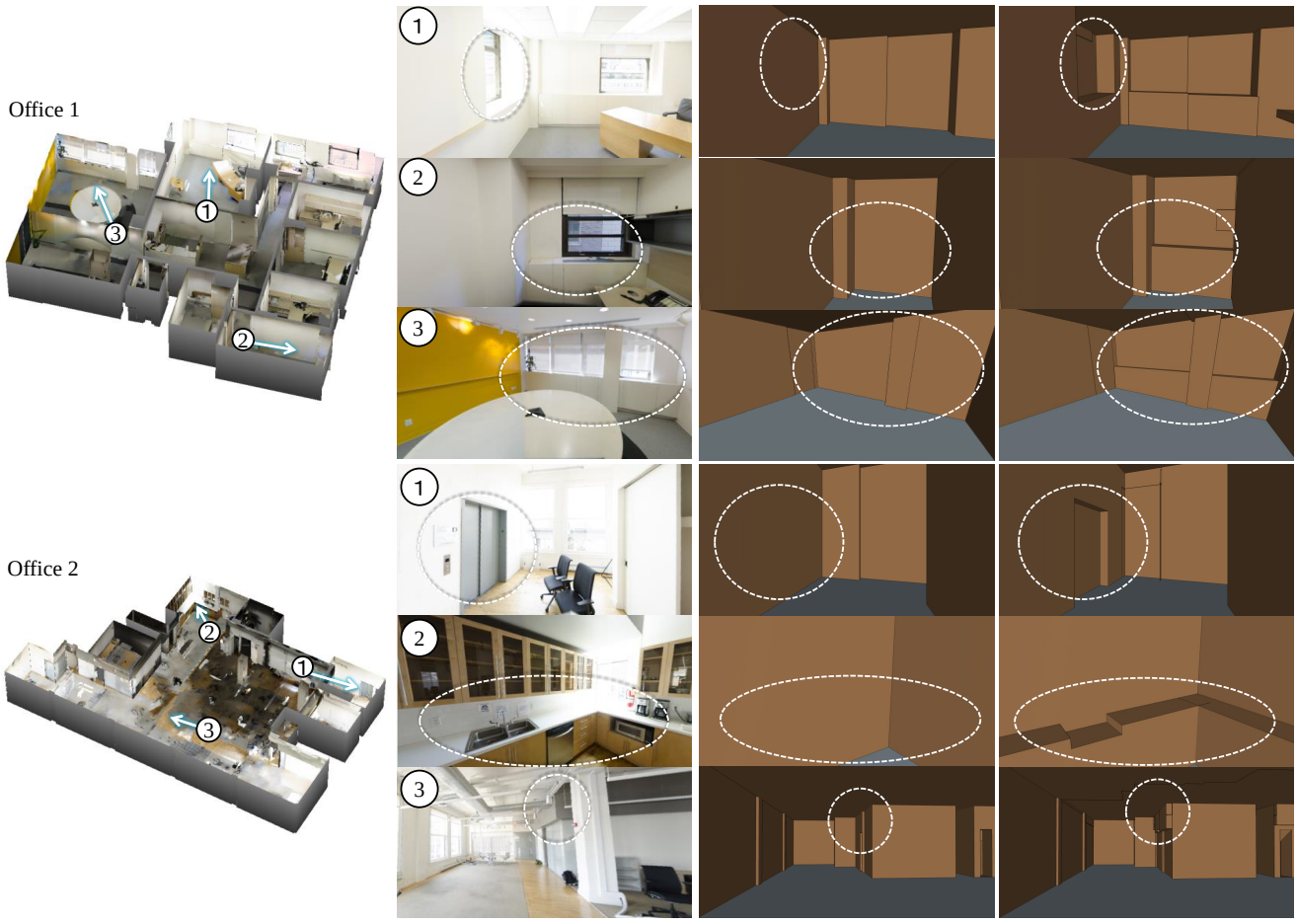
1. Additional Experimental Results

The section presents complete experimental results and evaluations. Please see the caption for the explanation of each figure.



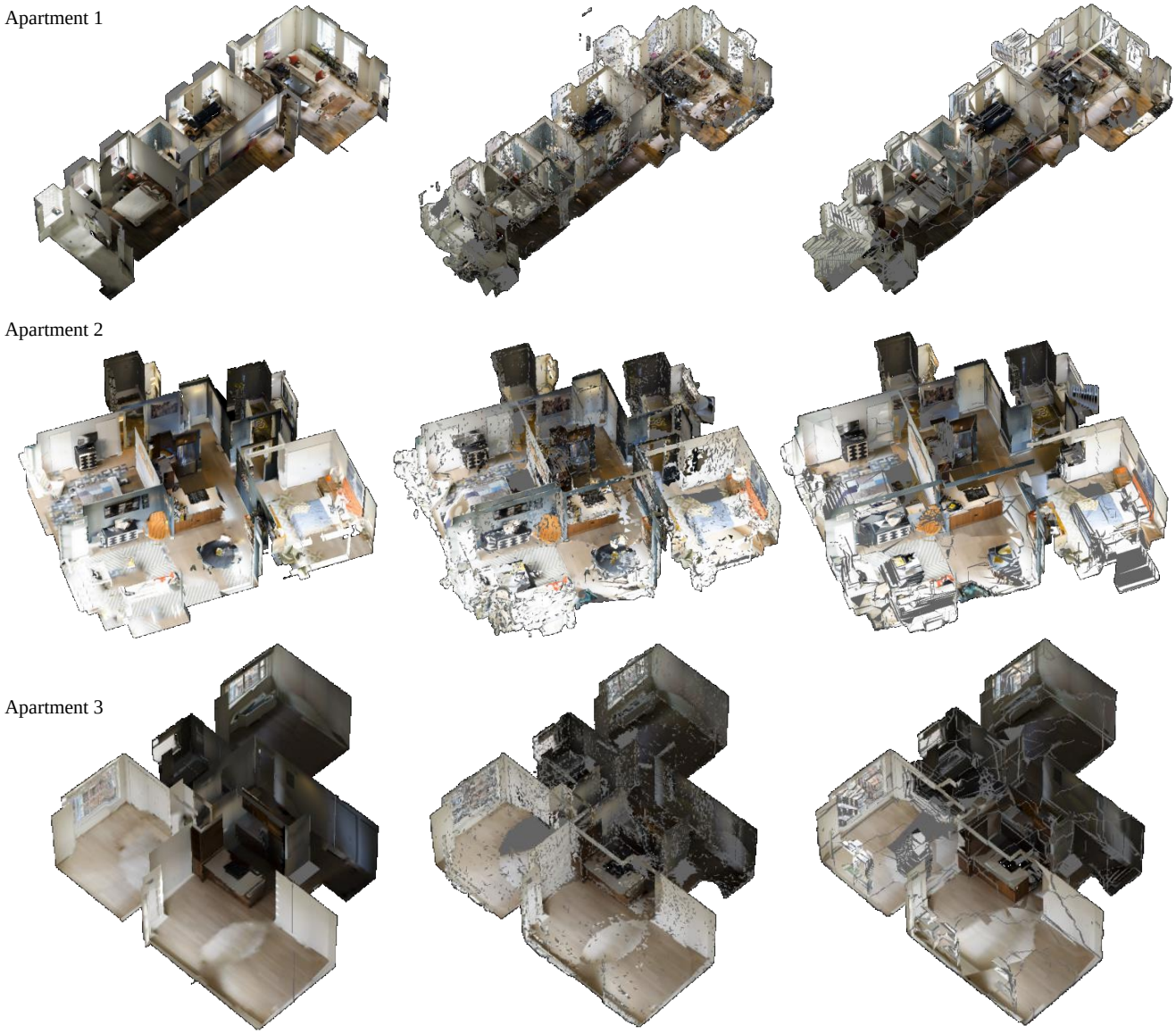
(a) Texture-mapped model (b) Target scene (c) w/o Details (c) w/ Details

Figure 1. The figure shows that our offset-map reconstruction algorithm is able to produce highly regularized and compact 3D structure. We show three examples for each dataset.



(a) Texture-mapped model (b) Target scene (c) w/o Details (c) w/ Details

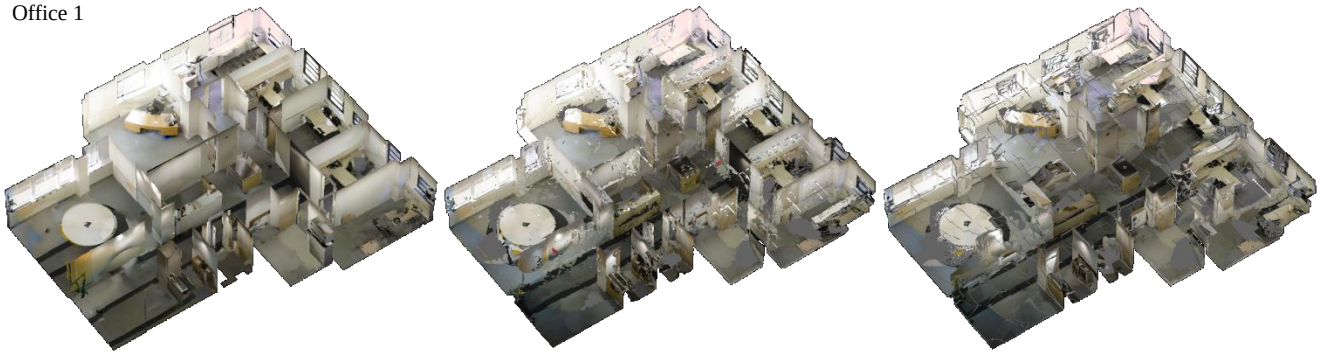
Figure 2. Continued.



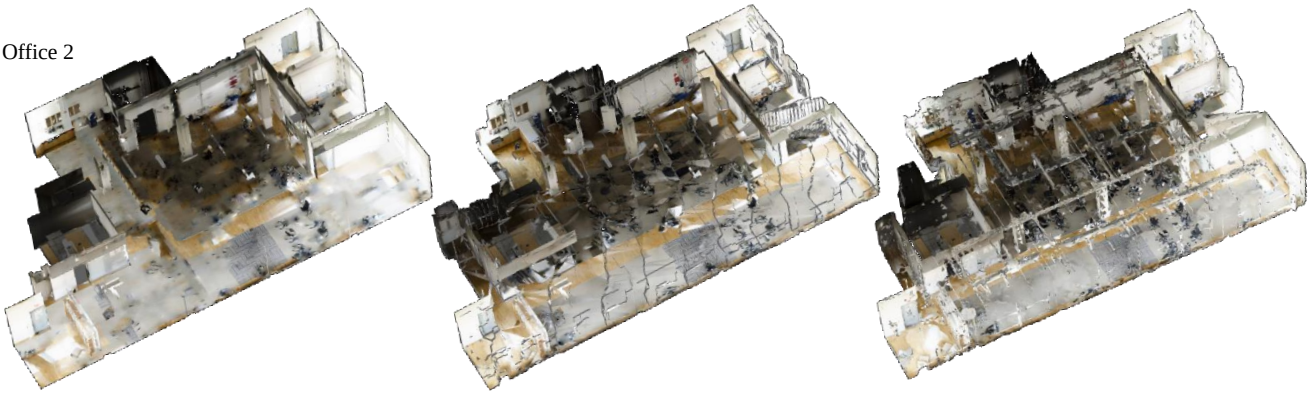
(a) Ours (b) Poission (high-res) (c) Vgcut (high-res)

Figure 3. Our structured model representation enables one to effectively hide (or add transparency to) back-facing surfaces such as ceilings and walls, at the level of structural elements as opposed to at the level of triangles. Meshes from existing algorithms only allow back-face culling per triangle and cause severe rendering artifacts for the aerial indoor scene visualization.

Office 1



Office 2



(a) Ours (b) Poission (high-res) (c) Vgcut (high-res)

Figure 4. Continued.

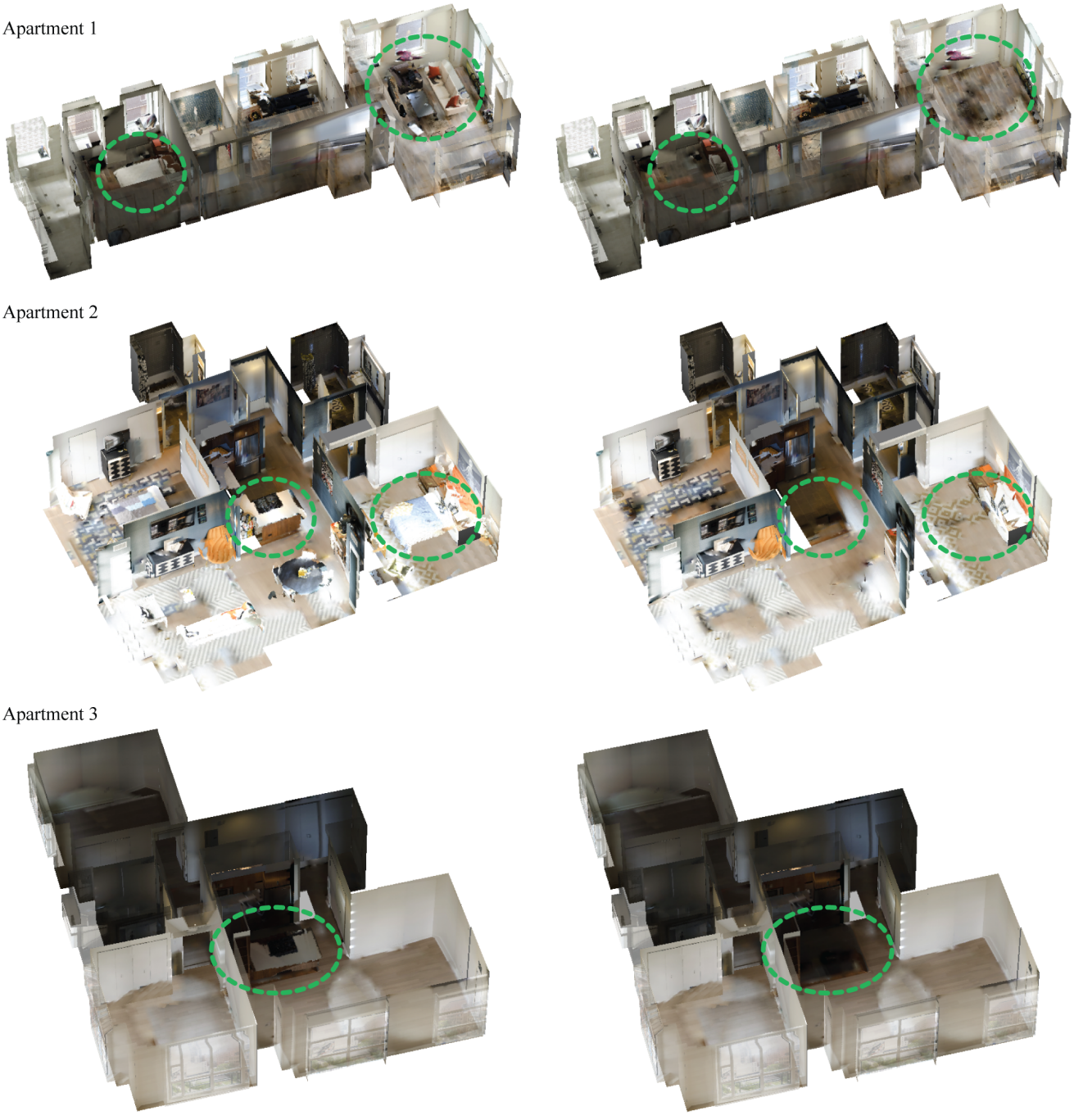


Figure 5. The left and right shows our model with and without objects rendered. Textures on the mesh models are computed for each piecewise planar surface by a combination of texture synthesis and inpainting techniques. Our structured representation essentially allows us to compute the texture for each structural element such as walls and floors, can effectively fill texture holes even behind objects.

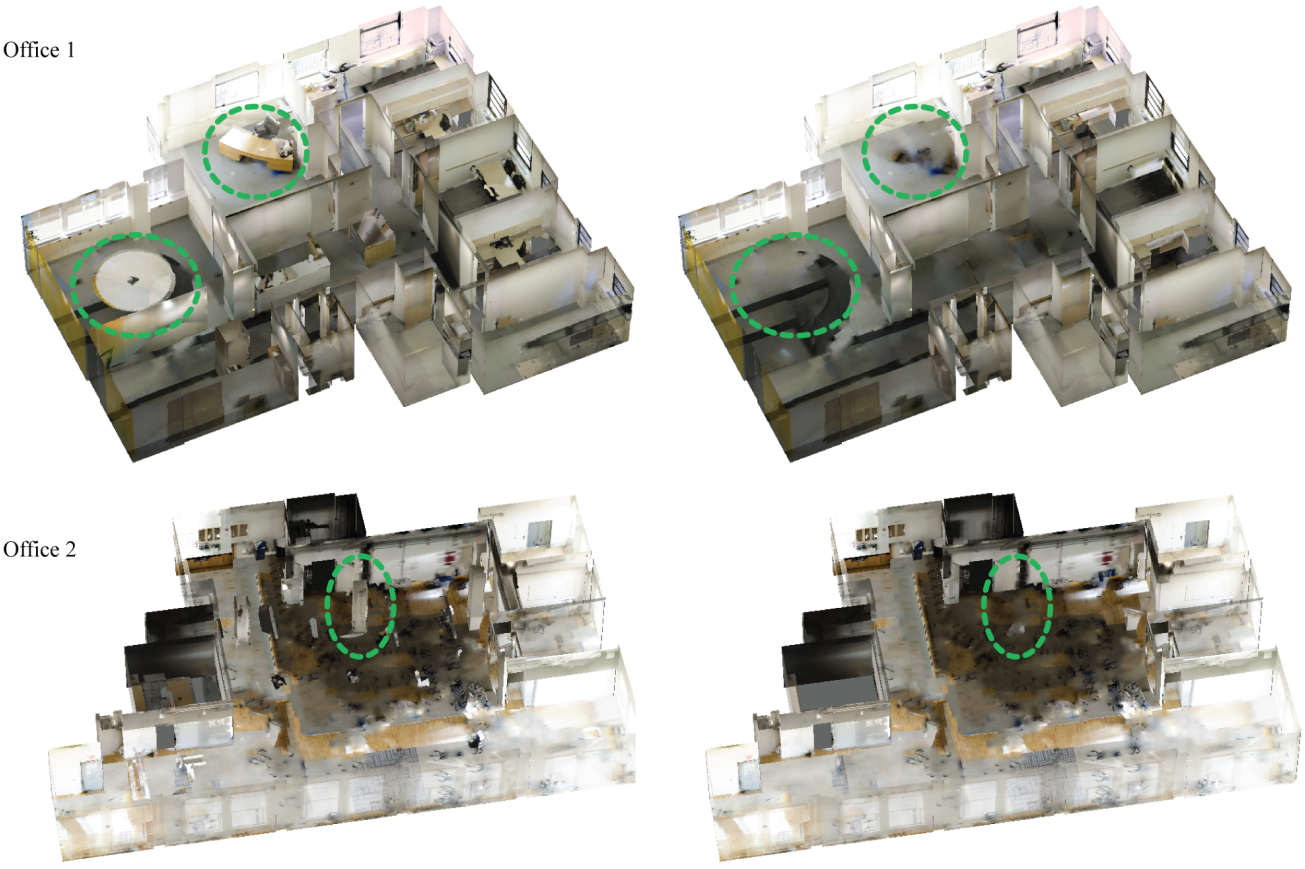
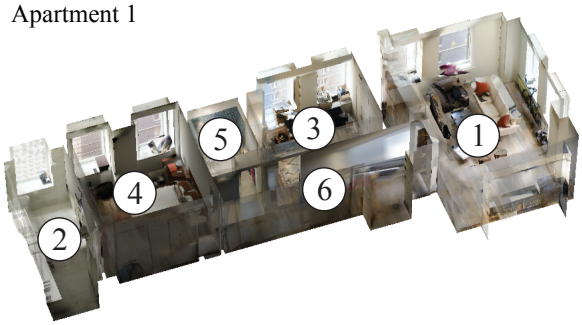


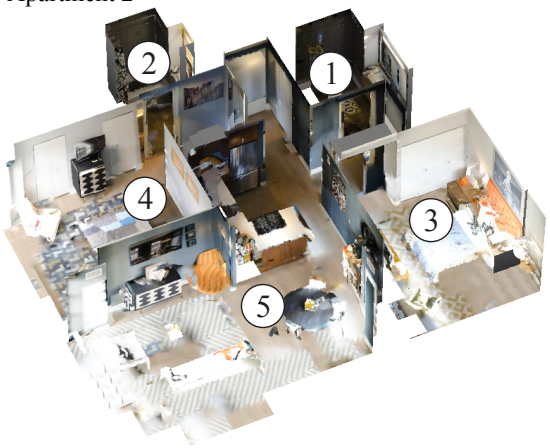
Figure 6. Continued.

Apartment 1



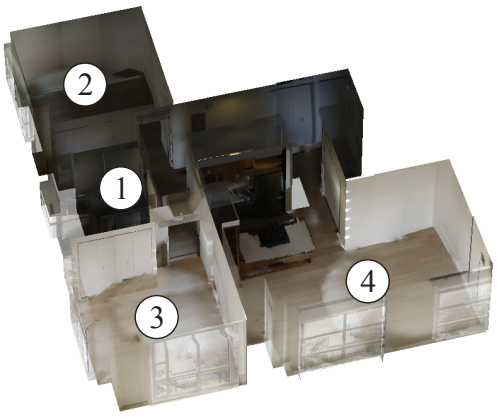
Room \ Algorithm	Uniform views closest panorama	Best view closest panorama	Best view nearby panoramas	Best view best panorama
1	living_room	living_room	living_room	kitchen
2	shower	shower	shower	shower
3	shower	living_room	living_room	living_room
4	bedroom	bedroom	bedroom	bedroom
5	shower	shower	shower	shower
6	corridor	corridor	corridor	corridor

Apartment 2



Room \ Algorithm	Uniform views closest panorama	Best view closest panorama	Best view nearby panoramas	Best view best panorama
1	shower	shower	shower	shower
2	closet	closet	closet	closet
3	hotel_room	hotel_room	hotel_room	hotel_room
4	bedroom	bedroom	bedroom	bedroom
5	living_room	kitchen	kitchen	dinette/home

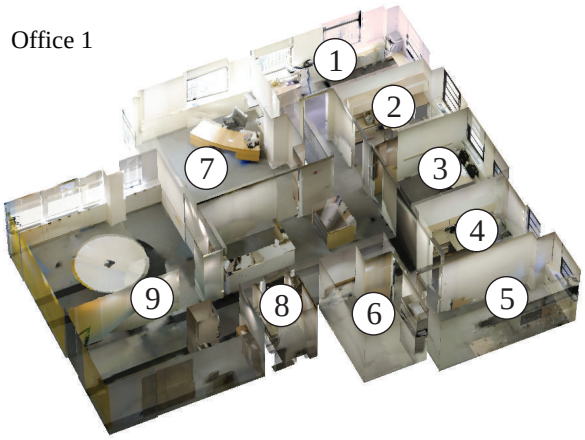
Apartment 3



Room \ Algorithm	Uniform views closest panorama	Best view closest panorama	Best view nearby panoramas	Best view best panorama
1	closet	closet	closet	living_room
2	corridor	corridor	corridor	corridor
3	closet	doorway/outdoor	doorway	living_room
4	closet	closet	corridor	living_room

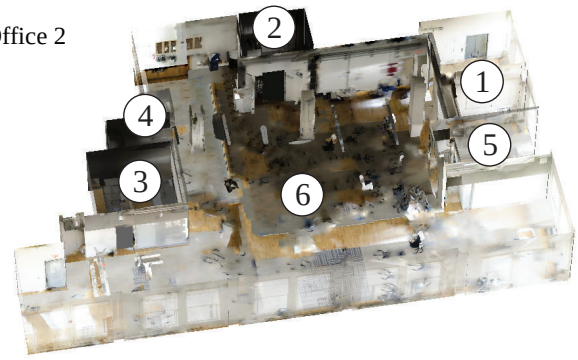
Figure 7. Room annotations are obtained by using the state-of-the-art scene recognition system [7]. We experimented four different algorithms in choosing images to represent each room, yielding large performance variations. The annotation result for each room by each algorithm is shown in the table. We manually verified the results and highlight mistakes in red.

Office 1



Room \ Algorithm	Uniform views closest panorama	Best view closest panorama	Best view nearby panoramas	Best view best panorama
1	shower	corridor	corridor	corridor
2	shower	basement	office	office
3	basement	closet	closet	classroom
4	closet	office	office	office
5	sky	basement	basement	corridor
6	shower	closet	basement	basement
7	office	office	office	office
8	closet	reception	reception	reception
9	corridor	corridor	corridor	corridor

Office 2



Room \ Algorithm	Uniform views closest panorama	Best view closest panorama	Best view nearby panoramas	Best view best panorama
1	shower	art_gallery	office	office
2	jail_cell	jail_cell	jail_cell	jail_cell
3	shower	conf_room	conf_room	conf_room
4	sky	sky	sky	sky
5	office	shower	shower	shower
6	office	office	office	office

Figure 8. Continued.

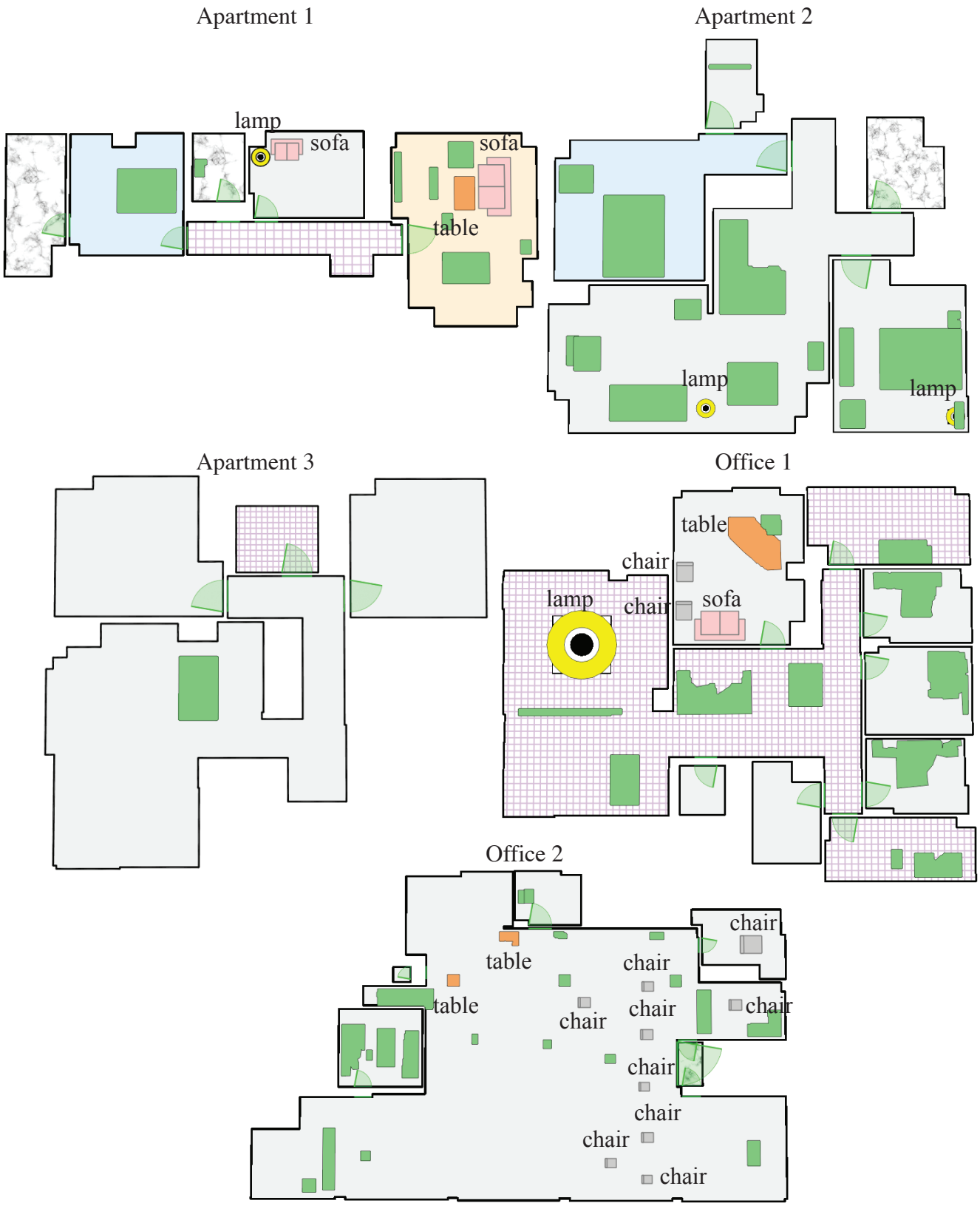


Figure 9. The object annotations turn out to be much more difficult than the room annotation. No texts are shown if an annotation cannot be associated. The segmentation results are good for major objects, and annotations, once obtained, are usually accurate. However, most objects are missing annotations, especially in cluttered and small indoor rooms. A lot more improvements are to be made in this space.

Figure 10. The figure illustrates our object reconstruction process for three rooms in our datasets. From top to bottom: 1) a sample panorama image in the room; 2) the point-cloud in the room; 3) a point-cloud cluster for each object after removing points nearby the walls, the floors, or the ceilings; 4) point-cloud clusters after removing clusters that are too small or float in the air; and 5) the corresponding generated floorplan image in the room.

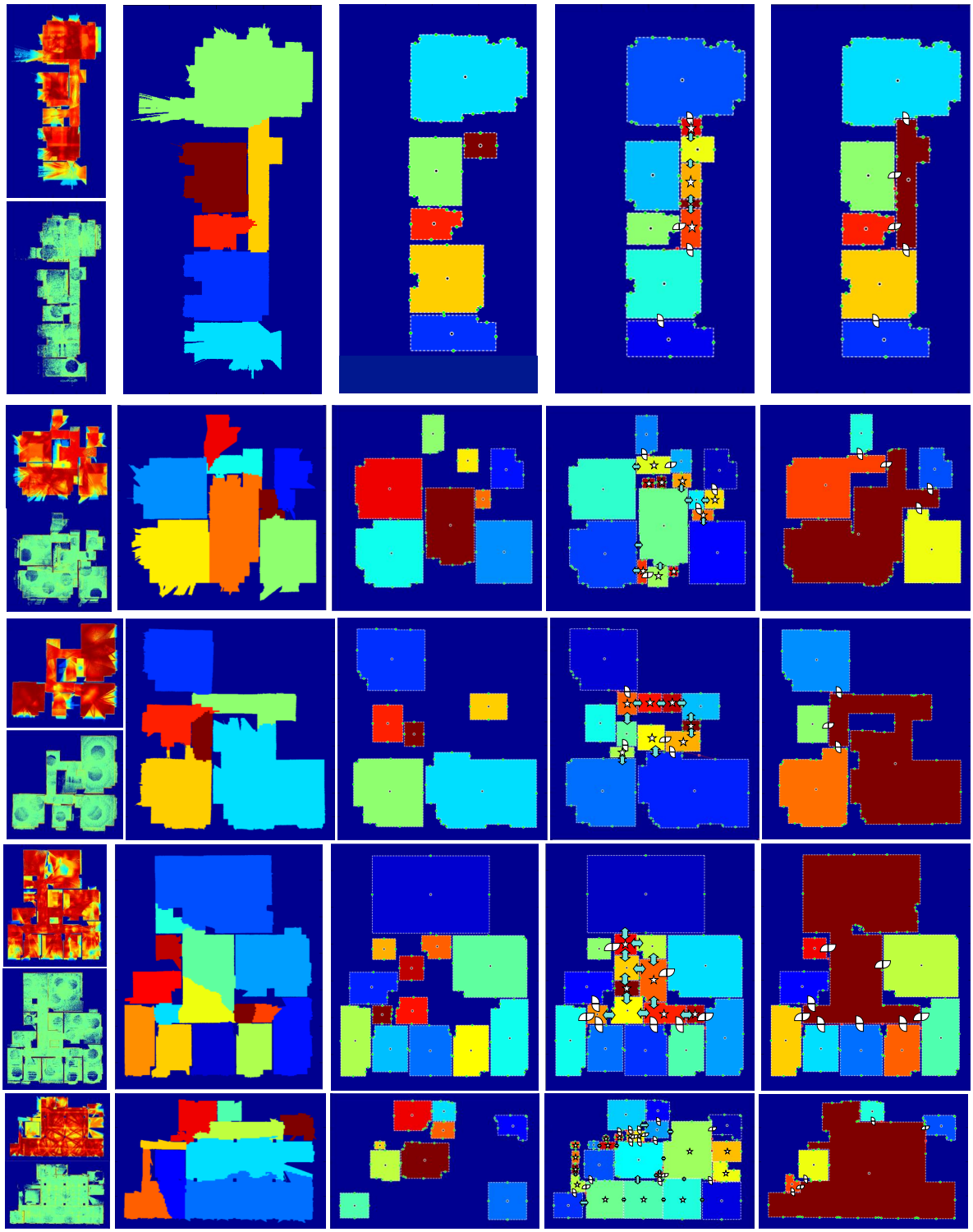


Figure 11. Starting from the generation of the free-space end point evidence (left), we perform the room-segmentation and reconstruction (2-nd and 3-rd column). More rooms are added and the room connection types are classified (4-th column). The white icon represents the "door", while the blue icon represents the "merge" classification. We finally get the structured graph representation as shown in the last column. Here we only show the spatial relationships for simplicity.

2. Structure graph - theory

The main paper provides the key ideas behind the structured modeling formulation. The supplementary material presents the complete mathematical definition, constraints, and proofs.

2.1. Definitions

Let v denote a node in a structure graph and its associated geometry with abuse of notation. We consider a 2D surface representation of v, where $\Gamma(v)$ denotes its boundary. The graph has directed and undirected edges. The *attachment* edges in the main paper are ignored in this analysis, as they do not exhibit geometric constraints across elements (except for intersections, which will be discussed later).

The right figure shows that potential connections at a node are a parent V_1^P, neighbors V_j^N, and children V_k^C. We define

$$\Gamma(e_1^P) = \Gamma(v) \cap \Gamma(v_1^P),$$
$$\Gamma(e_k^C) = \Gamma(v) \cap \Gamma(v_k^C),$$

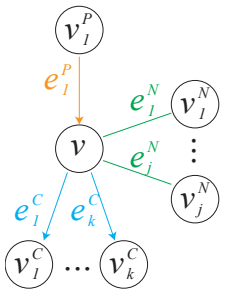


Figure 12. Three types of connected nodes: V_1^P, V_j^N, and V_k^C.

Edge $\Gamma(e_1^P)$ conveys a part of the boundary condition of v_1^P that must be satisfied by v for manifold-ness. The same is true for $\Gamma(e_k^C)$. The boundary associated with an edge must not be $\emptyset$, otherwise the edge should not be connected. $\Gamma(e_j^N)$ conveys the surface boundary where the two incident nodes must match exactly for manifold-ness. Its definition is not the simple intersection and will be later derived from the constraint. We use $\mathcal{P}$(arents), $\mathcal{N}$(eighbors), and $\mathcal{C}$(hildren) to denote a set of nodes that are connected by e_1^P, e_j^N, and e_k^C, respectively. Note that we will have a constraint not to allow multiple parents, and hence P can have at most one node.

2.2. Constraints

C1: The geometry v is a single connected component.

C2: The geometry $\bigcup_k v_k^C$ is a single connected component, if $\mathcal{C} \neq \emptyset$.

C3: $\Gamma(v) = \Gamma(e_1^P) \cup \left[\bigcup_j \Gamma(e_j^N)\right]$ if $\mathcal{P} \neq \emptyset$ or $\mathcal{N} \neq \emptyset$.
$\{\Gamma(e_1^P), \Gamma(e_1^N), \Gamma(e_2^N) \cdots\}$ are mutually exclusive.

C4: $\Gamma(v) = \bigcup_k \Gamma(e_k^C)$ if $\mathcal{C} \neq \emptyset$.
$\{\Gamma(e_1^C), \Gamma(e_2^C) \cdots\}$ are mutually exclusive.

C5: The graph is acyclic with respect to the directed edges.

C6: A node cannot have multiple parents.

The structure graph must satisfy these constraints. C2 states that the surface formed by the children must be a single connected component. This appears restrictive but is not. For modeling multiple connected components from a single surface, one should first split the domain, then add a component to each sub-domain. C3 states that the boundary condition of a node is satisfied by its neighbors and parent. C4 states that the boundary condition of v is propagated to its children.

2.3. Proofs

Lemma 2.1. $\Gamma(e_j^N) = \Gamma(v_j^N) \cap \left[\Gamma(v) \setminus \Gamma(e_1^P)\right]$.

Proof. C3 guarantees no intersection between $\Gamma(e_1^P)$ and $\Gamma(e_j^N)$. Therefore, the first line of C3 can be converted to

$$\Gamma(v) \setminus \Gamma(e_1^P) = \bigcup_j \Gamma(e_j^N).$$

Again $\{\Gamma(e_j^N)\}$ are mutually exclusive, and

$$\Gamma(e_j^N) \cap \left[\Gamma(v) \setminus \Gamma(e_1^P)\right] = \Gamma(e_j^N) \cap \left[\bigcup_j \Gamma(e_j^N)\right],$$
$$\Gamma(e_j^N) \cap \left[\Gamma(v) \setminus \Gamma(e_1^P)\right] = \Gamma(e_j^N).$$

□

Lemma 2.1 implies that only the boundary condition that has not been satisfied by the parent should be satisfied by the neighbors.

Lemma 2.2. *The nodes in $\mathcal{C}$ form a single connected component with respect to the undirected edges.*

Proof. Suppose $\mathcal{C}$ forms multiple connected components with respect to the undirected edges. Due to C2, there must be a pair of nodes (v, v_j^N) that shares the parent, are physically connected, but are not connected by e_j^N, that is, $\Gamma(e_j^N) = \emptyset$. We use γ to denote the physical surface boundary shared by these two nodes:

$$\gamma = \Gamma(v_j^N) \cap \Gamma(v).$$

From Lemma 2.1, the following holds

$$\Gamma(v_j^N) \cap \left[\Gamma(v) \setminus \Gamma(e_1^P)\right] = \emptyset.$$

Since v and v_j^N has a common boundary γ, $\Gamma(e_1^P)$ must contain γ for the above to hold.

$$\gamma \subseteq \Gamma(e_1^P).$$

v and v_j^N are interchangeable, and shares the parent. Therefore, this relationship is also true for another child of v_1^P. However, this violates the mutual exclusiveness in C4. □

Note that C2 states that $\mathcal{C}$ forms a single connected component in terms of its geometry. However, this does not provide a proof, as $\Gamma(e_j^N)$ can be $\emptyset$. Therefore, the above derivation completes the proof. Now, let us define that a *leaf node* does not have any out-going edge for the next lemma.

Lemma 2.3. *Given a structure graph, geometries at all the leaf nodes form a manifold,*

Proof. Given geometries at all the leaf nodes, we prove that the boundary conditions are satisfied at every node by induction. Let us first calculate the "distance" from the leaf-node as the fewest number of directed edges to reach any leaf. The induction is performed based on these distances.

First, at a leaf node, a geometry is created, and C3 ensures that its boundary conditions are satisfied at its boundary edges $\{e_j^N\}$ and the in-coming edge e_1^P. Second, suppose boundary conditions $\Gamma(v)$ at a node v are satisfied where the distances are less than or equal to k. Given a node whose distance is $k+1$, the boundary conditions at the out-going directed edges are satisfied, because the edges point to the nodes whose distances are at most k. C4 ensures that the entire boundary of v is generated by its out-going edges, all of which are now satisfied. Then, C3 again confirms that the boundary conditions are satisfied at its boundary and in-coming edges further up the tree. By induction, the boundary conditions are satisfied at every node. □

Our mesh compilation process generates a mesh after pruning an arbitrary set of nodes. This pruning operation must not leave any dangling boundary edges.

Lemma 2.4. *The structure graph group (a set of valid structure graphs) is closed under the pruning operation.*

Proof. We will check that every constraint is satisfied after any pruning operation.

C1: Independent of the pruning.

C2: Lemma 2.2 shows that all the children nodes must be pruned ($\mathcal{C}$ becomes $\emptyset$) or stay connected ($\mathcal{C}$ does not change).

C3: It suffices to prove for a node v that has not been pruned. Since no dangling boundary edge remains, all the neighbor nodes v_j^N should also stay in the graph. v_1^P is a parent of v and not pruned. Therefore, the equality stay unchanged. Mutual exclusiveness is independent of the pruning.

C4: Due to Lemma 2.2, all the children nodes must stay or be pruned. Therefore, either the equality does not change or $\mathcal{C}$ becomes $\emptyset$. Mutual exclusiveness is independent of the pruning.

C5: The graph loses connections by pruning, and is still acyclic.

C6: Pruning will not add new parents.

□

Theorem 2.5. *The structure graph produces a manifold-mesh after dropping an arbitrary set of nodes, as long as no dangling boundary edges remain.*

Proof. Lemma 2.3 and Lemma 2.4 prove. □

2.4. Practical considerations

It is easy to verify that our indoor structure grammar satisfies C5 and C6. It is also easy to see that C1 and C2 can be easily checked or enforced. The problems are C3 and C4, which impose non-trivial boundary conditions. The key observation is that the constraints act only on the surface boundaries. Therefore, our approach is to first fix the boundaries of each node to satisfy the constraints, then perform reconstruction algorithms subject to the given boundary conditions. This section shows that it is relatively straightforward to enforce these boundary conditions for each of the major geometric representations.

The first scenario is the 2D offset-map reconstruction on a quad patch (*e.g.*, wall detail reconstruction), where the boundary condition is at the boundary of the quad. We simply fix the depth values along the boundary, then the final 2D surface is guaranteed to match the specified boundary.

The second case is the volumetric scalar field followed by the Marching Cube algorithm for the mesh generation. Given the boundary condition, we know that the nearby space (voxels) must be either interior of exterior, which can be enforced by fixing the scalar values at the corresponding voxels [6]. The same goes for the point-cloud reconstruction.

Intersections in addition to the boundary condition can also be enforced in a similar manner. For an offset-map estimation algorithm, for example, infinite data terms can be used to avoid the reconstruction to go inside the interiors of the other geometries. The same goes for the volumetric scalar field and the point-cloud reconstruction, where the voxel values can be fixed inside the interiors of the other geometries. In our experiments, we do not enforce the intersection check in the reconstruction algorithms, as self-intersections are rare in practice. Furthermore, even if there exist, they do not pose problems to any of our applications.

Lastly, point-clouds are not converted to meshes for objects in our experiments for two reasons. First, point-clouds suffice for the visualization applications. Second, proper meshing of complex objects in the indoor scenes is a challenging problem by itself with very active on-going research. Note that our framework allows one to use any object reconstruction algorithm, even different algorithms for different object types, and we can easily exploit any other object reconstruction algorithm as a component.

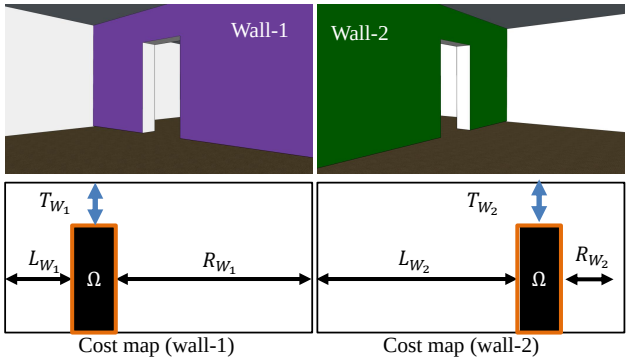


Figure 13. Once the rectangle Ω is extracted, we compute the left, the right, and the top margins between the rectangle and the wall.

3. Remaining structure grammar

This section presents the three remaining grammar rules.

Room merging: The rule takes two rooms and merges into a single room node, while discarding all the details and doors in each room (transformation). The room connection type must be classified as "merge"(pre-condition). The details of the connection classification will be given below. Given a "merge" connection classification, we connect the core free-space regions of the two rooms and simply apply the room reconstruction rule.

Room addition: In the process of room reconstruction and merging, the domain Ψ for the room segmentation (Sect. 4.3 in the main paper) may have a region that does not belong to any room. The rule adds a new room node to a scene (transformation). There are three pre-conditions. We identify the axis-aligned rectangle with the most area inside such uncovered region. The first condition is that the area of the rectangle is larger than 0.05 times the average room area. The next two conditions prevents generating rooms outside the windows, where 3D points suffer from strong structured noise. Second, find the closest room from every pixel inside the rectangle. There must be at least two rooms found. Third, the sum of the point evidence inside the rectangle must not be too small, in particular, more than half the sum of the free-space evidence.

Door addition: Given two rooms, whose connection type is classified as "door" (pre-condition), a rectangular hole (*i.e.*, doorway) is created in each wall. The door geometry, consisting of four planes connecting the two rectangular holes, is also created (transformation). Since the wall geometry or the detailed wall geometry are an 2D offset-map, it is easy to create a rectangular hole.

Room connection classification: This section details in the classification of the room connection, given the rectangles on the two opposing walls. We measure the three margins between the door and the wall boundary in each wall as in Fig. 13. Depending on the existence of the margins against some thresholds, each wall is classified into six different configurations (See Fig.14). Finally, the room connection type (either door or merge) is determined as shown in the table. The thresholds for the horizontal and the vertical margins are 0.13m and $0.05 \times \{\text{wall height}\}$, respectively.

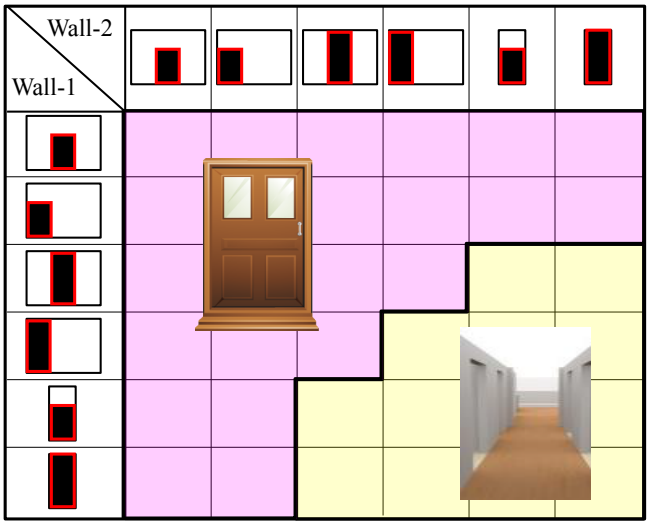


Figure 14. Classification table for the room connection types. Based on the configuration of the three margins, each wall is classified into six types. The combination of the two wall types classifies the room connection into either "door" (top-left) or "merge" (bottom-right).

4. Algorithmic details

The section provides algorithmic details that were not covered in the main paper.

4.1. Pre-processing

Points cloud acquired by multiple depth sensors are generally corrupted by mirrors, windows, reflective objects, and calibration errors, which may affect the quality of reconstruction. Therefore, we first remove noisy 3D points by the following two filters.

Intensity-based filtering: Most depth sensors emit the infrared light and record the pixel-wise intensity of the reflected light. At the presence of windows or reflective objects, corresponding intensity values become low. Therefore, we remove points if the intensity values are less than $\mu_I - \sigma_I$, where μ_I and σ_I are the mean and the standard deviation of the intensities in that image.

Connected-component filtering: The intensity thresholding is not effective against the mirrors, however these 3D points tend to be highly isolated. We compute the binary mask in exactly the same way as the computation of Ψ in the room segmentation, but this time, based only on one panorama depth image. We simply keep the largest con-

nected component and discard 3D points that project outside. This filtering is conducted for each panorama depth image. This criteria may not completely remove the mirror points, but is enough for our algorithm to suppress further noise in the reconstruction process.

4.2. Room reconstruction

The room reconstruction algorithm is based on the previous work [1], while we make an adjustment to the *core free-space region* extraction algorithm. The core free-space is computed as the space with very high free-space evidence, where the room boundary should not pass through. This caused a problem in our setting. Our input comes from a depth sensor, while their input is an MVS point-cloud. A depth sensor generates very strong outliers near the windows, and their core free-space region often crosses walls through the windows.

Our approach is to restrict the core free-space region to be an axis aligned rectangle. More specifically, we find the axis-aligned rectangle that fits inside the domain Ψ and minimizes the following objective function:

$$max(H, W) + \eta HW,$$

where H and W denote the width and height of the rectangle, and $\eta = 0.001$ is used in our experiments. This objective has an effect to maximize the rectangle while keeping its aspect ratio close to 1. We find the solution via the exhaustive search.

4.3. Wall/ceiling detail reconstruction

The initial offset-map in the wall/ceiling detail reconstruction is obtained by a simple 2D MRF with unary and pairwise terms:

$$E = \sum_{p} E_d(d_p) + \lambda \sum_{\{p,q\} \in \mathcal{N}} E_s(d_p, d_q).$$

p and q denote pixels, while d_p is the estimated offset-value at pixel p. The pairwise term E_s follows a $Potts$ model where $\lambda = 0.01$ is used in our experiments. The unary term E_d is defined as

$$E_d(d_p) = -P(p, d_p) \left(\sum_{d=d_p^{min}}^{d_p} F(p, d) \right).$$

$P(p, d_p)$ and $F(p, d_p)$ denote the point and free-space evidence at pixel p and offset d_p, respectively. Intuitively, the cost becomes smaller if the point evidence and the accumulation of the free-space evidence are large. The valid depth range of a pixel $[d_p^{min}, d_p^{max}]$ is computed from the domain Ψ for the room segmentation in the top-down view. More concretely, we simply cast rays in the two opposing directions orthogonal to the wall, and obtain the first intersection with the boundary of Ψ each. Finally, we get the intersection with the range $[-2\text{m}, 2\text{m}]$ to avoid unnecessarily large search space.

5. Application Details

This section presents the details of our applications demonstrated in the main paper and the supplementary video.

5.1. Floorplan generation

Figures 7 and 8 illustrate the results of the room annotation experiments. The room annotation did not work well with the full panorama images, and we render 400×300 standard perspective images with the horizontal field of view of 90 degrees. The results heavily depend on the choice of the viewing directions and we experimented the four algorithms in an increasing order of accuracy. The first algorithm picks the panorama closest to the room center, generates six uniformly overlapping images, and picks the scene type with the best average score. The second one exploits more scene information, and uses only one image out of six, in which the visible room area is the maximum in the top-down view. The third algorithm uses the second algorithm for all the panoramas inside the room, and uses the average to pick the best score. The last algorithm takes the same set of images as the third one, but only uses a single image, in which the room is the most visible. Surprisingly, this last algorithm works the best on all the datasets. An observation is that poorly positioned panoramas tend to yield incorrect results but with very high confidence. A rather better strategy is to use the best viewing direction from the best panorama.

Figure 9 shows the object annotation results, which turn out to be much more challenging than the room annotations, even with the state-of-the-art object detectors [3]. Our object annotation has the following three steps. First, object detector is applied to each panorama image, where two images are generated from each panorama by shifting the left image boundary by half the image width to avoid having objects across the image boundary. Object detection results are simply merged for each panorama. Second, the structure graph is rasterized into each panorama with a depth testing, while keeping track of the structural element ID. Lastly, starting from the object detection with the highest detection score, we identify the object element that has the most rasterized pixels inside the bounding-box of the object detection. The process repeats after removing the matched object detection and the element. We only keep the annotations that are related to furniture in the indoor scenes (*i.e.*, chair, lamp, table, desk, sofa, and bookshelf).

We change the rendering style and shape of object icons depending on the object recognition results. For example, we have a special icon for desk, sofa, chair, and a lamp,

respectively. The chair and the sofa have orientations, and we simply orient the icon so that the front side becomes the closest to the center of the room. Ideally, an image information should be used to determine the orientations of the icons. The outlines of the desk and the remaining (unannotated) objects are computed from the segmented 3D point clouds. After projecting the 3D points on the X-Y plane, a 2D version of the Marching Cube [4] algorithm is used to extract the polygon, followed by a mesh simplification algorithm.

5.2. Indoor scene viewer

Our indoor scene viewer enables seamless visualization across ground-to-air view-points under various rendering styles. There are a few important implementation details to be shared.

First, the texture map for the architectural components are computed for each piecewise planar surface in a single structural element independently. This is a major advantage enabled by our structure graph. Due to severe occlusions, we need to inpaint very large texture holes extensively, and likely misuse wall texture for the floor geometry with standard techniques, for example. Thanks to the compact mesh models, piecewise planar surfaces are very large in general, and we essentially compute the texture for each structural element one by one. Our approach benefits from both the texture synthesis and the texture inpainting literature. More concretely, after discretizing the piecewise planar surface into 2D array of texels, we first project nearby images to the surface with the depth testing. Due to occlusions, the projected textures have many holes. Starting from the projected texture with the least hole pixels, we assign a color to each texel, and repeat for all the projected textures. We then use a classical texture synthesis algorithm [2] to synthesize texture for the hole pixels, while fixing the colors of the non-hole pixels. Poisson-blending [5] is finally used to smooth our texture edges. Note that we also tried an off-the-shelf texture inpainting tool on PhotoShop (*i.e.*, content-aware fill) to fill holes, but their produced results were very poor.

Second, the structure graph enables us to render back-facing structures effectively. First, we can simply ignore ceiling elements, which is not easy for a polygon-soup mesh model. As shown in Figs. 3 and 4, simple back-face culling leaves numerous small triangle pieces in the views. In general, our system can render back-facing structural element as opaque, culled, or translucent. Note that this rendering style is enforced at the level of structural elements (*e.g.*, a wall), as opposed to triangles, yielding effective visualize effects without artifacts.

Lastly, the viewer enables a quick room-to-room ground-based navigation. We compute a path connecting the camera center locations by a simple shortest path algorithm with the visibility testing in real time.

5.3. Tunable reconstruction

A user specifies the maximum number of polygons allowed for an entire scene, a room, or a wall. Each number simply puts a constraint at the corresponding node in a structure graph. Our pruning algorithm is simple and greedy. Starting from the bottom of the tree, we identify either a wall, a room, or a scene node that violates this constraint. Then, we find the set of nodes that can be dropped below the node and has the fewest number of polygons. We drop the nodes and repeat the process until all the constraints are satisfied. In the worse case, when the numbers are tight, the system might drop entire rooms.

References

[1] R. Cabral and Y. Furukawa. Piecewise planar and compact floorplan reconstruction from images. In *Computer Vision and Pattern Recognition*, 2014.

[2] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE, 1999.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2015.

[4] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[5] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.

[6] Q. Shan, B. Curless, Y. Furukawa, C. Hernandez, and S. M. Seitz. Occluding contours for multi-view stereo. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 4002–4009. IEEE, 2014.

[7] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition. *Advances in Neural Information Processing Systems*, 2014.