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

Abstract

Volumetric design is the first and critical step for professional building design, where architects not only depict the rough 3D geometry of the building but also specify the programs to form a 2D layout on each floor. Though 2D layout generation for a single story has been widely studied, there is no developed method for multi-story buildings. This paper focuses on volumetric design generation conditioned on an input program graph. Instead of outputting dense 3D voxels, we propose a new 3D representation named voxel graph that is both compact and expressive for building geometries. Our generator is a cross-modal graph neural network that uses a pointer mechanism to connect the input program graph and the output voxel graph, and the whole pipeline is trained using the adversarial framework. The generated designs are evaluated qualitatively by a user study and quantitatively using three metrics: quality, diversity, and connectivity accuracy. We show that our model generates realistic 3D volumetric designs and outperforms previous methods and baselines.

1. Introduction

Volumetric design (also called massing design or schematic design) is the first step when an architect designs a building on a given land site. Based on the local building codes applied to the site, the building can only be designed within a valid design space, which is usually not a regular cuboid. For instance, the daylight restrictions prevent the building from casting too much shadow over its neighboring building by drawing a slant line as upper bound. Within the valid design space, a volumetric design not only depicts the volumetric 3D shape of the building, but also produces 2D program layouts for each story. An example is illustrated in Figure 2. The architect then uses the finalized volumetric design to gradually develop all the details for construction, including façade design, interior design, structure systems, etc. While volumetric design is the foundation of the design and construction process, making a good volumetric design usually requires a significant amount of time and effort. An efficient pipeline to generate volumetric design will bring a great impact on the architecture and construction industry.

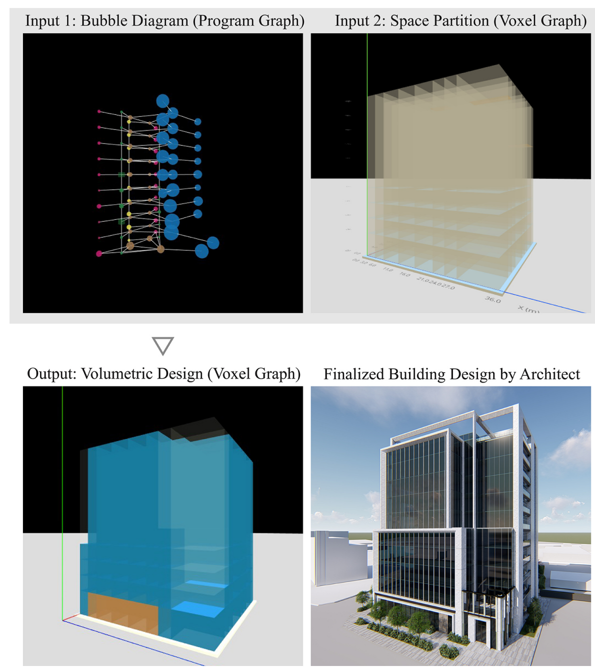

Figure 1. Our model takes in a program graph (also called bubble diagram) and a design space in voxel graph representation, and outputs a variety of volumetric designs. Professional architects can convert the output into detailed building design efficiently.

Generating realistic 2D room layouts has been a popular topic for many years. Existing methods include optimization-based [15, 1] and learning-based [30, 18, 12, 5] approaches. Recently, researchers start looking at how to integrate program graphs into layout generation tasks using graph neural networks (GNNs) [18, 12, 5]. Program graph, also called bubble diagram, is a graph that illustrates the relations between programs or rooms and is a common representation used by professional architects to explore design ideas. Similar to House-GAN [18], this paper also focuses on the graph-conditioned layout generation task. The task

*Contributed equally. chin-yi.cheng@autodesk.com

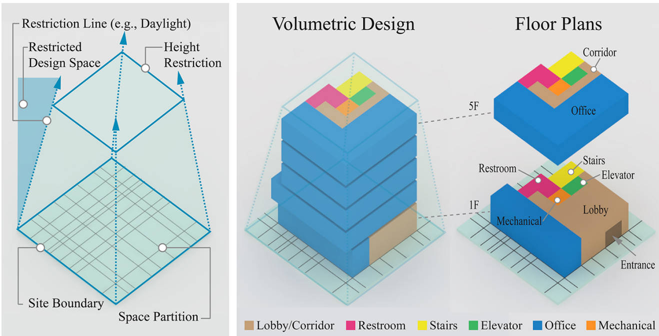

Figure 2. Left: an example of valid design space. Right: an example of volumetric design within the valid design space

requires the output layouts to be compatible to the condition input *program graphs*. However, there is no literature on extending the task to 3D. Our goal is to produce multiple layouts, which stack up and form a volumetric design for a multi-story building.

Though it might seem straight-forward to transfer previous 2D approaches to 3D, there are several challenges and limitations when applying previous approaches:

- Compared to the 2D counterparts, 3D program graphs are not only larger in size, but also more complex with additional inter-story relations. The output design space also increases by the number of stories.
- The raw rasterized output used in previous works cannot produce clean corners and edges due to the fine discretization of pixels. For instance, boundaries are usually jagged, rooms can be poorly aligned and overlapping each other, there might be small dents or bulges in some rooms, etc.
- Volumetric images (usually defined as 3D regular grids with uniformly discretized voxels) have the closest structural similarity to rectangular buildings than other 3D representations, such as point clouds or meshes. However, it is not computational and memory efficient to use this dense representation for polygonal rooms. Moreover, there are voxels within the regular grid but not in the irregular valid design space that take unneeded memory and computation.

To overcome these challenges and limitations, we propose *voxel graph*, a novel 3D representation that can encode irregular voxel grids with non-uniform space partitioning. To bridge between the input *program graph* and the output *voxel graph*, we design a pointer-based cross-modal modules in our generative adversarial graph network. The pointer module can be used not only for message passing, but also as a decoder to output probability over a dynamic set of valid programs.

We also work with professional architects to create a synthetic dataset that contains 120,000 volumetric designs based on realistic building requirements. We evaluate our model qualitatively and quantitatively, and it outperforms existing method by a large margin in all the three metrics: quality, diversity, and connectivity accuracy.

In summary, our main contributions are: 1) a new 3D representation, *voxel graph*; 2) a graph-conditioned generative adversarial network (GAN) using GNN and pointer-based cross-modal module; 3) an automated pipeline to generate valid volumetric designs through simple interaction; and 4) a synthetic dataset that contains 120,000 volumetric design and their corresponding program graphs. We will share the code, model, and dataset.

2. Related Work

2.1. Voxel Representations

Regular grid representation using voxels, such as occupancy grids, has been studied since the 3D extension of 2D convolution. To achieve 3D shape synthesis, researchers build encoder-decoder models, such as deep belief network [31], variational auto-encoder (VAE) [13], generative adversarial network (GAN) [29, 23], and energy-based model [32]. However, due to the dense representation for sparse occupancy, voxel representation is notorious for its cubic computational cost and poor scalability to higher resolutions and larger sizes. Existing methods to mitigate the problem include sparse convolution [8, 7, 4] and octree representation [21, 27, 28].

Our proposed voxel graph combines voxel-based and graph-based representations by encoding voxels into graph nodes. Similar idea was proposed in Point-Voxel CNN [14]. To enhance the local modeling capability, it has a high-resolution point-based branch as well as a low-resolution voxel-based branch for point cloud encoding. Another feature of our voxel graph is the ability to support non-uniform space partition. [16, 6, 24, 11] reconstruct 3D models by selecting space partition planes extracted from point clouds or images. BSP-Net [3] learns to generate compact meshes using binary space partitioning. NeuralSim and NeuralSizer [2] also use graphs to represent structure grids (i.e., columns and beams) of buildings instead of dense voxels.

2.2. Graph-conditioned Layout Generation

To the best of our knowledge, there is no prior works on learning-based 3D layout generation. Alternatively, we review several work on graph-conditioned 2D layout generation. Graph2Plan [12] generates bounding boxes for each room, and refines box locations with a cascaded refinement network. The input graphs are retrieved based on user constraints and outline similarity. The user can get various layouts by feeding different graphs, but the model itself can-

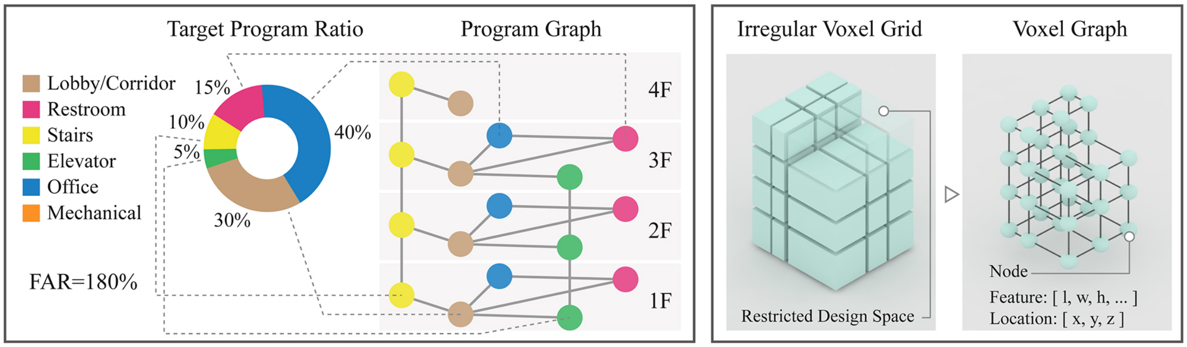

Figure 3. Left: the hierarchical program graph. Right: the irregular grid with non-uniform voxel size and the equivalent voxel graph.

not produce variation. House-GAN [18] proposes a graph-conditioned GAN, where the generator and discriminator are built upon relational architecture - ConvMPN [33]. Xinhan Di *et al.* [5] uses a similar adversarial approach on interior design with doors, windows, and furniture. LayoutGMN [20] learns to predict structural similarity between two layouts with an attention-based graph matching network. Wamiq Para *et al.* [19] explores the idea of generative modeling using constraint generation for layouts.

3. Representation and Data Collection

The goal of this paper is to generate 3D volumetric designs given a program graph and a valid design space. The program graph illustrates the intra-story and inter-story relations between programs. Besides program graph and valid design space, there are other design conditions that are considered by architects in industry practice. Floor area ratio (FAR, derived by dividing the total area of the building by the total area of the parcel), should not exceed a regulation limit. In addition, target program ratio (TPR) defines the approximate ratio between programs. For example, office : corridor : restroom : elevator : stairs $= 50 : 20 : 15 : 5 : 10$. Both TPR and FAR are encoded into the program graph as described in Section 3.2 and are used as the model input.

Another input is a valid design space, which may be irregular due to building codes. The design space can be further partitioned freely based on architect's decisions or statistical heuristics. In practice, before starting the design process, architects usually partition the space by considering construction standards, structure systems, and conventional modules. Inspired by this partitioning process, we invent the representation, *voxel graph*, as described in Section 3.3.

3.1. Data Collection

Since there is no publicly available dataset for volumetric designs from real buildings, we create a synthetic dataset with 120,000 volumetric designs for commercial buildings using parametric models. The site of each design is bounded within $40 \times 40 \times 50m^3$, where different site conditions are randomly generated. The heuristics behind the parametric models are based on the rules and knowledge provided by professional architects. Although these parametric models are able to explore possible volumetric designs, they are not capable of fitting the constraints. Therefore, we generate the designs first and then compute the voxel graph, program graph, FAR, and TPR for each design. Please refer to the supplementary for more details and visualization of the synthetic dataset. The dataset can also be used to explore other learning-based design tools or relevant tasks in computer vision and graphics.

3.2. Hierarchical Program Graph

Given a building datum, we first construct 2D program graphs for each story. Each program node feature includes the program type and the story level. Here, we consider 6 program types: lobby/corridor, restroom, stairs, elevator, office, and mechanical room. A program edge shows the two programs are connected by a door or opening. To construct the 3D program graph, we stack all 2D program graphs and chain the stairs and elevators, since they are the only paths for moving vertically. In practice, the 3D program graph also represents the circulation of the building.

Recall that there are two other design condition inputs: FAR and TPR. The FAR limit is stored as a graph-level feature. As for TPR, we add one hierarchy on top of the 3D program graph. We create one master program node for each program type and connect them to all program nodes of the same type. The edges allow the master node to allocate different area sizes on each program node through message passing. Please refer to left of Figure 3.

3.3. Voxel Graph

To overcome the challenges and limitations listed in Section 1, we invent a 3D representation called *voxel graph*. Each node represents a voxel and the voxel information (coordinate and dimension) is stored as node features. Different from volumetric images with voxel grids, voxel graph

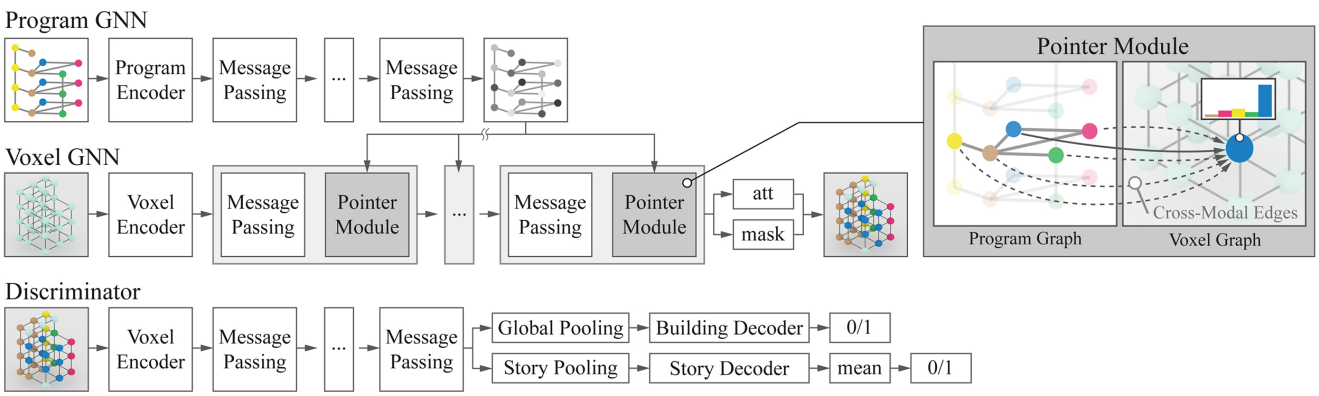

Figure 4. An overview of Building-GAN. Top: the Program GNN, Voxel GNN, and Cross-Modal Pointer Module for the generator. Bottom: the discriminator with the building and story level decoders.

does not assume regular grids and consumes memory only for occupied voxels. Moreover, it allows non-uniform space partitioning, which avoids over-discretization when using the uniform voxel size.

Theoretically, voxel nodes can encode arbitrary 3D primitives, but in this paper, only cuboids with varying sizes are used to build up the approximated valid design space. When parsing the data, the space partition is defined by the projection of all 2D layouts. In real-world practice, walls tend to align across different stories for structural stability or construction considerations, which leads to a reduced amount of voxels in the space partition. Next, we turn the voxels into graph nodes and store the voxel information (location and dimension) as node features and program type as node labels. Node mask is also stored in case of nodes that are left unused and does not have any program type. Lastly, a voxel edge connects two voxel nodes if they share a face. The final voxel graph should look like an irregular cubic lattice as illustrated in the right of Figure 3.

4. Method

We formulate the framework as a graph-conditioned GAN. The generator is composed by two GNNs for the program graph and voxel graph, connected by a cross-modal pointer module. The discriminator is composed by a GNN with two decoders to evaluate design from both building and story level. An overview of our model is illustrated in Figure 4.

4.1. Generator

4.1.1 Program GNN

Our generator starts with a program graph neural network to encode the input program graph. Denote random program noise as z^p, FAR limit as F, program node feature i as x_i, neighbor of node i as $Ne(i)$, node cluster of i's program type as $Cl(i)$, target program ratio of i's program type as $r_{Cl(i)}$, multi-layer perceptron as MLP, mean pooling as $Mean$, and concatenation operator as $[\cdot,\cdot]$. We first map the node feature to the embedding space (1), then compute message passing T times. In each message passing step, we compute the message from neighboring nodes (2) and mean pool all nodes with the same program type as the master node embedding (3). Lastly, we update the node embeddings with residual learning to avoid gradient vanishing (4). After $T = 5$ steps of message passing, the final embedding of program node i is denoted as x_i^T.

$$x_i^0 = MLP^p_{enc}([x_i, z_i^p, F]) \tag{1}$$

$$m_i^t = \frac{1}{|Ne(i)|}\sum_{j\in Ne(i)} MLP^p_{message}([x_i^t, x_j^t]) \tag{2}$$

$$c_i^t = Mean_{j\in Cl(i)}(\{x_j^t\}) \tag{3}$$

$$x_i^{t+1} = x_i^t + MLP^p_{update}([x_i^t, m_i^t, r_{Cl(i)}c_i^t, F]) \tag{4}$$

4.1.2 Voxel GNN

The input voxel features v_k and voxel noise z_k^v are first encoded by the voxel GNN encoder. To better encode the story index, we choose positional encoding (PE) as proposed in [25] and add it to the processed embedding (5). Instead of appending the absolute coordinates in voxel features, we use the relative displacements $p_k - p_l$ in message computation (6). Voxel node embeddings are updated with residual learning (7).

$$v_k^0 = MLP^v_{enc}([v_k, z_k^v]) + PE(story_k) \tag{5}$$

$$n_k^t = \sum_{l\in Ne(k)} MLP^v_{message}([v_k^t, v_l^t, p_k - pl]) \tag{6}$$

$$v_k^t = v_k^t + MLP^v_{update}(v_k^t, n_k^t) \tag{7}$$

4.1.3 Pointer-based Cross-Modal Module

After processing the program graph with the program GNN, the final embedding of program nodes can be viewed as the virtual "blueprint" of a design. Therefore, it is necessary to "look" at this blueprint to generate the output. To bridge between the program graph and the voxel graph, we introduce a pointer-based cross-modal module. Inspired by the application [22, 17] of the Pointer Network [26] in natural language processing and mesh generation tasks, we construct a pointer module to achieve message passing between the voxel nodes and all the program nodes on the same story. We cannot use a fixed length output to model program type distribution since 1) different stories can have different numbers of program nodes to choose from, for example, one floor has five rooms and another one has seven rooms; and 2) if there are two program nodes with the same program type, we want to differentiate between the two nodes, such as two restrooms in the same floor.

The pointer module returns three terms: $mask_k$, att_k, and v_k^{t+1} (8). $mask_k$ is used as a soft prediction whether the voxel node k is used or not (9). If it is not used, it is left unused and has no program type. Otherwise, att_k is the attention distribution over the set of program nodes on the same floor (10, 11). An updated embedding v_k^{t+1} is computed by the weighted sum of the program embeddings x_i^T multiplied by the soft prediction $mask_k$ with residual learning (12).

$$mask_k, att_k, v_k^{t+1} = Pointer(v_k^t, \{x_i^T\}) \tag{8}$$

$$mask_k = \sigma(MLP(v_k^t)) \tag{9}$$

$$e_{k,i} = \theta^T \tanh(W_x x_i^T + W_v v_k^t) \tag{10}$$

$$att_k = \text{gumbel softmax}(e_k) \tag{11}$$

$$v_k^{t+1} = v_k^t + mask_k \sum_i att_{k,i} x_i^T \tag{12}$$

We experiment different ways to integrate the pointer module. It can be placed after every several message passing steps in voxel GNN. Our baseline model uses 12 steps of message passing and call the pointer module once every 2 steps. Please refer to the supplementary for the complete model and algorithm. Conceptually, these pointer modules should gradually improve the design. Note that the output att_k indicates which program node is associated to the program type of the voxel node, instead of merely the program type prediction.

4.2. Discriminator

Our discriminator is trained to distinguish if a given design is generated by the generator or sampled from the dataset. Therefore, we take a similar architecture as voxel GNN, but without using the pointer modules. The program type predictions are concatenated to the encoded voxel node features. After $T = 12$ message passing steps, two separate decoders are used. A graph-level max-pooling decoder evaluates the design as a whole while a story-level max-pooling decoder evaluates the per-story layouts individually.

$$o^{global} = MLP_{global}^{dec}(\sum_k v_k^T) \tag{13}$$

$$o^{story} = Mean_{story\ s}(MLP_{story}^{dec}(\sum_{k \in s} v_k^T)) \tag{14}$$

4.3. Loss

We use the WGAN-GP [9] loss with gradient penalty set to 10. The two decoder outputs from the discriminator are equally weighted. The gradient penalty is computed by linearly interpolating the cross-modal attention between real data and generated output, while fixing the voxel graph connectivity.

4.4. Evaluation Metric

We evaluate the generated design in terms of quality, diversity, and connectivity accuracy. The quality and diversity of the output design is evaluated with the Fréchet Inception Distance (FID) score [10]. FID score has demonstrated high correlation to human judgement and has been widely used in many 2D and 3D studies. Our reference model is based on a larger version of 3D Descriptor Net [32]. We replace all convolution layers with 6 residual blocks due to the higher complexity of our data. Then we flatten the embedding to a 128-dimension tensor using convolution operation and pass it to a dense layer for loss computation. The FID score is measured over 10,000 samples. We also run a user study with architects to measure the quality in Section 5.5.

The connectivity accuracy (Con.) is measured by the number of the program (room) connections observed from both the generated design and in the program graph, divided by the amount of all edges in the program graph. Note that only when two rooms are connected in the program graph but disconnected in the voxel graph, it is considered as inaccurate, since there is no shared wall to put a door. It is accurate when two rooms are connected in voxel graph but disconnected in program graph, because designers can decide not to put a door on the shared walls.

For more details about model implementation, hyper parameters, training environment, and user study, please refer to the supplementary.

5. Experiments

5.1. Baseline Comparison and Visualization

We compare our model to a slightly modified version of House-GAN [18]. A major difference between our model

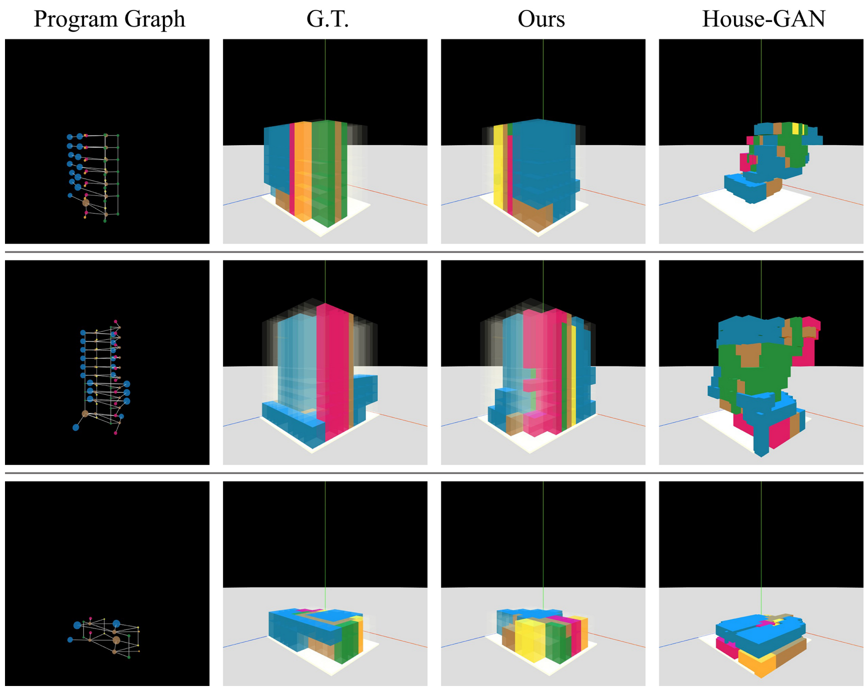

Figure 5. For each program graph, volumetric designs are generated by our model and by House-GAN [18].

and House-GAN is that House-GAN directly generates layout masks of size 40×40 on the nodes of the program graph. Since House-GAN does not use voxel graph representation, it assumes that the valid design space is a regular grid. House-GAN discriminator places the generated masks back to the program nodes as features and determines if the program graph is valid. To extend the House-GAN to 3D, we append the story index of each program node to its feature.

Figure 5 shows example designs from ground truth, our model, and House-GAN. Our model shows capability of generating designs that have realistic 3D shapes and clean facade surfaces. We also observe that the number of jagged boundaries are reduced due to the usage of voxel graph representation. In addition, the sliced layouts of individual stories are reasonable. For instance, the functional programs such as elevators, stairs, and restrooms are arranged as clusters and connected by corridors. Last but not least, the functional programs are nicely aligned in the vertical direction. In contrast, though House-GAN seems to generate reasonable layouts story-by-story, they don't align well when stacked vertically. Quantitative results are presented in Table 1. Our model outperforms House-GAN in both FID and connectivity accuracy.

Method	Parameter	FID	Con.
House-GAN	-	17.6003	0.403
Ours	-	0.0845	0.569
House-GAN (sliced)	-	52.256	0.612
Ours (sliced)	-	17.479	0.536
Voxel Layer (Pointer Frequency = every 2 steps)	4	1.0463	0.432
	6	0.4139	0.501
	8	0.2365	0.497
	10	0.0997	0.534
	12	0.0845	0.569
Pointer Frequency (Voxel Layer = 12)	first + last	3.3818	0.578
	every 6 steps	0.2179	0.547
	every 3 steps	0.1473	0.541
	every 2 steps	0.0845	0.569

Table 1. Quantitative evaluation using FID score and connectivity accuracy. We compare our baseline model to House-GAN and experiment baseline models with different numbers of voxel layer and pointer frequencies. We also evaluate sliced layouts from generated designs of both models.

In Table 1, we also compare models with different hyperparameter set-ups. First, we fix the frequency of applying the pointer module to 2 and experiment different numbers of message passing layers in voxel GNN. The result shows that using 12 voxel layers yields best performance in both FID and connectivity accuracy. This is not surprising: using more voxel layers allows computing longer-range relations between voxel nodes, which is especially critical for achieving vertical alignment in taller buildings. Next, we fix the number of voxel layer and evaluate the impact of different pointer frequency. The model (first + last) which uses the pointer module only before and after message passing fails

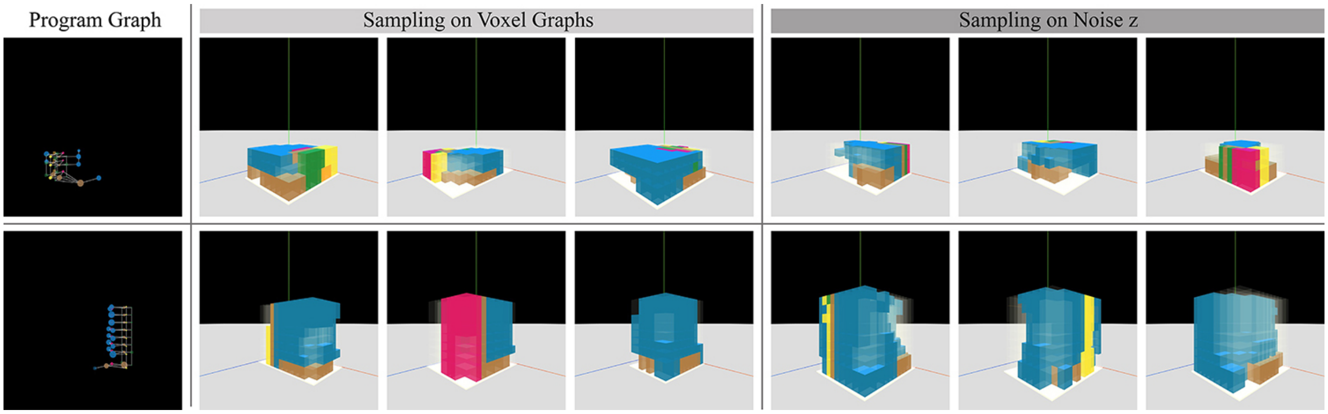

Figure 6. Design variations generated by fixing the program graph while changing the voxel graph and noise.

to converge. We also run the pointer module every 2, 3, 6 message passing steps in voxel GNN. Using the pointer module every 2 message passing steps yields the best performance in both FID and connectivity accuracy. Reviewing the program graph multiple times during the message passing process might ensure that the information from the program graph is always considered and provide shorter paths for gradient back propagation.

5.2. Variation Study

In Figure 6, we visualize examples generated by fixing the program graph while changing the space partition in the voxel graph and noise. The model is able to generate different designs with different patterns, orientations, etc. based on the given noise and design space partition.

5.3. Ablation Study

We run ablation studies on discriminator, positional encoding, and relative position. The results are summarized in Table 2. We found that it is necessary to use both story discriminator and building discriminator as using only one leads to inferior performance. Though having message-passing in the voxel GNN, story discriminator has difficulty evaluating inter-story relations and the overall 3D geometry. The better connectivity accuracy with * actually results from noisy low-quality outputs where the layouts are fragmented. Building discriminator proves to play a more crucial role in learning the task, but adding story discriminator significantly improves the output design quality. We also show that using positional encoding (PE) to encode story indices performs better than directly using it (i.e. $1, 2 \ldots$). Lastly, training without relative position (RP) ends up generating low quality designs. It shows that relative position is an indispensable component for our model, since it can capture the direct spatial relationships between connected voxels.

Ablation Study	FID	Con.
Ours	0.0845	0.569
Story discriminator only	6.8061	*0.777
Building discriminator only	1.0464	0.459
No PE	0.1512	0.507
No PE + No RP	0.8333	0.489

Table 2. Ablation study results on discriminator, positional encoding (PE), and relative position (RP). * The higher accuracy here is caused by fragmented low-quality outputs.

5.4. Intermediate Results

In voxel GNN, we run 12 message passing layers and use the pointer module every 2 layers. Since every mask and attention computed by the pointer modules represents a design solution, we are curious to see the ”design process” of our model by visualizing the intermediate designs during inference. As shown in Figure 7, before the voxel message passing, the first attention initializes a seemly random design, trying to allocate only the lobby/corridor type. It makes sense since the decision is only based on the program graph and individual voxel nodes. Interestingly, starting from the second attention, the model chooses to start over and gradually grow the voxels. This behavior aligns with the message passing process since the information from a far distance will flow in with more passing steps. The model also tries to refine the design by overwriting some of the decisions made in previous steps. For example, in the first row of Figure 7, at layer 6, the isolated restroom (in magenta color) is eliminated at layer 8.

5.5. User Study

To further examine the quality, we conduct user study with 20 professional architects. Each architect is given 48 design pairs that cover all the combinations of the ground truth, House-GAN, and our model. Given a pair of designs,

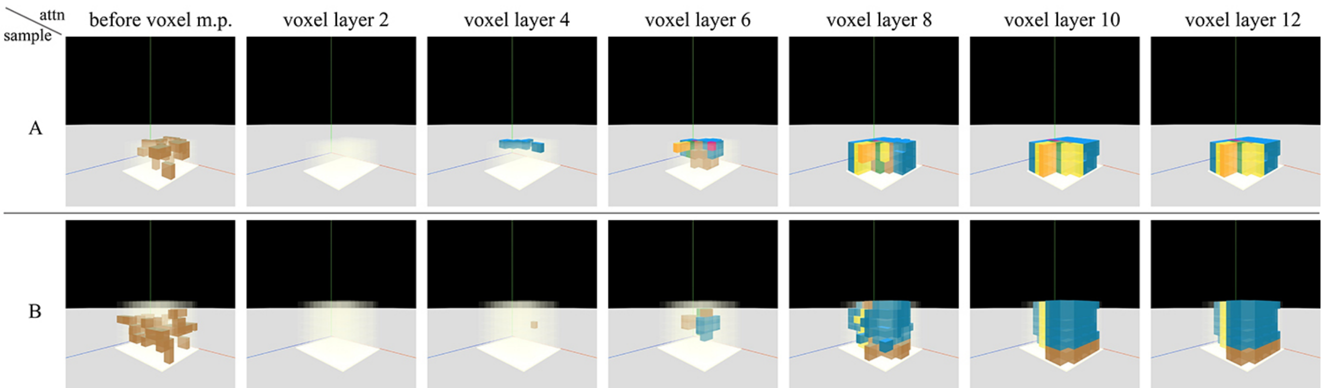

Figure 7. Visualization of designs generated by all pointer modules for every 2 message passing steps in Voxel GNN during inference.

	H.G.	Ours	G.T.
H.G.		-0.85	-0.92
Ours	0.85		-0.37
G.T.	0.92	0.37	

Figure 8. The pairwise quality scores between ground truth (G.T.), our model, and House-GAN(H.G.).

the better design gets 1 point while the worse one gets -1 point. If it's a tie, both get 0 scores. The average score of a method should range between 1 and -1. The results are shown in Figure 8 and it should be read row-by-row. Our model and ground truth defeats House-GAN with scores 0.85 and 0.92 respectively. The ground truth score is only 0.37 when compared to our model, which means in many cases architects cannot clearly tell the difference between the ground truth and ours.

5.6. Case Study

To understand if our pipeline can be useful to the professional building design process, we invite an architect to create the volumetric design using our pipeline and then complete a detailed building design. As shown in the Figure 1, the results are realistic and aesthetically appealing. The user does feel the pipeline largely increased the efficiency of the design process. For the creation process and detailed feedback, please refer to the supplementary.

5.7. Failure Case

We observe three types of common flaws in the volumetric designs generated from our model: 1) missing nodes; 2) missing edges; and 3) disconnected rooms, as visualized in Figure 9. One potential cause of these flaws is that our discriminator only considers the program type on each voxel node instead of the attention between voxel nodes and program graph. Therefore, the discriminator lacks information regarding the specific program nodes which the voxel nodes point to. Some of our failed attempts to resolve these flaws are introduced in the supplementary material and we leave the solution for future work.

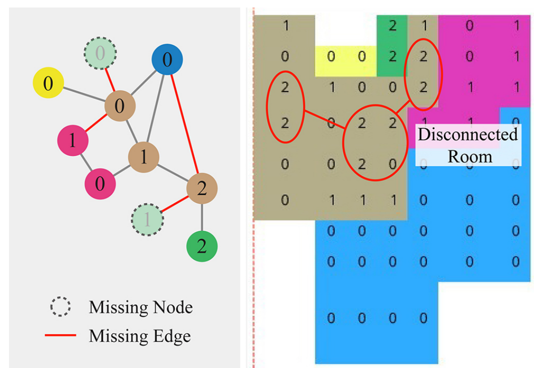

Figure 9. Visualization of the three common flaws in generated volumetric designs. Left: the input program graph. Right: a floor plan in the generated volumetric design given this program graph

6. Conclusion

In this paper, we try to provide a novel pipeline, Building-GAN, to improve the efficiency on a realistic professional task, volumetric design in the architectural and construction industry. We invent a 3D representation, voxel graph, to represent building designs, and design a generator with a cross-modal pointer module to connect the program graph and voxel graph. Our extensive evaluations, including user testing and user study, show that architects can create numerous valid and valuable designs by interacting with Building-GAN. Future works include enforcing the constraints, such as connectivity, TPR, and FAR, as well as extending the voxel graph for non-cuboid geometries. We will release our code, model, and dataset, and invite the research community to work together on design-related problems in the industries.

References

[1] Fan Bao, Dong-Ming Yan, Niloy J. Mitra, and Peter Wonka. Generating and exploring good building layouts. *ACM Transactions on Graphics*, 32(4), 2013.

[2] Kai-Hung Chang and Chin-Yi Cheng. Learning to simulate and design for structural engineering. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1426–1436, 2020.

[3] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 45–54, 2020.

[4] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[5] Xinhan Di, Pengqian Yu, Danfeng Yang, Hong Zhu, Changyu Sun, and YinDong Liu. End-to-end generative floor-plan and layout with attributes and relation graph. *arXiv preprint arXiv:2012.08514*, 2020.

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

[7] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9224–9232, 2018.

[8] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[10] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[11] Henry Howard-Jenkins, Shuda Li, and V. Prisacariu. Thinking outside the box: Generation of unconstrained 3d room layouts. In *ACCV*, 2018.

[12] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *arXiv preprint arXiv:2004.13204*, 2020.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32:965–975, 2019.

[15] Paul Merrell, Eric Schkufza, and Vladlen Koltun. Computer-generated residential building layouts. In *ACM SIGGRAPH Asia 2010 papers*, pages 1–12. 2010.

[16] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2353–2361, 2017.

[17] Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International Conference on Machine Learning*, pages 7220–7229. PMLR, 2020.

[18] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. pages 162–177, 2020.

[19] Wamiq Para, Paul Guerrero, Tom Kelly, Leonidas Guibas, and Peter Wonka. Generative layout modeling using constraint graphs. *arXiv preprint arXiv:2011.13417*, 2020.

[20] Akshay Gadi Patil, Manyi Li, Matthew Fisher, Manolis Savva, and Hao Zhang. Layoutgmn: Neural graph matching for structural layout similarity. *arXiv preprint arXiv:2012.06547*, 2020.

[21] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[22] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[23] Edward Smith and David Meger. Improved adversarial systems for 3d object generation and reconstruction. *arXiv preprint arXiv:1707.09557*, 2017.

[24] Sinisa Stekovic, F. Fraundorfer, and V. Lepetit. General 3d room layout from a single view by render-and-compare. In *ECCV*, 2020.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[26] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28:2692–2700, 2015.

[27] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[28] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[29] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29:82–90, 2016.

[30] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019.

[31] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[32] Jianwen Xie, Zilong Zheng, Ruiqi Gao, Wenguan Wang, Song-Chun Zhu, and Ying Nian Wu. Learning descriptor networks for 3d shape synthesis and analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8629–8638, 2018.

[33] Fuyang Zhang, Nelson Nauata, and Yasutaka Furukawa. Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2798–2807, 2020.