Generating 3D House Wireframes with Semantics
(Supplementary Materials)

Xueqi Ma, Yilin Liu, Wenjun Zhou, Ruowei Wang, and Hui Huang*

Visual Computing Research Center, Shenzhen University
hhzhiyan@gmail.com

1 Data

3D House Wireframe Dataset: Existing datasets for house models primarily feature 2D floor plans, which do not suffice for creating 3D house wireframes. Recognizing this deficiency, we have developed a comprehensive 3D house wireframe dataset. A typical 3D wireframe of a house encompasses 3 main components: the roof, exterior walls, and interior rooms. Our methodology initiated with extracting the 2D layout for critical room and exterior wall segments from the RPLAN dataset [10]. Subsequent steps involved lifting the corner points of these segments to establish the groundwork for the house's basic wireframe structure. To complete the wireframe model, we utilized the straight skeleton algorithm [1] on the exterior walls, facilitating the construction of the roof's wireframe. Our analysis focused on wireframes containing fewer than 400 line segments, as depicted in Fig. 1. This selection process resulted in the accumulation of 78,791 wireframes. We allocated these wireframes into training and test sets following a 9:1 ratio and normalized each wireframe to ensure its central alignment at the origin within a cubic space measured as $[-1, 1]^3$. For more information on downloading and using the dataset, please visit our GitHub repository: https://github.com/3d-house-wireframe/3d-house-wireframe-dataset.

Data augmentation: Throughout the training phase, we applied various data augmentation techniques to improve the model's generalization ability. Our augmentation strategies included: 1) Rotating the wireframes at predetermined angles, specifically $\{0°, 90°, 180°, 270°\}$, to simulate different orientations; 2) Applying mirror flips across the YOZ plane to reflect the wireframes, thereby increasing the diversity of the training data; 3) Adjusting the scale and position of the wireframe vertices in the x, y, and z dimensions. The scaling adjustments were confined to a range of $[0.9, 1.1]$, and translations were applied within a range of $[-0.1, 0.1]$. These transformations were intended to simulate variations in size and spatial alignment, further contributing to the robustness of the training.

* Corresponding author

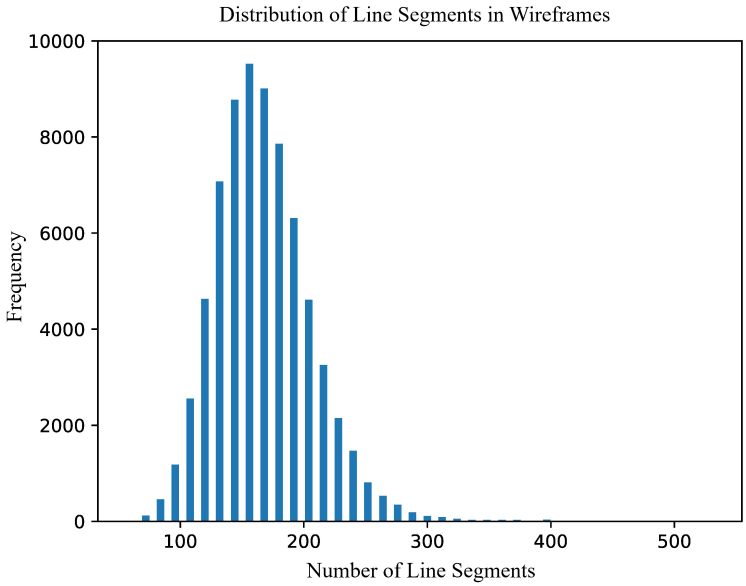

Fig. 1: The distribution of the number of line segments in wireframes within the 3D house wireframe dataset. The horizontal axis represents the count of line segments, and the vertical axis shows the number of wireframe samples with that count.

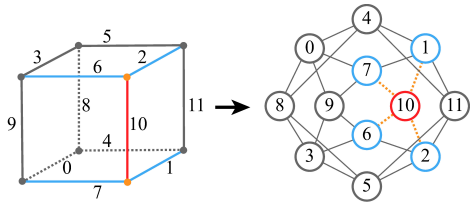

Fig. 2: The process of transforming a wireframe into a graph.

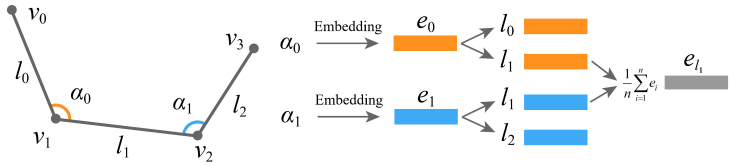

Fig. 3: Embedding of angular features between adjacent line segments.

2 Method Details

2.1 Autoencoder

Graph construction for feature learning: To facilitate effective feature learning of individual line segments, we convert the 3D wireframe models into graph representations, where nodes represent line segments and edges represent junctions. As illustrated in Fig. 2, the left part of the figure presents the cube's wireframe, indicating each line segment with specific indices. Conversely, the right part displays the generated graph. Here, a particular line segment from the wireframe, such as segment 10 depicted in red, aligns with node 10 in the graph. The segments that connect to it, identified by indices 1, 2, 6, and 7 in the wireframe, are mirrored as connected nodes in blue within the graph. Furthermore, the graph visualizes the junctions, shown in yellow on the wireframe, as edges in the graph, effectively mapping the structural connections between line segments.

Features: Our encoder consists of a graph convolution module and a Local Multi-Head Attention module. We introduced multiple features for each node in the graph (representing a line segment). These features include 6 coordinates of the line segment, its length, its direction, and the 3 coordinates of the segment's midpoint. To further enhance the node features, we calculate the angles between each pair of adjacent line segments and integrate these angular features into the corresponding line segment features, as shown in Fig. 3.

Network: In constructing our neural network model, we first introduced a 5-layer Graph Convolutional Network (GCN) [3] as the initial encoder E_G. The feature dimensions of these layers are $[64, 128, 256, 256, 384]$. Subsequently, we added a 4-layer Local Multi-Head Attention (LMH Attention) module E_A $[2, 8]$ to the encoder, with each layer having a dimension of 384. In the decoder, we first employed a 2-layer LMH Attention module D_A with each layer dimensioned at 384, followed by a 1D ResNet34 [4] as the second module D_R. D_R comprises four sets of residual blocks, with the number of blocks in each set being $[3, 4, 6, 3]$, and the feature dimensions sequentially are $[128, 192, 256, 384]$.

Our decoder is designed to map the features of line segments into a 128^3 cubic space, facilitating the generation of discretized line segments. The output comprises the logits of 6 discrete coordinates for each line segment in this cubic space. For the LMH Attention, we use a window size of 64 and a dimension of 32 for each head. Additionally, the size of our codebook is set at 8192.

For the transformer model, we implemented a phased strategy that progresses from coarse to fine [5] for autoregressively predicting the indices in the codebook. In this process, the transformer at the coarse stage is configured with 12 layers, a feature dimension of 512, and 8 heads. Following this, the transformer at the fine stage consists of 2 layers with a feature dimension of 512 and 8 heads. Both MeshGPT and our model have a maximum sequence length of 1624, whereas PolyGen's maximum is 816 for both vertices and segments due to its two-stage

generation process. The temperature of all methods is set to 1.0, and the generation process stops when the termination symbol is predicted or the maximum sequence length is reached.

2.2 Residual LFQ

As shown in MeshGPT [9], the quantized feature is crucial for the transformer model to predict high-quality 3D models. We used Residual LFQ [5,11] to quantize the vertex features of the line segments. Residual LFQ quantizes vertex features by treating them as Cartesian products of single-dimensional variables. Specifically, for a vertex feature vector z, its quantized representation $f(z)$ is defined as the value closest to each dimension of z in the codebook C_i. Since each C_i only contains two values, -1 and 1, the quantized result for each dimension $f(z_i)$ can be directly determined by the sign of z_i:

$$f(z_i) = \mathbf{sign}(z_i) = \begin{cases} -1 & \text{if } z_i \leq 0 \\ 1 & \text{if } z_i > 0, \end{cases} \tag{1}$$

where z_i is the ith dimension of z.

LFQ eliminates the need for the codebook lookup step typically required in traditional quantization, as each dimension's quantization index is obtained simply by $f(z_i) = \text{sign}(z_i)$. The token index for $f(z)$ is then calculated by

$$\text{Index}(z) = \sum_{i=1}^{n} 2^{i-1} \cdot \mathbb{I}\{z_i > 0\}, n = \log_2 K, \tag{2}$$

where K is the codebook size, and $\mathbb{I}\{z_i > 0\}$ is the indicator function, which equals 1 if $z_i > 0$, and 0 otherwise.

We adopt commit loss [5] to impose constraints on the quantization process. Additionally, to enhance the utilization of the codebook, we employ an entropy penalty [11]. This not only aids the network in making more confident predictions but also encourages using more codes from the codebook.

2.3 Loss Function for Autoencoder

In our method, a line segment consists of two vertices, A and B, with each vertex's coordinate predicted from a discrete set of possible values ranging from 0 to 127. We use the cross-entropy loss function to optimize the autoencoder, which measures the discrepancy between the predicted probabilities and the ground truth discrete coordinates.

The predicted coordinate is represented as a probability distribution across the possible coordinate values for a given vertex on a line segment. The probability that the model predicts the coordinate c of vertex j (with $j = 1$ for vertex A and $j = 2$ for vertex B) of line segment i to be a particular value k is denoted

by $p_{i,j,c,k}$. We uses smoothed one-hot encoding for true vertex coordinates, reducing penalties for physically closer coordinates. The true coordinate for this vertex is represented by $y_{i,j,c,k}$.

The cross-entropy loss for each vertex is calculated using the formula:

$$L_{i,j} = -\frac{1}{3}\sum_{c=1}^{3}\sum_{k=0}^{127} y_{i,j,c,k} \cdot \log(p_{i,j,c,k}),$$

Since each line segment has two vertices, the loss for line segment i is calculated as the average of the losses for both vertices A and B, $L_i = \frac{1}{2}(L_{i,1} + L_{i,2})$. The overall reconstruction loss for the model is the average of the total losses for each segment:

$$L_{\text{recon}} = \frac{1}{N}\sum_{i=1}^{N} L_i,$$

where N is the total number of line segments in the wireframe and L_{recon} is the reconstruction loss function for the autoencoder.

2.4 Loss Function for Transformer

Our method uses a 2-layer residual quantization, representing each vertex by two tokens corresponding to indices in the codebook. Each line segment comprises two vertices, therefore being represented by 4 tokens. With a total of N line segments, this equates to $4N$ tokens. Our codebook size, $|C|$, is 8192, allowing each token to have $|C|$ possible values.

The model predicts a probability for every possible value of each token. The probability that the model assigns to the jth token (where $j \in \{1, 2, 3, 4\}$) of the ith line segment being the cth token in the vocabulary is denoted as $p_{i,j,c}$. The ground truth token is represented by $y_{i,j}$. Hence, the overall loss function is defined as:

$$L_t = -\frac{1}{4N}\sum_{i=1}^{N}\sum_{j=1}^{4}\sum_{c=1}^{|C|} \mathbb{I}\{y_{i,j} = c\} \cdot \log(p_{i,j,c}),$$

where $\mathbb{I}\{y_{i,j} = c\}$ is the indicator function, which equals 1 if the true token $y_{i,j}$ is equal to c, and 0 otherwise, L_t is the loss function for the transformer.

2.5 Baselines

For PolyGen [7], we utilize the official TensorFlow implementation provided by the authors. Regarding MeshGPT [9], we replicate it based on the detailed descriptions provided in the paper. Given that MeshGPT was initially designed to generate triangular meshes, and we aim to generate wireframes, we adapt the MeshGPT during the replication process to shift its focus from predicting triangular faces to predicting line segments. We employ the same dataset for training and testing purposes for all methods under study.

3 Experiment Details

3.1 Metric Details

Following previous works on 3D generative models [6, 12, 13], we adopted COV, MMD, and 1-NN as our evaluation metrics. COV stands for *Coverage*, measuring the extent to which generated samples cover the real samples. MMD, or Maximum Mean Discrepancy, quantifies the difference between the generated and real samples. Lastly, 1-NN, meaning Nearest Neighbor, assesses the similarity between generated samples and their nearest real counterparts.

The definitions of these metrics are as follows:

$$\mathrm{COV}(S_g, S_r) = \frac{|\{\arg\min_{y \in S_r} D(X, Y) | X \in S_g\}|}{|S_r|},$$

$$\mathrm{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} D(X, Y),$$

$$\text{1-NN}(S_g, S_r) = \frac{\sum_{x \in S_g} \mathbb{I}[N_x \in S_g] + \sum_{y \in S_r} \mathbb{I}[N_y \in S_r]}{|S_g| + |S_r|},$$

where S_g and S_r represent the generated and real samples, respectively, and D denotes the distance function.

We utilized Chamfer distance and Earth Mover's Distance (EMD) as metrics to measure the similarity of wireframes. Chamfer distance, a method for quantifying point cloud similarities through nearest point distances, and EMD, which assesses the minimal effort required to transform one point cloud into another, were both applied to evaluate the wireframe comparisons effectively. All the metrics are computed on 8192 generated samples, each with 4096 sample points on their segments. The generated samples are compared with normalized augmented data, including rotation and axis flip.

To evaluate the structural validity of our generated 3D wireframes, we analyzed the relationships between the vertices and the line segments. This analysis is based on the following assumptions: if a vertex is only connected to a single line segment, it may indicate that one end of the segment is not connected to any other segment, resulting in the segment floating in space; if a vertex is connected to two line segments, it could suggest that the segments are located within the interior of the model's edges; whereas a vertex connected to three or more line segments typically indicates a structurally plausible wireframe vertex. Based on this understanding, we designed two metrics for quantifying the analysis: the Two-Line-Connected Vertex Proportion (2L-CVP), which measures the proportion of vertices connected to at least two line segments, and the Three-Line-Connected Vertex Proportion (3L-CVP), which is the proportion of vertices connected to at least three line segments. These two metrics collectively aid in evaluating the wireframes' structural validity, ensuring the generated wireframes' accuracy and realism.

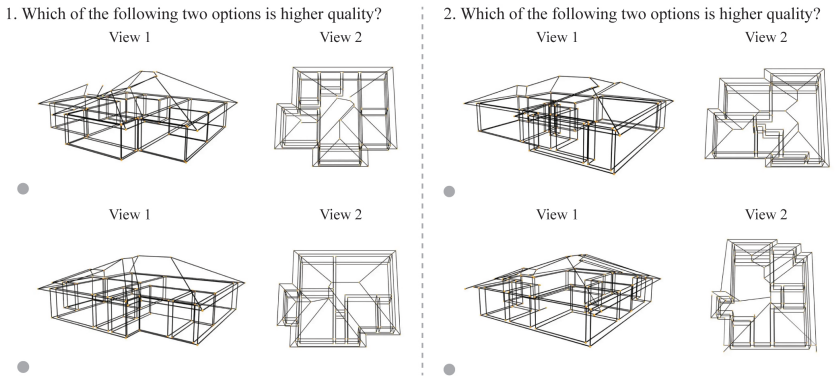

Fig. 4: User study interface.

3.2 User Study

As depicted in Fig. 4, we present the interface for our user study. Initially, we generated 1024 samples using various methods, and for comparison, we also randomly selected 1024 real wireframes from the dataset.

Since we have 4 methods (including ground truth), there are 6 possible pairings. For each pairing, we randomly selected 4 groups of samples, resulting in a total of 24 sample sets for our study.

These samples are shuffled before the presentation to ensure the display order does not influence the users' evaluations. Users could click on the images of each wireframe to view more details. We recorded the users' choices and calculated the win rates for each method. 60 participants were invited to participate in the survey, with each user evaluating 24 sets of samples. To assess the effects of different methods, we first calculated the proportion of user preferences between two methods. Taking our method and PolyGen [7] as examples, 92% of users preferred our method, while 8% preferred PolyGen. Calculating the score difference revealed that our method outperformed PolyGen by 0.84 points, which is $0.84 = 0.92$ (our selection rate) - 0.08 (PolyGen's selection rate).

3.3 Wireframe Novelty Analysis

We conducted another novelty analysis on the wireframes generated by our method. As demonstrated in Fig. 5, we compare these generated wireframes with the 3 most similar real wireframes from our training dataset. Our findings reveal a significant difference between our generated wireframes and those from the dataset, indicating that our method can produce diverse wireframes.

3.4 More Visual Results

Fig. 6 presents additional wireframe segmentation results. These wireframes are divided into multiple components, such as walls, roofs, and different rooms, based

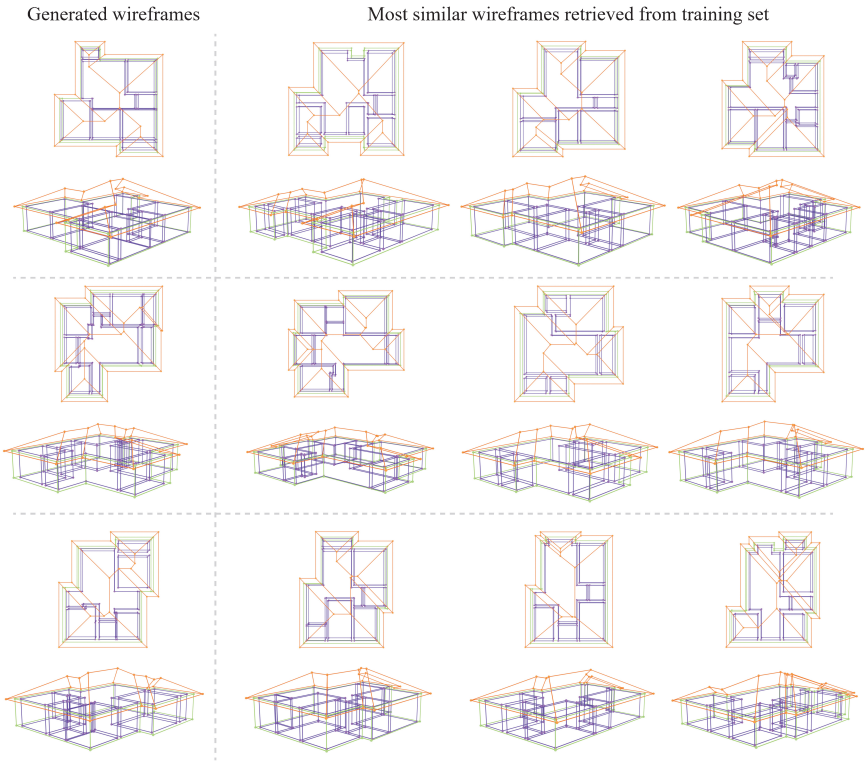

Fig. 5: Novelty analysis of generated wireframes. We present a comparison of a wireframe produced by our method against its 3 nearest neighbors from the 3D house wireframe training dataset, determined by Chamfer Distance (CD).

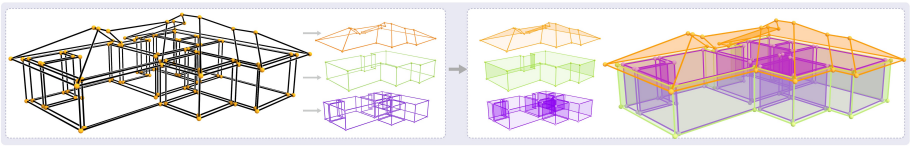

Fig. 6: The reconstructed wireframe model can be easily split into several components. We also show their corresponding mesh on the right.

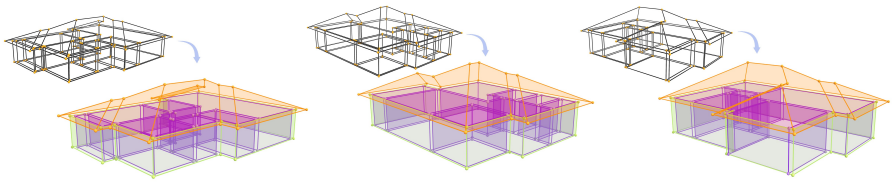

Fig. 7: The resulting wireframe can be easily converted into a mesh model.

on the connectivity of the line segments. Fig. 7 shows more results of converting wireframes into mesh models. These results further demonstrate that our wireframes can be easily converted into mesh models. Additionally, we utilize our method to generate a variety of 3D house wireframes, as shown in Fig. 8 and Fig. 9. We showcase the geometric characteristics of the wireframes and illustrate the diversity in different house layouts.

Acknowledgements

We thank all the anonymous reviewers for their insightful comments. This work was supported in parts by NSFC (U21B2023, U2001206, 62161146005), Guangdong Basic and Applied Basic Research Foundation (2023B1515120026), DEGP Innovation Team (2022KCXTD025), Shenzhen Science and Technology Program (KQTD20210811090044003, RCJC20200714114435012, JCYJ20210324120213036), Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), and Scientific Development Funds from Shenzhen University.

References

1. Aichholzer, O., Aurenhammer, F., Alberts, D., Gärtner, B.: A novel type of skeleton for polygons. Springer (1996)
2. Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer (2020)
3. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Adv. Neural Inform. Process. Syst. pp. 1024–1034 (2017)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 770–778 (2016)
5. Lee, D., Kim, C., Kim, S., Cho, M., Han, W.S.: Autoregressive image generation using residual quantization. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 11523–11532 (2022)
6. Luo, S., Hu, W.: Diffusion probabilistic models for 3d point cloud generation. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 2837–2845 (2021)
7. Nash, C., Ganin, Y., Eslami, S.A., Battaglia, P.: Polygen: An autoregressive generative model of 3d meshes. In: Int. Conf. Mach. Learn. pp. 7220–7229 (2020)
8. Roy, A., Saffar, M., Vaswani, A., Grangier, D.: Efficient content-based sparse attention with routing transformers. Trans. Assoc. Comput. Linguistics **9**, 53–68 (2021)
9. Siddiqui, Y., Alliegro, A., Artemov, A., Tommasi, T., Sirigatti, D., Rosov, V., Dai, A., Nießner, M.: Meshgpt: Generating triangle meshes with decoder-only transformers (2023)
10. Wu, W., Fu, X., Tang, R., Wang, Y., Qi, Y., Liu, L.: Data-driven interior plan generation for residential buildings. ACM Trans. Graph. **38**(6), 234:1–234:12 (2019)
11. Yu, L., Lezama, J., Gundavarapu, N.B., Versari, L., Sohn, K., Minnen, D., Cheng, Y., Gupta, A., Gu, X., Hauptmann, A.G., Gong, B., Yang, M.H., Essa, I., Ross, D.A., Jiang, L.: Language model beats diffusion – tokenizer is key to visual generation (2023)

12. Zeng, X., Vahdat, A., Williams, F., Gojcic, Z., Litany, O., Fidler, S., Kreis, K.: LION: latent point diffusion models for 3d shape generation. In: Adv. Neural Inform. Process. Syst. (2022)
13. Zhou, L., Du, Y., Wu, J.: 3d shape generation and completion through point-voxel diffusion. In: Int. Conf. Comput. Vis. pp. 5826–5835 (2021)

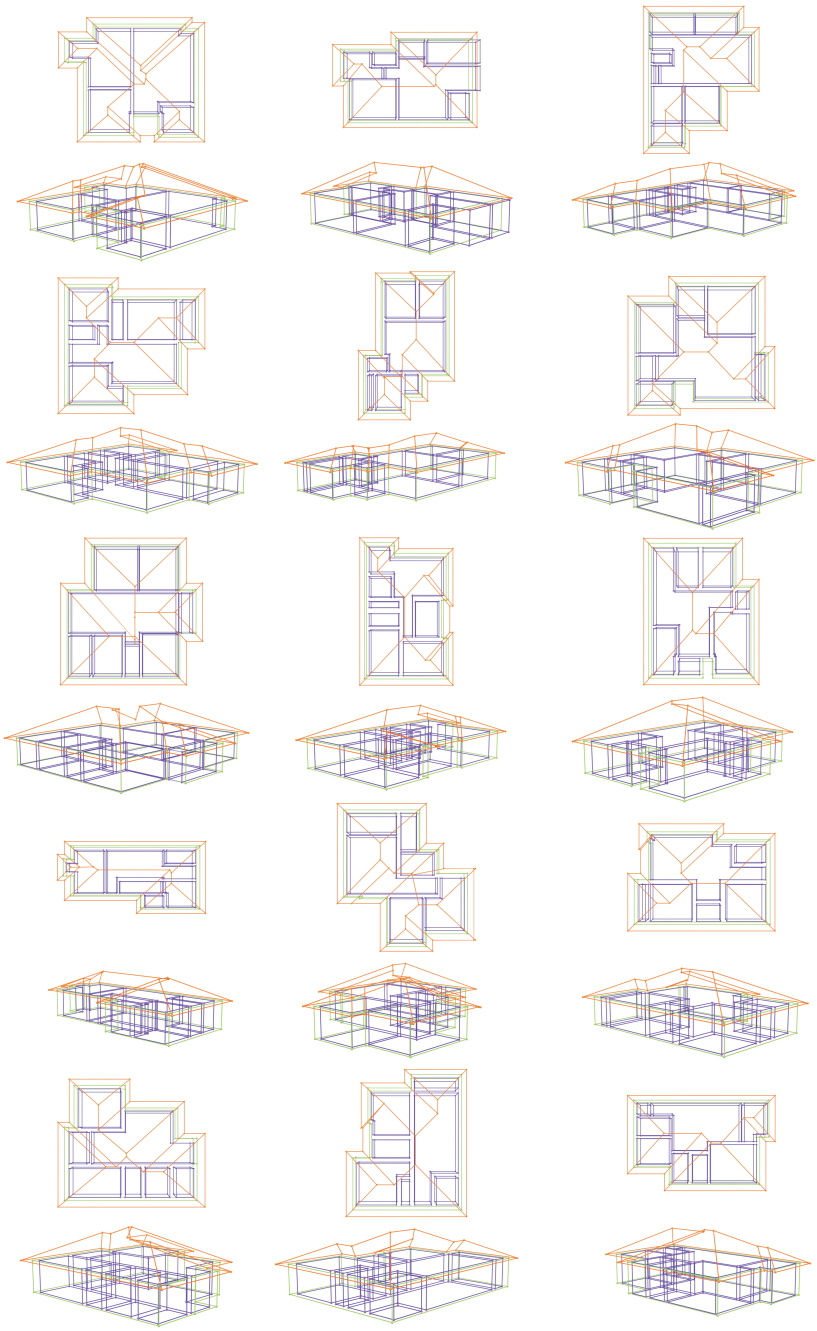

Fig. 8: Unconditionally generated 3D house wireframes from our method.

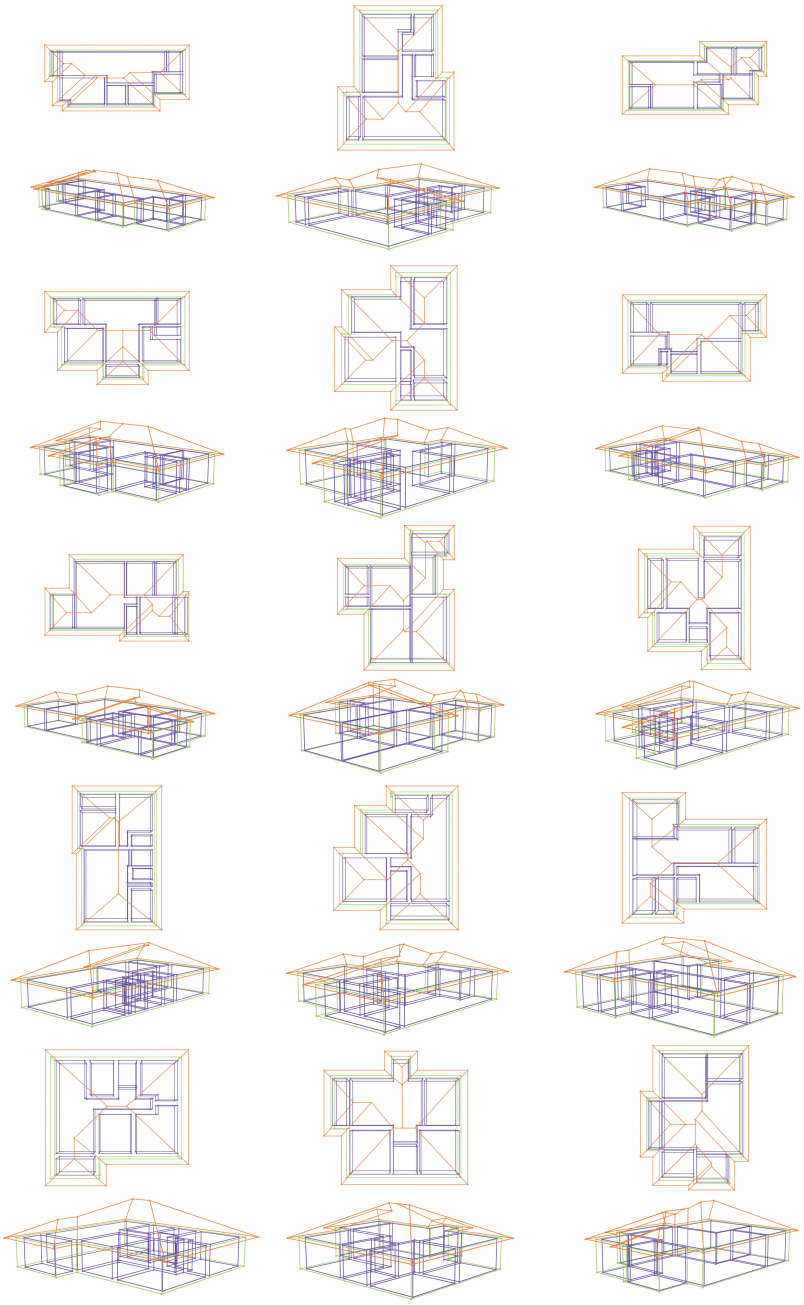

Fig. 9: Unconditionally generated 3D house wireframes from our method.