Supplementary:Learning to Construct 3D Building Wireframes from 3D Line Clouds

Yicheng Luo[1,2], Jing Ren[1,3]
Xuefei Zhe[1], Di Kang[1]
Yajing Xu[2], Peter Wonka[4]
Linchao Bao[1]

[1] Tencent AI Lab
[2] Beijing University of Posts And Telecommunications
[3] ETH Zurich
[4] KAUST

1 Implementation details

1.1 Network Architecture

In Fig. 1, we show the full details of our network structure, which can help to easily reproduce our network. Specifically, in our experiments, we sampled 256 line patches for junction prediction and sampled 1024 line patches for connectivity prediction for every building during training. During testing, we sampled line patches for all pairs of predicted junctions for connectivity prediction. We apply two transformer [8] encoders to extract a 256-dim feature for each line patch, where the first transformer encoder attends to the 32 neighboring line segments in each line patch, and the second transformer encoder attends to all the sampled line patches. We use layer normalization and ReLU activation in our transformer encoder. And the latent state size is 256. Our junction/connectivity classifier and junction regressor are formed by fully-connected layers. Note that our connectivity classifier predicts five labels for each selected pair of predicted junctions as discussed in Sec. 4.4 (main paper).

1.2 Training details

We use the ADAM [2] during the training. The learning rate and weight decay are set to 1×10^{-3} and 1×10^{-5}. All experiments are conducted on four Tesla V100 GPUs with a batch size of 32 (8 for each GPU). We train our junction predictor for 40 epochs and connectivity predictor for 25 epochs. We set the loss weight λ_v, λ_e to $1, 10$, respectively. We also randomly rotate, rescale and shift the line clouds as data-augmentation. We use our BuildingWF dataset for training and testing. Fig. 2 for some examples. We split training and testing dataset based on the number of vertices of wireframe model, making the distribution of the number of vertices in training set and test set the same.

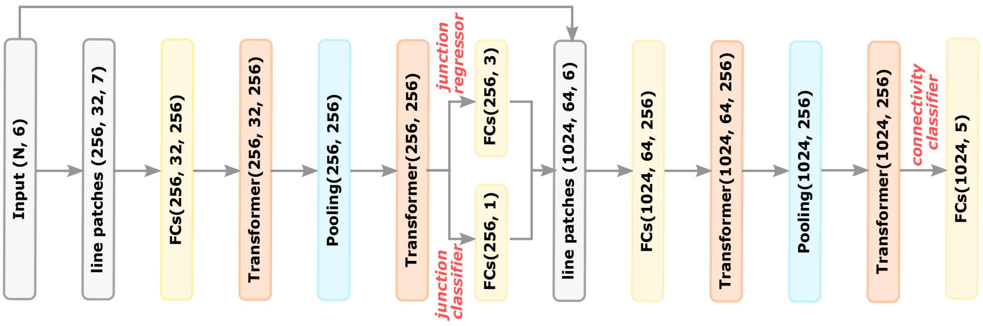

Figure 1: **Network Structure.** We report the size of the output tensor at each layer.

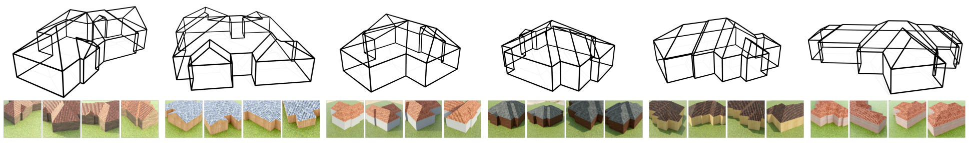

Figure 2: **BuildingWF Dataset**. ***Top***: GT 3D wireframes. ***Bottom***: multi-view images.

1.3 Comparison Configuration

For all the baselines, we fine-tuned the hyper-parameters using the publicly released codes. Specifically, for Line2Surface [3], we set the line-to-plane distance threshold ε to 2 in plane detection and set the weight of cost visibility to 1.4 in surface reconstruction. We use the default values for the rest parameters. We use the default implementation of PolyFit with region growing provided in CGAL[7], which extracts candidate planes for input of Polyfit.

In Tab. 1 in the main paper, we report the average runtime over 756 test buildings (without considering the data preparation and post-processing time). Here in Tab. 1 we report a more detailed runtime comparison. Specifically, for each method we report the data preparation time and wireframe/mesh reconstruction time. We also report the runtime for each individual step during the reconstruction. The runtime of post-processing (i.e., to simply a mesh into a wireframe or apply NMS to prune predicted wireframe) is also reported. For example, on average, it takes 19.5 second to prepare a line cloud for line3Dpp and our method, while it takes 63.8 second to prepare a point cloud for PolyFit and PC2WF from the same set of multi-view images. The line cloud reconstruction is more efficient than point cloud reconstruction due to the sparsity of line segments (point cloud reconstruction outputs about 80k points

Table 1: **Runtime Comparison (in second)**. We report the average runtime over 756 test building. Line3Dpp and line2Surf are tested on an Inter(R) Xeon(R) platinum 8255C 2.50GHz CPU. PC2WF and LC2WF are tested on Tesla V100 GPU. Polyfit and point cloud reconstruction are performed on a Quadro RTX 4000 GPU and an Intel(R) Core(TM) i9-9880H 2.30GHz CPU.

Method	Data prep.	pre-processing	junc. pred.	edge pred.	plane pred.	surf. recon.	post-processing	Total
line3Dpp	19.5	13.9	-	-	-	-	-	33.1
line2Surf	33.1	-	-	-	163.2	57.6	0.21	253.9
PolyFit	63.8	-	-	-	40.4	5.64	0.067	110.0
PC2WF	63.8	5.21	4.94	26.1	-	-	0.15	100.2
Ours	19.5	-	0.18	0.72	-	-	0.012	20.4

while line cloud reconstruction outputs 1400 line segments). This suggest that there is less overhead to construct wireframes from line clouds than from point clouds

2 Additional results

2.1 Evaluation Metrics

Precision & Recall: Precision is the fraction of the true positive predictions (e.g., predicted junctions/edges) whose distance to their nearest ground-truth is smaller than the threshold η, among all the predictions. Recall is the fraction of the ground-truth that is successfully retrieved by the predictions up to the threshold distance η. We can report different precision-recall pairs for different choices of η. A high precision and high recall suggests a high accuracy and a clean predicted model. In particular, for PC2WF and our method, a confidence score is associated with the predictions. We therefore compute the exact precision-recall (PR) curve w.r.t. the confidence scores and compute the area below PR-curve as the average precision. For simplicity, we denote the precision and recall for different methods as AP_η and Recall_η and report them in the same tables. However, line3Dpp, line2Surf, and Polyfit results let us only compute a single value for precision and recall and not an average stemming from a PR curve. In our experiments, we set η to 0.15m, 0.25m, and 0.35m. We also report the average AP_η and Recall_η over the three choices of η. Similar to [4], we report the precision and recall on both predicted junctions and wireframes:

(1) $v\text{AP}_\eta$ and $v\text{Recall}_\eta$ show the precision/recall on the predicted **junctions**.

(2) $s\text{AP}_\eta$ and $s\text{Recall}_\eta$ report the *structural* quality of the predicted **wireframes**. Specifically, it checks if a predicted edge is a true positive or if a ground-truth edge is retrieved according to the distances between the edge endpoints.

Wireframe Edit Distance (WED) reports the number of operations and the editing distances of adding/removing predicted junctions/edges that are needed to transform the graph structure of the predicted wireframe into the ground-truth wireframe. This metric shows the topological errors of the reconstructed wireframe. A smaller value suggests a more accurate topology. We follow [4] and use a simplified version of the edit distance to make it computationally tractable.

Table 2: Precision/Recall of predicted **junctions** (vAP/vRecall) and predicted **wireframe** models (sAP/sRecall) on **direct output** results. We highlight the best (in **red**) and the second best (in orange) results.

Method	output		$v\text{AP}_\eta/v\text{Recall}_\eta$ (%)				$s\text{AP}_\eta/s\text{Recall}_\eta$ (%)			
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\eta=0.15$	$\eta=0.25$	$\eta=0.35$	avg.	$\eta=0.25$	$\eta=0.35$	$\eta=0.50$	avg.
line3Dpp	240	120	8.90/**98.1**	8.92/**98.3**	8.93/**98.4**	8.92/**98.3**	24.7/80.8	25.4/83.0	25.9/84.6	25.3/82.8
line2Surf.	304	89	6.31/88.0	6.45/89.9	6.50/90.6	6.42/89.5	1.10/27.1	1.15/28.5	1.23/30.4	1.16/28.7
PolyFit	183	250	8.99/75.1	10.5/87.6	10.8/90.4	10.1/84.4	4.16/28.3	5.58/37.9	6.56/44.5	5.43/36.9
PC2WF	69	986	11.0/33.2	36.2/61.3	46.3/70.6	31.2/55.1	0.35/14.5	2.86/37.5	8.22/58.1	3.81/36.7
Ours	47	126	**91.2**/93.9	**93.3**/95.9	**93.9**/96.4	**92.8**/95.4	**84.3/90.9**	**86.8/93.4**	**87.7/94.5**	**86.3/92.9**

Table 3: **Wireframe Edit Distance** (WED) of the reconstructed **wireframes**. We report the number of operations (Num) and the editing distances in meters (Dist).

Method	Topology		(WED) **+vertex**		(WED) **+edge**		(WED) **-edge**		(WED) **Total**	
	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	Num.	Dist. (m)	Num.	Dist. (m)	Num.	Dist. (m)	Num.	Dist. (m)
ground-truth	22 ± 10	37 ± 16	-	-	-	-	-	-	-	-
line3Dpp [1]	240 ± 140	120 ± 70	0.104	59.27	2.074	8.917	49.60	255.7	51.77	323.9
Line2Surf. [3]	68 ± 68	89 ± 89	1.012	13.78	6.223	35.76	9.427	48.77	16.66	98.31
PolyFit [5]	29 ± 15	43 ± 21	1.681	3.170	4.811	21.41	0.969	5.285	7.463	29.86
PC2WF [4]	22 ± 7	34 ± 13	5.216	3.445	17.01	87.94	4.622	33.38	26.84	124.8
Ours	23 ± 10	39 ± 20	0.766	1.810	2.880	11.49	1.655	14.03	**5.301**	**27.33**

2.2 More results

As mentioned in Sec.5.3 (main paper), we post-process the output meshes into wireframes to make a fair comparision. Here, we report the evaluation on both direct outputs(Tab. 2,Fig. 3) and results after post-processing(Tab.2a and Fig.3 in main paper).

In Tab. 2 we measure the accuracy of the reconstructed buildings in the form of line clouds (line3Dpp), meshes (line2Surf and PolyFit), and wireframes (PC2WF and ours). Specifically, we report the average number of vertices ($|\mathcal{V}|$) and edges ($|\mathcal{E}|$) over all the tested buildings. We then evaluate the precision/recall of the predicted junctions and the predicted structure by considering all the predicted edges in the line clouds/meshes/wireframes. We can see that Line3Dpp has the highest recall on junction prediction (vRecall) and achieves the second best on structure prediction (sAP, sRecall). As a comparison, our method achieves comparable recall on junction prediction with 80% fewer predicted junctions. Moreover, our method obtains results with significantly higher structural accuracy compared to Line3Dpp.

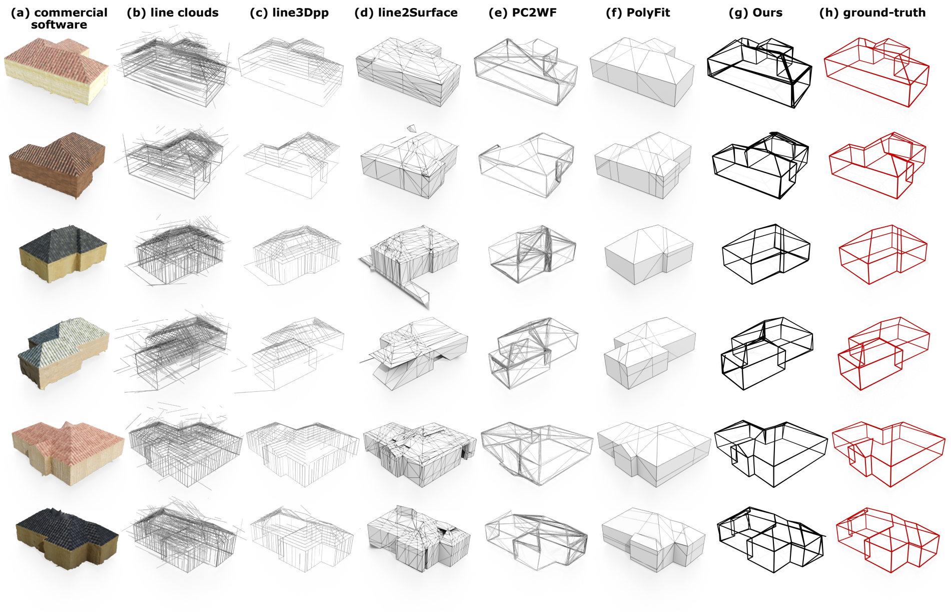

Figure 3: Direct output before post-processing using Line3Dpp [1], Line2Surface [3], PolyFit [5], PC2WF [4], and ours.

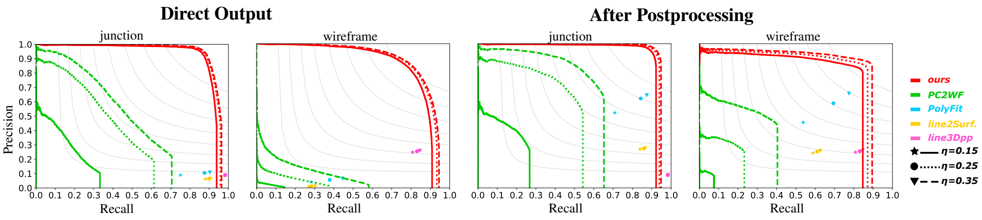

Figure 4: Precision-Recall curves of direct outputs (the left two figures) and results after post-processing (the right two figures), corresponding to Tab. 2 and Tab. 2a (main paper) respectively.

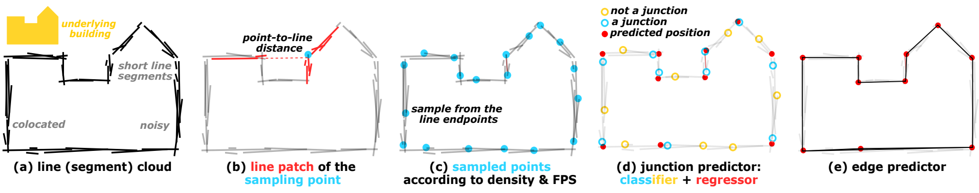

Figure 5: **2D illustration**. Given a noisy line cloud (a), we collect line patches based on point-to-line distance (b) for samples (c) to construct building wireframe, via a junction predictor (d) and a edge predictor (e). Note that our junction predictor (d) includes a classifier to tell whether a junction exists (blue circles) or not (yellow circles), and a regressor to predict/correct the junction position (red dots).

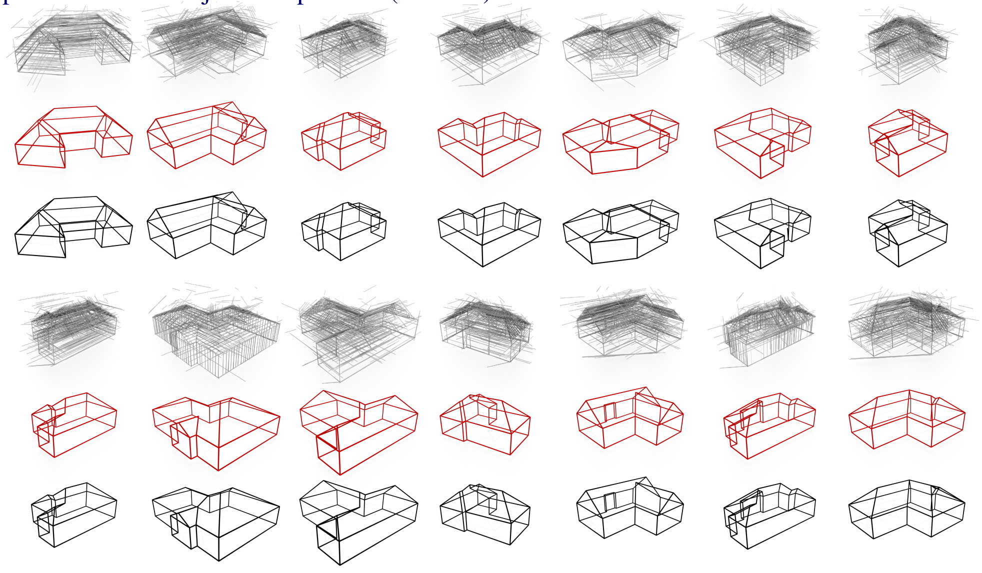
Figure 6: Input line clouds, ground-truth (red) and our reconstructed wireframes (black).

In Fig. 3 we show the direct output (without any post-processing steps) of different methods, which corresponds to Fig. 3 (main paper). Fig. 4 visualizes the precision-recall curves that correspond to Tab. 2 and Tab.2a (main paper). And we show more results of our reconstructed wireframe in Fig. 6.

Tab. 3 shows the full experiment results of the wireframe edit distance for different methods after post-process.

2.3 Ablation Study

LPT Structure We conduct an ablation study on the structure of our line-patch transformer[8] (LPT) as shown in Tab. 4. As discussed in Sec 4.2 (main paper), our LPT has two layers of transformers that first attend to the line segments in the neighborhood and then attend to all the line patches. We compare it to two trivial alternatives (1) replace the transformer by an MLP; (2) use a single-layer transformer that attends to all line segments in all sampled line patches at the same time. We can see that our design choice achieves the best precision/recall.

Table 4: Ablation on LPT ($v\text{AP}_\eta / v\text{Recall}_\eta$)

LPT Structure	$\eta = 0.15$	$\eta = 0.25$	$\eta = 0.35$	avg.
ours	90.3/95.2	91.6/96.6	92.1/97.2	91.3/96.3
MLP	85.0/94.6	86.1/96.2	86.6/96.9	85.9/95.9
Single-layer LPT	86.2/94.7	88.3/96.5	89.2/97.2	87.9/96.1

Sampling Strategy We also justify our sampling strategy for line patch construction via an ablation study as shown in Tab. 5. As illustrated in Fig. 5, there are two key steps in constructing line patches for wireframe prediction (Sec.4.3 main paper): (1) sample among the line endpoints based on density and FPS; (2) construct a line patch at each sampled point based on point-to-line distance. We compare our sampling strategy to another two alternatives: (1) only use FPS to sample without considering the density of points; (2) use the point-to-line-segment distance to collect neighboring line segments. The results show that ours is the most effective strategy.

Table 5: Ablation on Sampling ($v\text{AP}_\eta / v\text{Recall}_\eta$)

Sampling Strategy	$\eta = 0.15$	$\eta = 0.25$	$\eta = 0.35$	avg.
ours	90.3/95.2	91.6/96.6	92.1/97.2	91.3/96.3
FPS **w/out** density	49.4/80.7	58.9/91.2	61.7/94.0	56.7/88.6
point-to-**lineSeg**	88.1/94.9	89.4/96.4	89.9/97.1	89.1/96.1

Edge Labels As discussed in Sec.4.4 (main paper), we consider 5 categories of edge labels for a pair of predicted junctions, namely (a) at least one of the junctions are false positive, or the potential graph distance between the two junctions is (b) 0, (c) 1, (d) 2, (e) larger than 2. An alternative choice would be only considering 2 cases: i.e., whether the selected junction pair is connected or not. Tab. 6 shows such an ablation study. We can see that considering more detailed classifications for edges can help to learn more information for edge prediction and pruning during post-processing. For example, if two junctions are predicted to be true positive and have zero graph distance, only the junction with higher confidence will be kept during post-processing, which is not feasible if we only have two categories for edge labels.

Table 6: Ablation on Edge Labels ($s\text{AP}_\eta / s\text{Recall}_\eta$)

Edge Labels	$\eta = 0.15$	$\eta = 0.25$	$\eta = 0.35$	avg.
5 cate.	84.3/91.1	86.9/93.7	87.8/94.9	86.3/93.2
2 cate.	82.1/91.1	84.7/93.7	85.7/94.8	84.2/93.2
5 cate. + post.	76.8/84.7	80.6/87.1	83.9/89.5	80.4/87.1
2 cate. + post.	60.4/62.5	64.0/66.1	73.5/76.0	65.9/68.5

Partial Reconstruction As mentioned in main paper, our method can get complete wireframe model from multi-view images cover the overall region of the underlying buildings. We conduct an additional experiment by reconstructing meaningful 3D wireframe from partial line/point clouds. See Fig. 7 for some examples, where the line/point clouds are extracted from images where only half of the building is visible.

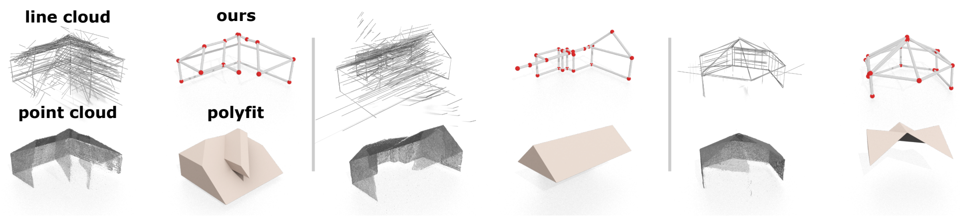

Figure 7: *Top*: **our** wireframe reconstruction from partial line clouds; *Bottom*: PolyFit reconstruction from partial point clouds.

3 Post-processing for LC2PW

Since the number of sample line patches is much more than the number of vertices/edges in the underlying wireframe, the predicted vertices and edges may be redundant. Recall that our junction regressor gives the predicted positions for the junctions $\mathcal{V}^{\text{pred}}$ and the junction classifier associates each predicted vertex p_k with a confidence score $c(p_k)$. Then for each pair of predicted junctions $(p_k, q_k) \in \mathcal{V}^{\text{pred}} \times \mathcal{V}^{\text{pred}}$, our connectivity classifier returns the probability distributions over five categories, where we use the probability of having the underlying graph distance to be 0 and 1, denoted as $c_0(p_k, q_k)$ and $c_1(p_k, q_k)$ respectively, for edge pruning.

To get a clean wireframe structure, we need to prune redundant edges and vertices. Algorithm 1 gives a detailed description for our post-processing steps. First, we use $c_0(\cdot,\cdot)$, the probability of a vertex pair to be identical, to prune the predicted junction (Line 3-4). For a vertex pair (p,q), the procedure PRUNE-BY-PROB will remove the vertex with a lower confidence $c(\cdot)$ if the value $c_0(p,q)$ is above a pre-defined threshold(Line 12-19). Then we prune vertex using PRUNE-BY-GRAPH. This procedure PRUNE-BY-GRAPH takes a graph $(\mathcal{V}, \mathcal{E})$ as input (Line 5). If the adjacency vectors[1] of the two test vertices have a small Hamming distance (i.e. share similar connected vertices) and the Euclidean distance between the two vertices is small, which indicates the two vertices are probably the same junction, then the vertex with a lower confidence $c(\cdot)$ will be removed (Line 21-29). We then use non-maximum suppression (Line 7) to remove redundant edges that are close to each other (Line 31-39) and we also remove the isolated edges (Line 8, 41-48) to get the final wireframe. The hyperparameters $\eta_0, \eta_1, \eta_2, \eta_3$ are set to $0.2, 0.3, 1.5, 1$ respectively.

We notice that sAP (the structural average precision) decreases about 5.9% after the post-processing (see Tab. 2 and Table 2 in the main paper). The reason is that during post-processing, the procedure LINE-NMS can remove short lines which are close to the ground-truth wireframe, and the procedure MERGE-BY-PROB can remove a lot of vertices that are close to ground-truth junctions and corresponding edges. In this case, we can get a clean wireframe with small graph editing distances to the ground-truth wireframe, but the sAP might get worse compared to the direct output.

4 BuildingWF Dataset Construction

Generate the line cloud. To get line cloud, we first need to synthesize multi-view images $\mathcal{I}$ of the building with synthetic textures based on the provided surface labels (facades or

[1]Adjacency vector of a vertex is a corresponding rows in the adjacency matrix of the graph $(\mathcal{V}, \mathcal{E})$

Algorithm 1 Post-Processing for LC2WF

```
Require: predicted junctions $\mathcal{V}^{\text{pred}}$; vertex confidence $c(\cdot)$; probability of graph distance 0 or 1: $c_0(\cdot,\cdot), c_1(\cdot,\cdot)$
    between two junctions; hyperparameters $\eta_0, \eta_1, \eta_2, \eta_3$.
procedure POST-PROCESS
    $\mathcal{V} \leftarrow \mathcal{V}^{pred}$
    $\tilde{\mathcal{E}} \leftarrow \{(p,q) \mid p,q \in \mathcal{V}\times\mathcal{V}, c_0(p,q) > \eta_0\}$
    $\mathcal{V} \leftarrow$ PRUNE-BY-PROB$(\tilde{\mathcal{E}}, \mathcal{V})$
    $\mathcal{V} \leftarrow$ PRUNE-BY-GRAPH$(\mathcal{V}, \{(p,q) \mid p,q \in \mathcal{V}\times\mathcal{V}, c_1(p,q) > \eta_1\})$
    $\mathcal{E} \leftarrow \{(p,q) \mid p,q \in \mathcal{V}\times\mathcal{V}, c_1(p,q) > \eta_1\}$
    $\mathcal{E} \leftarrow$ LINE-NMS$(\mathcal{E})$
    $\mathcal{V}, \mathcal{E} \leftarrow$ REMOVE-ISOLATED-EDGE$(\mathcal{V}, \mathcal{E})$
    return $(\mathcal{V}, \mathcal{E})$
end procedure

procedure PRUNE-BY-PROB$(\tilde{\mathcal{E}}, \mathcal{V})$
    sort $\tilde{\mathcal{E}}$ w.r.t merge probability values $c_0(\cdot,\cdot)$ in descending order
    for all $(v_1, v_2) \in \tilde{\mathcal{E}}$ do
        $v \leftarrow \arg\max_{v \in \{v_1, v_2\}} c(v)$
        $\mathcal{V} \leftarrow \mathcal{V} \backslash \{v\}$
    end for
    return $\mathcal{V}$
end procedure

procedure PRUNE-BY-GRAPH$(\mathcal{V}, \mathcal{E})$
    sort $\mathcal{V}$ w.r.t confidence values $c(\cdot)$ in descending order
    for all $v \in V$ do
        if $\exists v' \in \mathcal{V}\backslash\{v\} : Hamming(v,v') \leq 2$ and $Euclidean(v,v') < \eta_2$ then
            $\mathcal{V} \leftarrow \mathcal{V}\backslash\{v'\}$
        end if
    end for
    return $\mathcal{V}$
end procedure

procedure LINE-NMS$(\mathcal{E})$
    sort $\mathcal{E}$ w.r.t probability values $c_1(\cdot,\cdot)$ in descending order
    for all $e \in \mathcal{E}$ do
        if $\exists e' \in \mathcal{E}\backslash\{e\} : Dist(e,e') < \eta_3$ then
            $\mathcal{E} \leftarrow \mathcal{E}\backslash\{e'\}$
        end if
    end for
    return $\mathcal{E}$
end procedure

procedure REMOVE-ISOLATED-EDGE$(\mathcal{V}, \mathcal{E})$
    for $e = (v_1, v_2) \in \mathcal{E}$ do
        if $deg(v_1) == 1$ and $deg(v_2) == 1$ then
            $\mathcal{E} \leftarrow \mathcal{E}\backslash e$; $\mathcal{V} \leftarrow \mathcal{V}\backslash\{v_1, v_2\}$
        end if
    end for
    return $(\mathcal{V}, \mathcal{E})$
end procedure
```

roof). The synthetic textures are downloaded from the Internet. For each building, we render roughly 32 images by moving the camera around the building twice with pitch angle at 45 and 60 degree respectively. We also add noise to the camera poses. Then we project the 3D wireframe to each image plane using the corresponding camera parameters to get ground-truth 2D wireframes, during which the self-conclusion is considered. After that, we use

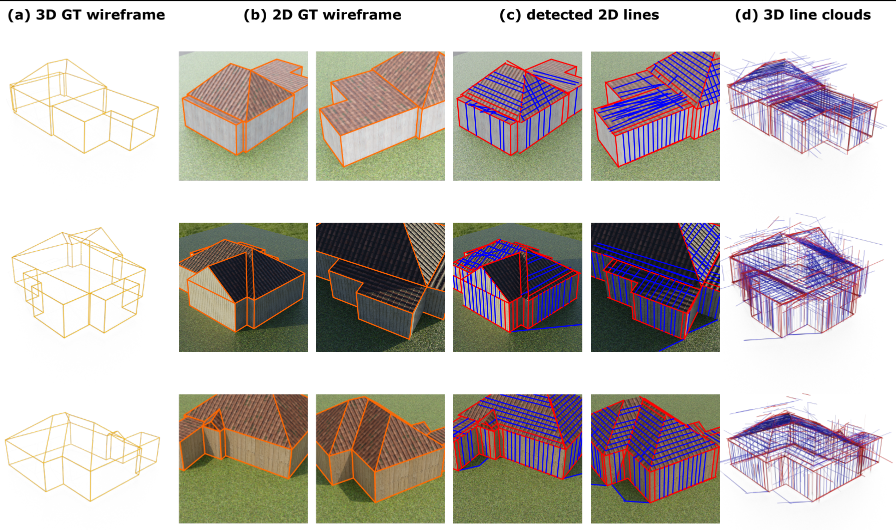

Figure 8: **Construction of our BuildingWF Dataset.** We collect the 3D building wireframe models from [6] (a), and synthesize 2D multi-view images. The ground-truth 3D wireframe can be projected to 2D image planes using the corresponding camera parameters to get the ground-truth 2D wireframes in different views (b). We then use state-of-the-art 2D lines detector to detect 2D lines from multi-view images, where the 2D ground-truth wireframes are used to annotate the positive lines (colored in red) and the negative noisy lines (colored in blue) as shown in (c). The detected 2D lines are then used to construct a 3D line colored with corresponding labels (d) for training.

ELSD [9] to detect 2D line segments of each image. Finally, we feed the detected 2D line segments and camera parameters to Line3Dpp [1] to reconstruct 3D line clouds.

Label the line cloud for training. Considering that the 3D line segments in the previously constructed line cloud is less accurate than the detected 2D line segments [9], we use the detected 2D line segments and the ground-truth 2D wireframe to label each line segment in the line cloud. Specifically, each 3D line l_i of the reconstructed line cloud is paired with a detected 2D line l_i^{2D}. We match each detected 2D line segment to the ground-truth 2D wireframe if their *line-to-line* distance is less than a threshold η. Then we can get the relation between the generated 3D line cloud and the ground-truth 3D wireframe through the 2D matching results. After that, we further associated it with two ground-truth junctions that are endpoints of the corresponding edge. Finally, we can use labeled line clouds to supervise our network training.

References

[1] Manuel Hofer, Michael Maurer, and Horst Bischof. Efficient 3d scene abstraction using line segments. *Computer Vision and Image Understanding*, 157:167–178, 2017. ISSN 1077-3142.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Pierre-Alain Langlois, Alexandre Boulch, and Renaud Marlet. Surface Reconstruction from 3D Line Segments. In *International Conference on 3D Vision (3DV)*, pages 553–563, Québec City, Canada, September 2019. IEEE. doi: 10.1109/3DV.2019.00067.

[4] Yujia Liu, Stefano D'Aronco, Konrad Schindler, and Jan Dirk Wegner. Pc2wf: 3d wireframe reconstruction from raw point clouds. In *International Conference on Learning Representations (ICLR)*, 2020.

[5] Liangliang Nan and Peter Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *International Conference on Computer Vision (ICCV)*, pages 2353–2361, 2017.

[6] Jing Ren, Biao Zhang, Bojian Wu, Jianqiang Huang, Lubin Fan, Maks Ovsjanikov, and Peter Wonka. Intuitive and efficient roof modeling for reconstruction and synthesis. *ACM Transactions on Graphics (TOG)*, 40(6):1–17, 2021.

[7] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4 edition, 2022. URL https://doc.cgal.org/5.4/Manual/packages.html.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[9] Haotian Zhang, Yicheng Luo, Fangbo Qin, Yijia He, and Xiao Liu. Elsd: Efficient line segment detector and descriptor, 2021.