[Supplementary] Building-GAN: Graph-Conditioned Architectural Volumetric Design Generation

Kai-Hung Chang[1] Chin-Yi Cheng[1] Jieliang Luo[1] Shingo Murata[2] Mehdi Nourbakhsh[1] Yoshito Tsuji[2]
[1]Autodesk Research, United States , [2]Obayashi AI Design Lab, Japan

1. Dataset

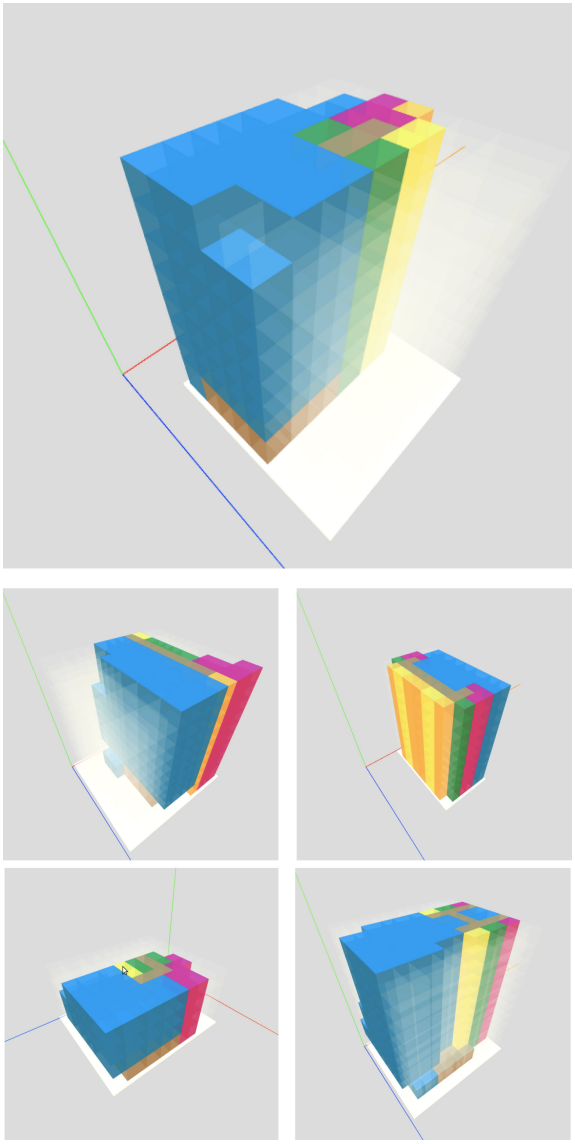

Figure 1. Volumetric design samples in the synthetic dataset

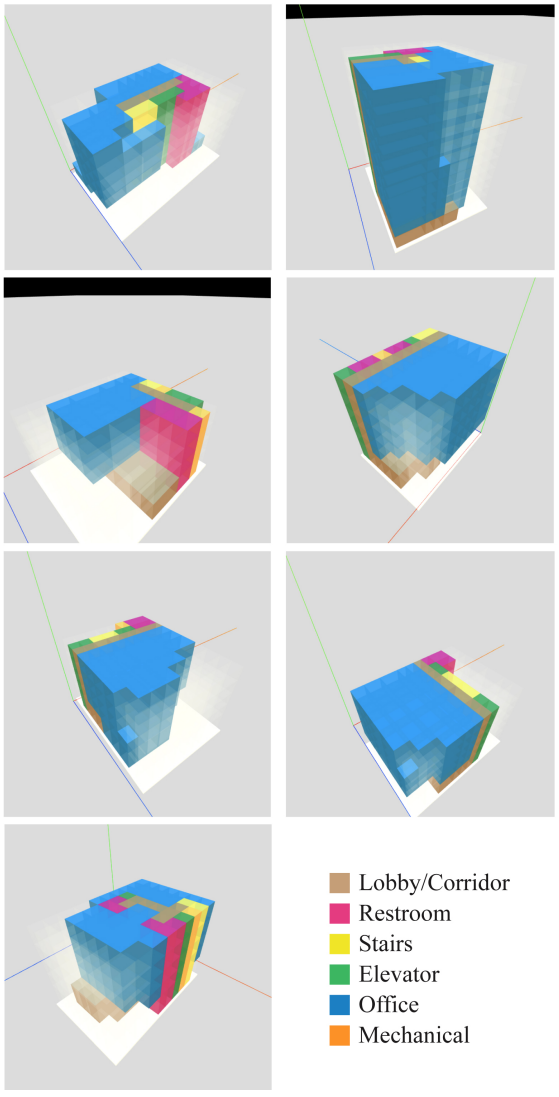

Figure 2. Continued

The synthetic dataset contains 120,000 volumetric de-

signs that are represented in voxel graph with JSON format. They can also be converted into either conventional voxels, meshes, or solid geometries. For each design, the site conditions such as the shape and maximum height are randomly generated from the distribution shown in Figure 3 and 4. The site conditions in Building-GAN are also represented in voxel graphs as part of the input conditions. Other conditions, such as the targeted TPR and FAR, and the input program graph (bubble diagram) are all restored in JSON format.

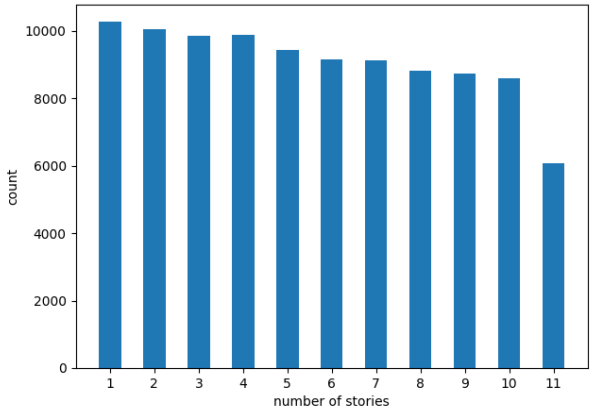

Figure 3. Number of stories distribution

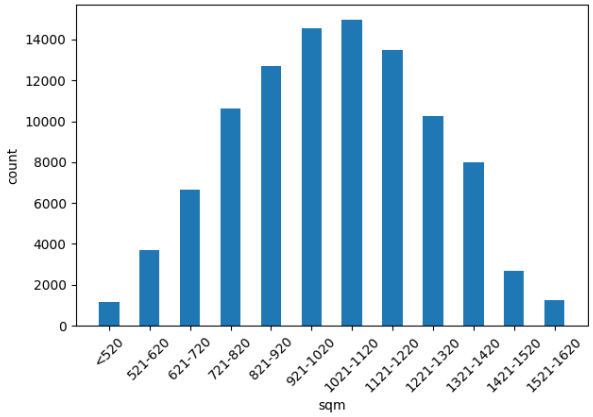

Figure 4. Site area distribution

Figure 1 and 2 present several examples in the synthetic dataset. We apply patterns and rules provided by professional architects to generate this dataset. For example, we can observe that some of the buildings have symmetric elevators facing each other (e.g., top left in Figure 1), while some corridors form left and right wings for the same set of functional spaces (e.g., bottom right in Figure 1. Also, the first story usually has a different layout than other stories due to the entrance and lobby requirements for commercial buildings. Moreover, the first story often has a higher ceiling compared to other floors.

Besides the interior functionalities, building façades, i.e., the exterior shapes of buildings, is also an important aesthetic factor to consider in the architectural design process. To increase the façade's variety in this dataset, we generate story partition patterns that divide stories into groups. For instance, the top two stories in the top left in Figure 1 have a different shape than the stories below them and therefore create a shape feature in 3D. In practice, such a shape feature can have functional purposes such as a terrace or roof garden. Please refer to Section 7 Case Study for a architect-generated building design example.

2. Implementation Detail

We implement our model in PyTorch. The latent dimension is 128 and the noise dimension (for both program and voxel graphs) is 32. The message passing steps in program and voxel GNN are 4 and 12 respectively. We summarize MLP specifications in Table 1 and the pseudo-code of our generator is given in Algorithm 1.

The 120,000 data is split to 96,000 data as training set and 24,000 data as test set. We train the models using two NVIDIA GV100 GPUs and an Intel i7 CPU with 16 cores. The total training epoch is 50 using Adam Optimizer ($b_1 = 0.5,\ b_2 = 0.999$) and batch size 8. The learning rates of the generator and the discriminator are both 0.0001. The generator updates its parameters every 5 discriminator update steps.

Algorithm 1 Generator

Input: program graph x, voxel graph v, noise z^p, z^v

1: Program GNN
$x^0 \leftarrow$ Program Graph Encoder(x, z^p)
for $t = 0$ **to** $T - 1$ **do**
  $x^{t+1} \leftarrow$ Program Graph Update(x^t)
end for

2: Voxel GNN
$v^0 \leftarrow$ Voxel Graph Encoder(v, z^v) + PE(v)
$\_, \_, v^0 \leftarrow$ Pointer(v^0, x^T)
for $t = 0$ **to** $T - 1$ **do**
  $v^{t+1} \leftarrow$ Voxel Graph Update(v^t)
  if $t < T - 1$ **then**
    $\_, \_, v^{t+1} \leftarrow$ Pointer(v^t, x^T) . . . (Optional)
  end if
end for
$mask^T, att^T, v^T \leftarrow$ Pointer(v^T, x^T)

Generator		
Program GNN	Encoder MLP	Dense(input=5+32, output=128)
	Message MLP	Dense(input=256, output=128)
	Update MLP	Dense(input=384, output=128,act=LeakyReLU)
Voxel GNN	Encoder MLP	Dense(input=4+32, output=128)
	Message MLP	Dense(input=256+3, output=128)
	Update MLP	Dense(input=256, output=128,act=LeakyReLU)
	Mask MLP	Dense(input=128, middle=64, output=2)
	Attention W_x, W_v	Dense(input=128, output=128)
Discriminator		
Voxel GNN	Feature Encoder MLP	Dense(input=4, output=128)
	Label Encoder MLP	Dense(input=6, output=128)
	Message MLP	Dense(input=512+3, output=256)
	Update MLP	Dense(input=512, output=256,act=LeakyReLU)
	Building Decoder MLP	Dense(input=256, middle=128, output=1,act=Sigmoid)
	Story Decoder MLP	Dense(input=256, middle=128, output=1,act=Sigmoid)

Table 1. Model specifications.

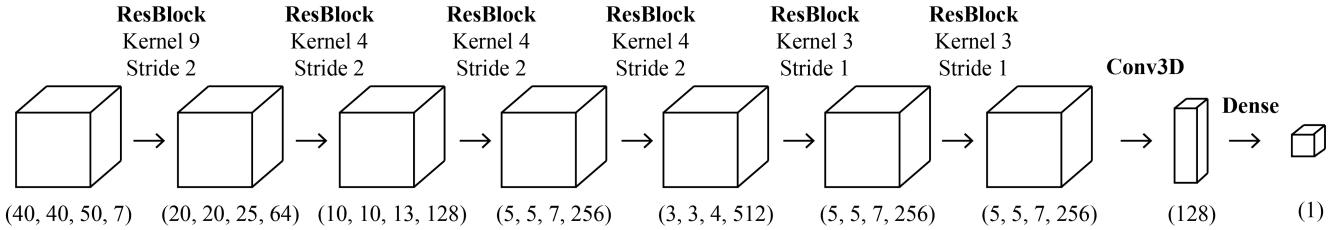

Figure 5. The network architecture of the 3D Descriptor Net used for FID computation

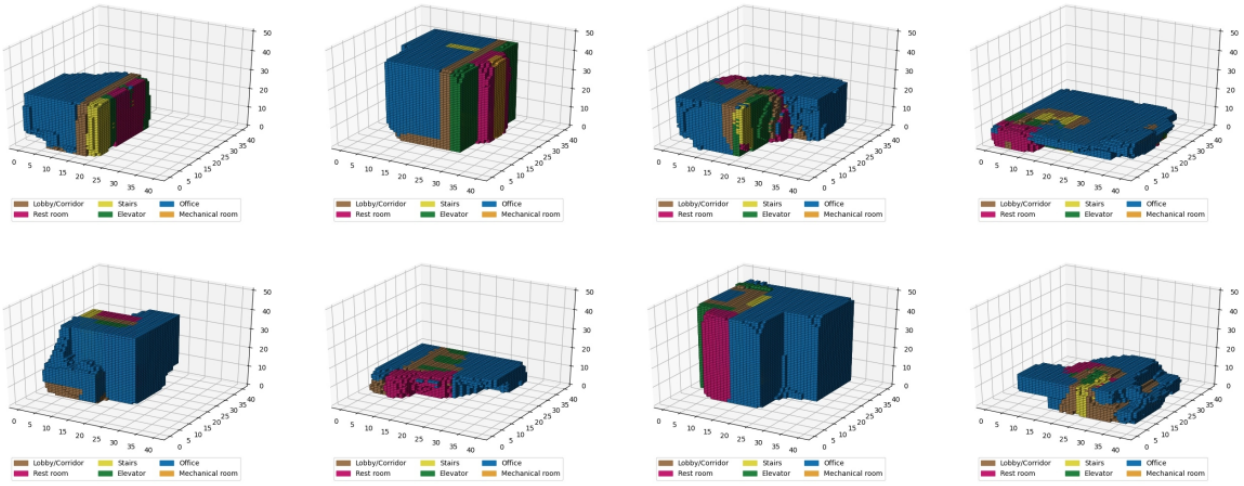

Figure 6. Synthesized examples by 3D Descriptor Net

3. 3D Descriptor Network for FID score

The backbone of the inference network used for FID computation is 3D Descriptor Net. We replace all convolution layers with 6 residual blocks due to the higher complexity of our data and insert a dense layer to convert the 128 dimension embedding to a scalar energy value. The model architecture is illustrated in Figure 5. We train the model for 400 epochs when the average mean square error decreases to 0.0121. Some synthesized volumetric designs generated unconditionally by the trained 3D Descriptor Network are visualized in Figure 6.

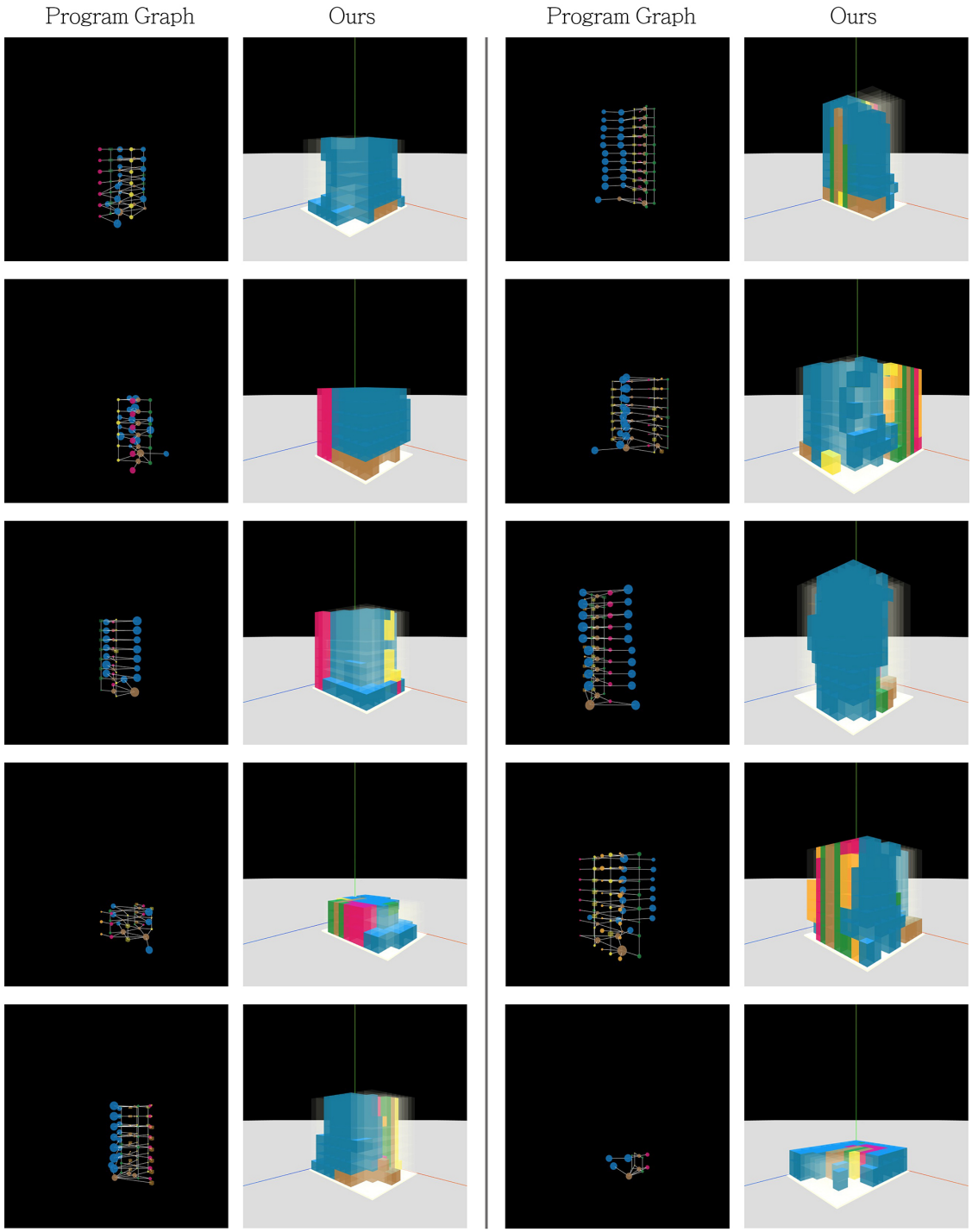

Figure 7. More results generated from Building-GAN. Left: Input Program Graph. Right: Output Volumetric Design.

4. Additional Results

4.1. Program Graph to Volumetric Design

Figure 7 to 9 shows additional results generated by Building-GAN given the program graphs (bubble diagrams).

4.2. Floor Plans from Volumetric Design

Figure 10 to 12 provide examples of floor plan layouts sliced from volumetric designs generated by Building-GAN.

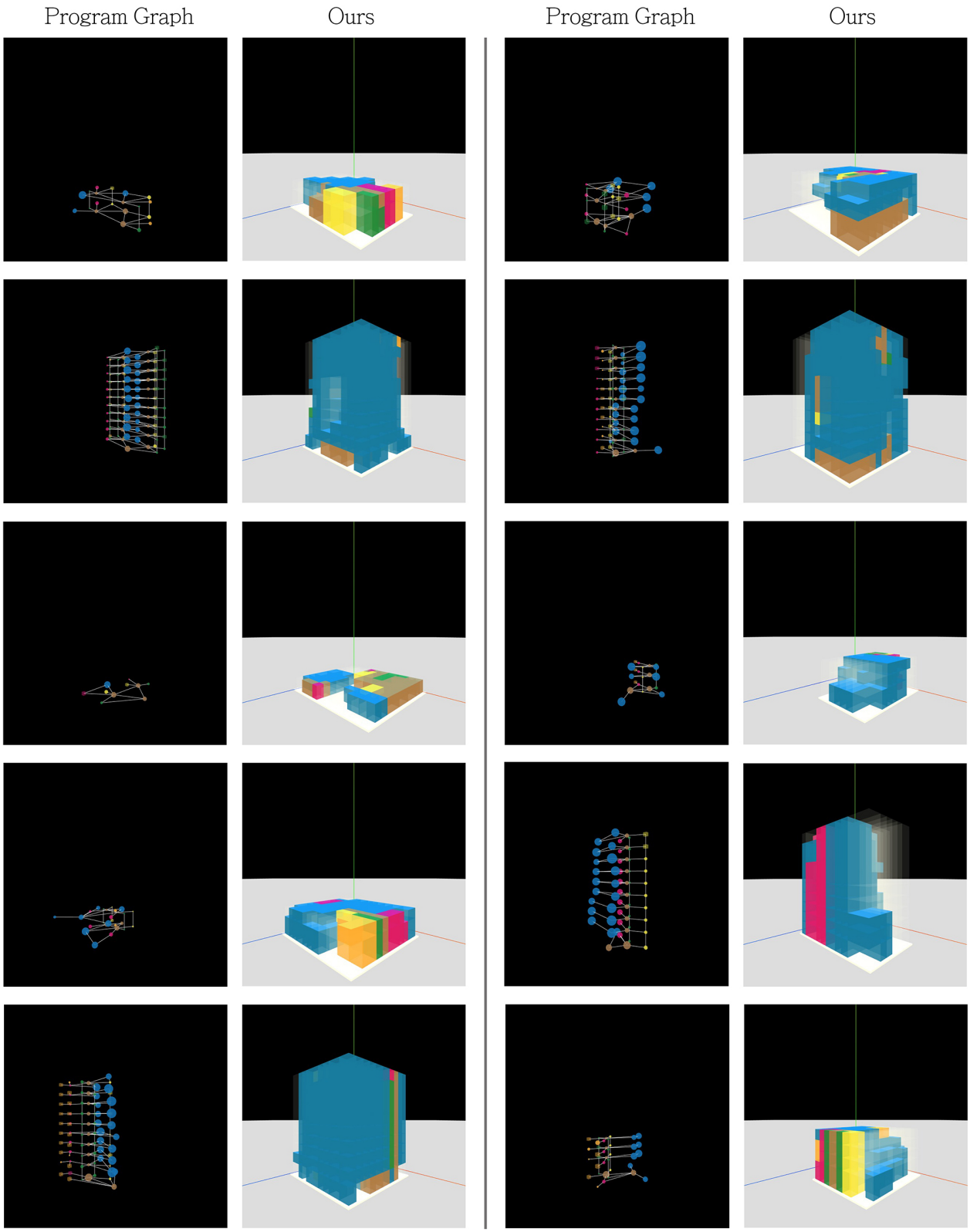

Figure 8. Continued.

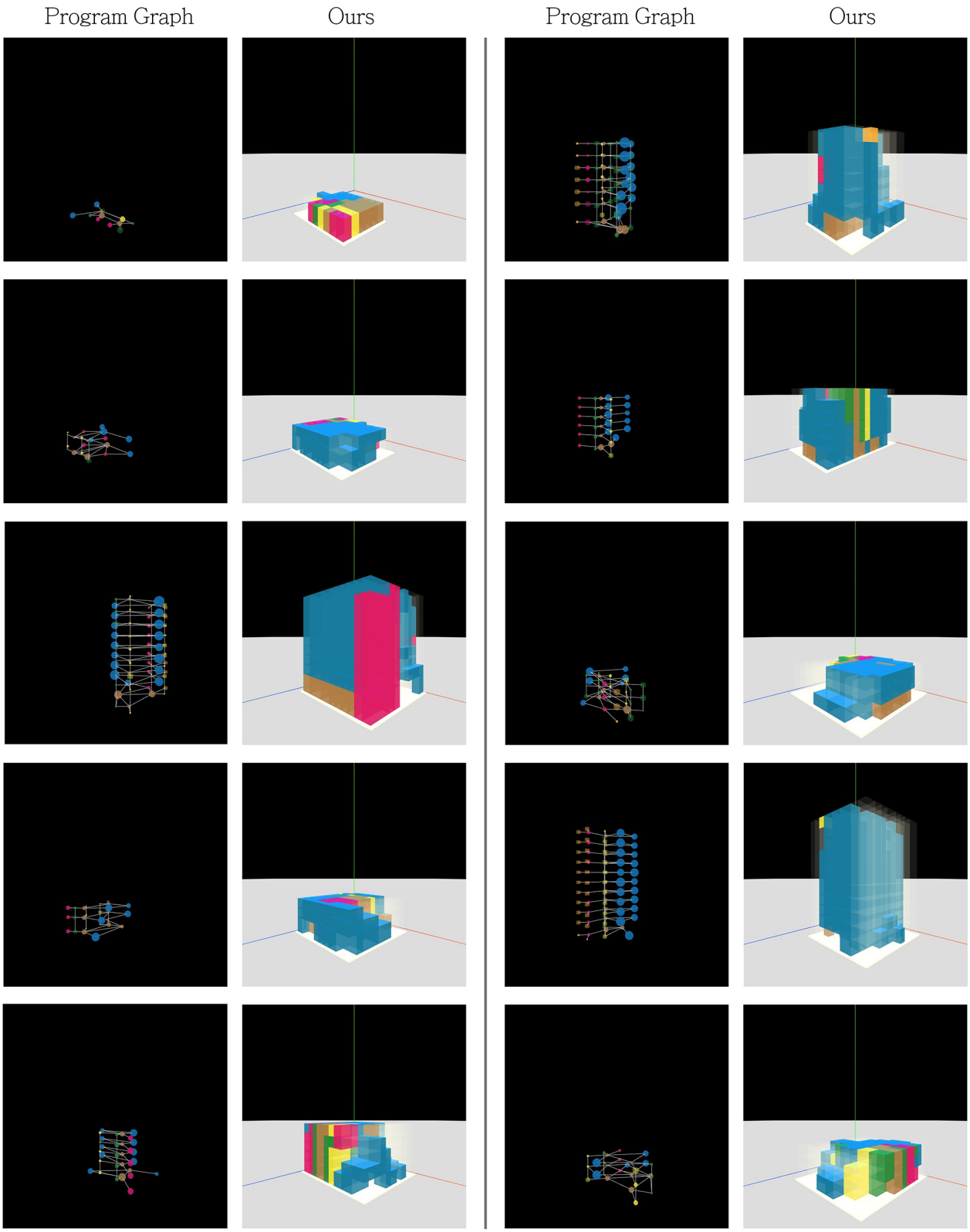

Figure 9. Continued.

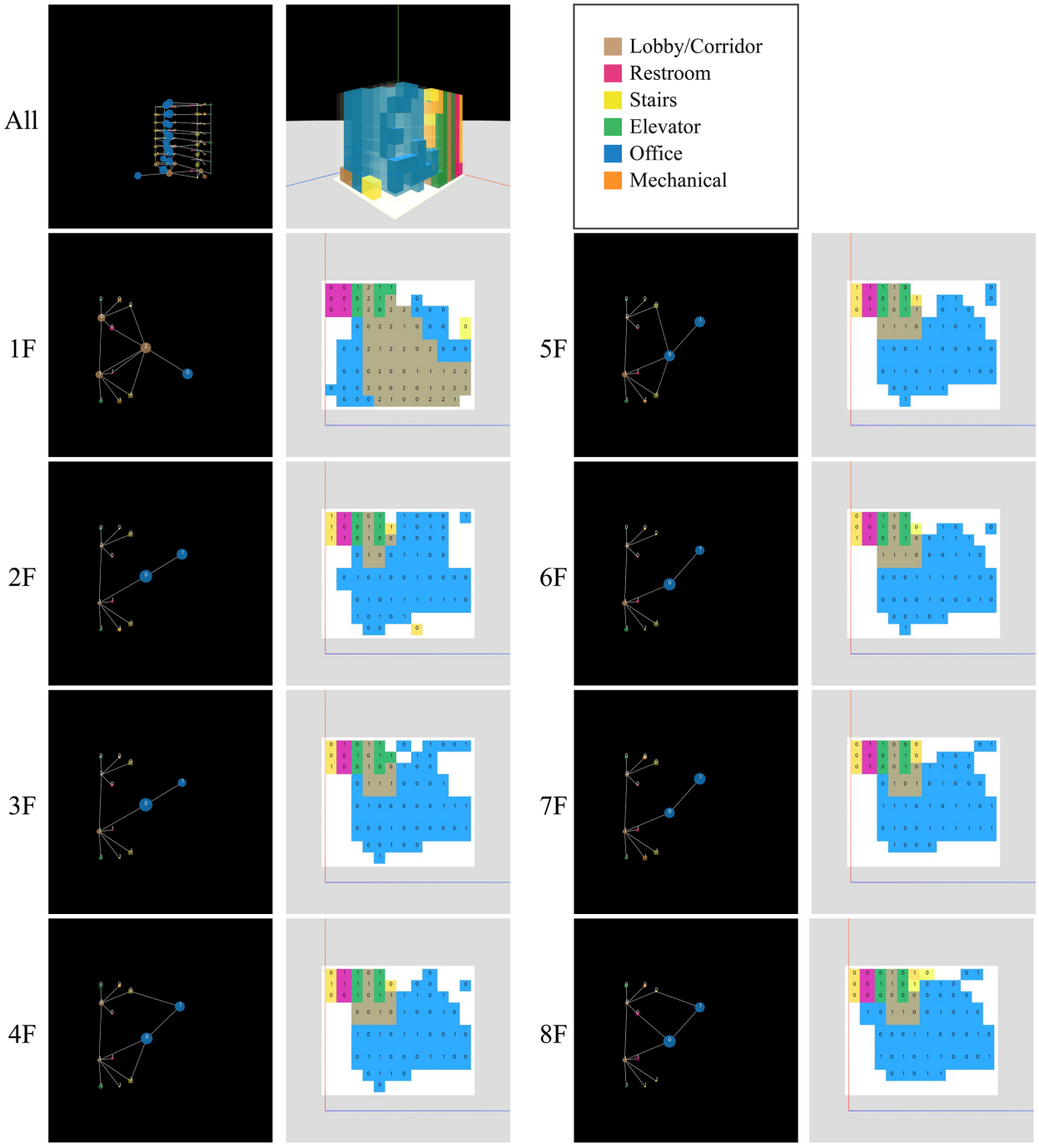

Figure 10. Floor plan layouts from volumetric design.

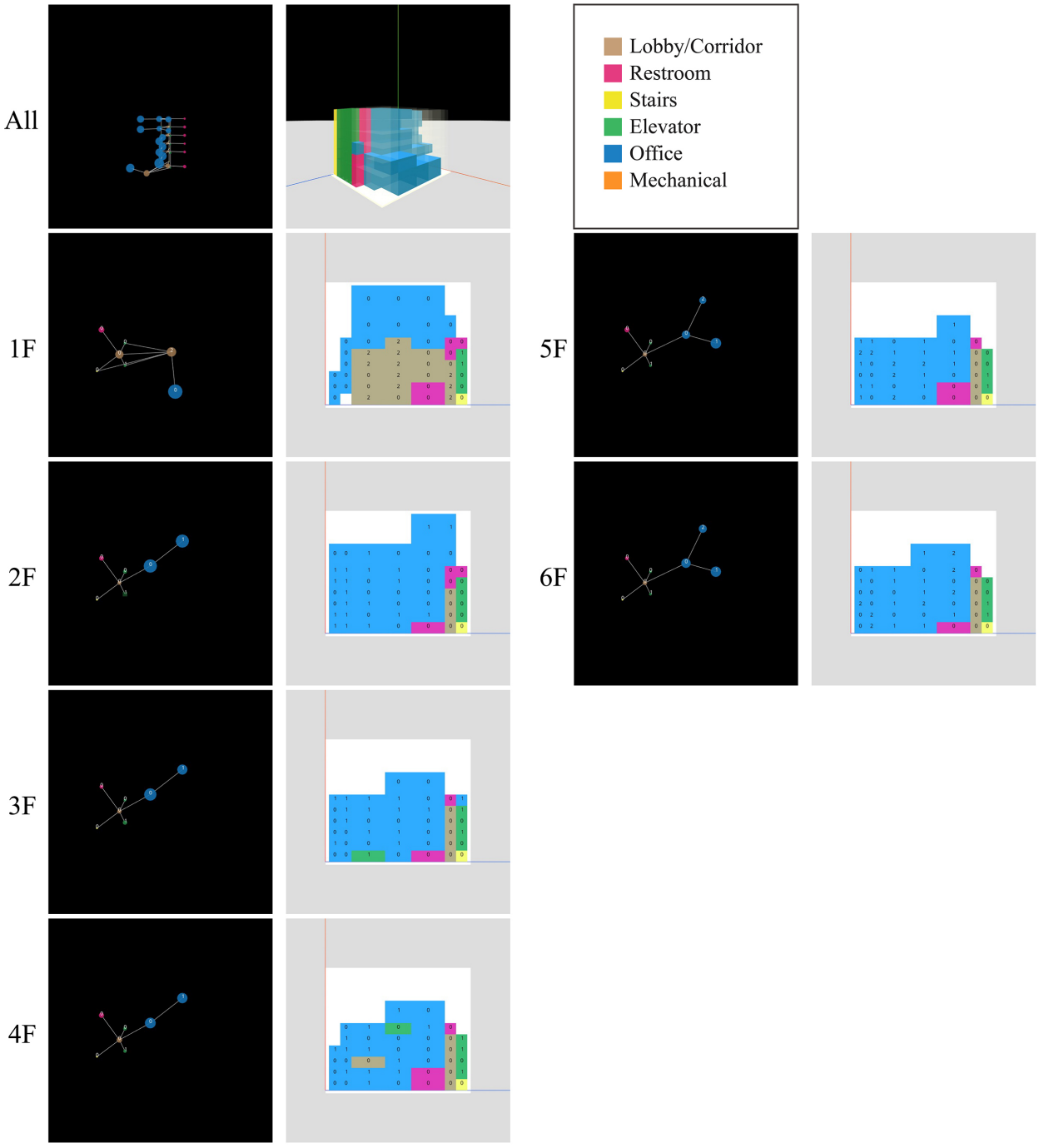

Figure 11. Continued.

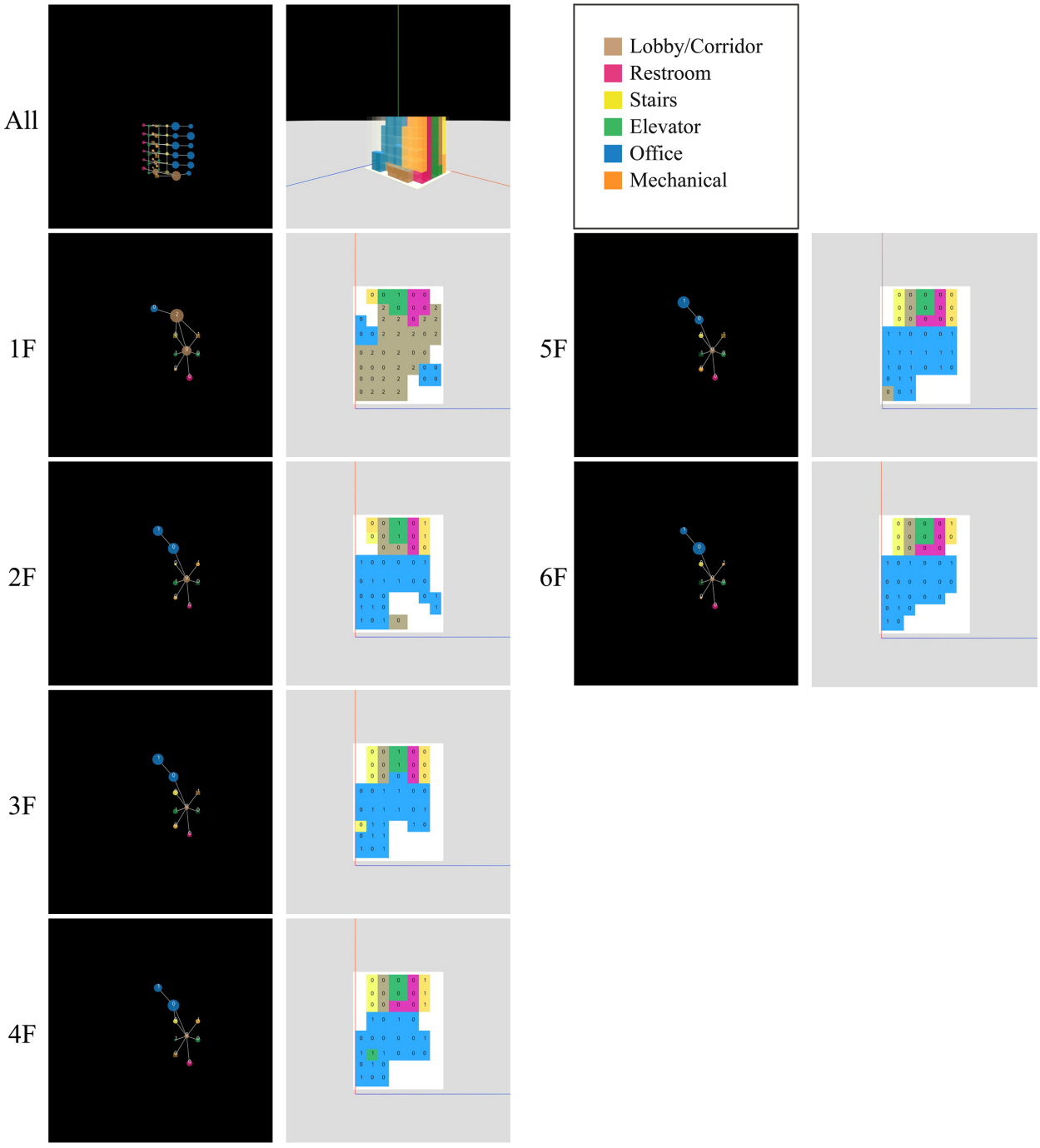

Figure 12. Continued.

5. FAR and TPR Study

We extend our comparative study and ablation study conducted in Section 5 on two more metrics: Floor Area Ratio (FAR) distance and Target Program Ratio (TPR) accuracy. The results are shown in Table 2 and Table 3 respectively. We use the difference between the actual FAR and the target FAR over the target FAR to calculate the FAR distance for each design. The TPR accuracy of each design is calculated as 1 minus the sum of the absolute difference between the actual program ratio and the target program ratio of each room type. All results are averaged over 10,000 samples.

Table 2 shows that TPR improves slightly when we increase the number of message passing layers, while FAR slightly decreases. We can also see that both FAR and TPR improve while raising the frequency for the pointer module. Table 3 shows that building discriminator helps significantly on both FAR and TPR since these two values are building-level properties.

Method	Parameter	FAR	TPR
House-GAN	-	0.853	0.528
Ours	-	1.210	0.772
Voxel Layer (Pointer Frequency = every 2 steps)	4	1.075	0.744
	6	1.159	0.759
	8	1.117	0.749
	10	1.144	0.754
	12	1.210	0.772
Pointer Frequency (Voxel Layer = 12)	first + last	2.026	0.512
	every 6 steps	1.290	0.743
	every 3 steps	1.393	0.734
	every 2 steps	1.210	0.772

Table 2. Quantitative evaluation using FAR distance and TPR accuracy. We compare our baseline model to House-GAN. We also experiment baseline models with different numbers of voxel layer and pointer frequencies.

Ablation Study	FAR	TPR
Ours	1.210	0.772
Story discriminator only	2.026	0.207
Building discriminator only	1.247	0.717
No PE	1.260	0.731
No PE + No RP	1.065	0.673

Table 3. Ablation study results of FAR distance and TPR accuracy on discriminator, positional encoding (PE), and relative position (RP).

In our best model, TPR is around 77% accurate, and FAR is around 1.2, which means about 15-20% error on each floor. Both numbers are acceptable given there are no explicit loss terms applied to these two conditions. However, we found that these two values are relatively trivial to learn compared to connectivity. First, FAR is highly dependent on the number of stories, i.e., higher buildings have higher FAR. Since the number of stories is given by the program graph, we can expect the FAR of the generated designs to have similar distribution if the model learns the overall shape distribution. Secondly, although TPR values vary in each program graph, they are following some prior design rules. For example, there should not be any building with elevators covering more than 30% of the floor area. Therefore, as long as the model learns the general type distribution on voxel graphs, TPR will not be significantly off. Based on the analysis above, we choose not to use TPR and FAR as our evaluation metrics.

6. User Study Details

Figure 13 shows a screenshot of our user study interface. A subject is presented with annotations of the room types on the top, followed by a pair of generated volumetric designs for each question. In addition, a reference page is provided to show a set of ground-truth volumetric designs, as shown in Figure 14. Each subject is given 48 questions and asked to choose one of the three possible answers ("A is better", "B is better", "Similar"), where the entire session takes around 20 minutes to be completed. The generated sample pairs come from pairs of randomly selected models among HouseGAN, Building-GAN, and ground-truth. We enforce that each possible pair of models is selected exactly 16 times during the entire session.

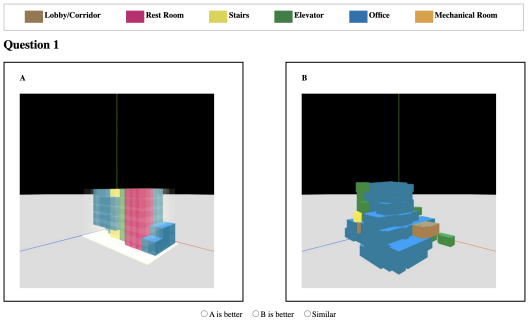

Figure 13. A screenshot of our user study interface: the annotations appear on the top, followed by a pair of generated samples for each question.

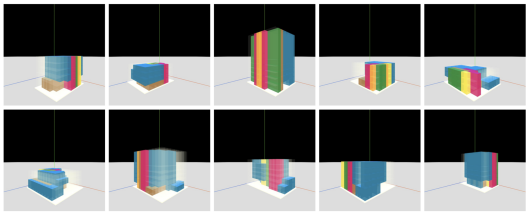

Figure 14. A screenshot of the reference page of our user study.

7. Case Study

7.1. Workflow and Setup

We invite an architect to go through the design process using Building-GAN and see if he can speed up the current workflow and create good volumetric designs. The typical workflow using conventional tools is as follow:

1. Collect the project information, including site boundary, height restriction, FAR, and the desired program requirements (TPR) from the client.
2. Create the bubble diagram and drawing the partitions or grids on the site.
3. Make 2D drawings, 3D models, or physical models using foam or foam board to create volumetric designs.
4. Arrange functional space, also called "core" in the architectural industry, including elevators, stairs, restrooms, mechanical rooms.
5. Repeat from step 2 to 4 until the design is satisfying.
6. Draw and model the interior details as well as the façade design.
7. Create drawings, renderings, and presentation slides.
8. Present to the client and repeating from any of the previous steps if the client disagrees with the current proposal.

Based on the evaluation from a professional architecture firm, this process usually takes about two to three weeks with three architectural designers. Building-GAN is designed for speeding up the loop from steps 2 to 5, where the user can quickly specify the requirements and explore the design options interactively by modifying the bubble diagram and voxel partitions. In this case study, we ask the architect to create the volumetric design using Building-GAN and complete step 6 and 7. We measure the time he spends on each step and collect feedback from him.

7.2. Results

Tasks	Time (hrs)
Draw program graph and site partitions	0.2
Explore the design options	0.25
Modify the generated volumetric design	1
Design the interior and facade	8
Choose materials and set up for rendering	2.5
Render images from different angles	3

Table 4. Time spent by the architect in each task of the case study.

As shown in Table 4, the architect uses only 10 minutes to set up the problem and 15 minutes to find the desired volumetric design. The total labor hours are reduced to less than two days. Although more adjustment and documentation work might be needed for a formal presentation to the client, the architect does feel a significant speed up on exploring valid design options while still being able to drive the design idea by himself. This is an important feedback for us–Building-GAN can help complete professional level tasks while making the architect feel their creativity is not limited nor replaced by providing real-time interaction with the system.

In addition, we found that the architect can easily modify the flawed results generated from Building-GAN. As long as the generated design is conceptually or roughly correct, meaningful, or inspiring, the architect can fix those flaws and move on to the next stage. Examples can be found in Figure 15. The rendered images of the completed building design are shown in Figure 17 and 18.

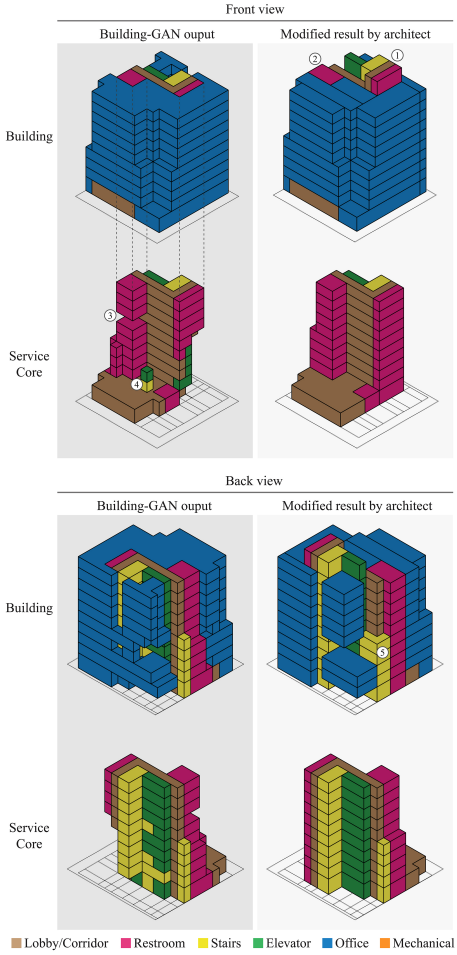

Figure 15. Comparison between the Building-GAN output and the modified volumetric design by the architect. There are five major types of modifications: 1) extending the stairs and elevators to the roof; 2) adjusting the boundaries of rooms for alignments; 3) filling the gaps; 4) removing redundant rooms; 5) expanding or shrinking the required space.

8. Failed Attempts

The failure cases shown in the paper is very likely because our discriminator only observes the program types on the voxel nodes to evaluate whether the entire design is realistic or not. Since the program graph is not taken as input, the discriminator cannot critic on missing nodes and missing edges on the program graph, nor the fragmented layouts (disconnected rooms). To resolve this issue, we experiment with aggregating the voxel node embeddings back to the program nodes. More specifically, the attention output from the generator indicates the program node selected by each voxel node, and we aggregate the voxel node embeddings that point to the same program node together as illustrated in Figure 16. The aggregated embedding is expected to provide information that tells if there's no voxel node pointing to the program node and if the voxel nodes form connected rooms. Though the attempt fails to generate promising results, we think it is helpful to share the lessons learned.

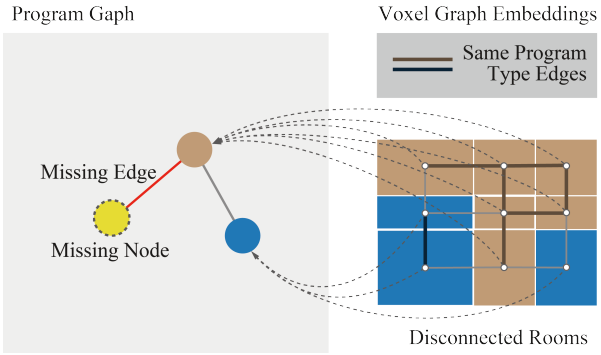

Figure 16. We tried aggregating the voxel embeddings back to program graph, but the GAN training is not stable.

We experiment with two approaches that apply this concept. First, a program discriminator can be designed by aggregating the voxel node embeddings to the program nodes. Then the program discriminator classifies each program node. Additional message passing can be added before the aggregation based on "same program type edges (shown in 16)" or after the aggregation to identify missing edges. However, the GAN training becomes unstable after adding the program discriminator. It is possibly due to the dynamic aggregation from the generated designs. This operation is also non-differentiable with respect to the attention output computed by the generator since it is used as aggregation indices. The other approach is to take the voxel embeddings from the generator and apply an explicit link prediction loss using contrastive learning. The positive and negative pairs are sampled from program edges and missing edges respectively. Cosine similarity and InfoNCE loss are used to encourage connected program nodes to have more similar aggregated embeddings. The link prediction loss is added to the GAN loss with a specific weight. Our experiments also show unstable GAN training after we add the link prediction loss. From the failed attempts, we learn that it is hard to provide stable gradients by aggregating the voxel embeddings back to program nodes. We leave this challenge for future work.

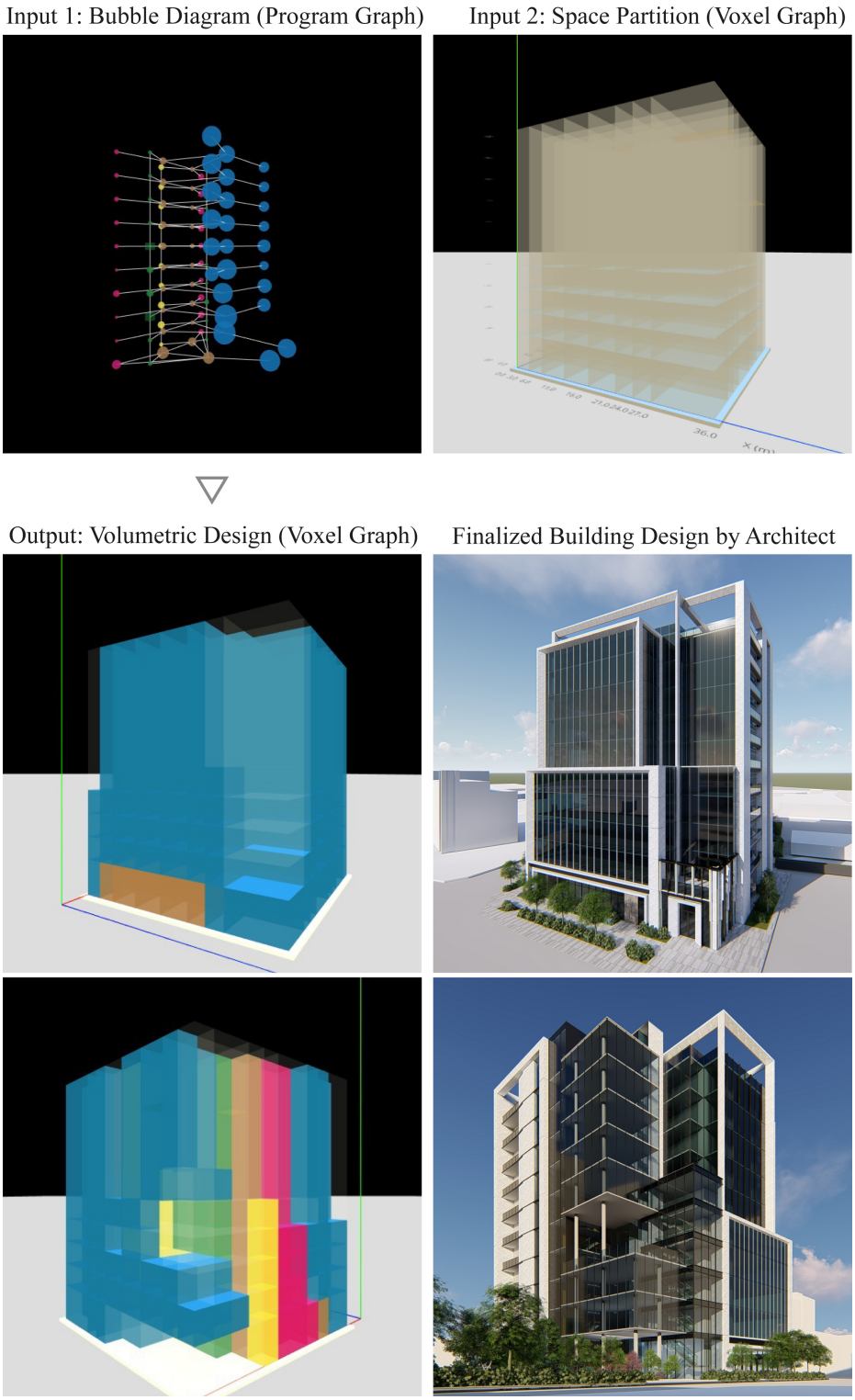

Figure 17. Results of the design process by the architect using Building-GAN.

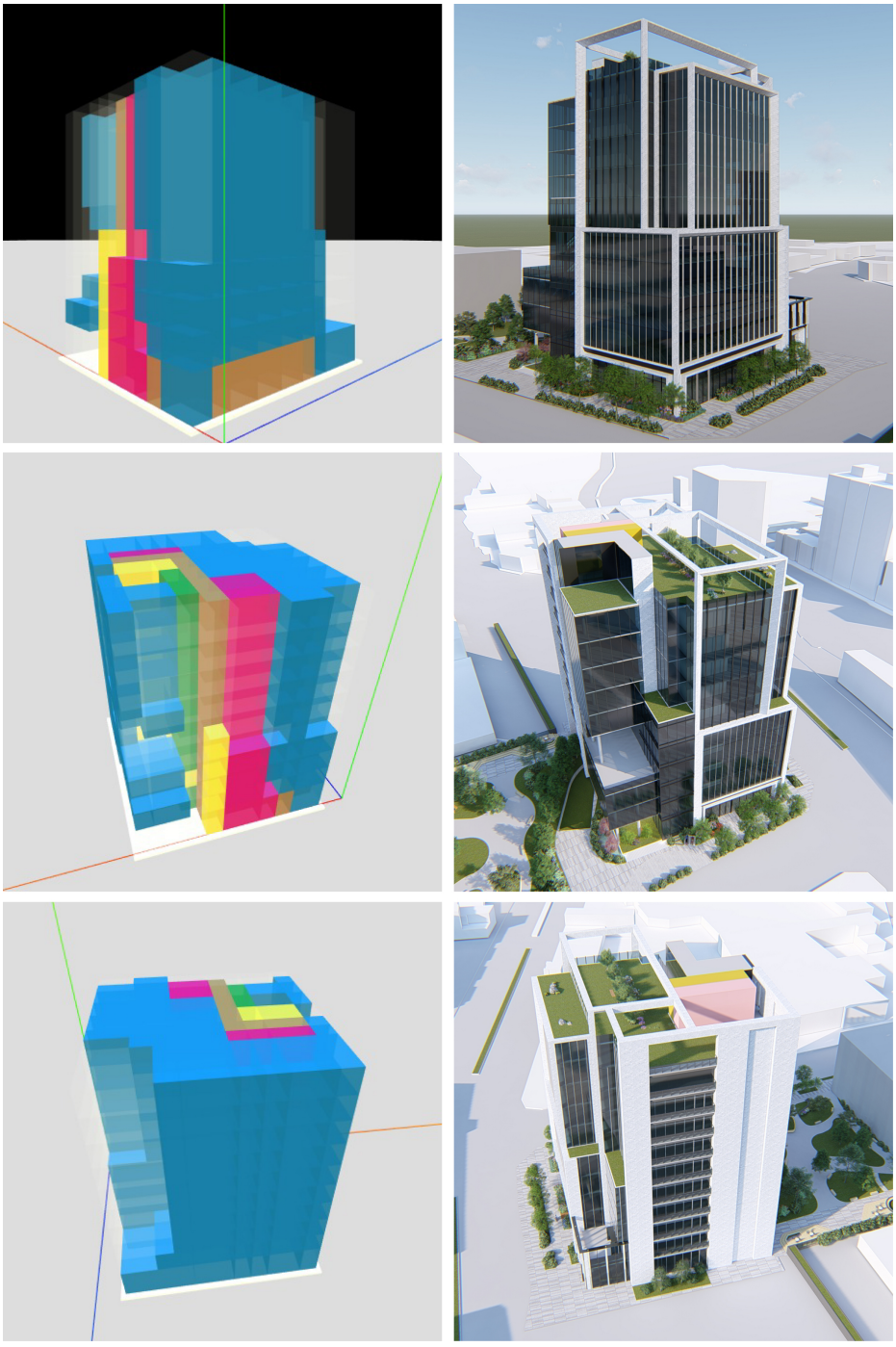

Figure 18. Continued.