WallPlan: Synthesizing Floorplans by Learning to Generate Wall Graphs

JIAHUI SUN, Hefei University of Technology, China
WENMING WU*, Hefei University of Technology, China
LIGANG LIU, University of Science and Technology of China, China
WENJIE MIN, Hefei University of Technology, China
GAOFENG ZHANG, Hefei University of Technology, China
LIPING ZHENG*, Hefei University of Technology, China

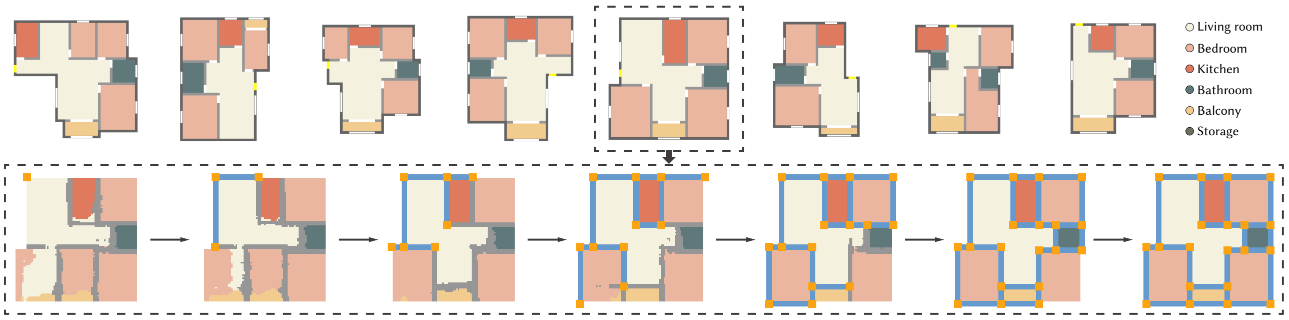


Fig. 1. We propose to synthesize floorplans by learning to generate wall graphs using deep neural networks. Top row: floorplans generated by our method with various input boundaries. Bottom row: our method generates the graph with wall junctions (in orange) as nodes and wall segments (in blue) as edges in an iterative manner for the example outlined in the dotted box. Room labels are represented in different colors, as shown in the upper-right.

Floorplan generation has drawn widespread interest in the community. Recent learning-based methods for generating realistic floorplans have made significant progress while a complex heuristic post-processing is still necessary to obtain desired results. In this paper, we propose a novel wall-oriented method, called *WallPlan*, for automatically and efficiently generating plausible floorplans from various design constraints. We pioneer the representation of the floorplan as a wall graph with room labels and consider the floorplan generation as a graph generation. Given the boundary as input, we first initialize the boundary with windows predicted by WinNet. Then a graph generation network GraphNet and semantics prediction network LabelNet are coupled to generate the wall graph progressively by imitating graph traversal. *WallPlan* can be applied for practical architectural designs, especially the wall-based constraints. We conduct ablation experiments, qualitative evaluations, quantitative comparisons, and perceptual studies to evaluate our method's feasibility, efficacy, and versatility. Intensive experiments demonstrate our method requires no post-processing, producing higher quality floorplans than state-of-the-art techniques.

*The corresponding authors

Authors' addresses: Jiahui Sun, Hefei University of Technology, China, jeffrey@mail.hfut.edu.cn; Wenming Wu, Hefei University of Technology, China, wwming@hfut.edu.cn; Ligang Liu, University of Science and Technology of China, China, lgliu@ustc.edu.cn; Wenjie Min, Hefei University of Technology, China, 2021171224@mail.hfut.edu.cn; Gaofeng Zhang, Hefei University of Technology, China, g.zhang@hfut.edu.cn; Liping Zheng, Hefei University of Technology, China, zhenglp@hfut.edu.cn.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; *Neural networks*; Probabilistic reasoning.

Additional Key Words and Phrases: Floorplan generation, graph traversal, deep learning

ACM Reference Format:
Jiahui Sun, Wenming Wu, Ligang Liu, Wenjie Min, Gaofeng Zhang, and Liping Zheng. 2022. WallPlan: Synthesizing Floorplans by Learning to Generate Wall Graphs. *ACM Trans. Graph.* 41, 4, Article 92 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530135

1 INTRODUCTION

Floor and building plans are fundamental to architectural design and indoor scene modeling. Automated generation of vectorized floorplans from design constraints (e.g., the boundary) has drawn recent interests in the community [Chaillou 2020; Laignel et al. 2021; Liu et al. 2013; Merrell et al. 2010; Wang et al. 2021; Wu et al. 2018].

Recent techniques [Hu et al. 2020; Nauata et al. 2021; Para et al. 2021; Wu et al. 2019] for generating realistic floorplans have made significant breakthroughs due to the surge of deep learning techniques. These methods either predict labels of the floorplan image at the pixel level [Chaillou 2020; Nauata et al. 2020, 2021; Wu et al. 2019], or bounding boxes of the floorplan rooms at the box level [Chen et al. 2020; Hu et al. 2020; Para et al. 2021]. They can be considered element-oriented methods as they actually generate different grains of discrete elements (labeled pixels or room boxes) as shown in Fig. 2. These methods cannot directly obtain the wall plan from discrete elements, and thus an optimization is always required to be adopted to generate the final vectorized wall plan of the floorplan as a post-processing. However, the optimization is generally designed with heuristic metrics and parameters, resulting in unstable results with tedious parameter tuning or problematic numerical computation. Moreover, these element-oriented methods are unable to meet various design constraints (e.g., the loading-bearing wall constraint) conducted on walls in practical floorplannings.

Unlike existing element-oriented methods, we propose an wall-oriented method, *WallPlan*, for generating wall plans, so as to obtain vectorized floorplans. Given the boundary as well as other design constraints, we aim to generate floorplans by placing walls. Our key idea is to consider the wall plan as a graph with wall junctions as nodes and wall segments as edges and its generation as a graph generation. To achieve it, we borrow the idea of graph traversal and design a network to generate the wall graph progressively by imitating graph traversal. Specifically, we design a graph generation network, called GraphNet, to predict the wall segments progressively in the order of graph traversal scheme. However, this can only produce floorplan geometry, learning the wall plan only is not enough to construct a complete floorplan due to the lack of room semantics. Thus we further design another network, called LabelNet, to predict room labels simultanously. GraphNet and LabelNet are well designed in a coupled manner so that both networks assist with each other during the learning procedure. The two networks are interlaced to generate both the local geometry and the global semantics. Therefore, the finally learned wall graph as well as room labels are output as the vectorized floorplan result. Furthermore, as our method *WallPlan* is actually a wall-oriented method, it is rather natural to realize various wall constraints.

WallPlan can be applied to various constrained floorplan designs, such as the bubble diagram and loading-bearing wall constraint. We conduct ablation experiments, qualitative evaluations, quantitative comparisons, and perceptual studies to evaluate it. Intensive experiments demonstrate *WallPlan* can produce higher quality floorplans than state-of-the-art techniques. We contribute the following:

- a novel wall-oriented method to generate floorplans by representing the floorplan as a wall graph and considering its generation as graph generation.
- a coupled geometry-semantics network that enables wall graph generation under various design constraints in an iterative manner of graph traversal.

WallPlan is able to automatically and efficiently generate plausible floorplans, and realizes abundant constrained floorplan designs, some of which are hard to achieve using state-of-the-art techniques. Fig. 1 shows several generated floorplans using our method. Our key contribution is to generate floorplans by synthesizing wall graphs, making a great improvement of generation quality over previous works. Code for this paper is at *https://github.com/cgjiahui/WallPlan*.

2 RELATED WORK

2.1 Space planning in computer graphics

Our work relates to the research of space planning, which is to study the reasonable arrangement of objects. Space planning is of great importance in many practical applications, such as architectural design, computer games, and virtual reality. Dozens of research on space planning have accumulated a large number of technical works, covering a variety of topics. Related works includes VLSI design [Chen et al. 2010; Sechen 2012; Singha et al. 2012], graphic design [Cao et al. 2014; O'Donovan et al. 2014; Pang et al. 2016; Yang et al. 2016; Zheng et al. 2019], urban planning [Peng et al. 2016, 2014; Vanegas et al. 2012; Yang et al. 2013], facade synthesis of residential buildings [Fan and Wonka 2016; Yeh et al. 2013], indoor scene synthesis [Li et al. 2019; Ritchie et al. 2019; Wang et al. 2019, 2018; Zhang et al. 2020; Zhou et al. 2019], game level design [Hendrikx et al. 2013; Ma et al. 2014; Yeh et al. 2012], layout design of composite buildings [Bao et al. 2013; Feng et al. 2016], etc. In this paper, we aim to generate floorplans, a blueprint of the room arrangement inside the building, which is also a classical problem in space planning.

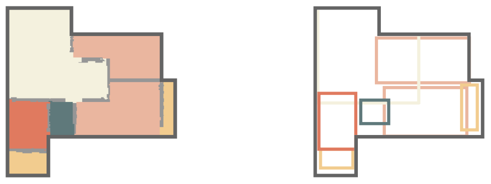

(a) Labeled pixels (b) Room boxes

Fig. 2. Existing element-oriented methods. (a) The floorplan is generated by the pixel level method. It is necessary to transform a raster image into a vectorized floorplan. (b) The floorplan is generated by the box level method. A heuristic post-processing is always required to construct a floorplan with the room box aligned and labeled.

2.2 Optimization-based floorplan generation

Floorplan generation typically takes a set of manually-defined constraints as input, and proposes a reasonable space allocation, providing room categories, locations, and boundaries. Earlier efforts [Arvin and House 2002; Medjdoub and Yannou 2000; Michalek et al. 2002; Michalek and Papalambros 2002] tackle this task using procedural or optimization methods. Given a set of high-level requirements, [Merrell et al. 2010] first learn room attributes based on a Bayesian network to generate architectural programs, then the detailed floorplan can be obtained through stochastic optimization. Following up this work, [Rosser et al. 2017] introduce a data-driven method to estimate room metrics, which can be used to construct appropriate spatial planning. [Liu et al. 2013] propose an interactive floorplan design method combining design constraints, user preferences, and manufacturing considerations to optimize the interior layout for pre-cast concrete-based buildings. [Wu et al. 2018] make full use of mixed integer quadratic programming to mathematically formulate the floorplan generation problem from high-level constraints and propose a hierarchical framework to generate floorplans in a coarse-to-fine manner. Recently, [Wang and Zhang 2020] present a framework to generate floorplans based on room adjacencies. [Laignel et al. 2021] introduce an optimization method to generate floorplans upon the apartment envelope and room specifications by gridding the space. [Shekhawat et al. 2021] propose to construct dimensioned floorplans from an adjacency graph and dimensional constraints with the graph-theoretical and optimization techniques.

However, these methods require careful constraint settings. Too few constraints may lead to unsatisfactory results, while too many could cause constraint contradictions and lead to no feasible solution. Furthermore, some constraints are difficult to model mathematically.

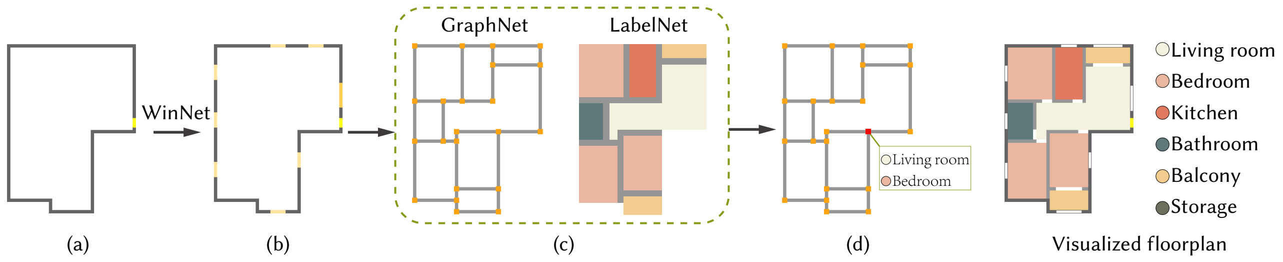


Fig. 3. Overview of *WallPlan*. (a) The building boundary, as well as the front door (in bright yellow), is given as input of *WallPlan*. (b) The boundary is initialized with windows (in dark yellow) predicted by WinNet. (c) *WallPlan* generates both the wall graph and floorplan semantics by a coupled structure of GraphNet and LabelNet from the input boundary and windows. (d) The generated wall graph and floorplan semantics are used to construct a wall graph with room labels as output of *WallPlan*. The vectorized floorplan can be directly obtained from the wall graph. Note that the drawing of windows and interior doors which are generated using a heuristic method is only used for visualization, also adopted by other figures in this paper.

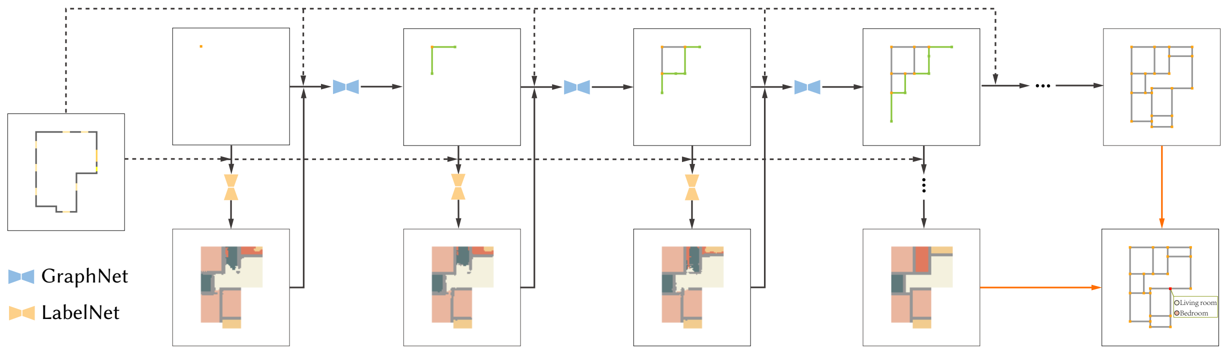


Fig. 4. Architecture of *WallPlan*. Given the boundary with windows as input, *WallPlan* generates both the wall graph and floorplan semantics progressively by imitating graph traversal, then a wall graph with room labels can be obtained. When no nodes can be generated, the generative process is terminated.

2.3 Learning-based floorplan generation

Another idea is to implicitly learn design principles from existing data, resulting in more efficient floorplan generation. Most of the focus has shifted to applying deep learning to floorplan generation in recent years. Most of the existing learning-based methods are element-oriented: pixel level or box level.

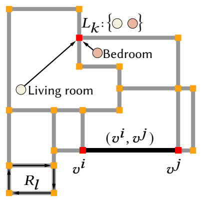
Pixel level. Given the building boundary as input, [Wu et al. 2019] present a two-stage convolutional neural network for automatically generating floorplans by locating rooms first and then locating walls. The generative adversarial network has also been explored in floorplan generation. [Chaillou 2020] proposes a series of deep networks to enable the generation of building footprints, floorplans, and even furniture placements. However, vectorized results cannot be obtained conveniently due to the representation of pixels. Transforming a raster floorplan image into the vector format is a necessary step to get visually plausible floorplans. Another limitation is that the user has little control over the generative process. [Nauata et al. 2020] present a generative model for floorplan generation through a generative adversarial network named *House-GAN*. Taking a bubble diagram as input, *House-GAN* can generate diverse floorplans to fit the diagram. As an extension of *House-GAN*, their following work *House-GAN++* [Nauata et al. 2021] is a straightforward integration of a relational GAN [Nauata et al. 2020] and a conditional GAN [Pathak et al. 2016]. *House-GAN++* has improved the quality of results compared to *House-GAN*, while still owning the major drawback of *House-GAN*, that is, floorplans are represented in raster images, further post-processing is still needed.

Box level. [Hu et al. 2020] achieve the conversion of layout graphs to floorplans by a deep network framework called *Graph2Plan* and allow users to control floorplan generation by imposing design constraints on layout graphs. However, this method is limited by the dataset and lacks generality since the layout graph is obtained by heuristic retrieval according to the similarity of boundaries. Meanwhile, they can only obtain discrete room boxes, post-processing is needed to get desired results. [Chen et al. 2020] focus on producing floorplans from language descriptions that describe house details based on graph convolutional network and generative adversarial network. Just similar to [Hu et al. 2020], further post-processing is also needed. More recently, [Wang et al. 2021] present a learning-based method to generate 3D floorplans by piecing together existing 3D rooms. [Para et al. 2021] propose a hybrid method composed of the generative model and optimization method to generate layouts, which first generates a layout graph with layout elements as nodes and constraints between layout elements as edges, and then the final layout is obtained by linear programming.

In summary, to produce good-looking floorplans, existing learning-based methods require a heuristic post-processing. In contrast, we aim to synthesize floorplans by learning to generate wall graphs, which can greatly reduce the complexity of post-processing. In addition, existing learning-based methods cannot achieve floorplan design based on walls, such as the loading-bearing wall constraint. *WallPlan* can realize abundant applications including floorplan design based on both the global bubble diagram constraint and the local loading-bearing wall constraint.

3 OVERVIEW

Representation. We represent the floorplan as a wall graph of the form $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a node set $\mathcal{V} = \{v^k\}_{k=0}^{M}$ indicating wall junctions and a edge set $\mathcal{E} = \{(v^i, v^j) \mid v^i, v^j \in \mathcal{V}\}$ indicating wall segments (right inset). The wall junction can be represented as $v^k = (\mathbf{p_k}, L_k)$ where $\mathbf{p_k}$ is the position of v^k, and L_k is a label list of rooms that contain v^k. A shortest cycle forms the room R_l, which is the single region surrounded by nodes. Once adopting a wall graph to represent floorplan, the vectorized floorplan can be directly obtained from the wall graph, and thus a natural idea is to transform floorplan generation to wall graph generation.



Dataset. We extract wall graphs from the *RPLAN* dataset [Wu et al. 2019], which contains more than 80K real-world floorplans with dense annotation from residential buildings. Specifically, we first extract wall junctions in the floorplan image, then find the adjacencies between the extracted junctions according to wall segments. Finally, room labels are obtained and added to the graph nodes. In turn, given a wall graph with room labels, rooms in the floorplan can be determined by finding all shortest cycles in the wall graph, and the common room label of all nodes on the cycle is used as the label of the room. Since we use image-based convolutional neural networks as the basic network architecture of our model, we convert the wall graph into a 120 × 120 image with fixed wall thickness (3 pixels in our experiments) during training.

Problem & Challenge. Given a building's outer boundary (the front door included) and other design constraints as input (Fig. 3a), our goal is to generate a plausible wall graph (Fig. 3d), then a vectorized floorplan can be obtained directly. There are two technical challenges. Firstly, given the boundary as well as other constraints,it is challenging to design deep learning networks to realize wall graph generation. Secondly, we also need room labels to construct a complete floorplan. Wall graph generation and room label prediction relate to each other and influence each other. The independent generation of the wall graph and room labels can not taking advantage of this connection apparently, while it is non-trivial to combine the generation of both.

Methodology. We enable wall graph generation by imitating graph traversal. *WallPlan* is shown in Fig. 3 and detailed generative process is given in Fig. 4. Given the boundary as well as other constraints as input, we first initialize the boundary with windows which can be predicted by WinNet. Starting from one node of the boundary, we propose a graph generation network GraphNet to iteratively predict new nodes and edges in a manner of graph traversal. The wall graph is progressively generated constrained on the boundary as well as other design constraints by feeding the inputs to GraphNet. To obtain room labels, we propose a semantics prediction network LabelNet to obtain the global floorplan semantics from the generated wall graph. GraphNet and LabelNet are coupled to generate a wall graph with room labels progressively. In each step, the generated wall graph by GraphNet will be used for room label prediction by LabelNet, and the result of LabelNet will assist GraphNet, in turn. This repeats until a complete wall graph can be obtained.

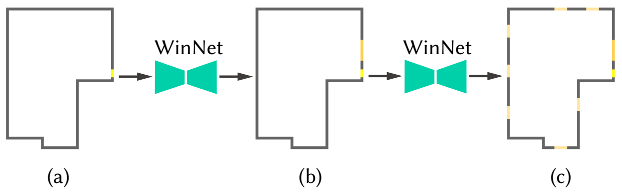


Fig. 5. Window prediction. (a) Given the boundary, (b) we first predict the living room window, then (c) the windows of other type rooms are predicted based on the boundary and the predicted living room window.

4 METHOD

WallPlan is mainly composed of two modules: wall graph generation (Section 4.2) and room label prediction (Section 4.3). We design a coupled structure of these two modules (Section 4.4). Before that, we propose a window-first strategy (Section 4.1) to initialize the generative process. For the convenience of description, we only consider the boundary as input to introduce our method, more input constraints are discussed in Section 5.

4.1 Initialization with windows

Window prediciton. We propose a window-first strategy that predicts windows on the boundary to initialize the generative process based on one key observation. The window position indicates the approximate location of the room, and the number of windows is often positively correlated with the number of rooms. Therefore windows can provide a weak constraint on floorplan design.

The window-first strategy brings two benefits. On one hand, windows can serve as a soft constraint to limit the solution space of floorplan design and thus facilitate us to obtain a locally optimal solution, improving the overall rationality of generated floorplans. On the other hand, the distribution of windows can guide floorplan generation, and the window-first strategy also provides a degree of control over the generative process. To set windows, we adopt a simple two-step prediction shown in Fig. 5. We first predict the living room window given the boundary, then the windows of other type rooms are predicted based on the boundary and the predicted living room window. The user is also allowed to adjust the windows to achieve the personalized floorplan design. We only consider two types of windows: the larger window assigned to the living room, and the smaller window to other type rooms, as some rooms usually have the same window specifications. Experiment results have shown that only two types of windows are sufficient for *WallPlan*.

WinNet. Windows are determined through a prediction network called WinNet. The original *RPLAN* dataset does not include windows. To build a training dataset, we directly use some empirical rules proposed by [Wu et al. 2019] to create windows for floorplans in *RPLAN* dataset. Setting windows based on the heuristics is pretty common in previous work [Hu et al. 2020; Wu et al. 2019] as the setting rules are relatively straightforward and simple. Therefore, it is available for our window prediction task, although windows are synthetic. WinNet aims at a multi-channel image as input and a semantic segmentation of windows as output. For the prediction of the living room window, the input includes the following information at each pixel, which defaults to 0:

- *Inside mask*: taking a value of 1 for the interior.
- *Boundary mask*: taking a value of 1 for the exterior walls.
- *Entrance mask*: taking a value of 1 for the front door.

For window prediction of other type rooms, the input to WinNet includes an additional channel *living room mask* taking a value of 1 for the living room window. We use a 13×5 square to represent the living room window and a 7×5 square for the window of other type rooms. We use a modified D-LinkNet [Zhou et al. 2018] as the network architecture. The original D-LinkNet is modified to use 120×120 multi-channel images as input. In the remainder of Section 4, the boundary we refer to contains windows by default. Windows are represented as a squared mask.

4.2 Graph generation

Traversal generation. Given the boundary which can be represented as a polygon $\mathcal{B} = \left\{v_B^k\right\}_{k=0}^{N}$, our goal is to generate a wall graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ constrained on $\mathcal{B}$. We refer to Breadth-First Search (BFS) of graph traversal. The traditional BFS algorithm starts with a source node and adds unvisited neighboring nodes of the source to a frontier queue. The nodes in the frontier queue are marked as visited and will be used for expansion in the subsequent traversal. In this way, the algorithm is executed repeatedly until no node remains unvisited. We borrow the idea of graph traversal and enable wall graph generation by imitating graph traversal. Starting from one node, the wall graph can be generated by iteratively predicting new nodes and edges based on the previously generated wall graph. Supposing that $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ is the wall graph in the t-iteration, the new nodes $\Delta\mathcal{V}_{t+1}$ and edges $\Delta\mathcal{E}_{t+1}$ of the wall graph $\mathcal{G}_{t+1}$ can be predicted from the wall graph $\mathcal{G}_t$ and the boundary $\mathcal{B}$. The joint distribution of $\mathcal{G}_T$ is then, by the chain rule:

$$p(\mathcal{G}_T \mid \mathcal{B}) = p(\mathcal{V}_T, \mathcal{E}_T \mid \mathcal{B}) = \prod_{t=0}^{T} p(\Delta\mathcal{V}_{t+1}, \Delta\mathcal{E}_{t+1} \mid \mathcal{G}_t, \mathcal{B}) \quad (1)$$

$$\mathcal{G}_{t+1} = \mathcal{G}_t \cup (\Delta\mathcal{V}_{t+1}, \Delta\mathcal{E}_{t+1}) \quad (2)$$

$\mathcal{G}_0$ only contains the starting point and T is the number of iterations.

To this end, we develop an algorithm for wall graph generation in Algorithm 1. Without loss of generality, we first choose the top-left node $v_B^{\textit{top-left}}$ obtained by pre-sorting of the boundary $\mathcal{B}$ as the starting point. Then an iterative pipeline is following, as shown in Fig. 6. In each step, GraphNet is used to produce the candidates of new nodes and edges by taking the boundary $\mathcal{B}$, wall graph $\mathcal{G}_t$, and source nodes $\Delta\mathcal{V}_t$ as input. It is worth noting that not all candidate nodes and edges can be added to the wall graph $\mathcal{G}$. We use "if...end" (Line 10-13 and 16-18 in Algorithm 1) to avoid adding wrong nodes and edges predicted by our method. Only if there exists an edge candidate between a node candidate and the source nodes, both the candidates of nodes and edges are added into the wall graph as the new nodes and edges. Meanwhile, we add an edge candidate into the wall graph if the edge candidate is built between two new nodes. This iterative process is terminated until GraphNet can no longer predict any new nodes. Note that we generate the wall graph from scratch based on the given boundary. To meet the boundary constraint, the generated wall graph $\mathcal{G}$ is trained to cover the boundary $\mathcal{B}$ during the training process.

ALGORITHM 1: Graph generation

```
Input: $\mathcal{B}$: the boundary; $v_B^{top\text{-}left}$: the top-left node of the boundary;
  $t$: the number of iteration
Output: $\mathcal{G}$: the wall graph constrained on $\mathcal{B}$
1  $t = 0$;
2  $\mathcal{G}_t = \left(\left\{v_B^{top\text{-}left}\right\}, \emptyset\right)$;   /* Wall graph in the $t$-iteration */
3  $\Delta\mathcal{V}_t = \left\{v_B^{top\text{-}left}\right\}$;   /* New nodes (source nodes) of $\mathcal{G}_t$ */
4  $\Delta\mathcal{E}_t = \emptyset$;   /* New edges of $\mathcal{G}_t$ */
5  while $\Delta\mathcal{V}_t \neq \emptyset$ do
6      $\Delta\mathcal{V}^c_{t+1}, \Delta\mathcal{E}^c_{t+1}$ = GraphNet($\mathcal{B}, \mathcal{G}_t, \Delta\mathcal{V}_t$);   /* Generate the
        candidates of new nodes and edges of $\mathcal{G}_{t+1}$ */
7      $\Delta\mathcal{V}_{t+1} = \emptyset$;
8      $\Delta\mathcal{E}_{t+1} = \emptyset$;
9      for $v^c \in \Delta\mathcal{V}^c_{t+1}, v^i \in \Delta\mathcal{V}_t$ do   /* Determine if each
        candidate of nodes and edges is added to $\mathcal{G}_{t+1}$ */
10         if edge $(v^c, v^i)$ in $\Delta\mathcal{E}^c_{t+1}$ then
11             Add $v^c$ into $\Delta\mathcal{V}_{t+1}$;
12             Add $(v^c, v^i)$ into $\Delta\mathcal{E}_{t+1}$;
13         end
14     end
15     for $v^i, v^j \in \Delta\mathcal{V}_{t+1}$ do   /* Determine if the edge candidate
        exists between two new nodes in $\mathcal{G}_{t+1}$ */
16         if edge $(v^i, v^j)$ in $\Delta\mathcal{E}^c_{t+1}$ then
17             Add $(v^i, v^j)$ into $\Delta\mathcal{E}_{t+1}$;
18         end
19     end
20     $\mathcal{G}_{t+1} = \mathcal{G}_t \cup (\Delta\mathcal{V}_{t+1}, \Delta\mathcal{E}_{t+1})$;   /* Update $\mathcal{G}_{t+1}$ */
21     $t = t + 1$;
22 end
23 $\mathcal{G} = \mathcal{G}_t$;
24 return $\mathcal{G}$;
```

GraphNet. We now introduce the core network GraphNet. To build the training dataset for GraphNet, we perform Breadth-First Search on graphs in the dataset and save the intermediate results during the graph traversal as the partial wall graphs which are used to train GraphNet. The input for GraphNet is also a multi-channel image, similar to WinNet. Except for the channels used in WinNet, we add several new channels which default to 0:

- *Window mask*: taking a value of 1 for the living room window and 2 for other windows.
- *Source node mask*: taking a value of 2 for the source nodes and 1 for other nodes. We use a small square (3×3) centered on the node to represent it.

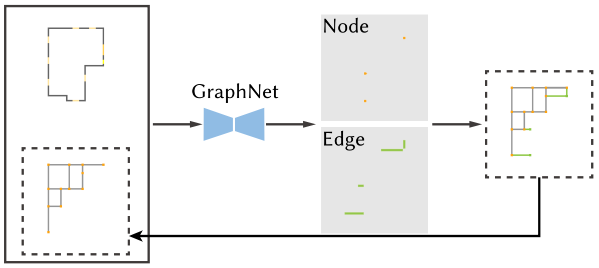


Fig. 6. A step of the wall graph generation. Given the boundary and wall graph, GraphNet outputs the semantic segmentation of the new nodes and edges, and we further extract new nodes and edges to construct a new wall graph. Please refer to the text for more details.

- *Wall graph mask*: taking a value of 1 for edges in the wall graph.

GraphNet outputs a semantic segmentation of the new nodes and edges with three labels: Node, Wall, and Nothing, as shown in Fig. 6. To determine the candidates of new nodes, we only need to search for predicted nodes in the top, bottom, left, and right of the source node due to the regularity of the wall graph. We determine the candidates of new edges by judging if there is a certain density of pixels belonging to new edges in the output segmentation. In other words, the edge exists if there are a certain number of edge pixels between two nodes in the local operation of each iteration. The higher the pixel density, the more confident the existence of edges. Once the candidates of new nodes and edges are determined, we add new nodes and edges into the wall graph using the algorithm of graph generation in Algorithm 1. Similar to WinNet, we use a modified D-LinkNet as the network architecture of GraphNet. The original application of D-LinkNet is proposed for road extraction, and we think D-LinkNet is beneficial for our task in view of the similarity between road extraction and graph generation as the road network can also be considered as a graph.

Discussions. Our method adopts the image-based methodology, like previous techniques [Chaillou 2020; Nauata et al. 2020, 2021; Wu et al. 2019], to learn the pixel semantics. However, previous methods need a post-processing optimization to generate the vectorized floorplan. Instead, we achieve this by performing operations in each iteration. Actually, this is equivalent to decomposing the complex post-processing optimization into a series of simpler local operations, achieving better and faster generation. We do not completely eliminate the post-processing brought by image-based generation, but greatly reduce the complexity of post-processing.

4.3 Room label prediction

LabelNet. We need to assign room labels for each node in the generated wall graph. Therefore, we propose a semantics prediction network called LabelNet to obtain the global floorplan semantics from the generated wall graph (Fig. 7). We determine the room label for each node of the wall graph based on the predicted floorplan semantics. Note that the input wall graph to LabelNet is not necessarily the complete wall graph. Similar to WinNet, the input to LabelNet is also a multi-channel image. Besides of the channels used in WinNet, we add several additional channels which default to 0:

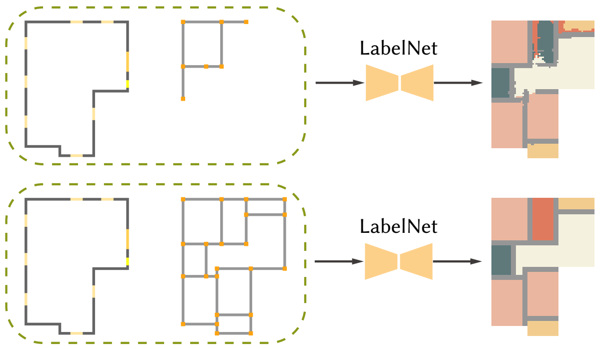


Fig. 7. Given a wall graph (not necessarily the complete wall graph as shown in the top row) and the boundary, we use LabelNet to predict floorplan semantics, so as to obtain room labels.

- *Window mask*: taking a value of 1 for the living room window and 2 for other windows.
- *Wall graph mask*: taking a value of 1 for edges in the wall graph.

The output of LabelNet is a semantic segmentation with room labels. To determine room labels for each node of the wall graph, we first find all the shortest cycles in the wall graph, that is, the room. For each room, we choose the label of the pixel on the room center as the room label. At the same time, we assign the room label to the nodes that enclose this room. Since LabelNet performs a semantic segmentation task, we use a modified D-LinkNet as the network architecture similar to WinNet and we train the network using pixel-wise cross-entropy loss. D-LinkNet performs better in the semantic segmentation of multi-scale objects.

4.4 Coupled geometry-semantics generation

Coupled generation. So far, we have proposed two modules to separately generate the wall graph and room labels. In the following, we do not simply chain these two modules together to construct a wall graph with room labels. What we want is to deeply couple the wall graph generation and room label prediction to improve the overall quality of the wall graph. In fact, the wall graph and room labels relate to each other, influence each other, and promote each other. Particularly, the input wall graph to LabelNet is not necessarily the complete graph (Fig. 7). Providing the partial wall graph, LabelNet is also able to predict the global floorplan semantics. The global floorplan semantics obtained from the wall graph can provide global guidance for GraphNet and improve the quality of the wall graph generated in the next step. In one word, providing the global semantics enables GraphNet to consider globally and produce a more reasonable space allocation. To this end, we embed the room label prediction module into the wall graph generation module and design a coupled structure in Fig. 4. The wall graph generated by GraphNet will be used for room label prediction, and the result of the LabelNet will, in turn, assist wall graph generation. In this way, the room label prediction and wall graph generation are coupled to generate a wall graph with room labels.

Coupled structure. The architecture of the coupled structure is shown in Fig. 4, mainly composed of two sub-networks: GraphNet

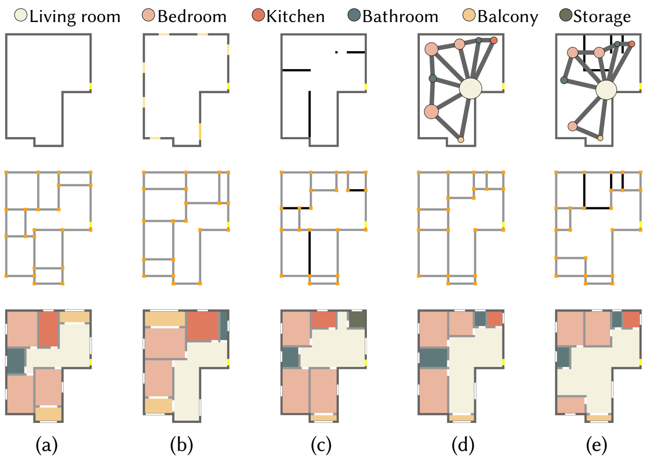


Fig. 8. Constrained floorplan generation. Top row: various design constraints applied to the same boundary. Middle row: output wall graphs. Bottom row: floorplan visualization of wall graphs. (a) Only the input boundary. (b) Window constraint. (c) Load-bearing wall constraint (black dots and line segments). (d) Bubble diagram constraint (represented as the layout graph). (e) Hybrid-constraint. The overall structure of the floorplan changes significantly with different constraints.

and LabelNet. The boundary $\mathcal{B}$ is the initial input. Given a wall graph $\mathcal{G}_t$ and the input boundary $\mathcal{B}$, LabelNet is used to predict the global floorplan semantics $\mathcal{S}_t$:

$$\mathcal{S}_t \leftarrow LabelNet\,(\mathcal{G}_t, \mathcal{B}) \tag{3}$$

Then GraphNet is used to produce a new wall graph $\mathcal{G}_{t+1}$ using the algorithm of graph generation in Algorithm 1 from the wall graph $\mathcal{G}_t$, boundary $\mathcal{B}$ as well as the floorplan semantics $\mathcal{S}_t$:

$$\mathcal{G}_{t+1} \leftarrow GraphNet\,(\mathcal{G}_t, \mathcal{B}, \mathcal{S}_t) \tag{4}$$

This can be executed repeatedly until a complete wall graph can be obtained. The network architectures of these two sub-networks are consistent with what was described before, except that the input of GraphNet will be added extra channels of the floorplan semantics.

5 IMPLEMENTATION

In this section, the detailed implementation of more design constraints and networks is given.

5.1 Constrained floorplan generation

Window constraint. Given the boundary as the only input, we can generate floorplans, which is the basic application of our method, as shown in Fig. 8a. In *WallPlan*, we use WinNet to predict windows for the input boundary. Windows on the boundary have a direct impact on the generated floorplan. Therefore, a certain degree of control over the floorplan generation can be enabled by setting desired windows, as shown in Fig. 8b. So simply specifying the windows by the user, various floorplans can be generated (Fig. 13).

Load-bearing wall constraint. Destruction of the load-bearing walls will affect the stability of the entire building. Therefore the loading-bearing walls are often preset. State-of-the-art methods [Chen et al. 2020; Hu et al. 2020; Nauata et al. 2020, 2021; Wu et al. 2019] cannot conveniently solve the loading-bearing wall constraint, while it is a straightforward application of our method, as shown in Fig. 8c. Given the boundary $\mathcal{B}$ as well as the loading-bearing walls $\mathcal{W}$, we define a wall graph generation problem constrained on $\mathcal{B}$ and $\mathcal{W}$:

$$p\,(\mathcal{G}_T \mid \mathcal{B}, \mathcal{W}) = \prod_{t=0}^{T} p\,(\Delta\mathcal{V}_{t+1}, \Delta\mathcal{E}_{t+1} \mid \mathcal{G}_t, \mathcal{B}, \mathcal{W}) \tag{5}$$

In this way, we can generate wall graphs by feeding the encoded inputs of the boundary and loading-bearing walls to *WallPlan*. There are two kinds of loading-bearing walls: loading-bearing columns (represented as points) and walls (represented as line segments). The generated wall graphs need to cover not only the boundary but also the loading-bearing columns and walls. In the implementation, we only need to add the loading-bearing wall constraint into the input of *WallPlan* without modifying the network architecture, by adding an extra *loading-bearing wall mask* taking a value of 1 for the loading-bearing columns and walls into the multi-channel input.

Bubble diagram constraint. The bubble diagram uses nodes to represent rooms and connections to indicate adjacencies between two rooms. Bubble diagrams can provide global guidance for floorplan generation and greatly limit the solution space of floorplans, which can serve as a flexible control tool for floorplan generation. It is worth mentioning that the bubble diagram includs spatial locations, similar to *Graph2Plan* [Hu et al. 2020]. Given the boundary $\mathcal{B}$ as well as the bubble diagram $\mathcal{D}$, we define a wall graph generation problem constrained on $\mathcal{B}$ and $\mathcal{D}$:

$$p\,(\mathcal{G}_T \mid \mathcal{B}, \mathcal{D}) = \prod_{t=0}^{T} p\,(\Delta\mathcal{V}_{t+1}, \Delta\mathcal{E}_{t+1} \mid \mathcal{G}_t, \mathcal{B}, \mathcal{D}) \tag{6}$$

Adding the bubble diagram into the input of *WallPlan* for encoding, *WallPlan* can convert a diagram into a floorplan that fulfills both the bubble diagram and boundary (Fig. 8d). In this way, *WallPlan* can provide a certain degree of control over the generative process by the bubble diagram constraint. Since *WallPlan* uses image-based convolutional neural networks as the main network architecture, we also convert the bubble diagram into an image representation to adapt to our image-based encoding. We visualize the bubble diagram as a raster image with a certain geometric size. Specifically, the graph nodes are represented as circles with a certain radius which is proportional to the room area, and the graph connections are visualized as line segments. The user can define the room area and the default setting is also provided. Note the room area here is only used as a reference for the relative room size. The generated floorplans may not strictly meet the area setting but will satisfy the relative size constraint. We extract bubble diagrams of floorplans from the *RPLAN* dataset [Wu et al. 2019] for training. Specifically, the node of the bubble diagram can be obtained by finding the geometric center of the room, and the room label and area are also added as the node attributes. The connections of the bubble diagram can be determined by finding the adjacencies of two rooms. There is no need to modify the network architecture of *WallPlan* except using several additional input channels to represent the bubble diagram: (1) *Diagram node mask* taking values of 1, 2, ... for different types of room nodes. (2) *Diagram connection mask* taking a value

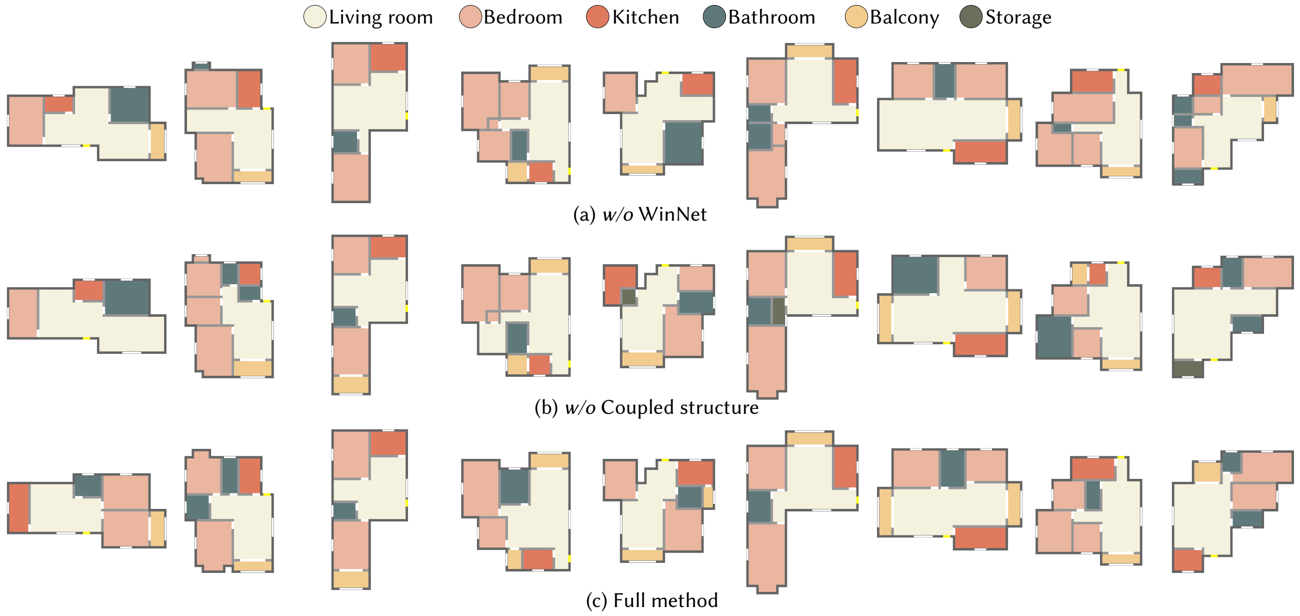


Fig. 9. Ablation studies. (a) We remove WinNet from *WallPlan* to generate floorplans. (b) We decouple the coupled structure into a two-stage network. (c) The floorplans are generated using our full method. Each column shares the same boundary as the input of floorplan generation.

of 2 for connections between the living room node and other room node and 1 for edges between any two non-living room nodes.

Hybrid-constraint. We currently have discussed three types of constraints: window, load-bearing wall, and bubble diagram. In general, we need to separately train the specific network for each type of constraint although they all have the same network architecture. It is not convenient for personalized floorplan design and not conducive to functional integration for a design tool. To this end, our method also achieves hybrid-constraint floorplan generation benefitting from the scalability of *WallPlan*, which provides a more flexible and versatile control tool for floorplan generation by combining multiple constraints, as shown in Fig. 8e. Given the boundary $\mathcal{B}$, load-bearing wall $\mathcal{W}$, and bubble diagram $\mathcal{D}$, we define the hybrid-constraint generation as a wall graph generation problem constrained on $\mathcal{B}$, $\mathcal{W}$, and $\mathcal{D}$:

$$p\left(\mathcal{G}_T \mid \mathcal{B}, \mathcal{W}, \mathcal{D}\right)=\prod_{t=0}^{T} p\left(\Delta \mathcal{V}_{t+1}, \Delta \mathcal{E}_{t+1} \mid \mathcal{G}_t, \mathcal{B}, \mathcal{W}, \mathcal{D}\right) \tag{7}$$

The input of *WallPlan* is scalable, so hybrid-constraint floorplan generation can be achieved by reserving the corresponding input channels for each constraint and setting the corresponding channels to 0 as default. Other design constraints not mentioned in this paper can also be added to our hybrid-constraint floorplan generation in a similar way. We would like to explore more in the future.

5.2 Network training

We use PyTorch to implement *WallPlan* and train our networks. All models are trained and tested on an NVIDIA GeForce GTX 3090 GPU. The *WallPlan* is trained on the *RPLAN* dataset, with 80% for training, 5% for validation, and 15% for testing. There are several networks in our method: WinNet, GraphNet, LabelNet, and these networks are trained separately since the input to the network is essential to obtain a good training effect. We first train WinNet from the input boundary. For the training of GraphNet and LabelNet, the ground truth of windows is provided as the input to networks. We only train single iterations sampled from the whole process, including the end condition. To train GraphNet, we perform Breadth-First Search on the ground-truth graphs and take the current search as input and the next as output. To train LabelNet, the output for the searched complete and partial graphs are all complete floorplan semantics. GraphNet and LabelNet are trained separately. We first train LabelNet, then we run LabelNet to obtain label maps in the training of GraphNet. The iterative process stops until GraphNet can no longer predict any new nodes. The final output of our networks is a connected graph. More details are given in the supplementary material. The execution time of different method is shown in Table. 1. It takes an average of 0.74 seconds to generate a wall graph with a given boundary. Although Graph2Plan is faster than ours, as we will show, the generation quality is not as good as ours.

Table 1. The average execution time of generating one floorplan.

Method	*RPLAN* [Wu et al. 2019]	*Graph2Plan* [Hu et al. 2020]	*WallPlan*
Time	4.00	0.40	0.74

6 EVALUATION

We conduct ablation experiments, qualitative evaluations, quantitative comparisons, and perceptual studies to evaluate our method.

6.1 Ablation study

In *WallPlan*, we introduce two core modules, the WinNet to predict floorplan windows and the coupled structure of GraphNet and LabelNet to generate the wall graph and room labels jointly. To prove these two modules are indeed beneficial, ablation studies are conducted. For the ablation study of WinNet, we just take it away from *WallPlan*, use this version to generate floorplans, and compare the performance to our full method. For the ablation study of the coupled structure, we directly decouple the original structure into a two-stage network, first generating the wall graph by GraphNet and then predicting room labels by LabelNet. From Fig. 9, we can clearly see that all ablated versions perform worse than the full method, demonstrating the critical role of each component of *WallPlan* in generating high-quality floorplans.

The lack of WinNet will bring two problems (Fig. 9a). Firstly, the generated rooms are not of appropriate size, and some bathrooms are too large while some bedrooms are over-small. Secondly, it may lead to some topological troubles, i.e., some bedrooms do not directly connect to the living room, which is not in line with the general principles of floorplan design. As mentioned in Section 4.1, windows can be regarded as a soft constraint on floorplan design and guide floorplan generation, improving the predictive accuracy and overall rationality of floorplans. Similar geometric and topological problems also appear in the results of the decoupled structure (Fig. 9b), illustrating the independent generation of the wall graph and room labels are not as good as the coupled generation. The latter proposes a more reasonable space allocation. From the results shown in each column, we see that providing the global semantics guidance enables our method to "consider" more globally.

6.2 Qualitative evalution

Boundary constrained generation. Since both *RPLAN* [Wu et al. 2019] and *Graph2Plan* [Hu et al. 2020] can generate floorplans from boundaries, we perform comparisons with these two methods. Fig. 10 shows the comparsion of the generated floorplans from boundaries. In column (b), *RPLAN* generates several crumbling walls in the first four rows due to complex post-processing on the raster image. Broken walls may lead to a problematic floorplan (e.g., semantic issue). In addition, *RPLAN* also generates floorplans with an insufficient number of rooms, such as the example in the fifth row. In column (c), *Graph2Plan* generates floorplans with some "tiny" rooms in the second, third and fifth rows, especially the second row where the generated bathroom box is almost invisible due to the post-processing: room box alignment and stacking. Moreover, bathrooms are "missing" in the first and fourth rows, as the generated bathroom box is totally covered by the bedroom box and has become invisible. The floorplan quality of *Graph2Plan* is severely impacted by the post-processing. It is worth mentioning our method can produce significantly different but still high-quality floorplans compared to the results of the ground truth, *RPLAN*, and *Graph2Plan* in the sixth row thanks to our graph-based learning method, illustrating the ability of our method to generate a variety of floorplans and generalize beyond its training dataset.

Bubble diagram constrained generation. Since *Graph2Plan* [Hu et al. 2020] can receive as input both the boundary and bubble diagram. we perform comparisons with *Graph2Plan*. It is worth mentioning that *Graph2Plan* encodes the discretized spatial position of the bubble diagram, which needs spatial locations as ours. Fig. 11 shows the comparison of the generated floorplans from the bubble diagram. The bubble diagram of the ground truth is provided as the generation constraint. In the first two rows, our method can generate plausible floorplans from the bubble diagram closer to the ground truth than the results of *Graph2Plan*, especially the room positions and sizes. In column (c), *Graph2Plan* generates floorplans with some "missing" rooms again in the third rows due to the improper box alignment or stacking order during post-processing. In addition, the fourth row shows the topology issue of *Graph2Plan*, obviously lowing the quality of the generated results. A deep corridor is commonly used to connect different areas and lengthens the visual effect of the home. In the last two rows, *Graph2Plan* cannot create the effect of deep corridors while our method enables the design of deep corridors as shown in the ground truth. We attribute this to our more advanced floorplan representation. Since *Graph2Plan* adopts the representation of room box, heuristic post-processing will directly destroy the effect of deep corridors. To sum up, our method achieves satisfactory floorplan generation. There are two reasons contributing to this. Firstly, the bubble diagram is strong to constrain the connection, type, location, and size of rooms. Secondly, since bubble diagram generation is not our work, the ground-truth bubble diagram is used, just for convenience. Combining these, it is natural to generate results close to the ground truth with good training. But for *Graph2Plan*, there exists a gap between the results and the ground truth in terms of the room numbers, sizes, and

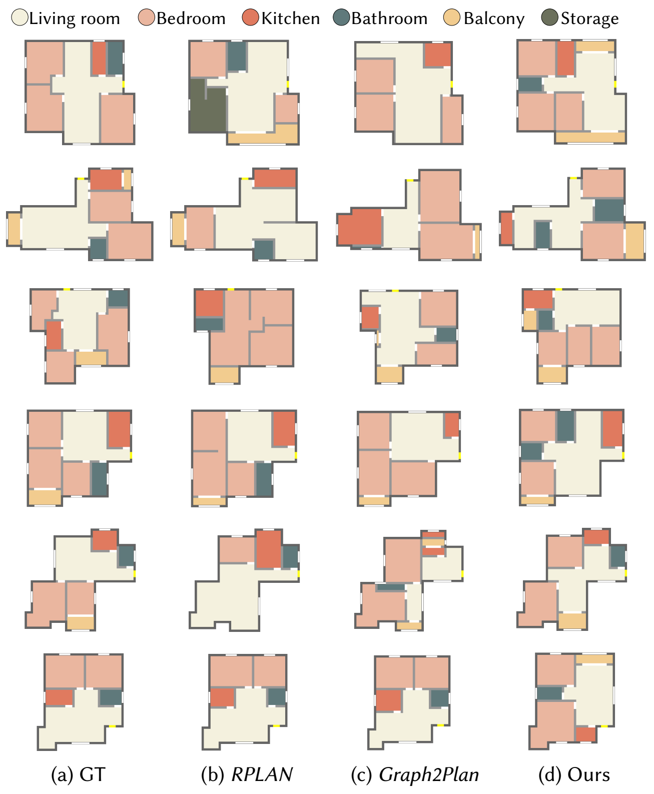


Fig. 10. Comparison to the state-of-the-arts with the only boundary constraint. Each row shows the results of different methods applied to the same boundary. (a) The ground truth floorplans, and the floorplans generated by (b) *RPLAN*, (c) *Graph2Plan* and (d) our method.

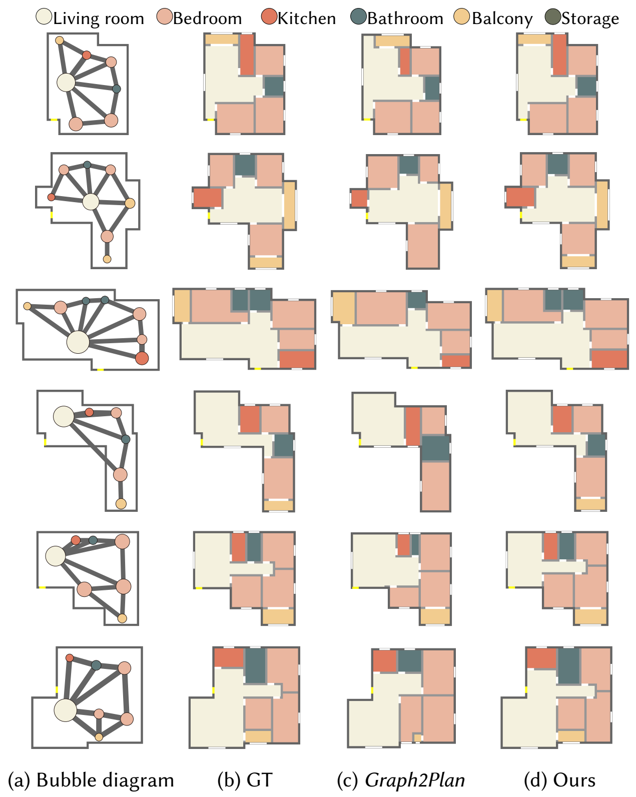


Fig. 11. Comparison to the state-of-the-arts with the bubble diagram constraint. Each row shows the results of different methods applied to (a) the same boundary and bubble diagram. (b) The ground truth floorplans, and the floorplans generated by (c) *Graph2Plan* and (d) our method.

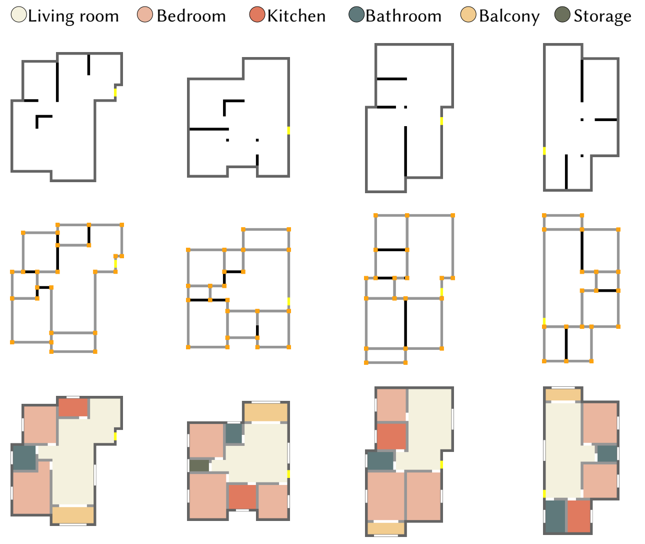


Fig. 12. Loading-bearing wall constrained generation. Top row: the input boundary as well as the loading-bearing walls. Middle row: the generated wall graphs with the loading-bearing walls (in black) using our method. Bottom row: the corresponding floorplans for visualization.

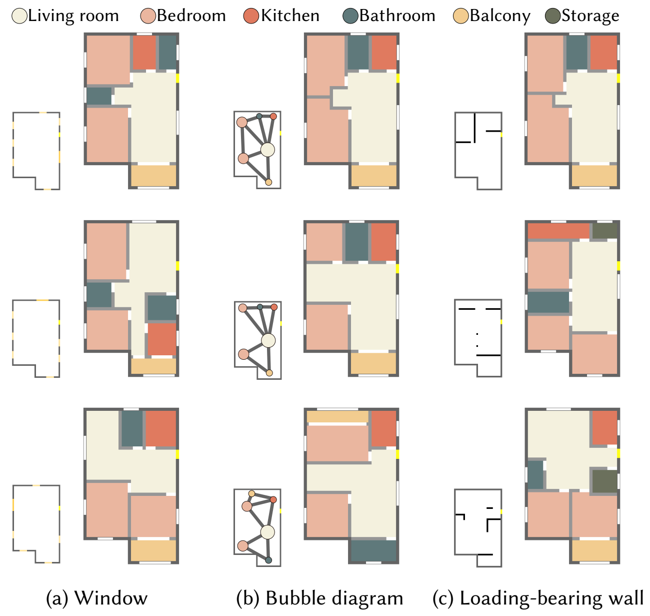


Fig. 13. Generating multiple floorplans from the same boundary with different constraints. Each column shows the different settings of the specific design constraint applied to the same boundary. (a) Window constraints. (b) Bubble diagram constraints. (c) Loading-bearing wall constraints.

connectivities. Even if the predicted room boxes are perfect, there always exists an inevitable gap between the floorplan constructed by the room box and the ground truth, which is the essential flaw of the representation of the room box.

Loading-bearing wall constrained generation. Fig. 12 shows several floorplans generated from the loading-bearing walls. We did not compare with other methods since state-of-the-art techniques based on deep learning still cannot well realize the loading-bearing wall constrained generation. In the top row of Fig. 12, various kinds of constraint settings are provided. By examining each column, it can be seen that the floorplans generated by our method satisfy the given boundary and adapt to the loading-bearing walls. Similar to the bubble diagram constraint, the loading-bearing wall constraint greatly limits the solution space of the floorplan design. Therefore, a certain degree of control over the floorplan generation can also be realized by the loading-bearing wall constraint.

Free generation. We also compare the floorplans generated by our method with the results by *House-GAN++* [Nauata et al. 2021], as shown in Fig. 15. *House-GAN++* aims to achieve free floorplan generation from only the bubble diagram while our method always needs the boundary as one part of the input. Note that the bubble diagram here is slightly different from the above in *Graph2Plan*, having no location information. Our method is enabled to generate floorplans freely with the bubble diagram as the only constraint via a simple trick. Specifically, a retrieval approach similar to *Graph2Plan* is adopted to match the input bubble diagram and retrieve the corresponding boundary. We can always get a lot of boundaries for one input bubble diagram in this way. These boundaries are used to achieve the bubble diagram constrained generation, producing

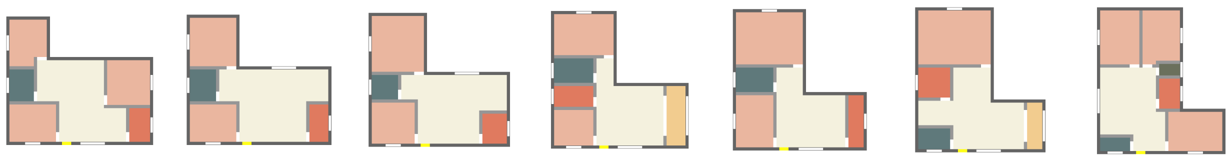


Fig. 14. Interpolation test. *WallPlan* generates floorplans from a set of interpolated boundaries. The first and last ones are the input boundaries for interpolation.

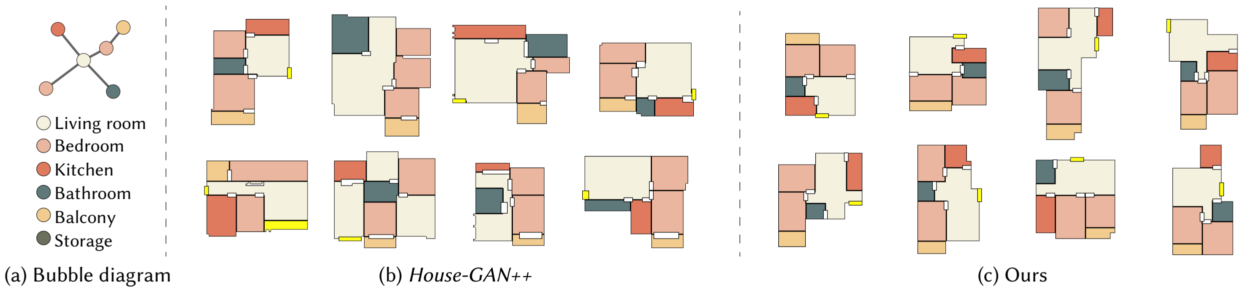


Fig. 15. Comparison to *House-GAN++*. An example of bubble diagram constrained floorplan generation using *House-GAN++* and our method are shown. To fairly compare the performance of the two methods, we adopt a drawing style similar to *House-GAN++*.

the "free" generation result. In Fig. 15, the generated floorplans of *House-GAN++* are unsatisfactory, and there is still much room for improvement. A lot of geometric and topological flaws can be found, such as short walls, weird shapes, wrong connections, etc. In contrast, our method can easily produce plausible floorplans. It shows that the quality of the generated floorplans by our method is much higher than that of *House-GAN++*. Our method outperforms generative models of mixing the image generation and vectorization, as our method directly synthesizes walls and greatly reduces the complexity of vectorization.

Generalization. Given the boundary as input, Fig. 10 shows our method can generate very different floorplans from the ground truth and not just memorize the training dataset. Our method can generate multiple floorplans from the same input boundary with different constraints. Fig. 13 shows a gallery of floorplans generated from the same boundary. Our method explores different possible settings of windows, bubble diagrams, and loading-bearing walls, and generates varying floorplans adapting to the corresponding constraints. Fig. 13 shows floorplans generated from user-defined constraints, which also indicates the results variety and generalization of our method.

To test the generalization capability, we interpolate two input boundaries and generated floorplans for each interpolated boundary using our method as shown in Fig. 14. As the input boundary slowly changes, the floorplan structure also changes accordingly and maintains a relatively reasonable spatial division. This demonstrates the desirable generalization capability of *WallPlan*.

Furthermore, we search for the nearest neighbor in the training dataset using 100 boundaries and our results are generally different from the nearest neighbors, indicating our method does not simply memorize the training dataset. The results are given in the supplementary material.

6.3 Quantitative evalution

FID comparison. Quantitative evaluation of the floorplan is not easy, we compare to the state-of-the-art techniques using *Fréchet Inception Distance* (FID) [Heusel et al. 2017], as most state-of-the-art methods [Hu et al. 2020; Nauata et al. 2021; Para et al. 2021] all use FID for evalution. FID is a global metric to calculate the distribution similarity between the generated images and the ground truth. The lower the FID score, the more similar the generated results to the ground truth. Note that the FID score is highly correlated to the example number.

We randomly select 9000 generated floorplans from each method with different constraints and calculate the respective FID score. Table 2 shows an obvious improvement in our FID score compared to all baselines. Given only the boundary as input, the FID score of our generated floorplans is reduced by 45% compared to *RPLAN*, almost reduced by half (1.48 vs. 2.70). The number is decreased by 15% when compared to *Graph2Plan* (1.48 vs. 1.75). When adding the ground truth bubble diagram as the constraint, the FID score of our generated floorplans reached a minimum level, FID = 0.46, which is significantly reduced by 64% compared to *Graph2Plan*, showing a significant improvement in our method (0.46 vs. 1.27). We also compute the FID score to quantitatively assess ablation studies in Section 6.1 and the FID scores further confirm our qualitative analysis and conclusions. Compared to the full method, the FID score has increased by 49% and 35% for the ablation study of WinNet (2.20 vs. 1.48) and decoupled structure (2.00 vs. 1.48), respectively. These results are not surprising, as both the windows and coupled structure provide a global guide to the generation and improve the quality of floorplan design. Each part of *WallPlan* is critical.

To compare the FID score with *House-GAN++*, we freely generate 500 example floorplans constrained on the given bubble diagram using *House-GAN++* and our method. Table 3 shows the results. By comparing, we see that our method is far superior to *House-GAN++* with more than one order of magnitude in terms of the FID score.

[Para et al. 2021] propose a hybrid method composed of the generative model and optimization method to generate floorplans

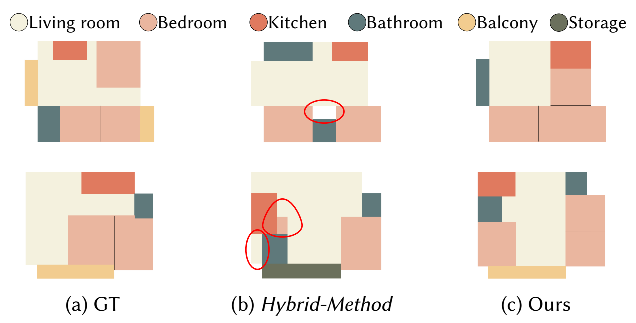


Fig. 16. Comparison to *Hybrid-Method* with the same boundary. (a) The ground truth floorplan. (b) The floorplan generated by *Hybrid-Method*. The red ellipse shows the problematic optimization. (c) The floorplan is generated by our method. We adopt a drawing style similar to *Hybrid-Method*.

(denoted as *Hybrid-Method*). To compare the FID score with *Hybrid-Method*, we directly generate floorplans from the only input boundary. The intersection of the test datasets of our method and *Hybrid-Method* is picked as the examples to calculate the FID score, with a total of 134 floorplans. From Table 4, we see that our method outperforms *Hybrid-Method*. As shown in Fig. 16, *Hybrid-Method* obtains the final floorplan by linear programming, therefore may produce problematic floorplans due to the optimization.

Table 2. FID comparison. For the boundary constraint, *Graph2Plan* uses the retrieved bubble diagram. For the bubble diagram constraint, *Graph2Plan* uses the ground truth bubble diagram. 9000 generated floorplans are selected to calculate the FID score.

Constraint	Method	FID
Boundary	*RPLAN* [Wu et al. 2019]	2.70
	Graph2Plan [Hu et al. 2020]	1.75
	Ours	1.48
Bubble diagram	*Graph2Plan* [Hu et al. 2020]	1.27
	Ours	0.46

Table 3. FID comparison to *House-GAN++* with the only bubble diagram constraint. 500 generated floorplans are selected to calculate the FID score.

Constraint	Method	FID
Bubble diagram	*House-GAN++* [Nauata et al. 2021]	66.18
	Ours	7.76

Table 4. FID comparison to *Hybrid-Method* with the only boundary constraint. 134 generated floorplans are selected to calculate the FID score.

Constraint	Method	FID
Boundary	*Hybrid-Method* [Para et al. 2021]	12.12
	Ours	7.59

Statistics comparison. FID measures the distribution similarity of the floorplan shape and other geometrics but is not good at evaluating floorplan topology. To improve our evaluation, statistics of the generated floorplans by different methods are conducted. Our method is compared with *RPLAN* and *Graph2Plan* since these two methods are more closely related to our work and we have sufficient test data for statistics. We randomly select 9000 example floorplans from each method with different constraints. We collected some statistics: the number of rooms (denoted as R), the number of rooms connected to the living room (denoted as C), the ratio between the number of rooms connected to the living room and the total number of non-living rooms (denoted as C^r), the number of living rooms (denoted as L^n), and the area of the living room (denoted as L^a). We average each statistic over all floorplans in the test dataset and calculate the ratio between each statistic and the corresponding statistic of the ground truth floorplans. The comparison result is shown in Table 5, illustrating that our method outperforms the state-of-the-art techniques in generating the topology of the floorplans.

Table 5. Statistics comparison. Each statistic is the ratio calculated based on the ground truth floorplans. The ratio close to 1 shows that our method can generate the topology similar to the ground truth.

Constraint	Method	R_{avg}	C_{avg}	C^r_{avg}	L^n_{avg}	L^a_{avg}
Boundary	*RPLAN* [Wu et al. 2019]	0.877	0.823	0.949	0.952	0.990
	Graph2Plan [Hu et al. 2020]	0.969	0.926	0.964	1.034	**1.008**
	Ours	**0.983**	**0.970**	**0.995**	**1.002**	1.018
Bubble diagram	*Graph2Plan* [Hu et al. 2020]	0.983	0.945	0.965	1.030	**0.994**
	Ours	**0.984**	**0.974**	**0.993**	**1.001**	1.018

6.4 Perceptual study

We have conducted the perceptual study to evaluate the generation quality and perceived realism of the generated floorplans. We have planned five groups of perceptual studies, and the competitors are *RPLAN* with the only boundary constraint (denoted as *RPLAN*^{boundary}), *Graph2Plan* with the only boundary constraint (denoted as *Graph2Plan*^{boundary}), the ground truth floorplan with the same boundary (denoted as *GT*^{boundary}), *Graph2Plan* with the boundary and ground truth bubble diagram constraint (denoted as *Graph2Plan*^{bubble}) and *House-GAN++* with the only bubble diagram constraint (denoted as *House-GAN++*^{bubble}), respectively. Each group of perceptual studies includes 20 comparison tasks where each pair of floorplans are generated by our method and the competitor. We also add an obviously simple comparison task as the vigilance test for each group. Note that only the user data that passes the vigilance test is valid. For each task, the user is asked to compare a pair of floorplans and choose one floorplan that is more plausible than the other or "Not sure". We recruited 34 participants to finish all of five groups of perceptual studies, all of which are undergraduate and graduate students. At last, 6 participants failed the vigilance test and we have collected 28 valid questionnaire data in total.

Fig. 17 shows a statistical violin plot of the results of perceptual studies, and more detailed voting data is shown in Fig. 18. Of 20 tasks, average 65.9% (AVG 13.18/20, STD 2.60) floorplans of our method are chosen by participants comparing to *Graph2Plan*^{boundary} (AVG 6.00/20, STD 2.62). The proportion 72.5% (AVG 14.50/20, STD 1.90) is higher when comparing to *RPLAN*^{boundary} (AVG 4.75/20, STD 1.76). In the perceptual study, our method obtains a similar score (AVG 8.68/20, STD 2.02) with *GT*^{boundary} (AVG 8.82/20, STD 2.11), which indicates the high quality of our generated floorplans and they are comparable to floorplans in the dataset. When comparing

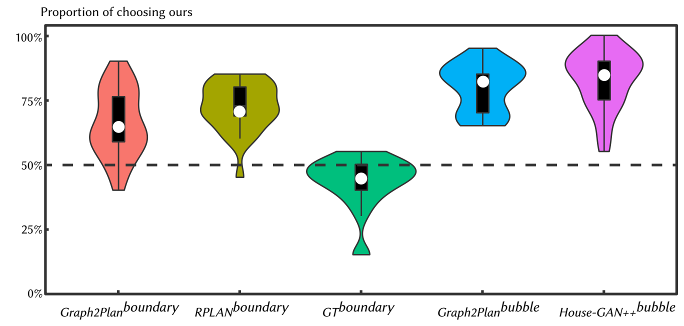


Fig. 17. The violin plot of perceptual studies' results. The X axis indicates comparisons to the five competitors. The Y axis shows the proportion of choosing the floorplans generated by our method, of the 20 tasks in each group of perceptual studies. The dots and the thick black lines indicate means and quartiles respectively. The dashed line on 50% indicates a random guess rate given the forced-choice task.

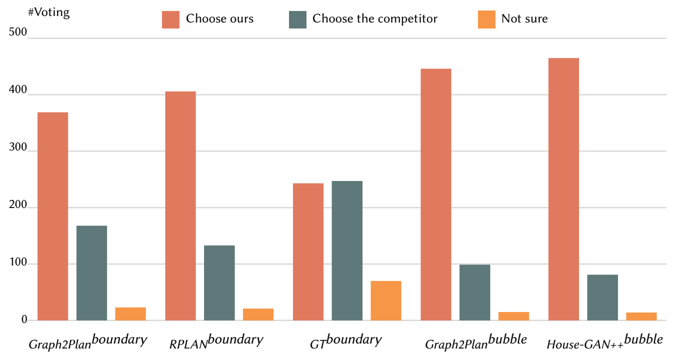


Fig. 18. Voting results of five groups of perceptual studies. Of all 560 valid tasks (20 tasks/participant × 28 participants) for each group, we record the voting number of our method, the competitor, and not sure, respectively.

to *Graph2Plan^{bubble}* and *House-GAN++^{bubble}*, our method shows a huge scoring advantage. Our method almost completely defeats *House-GAN++^{bubble}* with a score of 83.0% (AVG 16.61/20, STD 2.23). The statistical data of perceptual studies are highly consistent with our qualitative analysis in Section 6.2. Average 79.6% (AVG 15.93/20, STD 1.86) floorplans of our method are preferred by participants compared to *Graph2Plan^{bubble}* (AVG 3.54/20, STD 1.91). The results of perceptual studies verify the conclusions of the previous qualitative evaluation.

7 CONCLUSION

We propose a novel wall-oriented method for automatically and efficiently generating plausible floorplans. Different from previous techniques, our method synthesizes floorplans by learning to generate wall graphs. This is achieved by regarding floorplan generation as wall graph generation. A learning-based method *WallPlan* is proposed to generate both the local geometry of the wall and the global semantics of the floorplan by imitating the graph traversal. Intensive experiments demonstrate that our method supports various constraint generation, producing higher quality floorplans than state-of-the-art techniques.

Design constraints. The constrained floorplan generation is enabled by feeding the encoding of inputs to *WallPlan* and training to meet the constraints, instead of regarding the inputs as hard constraints. Therefore, we have no theoretical guarantee that the input constraints must be satisfied, although the case where the constraints are not satisfied is rare in our experiments. Fig. 19(a) show a counterexample of the bubble diagram. One possible solution is to generate subgraphs for each node in the bubble diagram instead of generating one whole graph.

Multiple solutions. Theoretically, *WallPlan* can only find a locally optimal solution for a given input. Thus our models cannot be conveniently sampled to obtain diverse generation results like some generative models such as *House-GAN* and *House-GAN++*. We will explore possible technical routes to diversify floorplan generation.

Door setting. In the generated floorplan, we set room doors for floorplan visualization. We use the rule-based algorithm in previous works: *RPLAN* [Wu et al. 2019] and *Graph2Plan* [Hu et al. 2020]. In addition, the load-bearing wall constraints are not considered in the door setting. Thus there may be a door set in the load-bearing wall. We would like to explore a smarter setting of doors in the future.

Graph generation. Our method is also limited by the type of graph (Fig. 19b). Currently, our method cannot deal with graphs with multiple connected components which will interrupt the continuous traversal process. One straightforward solution is to introduce a new node to start a new round of graph traversal. Besides, we require the graph edge can only be horizontal or vertical, making only axis-aligned floorplans can be generated. Although most floorplans in the real world are axis-aligned, we would like to expand to handle more general situations (Fig. 19c).

Image-based methodology. Technically, the image-based learning method can well represent and encode various space constraints in a unified form and efficiently encode regular spatial relationships of the floorplan. We also think that Graph Convolutional Network, Transformer, and other technologies have much potential to solve this problem. The task of positional graph generation with constraints needs to be redefined. It is exciting to explore these techniques and we leave them as future works.

In the future, we would like to extend the scope and capabilities of our graph-based generative method. We have discussed various floorplan constraint generations in the paper. More design constraints are expected to be considered, which is a critical step to build a truly practical design tool. In addition, it would be interesting to borrow the idea of our method to study other design problems. The proposed method can be further explored in other fields of space planning in computer graphics.

ACKNOWLEDGMENTS

We would like to thank perceptual study participants for evaluating our system and the anonymous reviewers for their constructive suggestions and comments. This work is supported by the National Natural Science Foundation of China (62102126, 62025207, 61972128) and the Fundamental Research Funds for the Central Universities of China (PA2021KCPY0050).

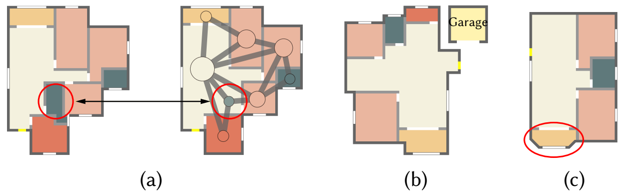


Fig. 19. (a) The bathroom (the red circle) of the bubble diagram is missing in the generated floorplan (Right). The ground truth floorplan is also shown (Left). (b) We cannot directly generate the floorplan with a separated garage. (c) We cannot generate non-axis aligned or curved walls (the red ellipse).

REFERENCES

Scott A Arvin and Donald H House. 2002. Modeling architectural design objectives in physically based space planning. *Automation in Construction* 11, 2 (2002), 213–225.

Fan Bao, Dong-Ming Yan, Niloy J Mitra, and Peter Wonka. 2013. Generating and exploring good building layouts. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Ying Cao, Rynson WH Lau, and Antoni B Chan. 2014. Look over here: Attention-directing composition of manga elements. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

Stanislas Chaillou. 2020. ArchiGAN: Artificial Intelligence x Architecture. In *Architectural Intelligence*. Springer, 117–127.

Guolong Chen, Wenzhong Guo, and Yuzhong Chen. 2010. A PSO-based intelligent decision algorithm for VLSI floorplanning. *Soft Computing* 14, 12 (2010), 1329–1337.

Qi Chen, Qi Wu, Rui Tang, Yuhan Wang, Shuai Wang, and Mingkui Tan. 2020. Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12625–12634.

Lubin Fan and Peter Wonka. 2016. A probabilistic model for exteriors of residential buildings. *ACM Transactions on Graphics (TOG)* 35, 5 (2016), 1–13.

Tian Feng, Lap-Fai Yu, Sai-Kit Yeung, KangKang Yin, and Kun Zhou. 2016. Crowd-driven mid-scale layout design. *ACM Trans. Graph.* 35, 4 (2016), 132–1.

Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9, 1 (2013), 1–22.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems* 30 (2017).

Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2020. Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 118–1.

Graziella Laignel, Nicolas Pozin, Xavier Geffrier, Loukas Delevaux, Florian Brun, and Bastien Dolla. 2021. Floor plan generation through a mixed constraint programming-genetic optimization approach. *Automation in Construction* 123 (2021), 103491.

Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. 2019. Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics (TOG)* 38, 2 (2019), 1–16.

Han Liu, Yong-Liang Yang, Sawsan AlHalawani, and Niloy J Mitra. 2013. Constraint-aware interior layout exploration for pre-cast concrete-based buildings. *The Visual Computer* 29, 6 (2013), 663–673.

Chongyang Ma, Nicholas Vining, Sylvain Lefebvre, and Alla Sheffer. 2014. Game level layout from design specification. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 95–104.

Benachir Medjdoub and Bernard Yannou. 2000. Separating topology and geometry in space planning. *Computer-aided design* 32, 1 (2000), 39–61.

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

Jeremy Michalek, Ruchi Choudhary, and Panos Papalambros. 2002. Architectural layout design optimization. *Engineering optimization* 34, 5 (2002), 461–484.

Jeremy Michalek and Panos Papalambros. 2002. Interactive design optimization of architectural layouts. *Engineering optimization* 34, 5 (2002), 485–501.

Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. 2020. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *European Conference on Computer Vision*. Springer, 162–177.

Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. 2021. House-GAN++: Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13632–13641.

Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2014. Learning layouts for single-pagegraphic designs. *IEEE transactions on visualization and computer graphics* 20, 8 (2014), 1200–1213.

Xufang Pang, Ying Cao, Rynson WH Lau, and Antoni B Chan. 2016. Directing user attention via visual flow on web designs. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–11.

Wamiq Para, Paul Guerrero, Tom Kelly, Leonidas J Guibas, and Peter Wonka. 2021. Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6690–6700.

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2536–2544.

Chi-Han Peng, Yong-Liang Yang, Fan Bao, Daniel Fink, Dong-Ming Yan, Peter Wonka, and Niloy J Mitra. 2016. Computational network design from functional specifications. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–12.

Chi-Han Peng, Yong-Liang Yang, and Peter Wonka. 2014. Computing layouts with deformable templates. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

Daniel Ritchie, Kai Wang, and Yu-an Lin. 2019. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6182–6190.

Julian F Rosser, Gavin Smith, and Jeremy G Morley. 2017. Data-driven estimation of building interior plans. *International Journal of Geographical Information Science* 31, 8 (2017), 1652–1674.

Carl Sechen. 2012. *VLSI placement and global routing using simulated annealing*. Vol. 54. Springer Science & Business Media.

Krishnendra Shekhawat, Nitant Upasani, Sumit Bisht, and Rahil N Jain. 2021. A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in Construction* 127 (2021), 103718.

T Singha, HS Dutta, and M De. 2012. Optimization of floor-planning using genetic algorithm. *Procedia Technology* 4 (2012), 825–829.

Carlos A Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel G Aliaga, and Pascal Müller. 2012. Procedural generation of parcels in urban modeling. In *Computer graphics forum*, Vol. 31. Wiley Online Library, 681–690.

Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. 2019. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. 2018. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Kai Wang, Xianghao Xu, Leon Lei, Selena Ling, Natalie Lindsay, Angel X Chang, Manolis Savva, and Daniel Ritchie. 2021. Roominoes: Generating Novel 3D Floor Plans From Existing 3D Rooms. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 57–69.

Xiao-Yu Wang and Kang Zhang. 2020. Generating layout designs from high-level specifications. *Automation in Construction* 119 (2020), 103288.

Wenming Wu, Lubin Fan, Ligang Liu, and Peter Wonka. 2018. MIQP-based Layout Design for Building Interiors. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 511–521.

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

Xuyong Yang, Tao Mei, Ying-Qing Xu, Yong Rui, and Shipeng Li. 2016. Automatic generation of visual-textual presentation layout. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 12, 2 (2016), 1–22.

Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. 2013. Urban pattern: Layout design by hierarchical domain splitting. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–12.

Yi-Ting Yeh, Katherine Breeden, Lingfeng Yang, Matthew Fisher, and Pat Hanrahan. 2013. Synthesis of tiled patterns using factor graphs. *ACM Transactions on Graphics (TOG)* 32, 1 (2013), 1–13.

Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–11.

Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. 2020. Deep generative modeling for scene synthesis via hybrid representations. *ACM Transactions on Graphics (TOG)* 39, 2 (2020), 1–21.

Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. 2019. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Lichen Zhou, Chuang Zhang, and Ming Wu. 2018. D-linknet: Linknet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 182–186.

Yang Zhou, Zachary While, and Evangelos Kalogerakis. 2019. Scenegraphnet: Neural message passing for 3d indoor scene augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7384–7392.