Intelligent Home 3D: Automatic 3D-House Design from Linguistic Descriptions Only

Qi Chen\textsuperscript{1,2*} Qi Wu\textsuperscript{3} Rui Tang\textsuperscript{4} Yuhan Wang\textsuperscript{4} Shuai Wang\textsuperscript{4} Mingkui Tan\textsuperscript{1†}

\textsuperscript{1}South China University of Technology, \textsuperscript{2}Guangzhou Laboratory, Guangzhou, China
\textsuperscript{3}Australian Centre for Robotic Vision, University of Adelaide, Australia, \textsuperscript{4}Kujiale, Inc.

sechenqi@mail.scut.edu.cn qi.wu01@adelaide.edu.au
{ati,daishu,luorui}@qunhemail.com mingkuitan@scut.edu.cn

Abstract

Home design is a complex task that normally requires architects to finish with their professional skills and tools. It will be fascinating that if one can produce a house plan intuitively without knowing much knowledge about home design and experience of using complex designing tools, for example, via natural language. In this paper, we formulate it as a language conditioned visual content generation problem that is further divided into a floor plan generation and an interior texture (such as floor and wall) synthesis task. The only control signal of the generation process is the linguistic expression given by users that describe the house details. To this end, we propose a House Plan Generative Model (HPGM) that first translates the language input to a structural graph representation and then predicts the layout of rooms with a Graph Conditioned Layout Prediction Network (GC-LPN) and generates the interior texture with a Language Conditioned Texture GAN (LCT-GAN). With some post-processing, the final product of this task is a 3D house model. To train and evaluate our model, we build the first Text–to–3D House Model dataset, which will be released at: https://github.com/chenqi008/HPGM.

1. Introduction

Everyone wants a dream home, but not everyone can design home by themselves. Home design is a complex task that is normally done by certificated architects, who have to receive several years of training on designing, planning and using special designing tools. To design a home, they typically start by collecting a list of requirements for a building layout. Then, they use trial-and-error to generate layouts with a combination of intuition and prior experience. This usually takes from a couple of days to several weeks and has high requirements for professional knowledge.

It will be fantastic if we can design our own home by ourselves. We may not have design knowledge and have no idea how to use those complicated professional designing tools, but we have strong linguistic ability to express our interests and desire. Thus, for time-saving and allowing people without expertise to participate in the design, we propose to use linguistic expressions as the guidance to generate home design plans (Figure 1). Thanks to the fast development of deep learning [6, 9, 11, 45, 46, 51, 52, 53, 55], especially Generative Adversarial Network (GAN) [3, 4, 8, 10] and vision-language research [13, 29, 38, 47, 50], we can turn this problem into a text-to-image generation problem, which has been studied in [22, 30, 32, 43, 44]. However, it is non-trivial to directly apply these methods on our new task because there exist two new technical challenges: 1) A floor plan is a structured layout which pays more attention to the correctness of size, direction, and connection of different blocks, while the conventional text-to-image task focuses more on pixel-level generation accuracy. 2) The interior texture such as floor and wall needs neater and more stable pixel generation than general images and should be...
To tackle the above issues, we propose a House Plan Generative Model (HPGM) to generate home plans from given linguistic descriptions. The HPGM first uses a Stanford Scene Graph Parser [33] to parse the language to a structural graph layout, where nodes represent room types associated with size, room floor (wall) colour and material. Edges between nodes indicate whether rooms are connected or not. We then divide the house plan generation process into two sub-tasks: building layout generation and texture synthesis. Both of them are conditioned on the above extracted structural graph. Specifically, we design a Graph Conditioned Layout Prediction Network (GC-LPN) which applies a Graph Convolutional Network [20] to encode the graph as a feature representation and predicts the room layouts via bounding box regressions. The predicted room layouts are sent to a floor plan post-processing step, which outputs a featured floor plan with doors, windows, walls, etc. To generate floor and wall textures, we design a Language Conditioned Texture GAN (LCT-GAN) that takes the encoded text representations as input and generates texture images with three designed adversarial, material-aware, and colour-aware losses. The generated floor plan and texture images are sent to an auto 3D rendering system to produce the final rendered 3D house plan.

For 3D house generation from linguistic description, we build the first Text-to-3D House Model dataset that contains a 2D floor plan and two texture (floor and wall) patches for each room in the house. We evaluate the room layout generation and texture generation ability of our model separately. The room layout accuracy is evaluated based on the IoU (Intersection over Union) between the predicted room bounding boxes and the ground-truth annotation. The generated interior textures are evaluated with popular image generation metrics such as Fréchet Inception Distance (FID) [12] and Multi-scale Structural Similarity (MS-SSIM) [39]. Our proposed GC-LPN and LCT-GAN outperform the baseline methods in a large margin. Besides, a generalisation ability evaluation of our LCT-GAN is carried out. We also perform a human evaluation on our final products – 3D house plans, which shows 39.41% pass it.

We highlight our principal contributions as follows:

- We propose a novel architecture, called House Plan Generative Model (HPGM), which is able to generate 3D house models with given linguistic expressions. To reduce the difficulty, we divide the generation task into two sub-tasks to generate floor plans and interior textures, separately.
- To achieve the goal of synthesising 3D building model from the text, we collect a new dataset consisting of the building layouts, texture images, and their corresponding natural language expressions.
- Extensive experiments show the effectiveness of our proposed method on both qualitative and quantitative metrics. We also study the generalisation ability of the proposed method by generating unseen data with the given new texts.

2. Related Work

**Building layout design.** Several existing methods have been proposed for generating building layouts automatically [1, 5, 24, 28, 40]. However, most of these methods generate the building layouts by merely adjusting the interior edges in a given building outline. Specifically, Merrel et al. [24] generate residential building layouts using a Bayesian network trained in architectural programs. Based on an initial layout, Bao et al. [1] formulate a constrained optimisation to characterise the local shape spaces and then link them to a portal graph to obtain the objective layout. Peng et al. [28] devise a framework to yield the floor plan by tiling an arbitrarily shaped building outline with a set of deformable templates. Wu et al. [40] develop a framework that generates building interiors with high-level requirements. More recently, Wu et al. [41] propose a data-driven floor plan generating system by learning thousands of samples. However, the above methods require either a given building outline or a detailed structured representation as the input while we generate the room layouts with human verbal commands.

**Texture synthesis.** Many existing works in terms of texture generation focus on transferring a given image into a new texture style [16, 21, 36] or synthesising a new texture image based on the input texture [7, 35, 42]. Different from that, we aim to solve the problem that generates texture images with given linguistic expressions. The closest alternative to our task is texture generation from random noise [2, 15]. Specifically, Jetchev et al. [15] propose a texture synthesis method based on GANs, which can learn a generating process from the given example images. Recently, to obtain more impressive images, Bergmann et al. [2] incorporate the periodical information into the generative model, which makes the model have the ability to synthesise periodic texture seamlessly. Even if these methods have a strong ability to produce plausible images, they have limited real-world applications due to the uncontrollable and randomly generated results. We use natural language as the control signal for texture generation.

**Text to image generation.** For generating an image from text, many GAN-based methods [17, 23, 30, 32, 43, 44, 48, 49, 54] have been proposed in this area. Reed et al. [32] transform the given sentence into text embedding and then generate image conditioning on the extracted embedding. Furthermore, to yield more realistic images, Zhang et al. [49] propose a hierarchical network, called StackGAN, which generates images with different sizes (from coarse to
fine). Meanwhile, they introduce a conditioning augmentation method to avoid the discontinuity in the latent manifold of text embedding. Based on StackGAN, Xu et al. [43] develop an attention mechanism, which ensures the alignment between generated fine-grained images and the corresponding word-level conditions. More recently, to preserve the semantic consistency, Qiao et al. [30] consider both text-to-image and image-to-text problems jointly.

3. Proposed Method

In this paper, we focus on 3D-house generation from requirements, which seeks to design a 3D building automatically conditioned on the given linguistic descriptions. Due to the intrinsic complexity of 3D-house design, we divide the generation process into two sub-tasks: building layout generation and texture synthesis, which produce floor plan and corresponding room features (i.e., textures of each room), respectively.

To complete the above two tasks, we propose a House Plan Generative Model (HPGM) to automatically generate a 3D home design conditioned on given descriptions. As shown in Figure 2, the proposed HPGM consists of five components: 1) text representation block, 2) graph conditioned layout prediction network (GC-LPN), 3) floor plan post-processing, 4) language conditioned texture GAN (LCT-GAN), and 5) 3D scene generation and rendering.

In Figure 2, the text representation is to capture the structural text information from given texts using a Stanford Scene Graph Parser [33]. Based on the text representations, GC-LPN is devised to produce a coarse building layout. To obtain a real-world 2D floor plan, we send the generated layout to a floor plan post-processing step to refine the coarse building layout to yield a floor plan with windows and doors. To synthesise the interior textures of each room, we further devise a Language Conditioned Texture GAN (LCT-GAN) to yield the controllable and neat images according to the semantic text representations. Last, we feed the generated floor plan with room features into a 3D rendering system for 3D scene generation and rendering. The details of each component are depicted below.

3.1. Text Representation

The linguistic descriptions of the building include the description of the number of rooms and room types, followed by the connections between rooms, and the designing patterns of each room. Although it follows a weakly structural format, directly using the template-based language parser is impractical due to the diversity of the linguistic descriptions. Instead, we employ the Stanford Scene Graph Parser [33] with some post-processing and merging to parse the linguistic descriptions to a structural graph format. For such a constructed graph, each node is a room with some properties (e.g., the room type, size, interior textures). The edge between nodes indicates the connectivity of two rooms. More details of the scene graph parser can be found in the supplementary materials.

We use different representation as inputs in building layout generation and texture synthesis, since these two tasks require different semantic information. In building layout generation, we define input vectors as $X \in \mathbb{R}^{N \times D}$, where $N$ refers to the number of nodes (i.e., rooms) in each layout and $D$ denotes the feature dimension. Each node feature $x_i = \{\alpha_i, \beta_i, \gamma_i\} \in \mathbb{R}^D$ is a triplet, where $\alpha_i$ is the type of room (e.g., bedroom), $\beta_i$ is the size (e.g., 20 squares) and $\gamma_i$ is the position (e.g., southwest). All features are encoded as one-hot vectors except the size is a real value. Moreover, to exploit the topological information elaborately, following [20], we convert the input features $X$ to an undirected graph $\mathcal{G}$ via introducing an adjacency matrix $A \in \mathbb{R}^{N \times N}$.

In the texture synthesis task, for a given text, we trans-
form the linguistic expression to a collection of vectors \( V \in \mathbb{R}^{2N \times M} \), where \( 2N \) refers to the number of textures in each layout and \( M \) denotes the dimension of each feature vector. For \( v_i \in \mathbb{R}^M \), we design \( v_i = \{ p_i, q_i \} \), where \( p_i \) indicates the material (e.g., log, mosaic or stone brick) and \( q_i \) refers to the colour. We pre-build a material and colour word vocabulary from training data so that we can classify the parsed attributes into the material or colour set.

3.2. Graph Conditioned Layout Prediction Network

To generate the building layouts satisfying the requirements, we propose a Graph Conditioned Layout Prediction Network (GC-LPN). We incorporate the adjacent information into the extracted features via a GCN, which facilitates the performance when generating the objective layouts.

Graph convolutional network. In order to process the aforementioned graphs in an end-to-end manner, we use a graph convolutional network composed of two graph convolutional layers. Specifically, we take the feature matrix \( X \in \mathbb{R}^{N \times D} \) as inputs and produce a new feature matrix, where each output vector is an aggregation of a local neighbourhood of its corresponding input vector. In this way, we obtain a new feature matrix, which introduces the information across local neighbourhoods of the inputs. Note that, since we only focus on generating the layouts of resident building, the order and size of corresponding graph are small. Therefore, it is sufficient to leverage a two-layer GCN model (as shown in Figure 2) when introducing the information of adjacent rooms. Mathematically, we have

\[
Y = g(X, A) = \text{Softmax} \left( \text{ReLU} \left( AXW_0 \right) W_1 \right),
\]

where \( W_0 \in \mathbb{R}^{D \times D} \) and \( W_1 \in \mathbb{R}^{D \times D} \) are the weights of two graph convolutional layers. Note that the adjacency matrix \( A \) only contains 1 and 0, which indicates whether pairs of nodes (rooms) are adjacent or not. \( Y \in \mathbb{R}^{N \times D} \) is the structured feature. Then, we add the extracted feature \( Y \) with the input feature \( X \) to get the feature \( S \in \mathbb{R}^{N \times D} \):

\[
S = X \odot Y,
\]

where “\( \odot \)” is the element-wise addition.

Bounding box regression. After reasoning on the graph with GCNs, we gain a set of embedding vectors, where each vector aggregates the information across the adjacent rooms. In order to produce the building layout, we must transform these vectors from the graph domain to the image domain. Thus, we define each room as a coarse 2D bounding box, which can be represented as \( b_i = (x_0, y_0, x_1, y_1) \). In this way, we cast the problem to that bounding box generation from given room embedding vectors.

In practice, we first feed the well-designed feature \( S \) into a two-layer perceptron network \( h(\cdot) \) and predict the corresponding bounding box of each node \( b_i = h(S_i) = (\hat{x}_0, \hat{y}_0, \hat{x}_1, \hat{y}_1) \). Then, we integrate all the predicted boxes and obtain the corresponding building layout. For training the proposed model, we minimise the objective function

\[
L_B = \frac{1}{N} \sum_{i=1}^{N} \| \hat{b}_i - b_i \|_2^2,
\]

where \( b_i \) is the ground-truth bounding box for \( i \)-th node (i.e., the bounding box that covers the room).

3.3. Floor Plan Post-processing

To transform the bounding box layout to a real-world 2D floor plan, we propose a floor plan post-processing (shown in Figure 3), which consists of five steps, i.e., (a)–(e). To be specific, in Step (a), we first extract boundary lines of all generated bounding boxes and then merge the adjacent segments together in Step (b). In Step (c), we further align the line segments with each other to obtain the closed polygon. In Step (d), we judge the belonging of each closed polygon based on a weight function:

\[
W_{ij} = \int \int \frac{1}{w_i h_i} \exp \left( -\frac{(x_j - c_{e_i})^2}{w_i} - \frac{(y_j - c_{e_i})^2}{h_i} \right) \, dx_j dy_j,
\]

where \( i = 1, 2, \ldots, n \) is the \( i \)-th original box (room) while \( j = 1, 2, \ldots, m \) is the \( j \)-th aligned polygon. \( W_{ij} \) denotes the weight of \( j \)-th polygon belonging to room \( i \). \( c_{e_i} \) and \( c_{w_i} \) indicate the central position while \( w_i \) and \( h_i \) are the half width and height of the \( i \)-th bounding box. \( x_j \) and \( y_j \) are the coordinates in the aligned polygon. We assign the \( j \)-th polygon with the room type, according to the corresponding original bounding box, which has maximum weight \( W \).

Finally, in Step (e), we apply a simple rule-based method to add doors and windows in rooms. Specifically, a door or open wall is added between the living room and any other room. We set the window on the longest wall of each room and set the entrance on the wall of the biggest living room. We find these rules work in most cases and good enough to set reasonable positions, but a learning-based method may improve this process and we leave it as the future work.

3.4. Language Conditioned Texture GAN

For better controlling the details of textures, we consider the texture images in terms of two fields, i.e., material and
we set $\lambda_1$ and $\lambda_2$ to 1 by default. We will elaborate on the modules that lead to these losses in the following sections.

**Adversarial loss.** To synthesise the natural images, we follow the traditional GAN [8], where the generator $G$ and discriminator $D$ compete in a two-player minimax game. Specifically, the generator $G$ tries to fool the discriminator $D$ while $D$ tries to distinguish whether the given image is real or fake/generated. Based on that, for our task, when optimising the discriminator $D$, we minimise the loss

$$L_{DA} = -\mathbb{E}_R [\log D(R)] - \mathbb{E}_{Z \sim P_z} [\log (1 - D(G(Z)))].$$

(6)

where $P_r$ and $P_z$ denote the distributions of real samples and noise, respectively. $Z$ refers to the input of $G$, as mentioned before, consisting of noise $Z'$ and conditions $p$ and $q$. On the other hand, when optimising network $G$, we use

$$L_{Adv} = -\mathbb{E}_{Z' \sim P_z} [\log D(G(Z'))].$$

(7)

**Material-aware loss.** To preserve the semantic alignment between generated textures and given texts, we propose a material-aware loss, which is sensitive to fine-grained material categories. To be specific, as mentioned in Section 3.1, we transform the linguistic descriptions to a structural format, which includes a label for each node to indicate its floor/wall material categories. We then add a material classifier on top of $D$, called $\phi$, which imposes the generated texture into the right category. In this way, we obtain the posterior probability $\phi(c_m | \cdot)$ of each entry image, where $c_m$ refers to the category of material. Thus, we minimise the training loss of $G$ and $D$ as

$$L_m = -\mathbb{E}_R [\log \phi(c_m | R)] - \mathbb{E}_{Z' \sim P_z} [\log \phi(c_m | G(Z'))].$$

(8)

**Colour-aware loss.** Similar to the above material-aware loss, instead of focusing on materials, colour-aware loss pays more attention to colour categories. Based on the given expressions of texture colour, we cast the colour alignment as a classification problem. Specifically, we reuse the discriminator $D$ as the feature extractor and replace the last layer to be a colour classifier $\varphi$. Then, in both $G$ and $D$, we try to minimise the loss

$$L_C = -\mathbb{E}_R [\log \varphi(c_i | R)] - \mathbb{E}_{Z' \sim P_z} [\log \varphi(c_i | G(Z'))],$$

(9)

where $\varphi(c_i | R)$ is the posterior probability conditioning on the given texture image $R$.

3.5. 3D Scene Generation and Rendering

For the better visualisation of the generated floor plan with textures, we introduce a 3D scene generator followed with a photo-realistic rendering process. Given generated floor plan and textures as shown in Figure 5, we generate walls from boundaries of rooms with fixed height and thickness. We set the height of walls to 2.85m and the thickness of interior walls to 120mm. The thickness of the exterior

---

Figure 4: Architecture of LCT-GAN. Generator $G$ transforms a conditional noise $Z$ into an RGB image $G(Z)$ with fully convolutional layers. Discriminator $D$, which is used to distinguish fake data from real ones, is fed either fake image $G(Z)$ or real image $R$. Two classifiers ($\phi$ and $\varphi$) have been added on top of $D$, which are used to impose the image into the right material and colour categories, respectively.

**Texture generator.** We first obtain the input noise $Z' \in \mathbb{R}^{w \times h \times d_1}$ from a Gaussian distribution $\mathcal{N}(0, I)$. After that, to incorporate the conditional information, we extend the aforementioned material and colour vectors $p \in \mathbb{R}^{1 \times 1 \times d_2}$ and $q \in \mathbb{R}^{1 \times 1 \times d_3}$ as the same size with the noise $Z'$ and then concatenate them together to obtain the objective input $Z \in \mathbb{R}^{w \times h \times (d_1 + d_2 + d_3)}$.

Conditioning on the input tensor $Z$, we generate the corresponding texture image by $G(Z) \in \mathbb{R}^{W \times H \times 3}$, where $W$ and $H$ denote the width and height of the generated image, respectively. Note that, in order to generate arbitrary size of texture, we design the generator $G$ with a fully convolutional network (FCN), which allows input $Z$ with various sizes when inferring. In practice, we establish our FCN model with only five blocks, where each block consists of a $2 \times 2$ upsampling interpolation, a convolutional layer, a batch normalisation [14] and an activation function.

On the other hand, to generate texture from an expression, the generator $G$ must: 1) ensure the generated images are natural and realistic; and 2) preserve the semantic alignment between given texts and texture images. To satisfy the above requirements, we propose an optimisation mechanism consisting of three losses $L_{Adv}$, $L_m$ and $L_C$, which indicate the adversarial loss, material-aware loss and colour-aware loss, respectively. Overall, the final objective function of the texture generator $G$ is

$$L_G = L_{Adv} + \lambda_1 L_m + \lambda_2 L_C,$$

(5)

where $\lambda_1$ and $\lambda_2$ are trade-off parameters. In experiments,
wall is set to 240mm while the length of the door is 900mm and the height is 2000mm. We simply set the length of the window to thirty percent of the length of the wall it belongs to. Besides, we develop a photo-realistic rendering based on Intel Embree [37], an open-source collection of high-performance ray tracing kernels for x86 CPUs. Photo-realistic renderer is implemented with Monte Carlo path tracing. By following the render equation [18], the path tracer simulates real-world effects such as realistic material appearance, soft shadows, indirect lighting, ambient occlusion and global illumination. In order to visualise the synthetic scenes, we deploy a virtual camera on the front top of each scene and capture a top-view render image.

4. Experiments

4.1. Experimental Settings

Dataset. To generate 3D building models from natural language descriptions, we collect a new dataset, which contains 2,000 houses, 13,478 rooms and 873\textsuperscript{1} texture images with corresponding natural language descriptions. These descriptions are firstly generated from some pre-defined templates and then refined by human workers. The average length of the description is 173.73 and there are 193 unique words. In our experiments, we use 1,600 pairs for training while 400 for testing in the building layout generation. For texture synthesis, we use 503 data for training and 370 data for testing. We put more dataset analysis in supplementary.

Evaluation metrics. We quantitatively evaluate our model and compare it with other models in threefold: layout generation accuracy, texture synthesis performance, and final 3D house floor plans. We measure the precision of the generated layout by Intersection-over-Union (IoU), which indicates the overlap between the generated box and ground-truth one, where the value is from 0 to 1. For the evaluation of textures, we use Fréchet Inception Distance (FID) [12]. In general, the smaller this value is, the better performance the method will have. Besides, to test the pair-wise similarity of generated images and identify mode collapses reliably [26], we use Multi-scale Structural Similarity (MS-SSIM) [39] for further validation. A lower score indicates a higher diversity of generated images (i.e., fewer model collapses). Note that, following the settings in [48], for a fair comparison, we resize all the images to 64 × 64 before computing FID and MS-SSIM. For the 3D house floor plans, which are our final products, we run a human study to evaluate them.

Implementation details. In practice, we set input \( Z \in \mathbb{R}^{w \times h \times (d_1 + d_2 + d_3)} \) of LCT-GAN with \( h = 5, w = 5, d_1 = 100, d_2 = 19 \) and \( d_3 = 12 \). All the weights of models (GC-LPN and LCT-GAN) are initialised from a normal distribution with zero-mean and standard deviation of 0.02. In training, we use Adam [19] with \( \beta_1 = 0.5 \) to update the model parameters of both GC-LPN and LCT-GAN. We optimise our LCT-GAN to generate texture images of size 160 × 160 with mini-batch size 24 and learning rate 0.0002.

4.2. Building Layout Generation Results

Compared methods. We evaluate the generated layout and compare the results with baseline methods. However, there is no existing work on our proposed text-guided layout generation task, which focuses on generating building layouts directly from given linguistic descriptions. Therefore, our comparisons are mainly to ablated versions of our proposed network. The compared methods are:

MLG: In “Manually Layout Generation” (MLG), we draw the building layouts directly using a program with the predefined rules, according to the given input attributes, such as type, position and size of the rooms. Specifically, we first roughly locate the central coordinates of each room conditioning on the positions. After that, we randomly pick the aspect ratio \( \rho \in (\frac{4}{3}, \frac{3}{4}) \) for different rooms, and then get the exact height and width by considering the size of each room. Finally, we draw the building layouts with such centre, height, width and type of each room.

C-LPN: In “Conditional Layout Prediction Network” (C-LPN), we simply remove the GCN in our proposed model. That means, when generating building layouts, the simplified model can only consider the input descriptions and ignore the information from neighbourhood nodes.

RC-LPN: In “Recurrent Conditional Layout Prediction Network” (RC-LPN), we yield the outline box of rooms sequentially like [34]. To be specific, we replace GCN with an LSTM and predict the building layout by tracking the history of what has been generated so far.

Quantitative evaluation. We evaluate the performance of our proposed GC-LPN by calculating the average IoU value of the generated building layouts. From Table 1, compared with the baseline methods, GC-LPN obtains higher value in IoU, which implies that the GC-LPN has the capacity to locate the outline of layout more precisely than other approaches. Models without our graph-based representation,
such as C-LPN and RC-LPN, have lower performance.

**Qualitative evaluation.** Moreover, we investigate the performance of our GC-LPN by visual comparison. From Figure 6, we provide two layout samples corresponding to “Text1” and “Text2” respectively. The results show that compared with the baseline methods, GC-LPN obtains more accurate layouts, whether simple or complex. We also present the generated 2D floor plans after post-processing, and the corresponding ground-truths in Figure 6.

![Figure 6: Visual comparisons between GC-LPN and baselines.](image)

In this section, we conduct two experiments to verify the generalisation ability of our proposed method. We first investigate the landscape of the latent space. Following the setting in [31], we conduct the linear interpolations between two input embeddings and feed them into the generator \( G \). As shown in Figure 9, the generated textures change smoothly when the input semantics (i.e., material or color) vary. On the other hand, to further evaluate the generalisation ability of our LCT-GAN, we feed some novel descriptions, which are not likely to be seen in the real world, into the generator \( G \). From Figure 9, the generated textures change smoothly when the input semantics (i.e., material or color) vary.

### 4.3. Texture Synthesis Results

**Compared methods.** For the conditional texture generation task, we compare the performance of our proposed method with several baselines, including ACGAN [26], StackGAN-v2 [48] and PSGAN [2]. Note that PSGAN can only generate image from random noise. Thus, to generate images in a controlled way, we design a variant of PSGAN, which introduces the conditional information when synthesising the objective texture like [25].

**Quantitative evaluation.** In this part, we compare the performance of different methods on our proposed dataset in terms of FID and MS-SSIM. In Table 2, our LCT-GAN achieves the best performance in FID, which implies that our method is able to yield more photo-realistic images than others. Moreover, for MS-SSIM, our LCT-GAN obtains the competitive result compared with PSGAN, which is also designed specifically for texture generation. It suggests that our method has the ability to ensure the diversity of synthesised images when preserving realism.

**Ablation studies.** To test the effect of each proposed loss, we conduct an ablation study to compare the generated results by removing some losses and show the quantitative results in Table 3. Note that the model only using adversarial loss \( L_{Adv} \) can not yield controllable images. Thus, we combine \( L_{Adv} \) with the other two losses (i.e., \( L_M \) and \( L_C \)) to investigate the performance. The results show that based on \( L_{Adv}, L_M \) and \( L_C \) are able to improve the performance very well. When using all the three losses into our model, we obtain the best results on both FID and MS-SSIM.

**Generalisation ability.** In this section, we conduct two experiments to verify the generalisation ability of our proposed method. We first investigate the landscape of the latent space. Following the setting in [31], we conduct the linear interpolations between two input embeddings and feed them into the generator \( G \). As shown in Figure 9, the generated textures change smoothly when the input semantics (i.e., material or color) vary. On the other hand, to further evaluate the generalisation ability of our LCT-GAN, we feed some novel descriptions, which are not likely to be seen in the real world, into the generator \( G \). From Figure 9, the generated textures change smoothly when the input semantics (i.e., material or color) vary.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Train Set FID</th>
<th>MS-SSIM</th>
<th>Test Set FID</th>
<th>MS-SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACGAN [26]</td>
<td>198.07</td>
<td>0.4584</td>
<td>220.18</td>
<td>0.4601</td>
</tr>
<tr>
<td>StackGAN-v2 [48]</td>
<td>182.96</td>
<td>0.6356</td>
<td>168.15</td>
<td>0.6225</td>
</tr>
<tr>
<td>PSGAN [2]</td>
<td>195.29</td>
<td>0.4162</td>
<td>217.12</td>
<td>0.4187</td>
</tr>
<tr>
<td>LCT-GAN (ours)</td>
<td>119.33</td>
<td>0.3944</td>
<td>145.16</td>
<td>0.3859</td>
</tr>
</tbody>
</table>

### Table 1: IoU results on Text-to-3D House Model dataset.

<table>
<thead>
<tr>
<th>Methods</th>
<th>FID</th>
<th>MS-SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCT-GAN (ours)</td>
<td>119.33</td>
<td>0.3944</td>
</tr>
</tbody>
</table>

### Table 2: FID and MS-SSIM results of generated textures.
Table 3: Impact of losses in conditional texture generation.

<table>
<thead>
<tr>
<th>LCT-GAN</th>
<th>L_{A,0}</th>
<th>L_{M}</th>
<th>L_{C}</th>
<th>Train Set</th>
<th>Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>MS-SSIM</td>
<td>FID</td>
<td>MS-SSIM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>√</td>
<td>√</td>
<td>134.06</td>
<td>0.4189</td>
<td>157.01</td>
<td>0.4191</td>
</tr>
<tr>
<td>√</td>
<td>√</td>
<td>134.61</td>
<td>0.4310</td>
<td>158.20</td>
<td>0.4263</td>
</tr>
<tr>
<td>√</td>
<td>√</td>
<td>119.33</td>
<td>0.3944</td>
<td>145.16</td>
<td>0.3859</td>
</tr>
</tbody>
</table>

Table 4: Results of HPGM v.s. human. “Tie” refers to the confusing results, which can not be clearly distinguished.

<table>
<thead>
<tr>
<th>Choice (%)</th>
<th>HPGM (ours)</th>
<th>Human</th>
<th>Tie</th>
</tr>
</thead>
<tbody>
<tr>
<td>39.41</td>
<td>47.94</td>
<td>12.65</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Interpolation results of generated texture images.

Figure 10: Generated textures with novel material-colour scenarios, which are impossible existing in the real world.

Figure 11: Comparison of our generated 3D house plans with ground-truth (human-made) counterparts.

4.4. 3D House Design

**Qualitative results.** For quality evaluation, we show the 3D house plans (in Figure 11) generated by our HPGM and the ground-truth counterparts with conditional text[^1], where the floor plan and corresponding room textures are drawn by architects. Our method has ability to produce competitive visual results, even compared with the human-made plans.

**Human study.** Since the automatic metrics can not fully evaluate the performance of our method, we perform a human study on the house plans. Inspired by [23, 27], we conduct a pairwise comparison between HPGM and human beings, using 100 house plans pairs with their corresponding descriptions. Then, we ask 20 human subjects (university students) to distinguish which is designed by human beings. Finally, we calculate the ratio of choice and obtain the final metrics. From Table 4, 39.41% generated samples pass the exam, which implies that compared with the manually designed samples, the machine-generated ones are exquisite enough to confuse the evaluators.

5. Conclusion

3D house generation from linguistic descriptions is non-trivial due to the intrinsic complexity. In this paper, we propose a novel House Plan Generative Model (HPGM), dividing the generation process into two sub-task: building layout generation and texture synthesis. To tackle these problems, we propose two modules (i.e., GC-LPN and LCT-GAN), which focus on producing floor plan and corresponding interior textures from given descriptions. To verify the effectiveness of our method, we conduct a series of experiments, including quantitative and qualitative evaluations, ablation study, human study, etc. The results show that our method performs better than the competitors, which indicates the value of our approach. We believe this will be a practical application with further polish.

**Acknowledgments**

This work was partially supported by Guangdong Provincial Scientific and Technological Funds under Grant 2018B010107001 and Grant 2019B010155002, National Natural Science Foundation of China (NSFC) 61836003 (key project), Fundamental Research Funds for the Central Universities under Grant D2191240, Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X183, Tencent AI Lab Rhino-Bird Focused Research Program under Grant JR201902, Guangdong Special Branch Plans Young Talent with Scientific and Technological Innovation under Grant 2016TQ03X445, Guangzhou Science and Technology Planning Project under Grant 201904010197, Natural Science Foundation of Guangdong Province under Grant 2016A030313437.
References


