

Learning to Reconstruct 3D Manhattan Wireframes from a Single Image

Yichao Zhou^{1,2,*} Haozhi Qi¹ Yuexiang Zhai¹ Qi Sun² Zhili Chen² Li-Yi Wei² Yi Ma¹

¹UC Berkeley

²Adobe Research

Abstract

In this paper, we propose a method to obtain a compact and accurate 3D wireframe representation from a single image by effectively exploiting global structural regularities. Our method trains a convolutional neural network to simultaneously detect salient junctions and straight lines, as well as predict their 3D depth and vanishing points. Compared with the state-of-the-art learning-based wireframe detection methods, our network is much simpler and more unified, leading to better 2D wireframe detection. With global structural priors such as Manhattan assumption, our method further reconstructs a full 3D wireframe model, a compact vector representation suitable for a variety of high-level vision tasks such as AR and CAD. We conduct extensive evaluations on a large synthetic dataset of urban scenes as well as real images. Our code and datasets will be released.

1. Introduction

Recovering 3D geometry of a scene from RGB images is one of the most fundamental and yet challenging problems in computer vision. Most existing off-the-shelf commercial solutions to obtain 3D geometry still requires *active* depth sensors such as structured lights (e.g., Apple ARKit and Microsoft Mixed Reality Toolkit¹) or LIDARs (popular in autonomous driving). Although these systems can meet the needs of specific purposes, they are limited by the cost, range, and working conditions (indoor or outdoor) of the sensors. The representations of final outputs are typically dense point clouds, which are not only memory and computation intense, but also may contain noises and errors due to transparency, occlusions, reflections, etc.

On the other hand, traditional image-based 3D reconstruction methods, such as Structure from Motion (SfM) and visual SLAM, often rely on local features. Although the efficiency and reliability have been improving (e.g., Microsoft Hololens, Magic Leap), they often need multiple cameras with depth sensors [14] for better accuracy. The final scene

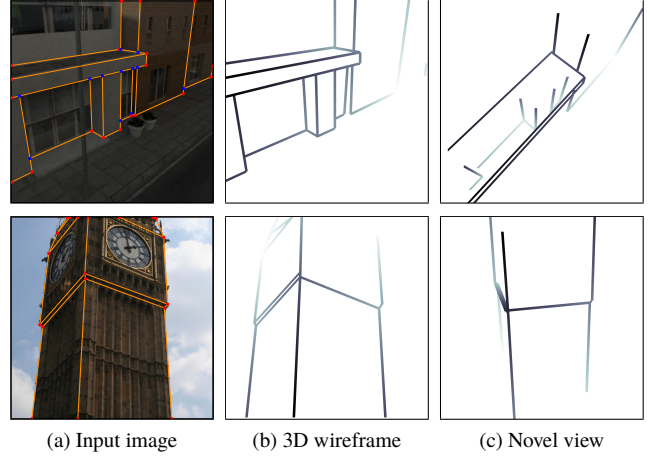


Figure 1: Results of our method tested on a single synthetic image (top row) and a real image (bottom row). Column (a) shows the input images overlaid with the groundtruth wireframes, in which the red and blue dots represent the C- and T-type junctions, respectively. Column (b) shows the predicted 3D wireframe from our system, with grayscale visualizing depth. Column (c) shows alternative views of (b). Note that our system recovers geometrically salient wireframes, without being affected by the textural lines, e.g., the vertical textural patterns on the Big Ben facade.

representation remains quasi-dense point clouds, which are typically incomplete, noisy, and cumbersome to store and share. Consequently, complex post-processing techniques such as plane-fitting [12] and mesh refinement [15, 20] are required. Such traditional representations can hardly meet the increasing demand for high-level 3D modeling, content editing, and model sharing from hand-held cameras, mobile phones, and even drones.

Unlike conventional 3D geometry capturing systems, the human visual system does not perceive the world as uniformly distributed points. Instead, humans are remarkably effective, efficient, and robust in utilizing geometrically salient *global* structures such as lines, contours, planes, and smooth surfaces to perceive 3D scenes [2]. However, it remains challenging for vision algorithms to detect and utilize

*This work was done when Y. Zhou was an intern at Adobe Research.

¹<https://github.com/Microsoft/MixedRealityToolkit-Unity>

such global structures from local image features, until recent advances in deep learning which makes learning high-level features possible from labeled data. The examples include detecting planes [30, 19], surfaces [10], 2D wireframes [13], room layouts [35], key points for mesh fitting [31, 29], and sparse scene representations from multiple images [6].

In this work, we infer global 3D scene layouts from learned line and junction features, as opposed to local corner-like features such as SIFT [8], ORB [22], or line segments [11, 5, 24] used in conventional SfM or visual SLAM systems. Our algorithm learns to detect a special type of wireframes that consist of junctions and lines representing the corners and edges of buildings. We call our representation the *geometric wireframe* and demonstrate that together with certain global priors (say globally or locally Manhattan [3, 8, 24]), the wireframe representation allows effective and accurate recovery of the scene’s 3D geometry, even from a single input image. Our method trains a neural network to estimate global lines and two types of junctions with depths, and constructs full 3D wireframes using the estimated depths and geometric constraints.

Previously, there have been efforts trying to understand the indoor scenes with the help of the 3D synthetic datasets such as the SUNCG [25, 32]. Our work aims at natural urban environments with a variety of geometries and textures. To this end, we build two new datasets containing both synthetic and natural urban scenes. Figure 1 shows the sampled results of the reconstruction and Figure 2 shows the full pipeline of our system.

Contributions of this paper. Comparing to existing wireframe detection algorithms such as [13], our method

- jointly detects junctions, lines, depth, and vanishing points with a single neural network, exploiting the tight relationship among those geometric structures;
- learns to differentiate two types of junctions: the physical intersections of lines and planes “C-junctions”, and the occluding “T-junctions”;
- recovers a full 3D wireframe of the scene from the lines and junctions detected in a single RGB image.

2. Methods

As depicted in Figure 2, our system starts with a neural network that takes a single image as input and jointly predicts multiple 2D heatmaps, from which we vectorize lines and junctions as well as estimate their initial depth values and vanishing points. We call this intermediate result a 2.5D wireframe. Using both the depth values and vanishing points estimated from the same network as the prior, we then lift the wireframe from the 2.5D *image-space* into the full 3D *world-space*.

2.1. Geometric Representation

In a geometric wireframe $W = (V, E)$ of the scene, V and $E \subseteq V \times V$ are the junctions and corresponding lines. Specifically, E represents lines from physical intersections of two planes while V represents (physical or projective) intersections of lines among E . Unlike [11, 13], our E excludes planar textural lines, such as the vertical textures of Big Ben in Figure 1. The so-defined W aims to capture global scene geometry instead of local textural details.² To specify a wireframe that extends to 3D, extra annotations on W are necessary. For each junction $w \in V$, our key idea is to assign it a junction type $J_w \in \{C, T\}$, representing whether w is a *C-junction* ($J_w = C$) or a *T-junction* ($J_w = T$). Corner C-junctions are actual intersections of physical planes or edges, while T-junctions are generated by occlusion. Examples of T-junctions (in blue) and C-junctions (in red) can be found in Figure 1. We denote them as two disjoint sets $V = V_C \cup V_T$, in which $V_C = \{w \in V \mid J_w = C\}$ and $V_T = \{w \in V \mid J_w = T\}$. Junction types are important for inferring 3D wireframe geometry, as different 3D priors will be applied to each type.³ For each C-junction u , define z_u as the depth of vertex u , i.e., the z coordinate of u in the *camera space*. For each T-junction v , there are two non-intersecting 3D lines involved. We define z_v as the depth on the occluded line in the background because the foreground line depth can always be recovered from other junctions. With depth information, 3D wireframes that are made of C-junctions, T-junctions and lines give a compact representation of the scene geometry. Reconstructing such 3D wireframes from single images is our goal.

2.2. From Single Image to 2.5D Representation

Our first step is to train a neural network that learns the desired junctions, lines, depth, and vanishing points from our labeled datasets. We first briefly describe the desired outputs from the network and the architecture of the network. The associated loss functions for training the network will be specified in detail in the next sections.

Given the image I of a scene, the pixel-wise outputs of our neural network consist of five outputs – junction probability J , junction offset O , edge probability E , junction depth D , and vanishing points V :

$$Y \doteq (J, O, E, D, V), \quad \hat{Y} \doteq (\hat{J}, \hat{O}, \hat{E}, \hat{D}, \hat{V}), \quad (1)$$

where symbols with and without hats represent the ground truth and the prediction from the neural network. The meaning of each symbol is detailed in Section 2.2.2.

²In urban scenes, lines from regular textures (such as windows on a facade) do encode accurate scene geometry [33]. The neural network can still use them for inferring the wireframe but only not to keep them in the final output, which is designed to give a compact representation of the geometry only.

³There is another type of junctions which are caused by lines intersecting with the image boundary. We treat them as C-junctions for simplicity.

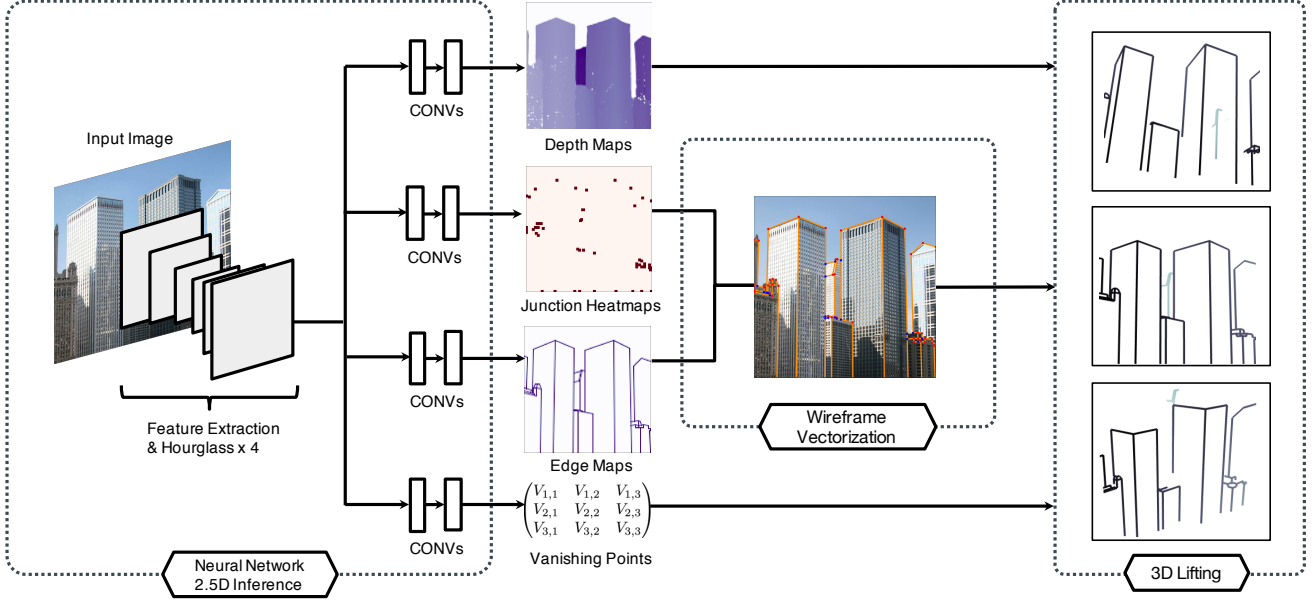


Figure 2: Overall pipeline of the proposed method.

2.2.1 Network Design

Our network structure is based on the stacked hourglass network [23]. The input images are cropped and re-scaled to 512×512 before entering the network. The feature-extracting module, the first part of the network, includes strided convolution layers and one max pooling layer to downsample the feature map to 128×128 . The following part consists of S hourglass modules. Each module will gradually downsample then upsample the feature map. The stacked hourglass network will gradually refine the output map to match the supervision from the training data. Let the output of the j th hourglass module given the i th image be $F_j(I_i)$. During the training stage, the total loss to minimize is:

$$L^{\text{total}} \doteq \sum_{i=1}^N \sum_{j=1}^S L(Y_i^{(j)}, \hat{Y}_i) = \sum_{i=1}^N \sum_{j=1}^S L(F_j(I_i), \hat{Y}_i),$$

where i represents the index of images in the training dataset; j represents the index of the hourglass modules; N represents the number of training images in a batch; S represents the number of stacks used in the neural network; $L(\cdot, \cdot)$ represents the loss of an individual image; $Y_i^{(j)}$ represents the predicted intermediate representation of image I_i from the j th hourglass module, and \hat{Y}_i represents the ground truth intermediate representation of image I_i .

The loss of an individual image is a superposition of the loss functions L_k specified in the next section:

$$L \doteq \sum_k \lambda_k L_k, \quad k \in \{J, \mathbf{O}, E, \mathcal{D}, \mathbf{V}\}.$$

The hyper-parameters λ_k represents the weight of each sub-loss. During experiments, we set λ so that $\lambda_k L_k$ are of similar scales.

2.2.2 Output Maps and Loss Functions

Junction Map J and Loss L_J . The ground truth junction map \hat{J} is a down-sampled heatmap for the input image, whose value represents whether there exists a junction in that pixel. For each junction type $t \in \{C, T\}$, we estimate its corresponding junction heatmap

$$\hat{J}_t(\mathbf{p}) = \begin{cases} 1 & \exists \mathbf{w} \in \mathbf{V}_t : \mathbf{p} = \lfloor \frac{\mathbf{w}}{4} \rfloor, \\ 0 & \text{otherwise.} \end{cases}$$

Following [23], the resolution of the junction heatmap is 4 times less than the resolution of the input image.

Because some pixels may contain two types of junctions, we treat the junction prediction as two per-pixel binary classification problems. We use the classic softmax cross entropy loss to predict the junction maps:

$$L_J(J, \hat{J}) \doteq \frac{1}{n} \sum_{t \in \{C, T\}} \sum_{\mathbf{p}} \text{CrossEntropy}(J_t(\mathbf{p}), \hat{J}_t(\mathbf{p})),$$

where n is the number of pixels of the heatmap. The resulting $J_t(x, y) \in (0, 1)$ represents the probability whether there exists a junction with type t at $[4x, 4x + 4) \times [4y, 4y + 4)$ in the input image.

Offset Map \mathbf{O} and Loss L_O . Comparing to the input image, the lower resolution of J might affect the precision of junction positions. Inspired by [28], we use an offset map to store the difference vector from \hat{J} to its original position

with sub-pixel accuracy:

$$\hat{\mathbf{O}}_t(\mathbf{p}) = \begin{cases} \frac{\mathbf{w}}{4} - \mathbf{p} & \exists \mathbf{w} \in \mathbf{V}_t : \mathbf{p} = \lfloor \frac{\mathbf{w}}{4} \rfloor, t \in \{C, T\}. \\ 0 & \text{otherwise} \end{cases}$$

We use the ℓ_2 -loss for the offset map and use the heatmap as a mask to compute the loss only near the actual junctions. Mathematically, the loss function is written as

$$L_O(\mathbf{O}, \hat{\mathbf{O}}) \doteq \sum_{t \in \{C, T\}} \frac{\sum_{\mathbf{p}} \hat{J}_t(\mathbf{p}) \|\mathbf{O}_t(\mathbf{p}) - \hat{\mathbf{O}}_t(\mathbf{p})\|_2^2}{\sum_{\mathbf{p}} \hat{J}_t(\mathbf{p})},$$

where $\mathbf{O}_t(\mathbf{p})$ is computed by applying a sigmoid and constant translation function to the last layer of the offset branch in the neural network to enforce $\mathbf{O}_t(\mathbf{p}) \in (-0.5, 0.5)^2$. We normalize L_O by the number of junctions of each type.

Edge Map E and Loss L_E . To estimate line positions, we represent them in an edge heatmap. For the ground truth lines, we draw them on the edge map using an anti-aliasing technique [34] for better accuracy. Let $\text{dist}(\mathbf{p}, e)$ be the shortest distance between a pixel \mathbf{p} and the nearest line segment e . We define the edge map to be

$$\hat{E}(\mathbf{p}) = \begin{cases} \max_e 1 - \text{dist}(\mathbf{p}, e) & \exists e \in E : \text{dist}(\mathbf{p}, e) < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, $E(\mathbf{p}) \in [0, 1]$ represents the probability of a line close to point \mathbf{p} . Because the range of the edge map is always between 0 and 1, we can treat it as a probability distribution and use the sigmoid cross entropy loss on the E and \hat{E} :

$$L_E(E, \hat{E}) \doteq \frac{1}{n} \sum_{\mathbf{p}} \text{CrossEntropy}(E(\mathbf{p}), \hat{E}(\mathbf{p})).$$

Junction Depth Maps \mathcal{D} and Loss L_D . To estimate the depth z_w for each junction w , we define the junction-wise depth map as

$$\hat{\mathcal{D}}_t(\mathbf{p}) = \begin{cases} z_w & \exists w \in \mathbf{V}_t : \mathbf{p} = \lfloor \frac{\mathbf{w}}{4} \rfloor, \\ 0 & \text{otherwise.} \end{cases}$$

In many datasets with unknown depth units and camera intrinsic matrix K , z_w remains a relative scale instead of absolute depth. To remove the ambiguity from global scaling, we use scale-invariant loss (SILog) which has been introduced in single image depth estimation literature [4]. It removes the influence of the global scale by summing the log difference between each pixel pair.

$$L_D(\mathcal{D}, \hat{\mathcal{D}}) \doteq \sum_t \frac{1}{n_t} \sum_{\mathbf{p} \in \mathbf{V}_t} (\log \mathcal{D}_t(\mathbf{p}) - \log \hat{\mathcal{D}}_t(\mathbf{p}))^2 - \sum_t \frac{1}{n_t^2} \left(\sum_{\mathbf{p} \in \mathbf{V}_t} \log \mathcal{D}_t(\mathbf{p}) - \log \hat{\mathcal{D}}_t(\mathbf{p}) \right)^2.$$

Vanishing Point Map \mathbf{V} and Loss L_V . Lines in man-made outdoor scenes often cluster around the three mutually orthogonal directions. Let $i \in \{1, 2, 3\}$ represent these three

directions. In perspective geometry, parallel lines in direction i will intersect at the same vanishing point $(V_{i,x}, V_{i,y})$ in the image space, possibly at infinity. To avoid $V_{i,x}$ or $V_{i,y}$ becoming too large, we normalize the vector so that

$$\mathbf{V}_i = \frac{1}{V_{i,x}^2 + V_{i,y}^2 + 1} [V_{i,x}, V_{i,y}, 1]^T, \quad (2)$$

similar to the Gaussian Sphere representation [1]. Because the two horizontal vanishing points \mathbf{V}_1 and \mathbf{V}_2 are order agnostic from a single RGB image, we use the Chamfer ℓ_2 -loss for \mathbf{V}_1 and \mathbf{V}_2 , and the ℓ_2 -loss for \mathbf{V}_3 (the vertical vanishing point):

$$L_V(\mathbf{V}, \hat{\mathbf{V}}) \doteq \min(\|\mathbf{V}_1 - \hat{\mathbf{V}}_1\|, \|\mathbf{V}_2 - \hat{\mathbf{V}}_1\|) + \min(\|\mathbf{V}_1 - \hat{\mathbf{V}}_2\|, \|\mathbf{V}_2 - \hat{\mathbf{V}}_2\|) + \|\mathbf{V}_3 - \hat{\mathbf{V}}_3\|_2^2.$$

2.3. Heatmap Vectorization

As seen from Figure 2, the outputs of the neural network are essentially image-space 2.5D heatmaps of the desired wireframe. Vectorization is needed to obtain a compact wireframe representation.

Junction Vectorization. Recovering the junctions \mathbf{V} from the junction heatmaps J is straightforward. Let ϑ_C and ϑ_T be the thresholds for J_C and J_T . The junction candidate sets can be estimated as

$$\mathbf{V}_t = \{\mathbf{p} + \mathbf{O}_t(\mathbf{p}) \mid J_t(\mathbf{p}) \geq \vartheta_t\}, t \in \{C, T\}. \quad (3)$$

Line Vectorization. Line vectorization has two stages. In the first stage, we detect and construct the line candidates from all the corner C-junctions. This can be done by enumerating all the pairs of junctions $\mathbf{u}, \mathbf{v} \in \mathbf{V}_C$, connecting them, and testing if their line confidence score is greater than a threshold $c(\mathbf{u}, \mathbf{v}) \geq \vartheta_E$. The confidence score of a line with two endpoints \mathbf{u} and \mathbf{v} is given as $c(\mathbf{u}, \mathbf{v}) = \frac{1}{|\mathbf{uv}|} \sum_{\mathbf{p} \in P(\mathbf{u}, \mathbf{v})} E(\mathbf{p})$ where $P(\mathbf{u}, \mathbf{v})$ represents the set of pixels in the rasterized line $\mathbf{u}\mathbf{v}$, and $|\mathbf{uv}|$ represents the number of pixels in that line.

In the second stage, we construct all the lines between ‘‘T-T’’ and ‘‘T-C’’ junction pairs. We repeatedly add a T-junction to the wireframe if it is tested to be close to a detected line. Unlike corner C-junctions, the degree of a T-junction is always one. So for each T-junction, we find the best edge associated with it. This process is repeated until no more lines could be added. Finally, we run a post-processing procedure to remove lines that are too close or cross each other. By handling C-junctions and T-junctions separately, our line vectorization algorithm is both efficient and robust for scenes with hundreds of lines. A more detailed description is discussed in the supplementary material.

2.4. Image-Space 2.5D to World-Space 3D

So far, we have obtained vectorized junctions and lines in 2.5D image space with depth in a relative scale. However, in

scenarios such as AR and 3D design, absolute depth values are necessary for 6DoF manipulation of the 3D wireframe. In this section, we present the steps to estimate them with our network predicted vanishing points.

2.4.1 Calibration from Vanishing Points

In datasets such as MegaDepth [17], the camera calibration matrix $K \in \mathbb{R}^{3 \times 3}$ of each image is unknown, although it is critical for a full 3D wireframe reconstruction. Fortunately, calibration matrices can be inferred from three mutually orthogonal vanishing points if the scenes are mostly Manhattan. According to [21], if we transform the orthogonal vanishing points V_i to the calibrated coordinates $\bar{V}_i \doteq K^{-1}V_i$, then \bar{V}_i should be mutually orthogonal, i.e.,

$$V_i K^{-T} K^{-1} V_j = 0, \quad \forall i, j \in \{1, 2, 3\}, i \neq j.$$

These equations impose three linearly independent constraints on $K^{-T} K^{-1}$ and would enable solving up to three unknown parameters in the calibration matrix, such as the optical center and the focal length.

2.4.2 Depth Refinement with Vanishing Points

Due to the estimation error, the predicted depth map may not be consistent with the detected vanishing points V_i . In practice, we find the neural network performs better on estimating the vanishing points than predicting the 2.5D depth map. This is probably because there are more geometric cues for the vanishing points, while estimating depth requires priors from data. Furthermore, the unit of the depth map might be unknown due to the dataset (e.g., MegaDepth) and the usage of SILog loss. Therefore, we use the vanishing points to refine the junction depth and determine its absolute value. Let $\tilde{z}_w \doteq \mathcal{D}_{J_w}(\mathbf{w})$ be the predicted depth for junction \mathbf{w} from our neural network. We design the following convex objective function:

$$\min_{\mathbf{z}, \alpha} \sum_{i=1}^3 \sum_{(\mathbf{u}, \mathbf{v}) \in A_i} \|(z_u \bar{\mathbf{u}} - z_v \bar{\mathbf{v}}) \times \bar{\mathbf{V}}_i\|_2 + \lambda_R \sum_{\mathbf{w} \in V} \|z_w - \alpha \tilde{z}_w\|_2^2 \quad (4)$$

$$\text{subject to } z_w \geq 1, \quad \forall \mathbf{w} \in V, \quad (5)$$

$$\lambda z_u + (1 - \lambda) z_v \leq z_w, \quad (6)$$

$$\text{where } \mathbf{w} \in V_T, (\mathbf{u}, \mathbf{v}) \in E : \mathbf{w} = \lambda \mathbf{u} + (1 - \lambda) \mathbf{v},$$

where A_i represents the set of lines corresponding to vanishing point i ; α resolves the scale ambiguity in the depth dimension; $\bar{\mathbf{u}} \doteq K^{-1}[u_x \ u_y \ 1]^T$ is the vertex position in the calibrated coordinate. The goal of the first term in Equation (4) is to encourage the line $(z_u \bar{\mathbf{u}}, z_v \bar{\mathbf{v}})$ parallel to vanishing point $\bar{\mathbf{V}}_i$ by penalizing over the parallelogram area spanned by those two vectors. The second term regularizes z_w so that it is close to the network's estimation \tilde{z}_w up to a scale. Equation (5) prevents the degenerating solution $\mathbf{z} = \mathbf{0}$. Equation (6) is a convex relaxation of $\frac{\lambda}{z_u} + \frac{1-\lambda}{z_v} \geq \frac{1}{z_w}$, the

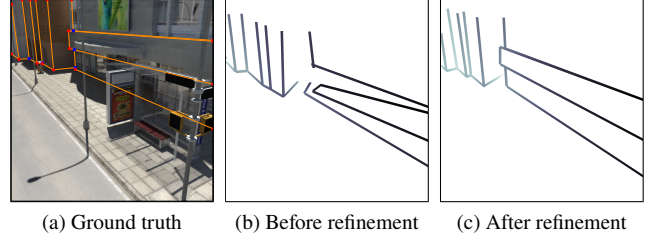


Figure 3: Depth refinement with vanishing points. The middle figure shows a rendering of the wireframe from \tilde{z}_w from a slightly different view, while the right figure shows the wireframe improved by the optimization in Section 2.4.2.

depth constraint for T-junctions. Figure 3 shows the effectiveness of the refinement.

3. Datasets and Annotation

One of the bottlenecks of supervised learning is inadequate dataset for training and testing. Previously, [13] develops a dataset for 2D wireframe detection. However, their dataset does not contain the 3D depth or the type of junctions. To the best of our knowledge, there is no public image dataset that has both wireframe and 3D information. To validate our approach, we create a hybrid dataset with a larger number of synthetic images of city scenes and smaller number of real images. The former has accurate 3D geometry and automatically annotated ground truth 3D wireframes from mesh edges, while the latter is manually labelled with less accurate 3D information.

Synthetic City Dataset. To obtain a large number of images with accurate geometrical wireframes, we use a progressively generated 3D mesh repository, SceneCity⁴. The dataset is made up of simple polygons with artist-tuned materials and textures. We extract the C-junctions from the vertices of the mesh and compute T-junctions using computational geometry algorithms and OpenGL. Our dataset includes 230 cities, each containing 8×8 city blocks. The cities have different building arrangements and lighting conditions by varying the sky maps. We randomly generate 100 viewpoints for each city based on criteria such as the number of captured buildings to simulate hand-held and drone cameras. The synthetic outdoor images are then rendered through global illumination by Blender modeler, which provides 23,000 images in total. We use the images of the first 227 cities for training and the rest 3 cities for validation.

Realistic Landmark Dataset. The MegaDepth v1 dataset [18] contains real images of 196 landmarks in the world. It also contains the depth maps of these images via structure from motion. We select about 200 images that approximately meet the assumptions of our method, manually label

⁴<https://www.cgchan.com/>

their wireframes, and register them with the rough 3D depth.

In our experiments, we pretrain our network on the synthetic dataset, and then use 2/3 of the real images to finetune the model. The remaining 1/3 is for testing.

4. Experiments

We conduct extensive experiments to evaluate our method and validate the design of our pipeline with ablation studies. In addition, we compare our method with the state-of-the-art 2D wireframe extraction approaches. We then evaluate the performance of our vanishing point estimation and depth refinement steps. Finally, we demonstrate the examples of our 3D wireframe reconstruction.

4.1. Implementation Details

Our backbone is a two-stack hourglass network [23]. Each stack consists of 6 stride-2 residual blocks and 6 nearest neighbour upsamplers. After the stacked hourglass feature extractor, we insert different “head” modules for each map. Each head contains a 3×3 convolutional layer to reduce the number of channels followed by a 1×1 convolutional layer to compute the corresponding map. For vanishing point regression, we use a different head with two consecutive stride-2 convolution layers followed by a global average pooling layer and a fully-connected layer to regress the position of the vanishing points.

During the training, the ADAM [16] optimizer is used. The learning rate and weight decay are set to 8×10^{-4} and 1×10^{-5} . All the experiments are conducted on four NVIDIA GTX 1080Ti GPUs, with each GPU holding 12 mini-batches. For the synthetic dataset, we train our network for 25 epochs. The loss weights are set as $\lambda_J = 2.0$, $\lambda_O = 0.25$, $\lambda_E = 3.0$, and $\lambda_D = 0.1$ so that all the loss terms are roughly equal. For the real-world dataset, we initialize the network with the one trained on the synthetic data and use a 10^{-4} learning rate to train for 5 epochs. We horizontally flip the input image as data-augmentation. Unless otherwise stated, the input images are cropped to 512×512 . The final output is of stride 4, i.e., with size 128×128 . During heatmap vectorization, we use the hyper-parameter $\theta_C = 0.2$, $\theta_T = 0.3$, and $\theta_E = 0.65$.

4.2. Evaluation Metrics

We use the standard AP (average precision) from object detection [7] to evaluate our junction prediction results. Our algorithm produces a set of junctions and their associated scores. The prediction is considered correct if its ℓ^2 distance to the nearest ground truth is within a threshold. By this criterion, we can draw the precision-recall curve and compute the *mean AP* (mAP) as the area under this curve averaging over several different thresholds of junction distance.

In our implementation, mAP is averaged over thresholds 0.5, 1.0, and 2.0. In practical applications, long edges

between junctions are typically preferred over short ones. Therefore, we weight the mAP metric by the sum of the length of the lines connected to that junction. We use AP^C and AP^T to represent such weighted mAP metric for C-junctions and T-junctions, respectively. We use the intersection over union (IoU) metric to evaluate the quality of line heatmaps. For junction depth map, we evaluate it on the positions of the ground truth junctions with the scale invariant logarithmic error (SILog) [4, 9].

4.3. Ablation on Joint Training and Loss Functions

We run a series of experiments to investigate how different feature designs and multi-task learning strategies affect the wireframe detection accuracy. Table 1 presents our ablation studies with different combinations of tasks to research the effects of joint training. We also evaluate the choice of ℓ_1 - and ℓ_2 -losses for offset regression and the ordinary loss [17] for depth estimation. We conclude that:

1. Regressing offset is significantly important for localizing junctions (7.4 points for AP^C and 3 points for AP^T), by comparing rows (a-c). In addition, ℓ_2 loss is better than ℓ_1 loss, probably due to its smoothness.
2. Joint training junctions and lines improve in both tasks. Rows (c-e) show improvements with about 1.5 points in AP^C , and 0.9 point in AP^T and line IoU. This indicates the tight relation between junctions and lines.
3. For depth estimation, we test the ordinal loss from [17]. To our surprise, it does not improve the performance on our dataset (rows (f-g)). We hypothesize that this is because the relative orders of sparsely annotated junctions are harder to predict than the foreground/background relationship in [17].
4. According to rows (f) and (h), joint training with junctions and lines slightly improves the performance of depth estimation by 0.55 SILOG point.

4.4. Comparison with 2D Wireframe Extraction

One recent work related to our system is [13], which extracts 2D wireframes from single RGB images. However, it has several fundamental differences from ours: 1) It does not differentiate between corner C-junctions and occluding T-junctions. 2) Its outputs are only 2D wireframes while ours are 3D. 3) It trains two separated networks for detecting junctions and lines. 4) It detects texture lines while ours only detects geometric wireframes.

In this experiment, we compare the performance with [13]. The goal of this experiment is to validate the importance of joint training. Therefore we follow the exact same training procedure and vectorization algorithms as in [13] except for the unified objective function and network structure. Figure 4 shows the comparison of precision and recall curves evaluated on the test images, using the same evaluation metrics as in [13]. Note that due to different network

	supervisions						metrics			
	J	O	E	\mathcal{D}			J	E	\mathcal{D}	
	CE	ℓ_1	ℓ_2	CE	SILog	Ord	AP^C	AP^T	IoU_E	SILog
(a)	✓						65.4	57.1	/	/
(b)	✓	✓					69.3	55.8	/	/
(c)	✓		✓				72.8	60.1	/	/
(d)				✓			/	/	73.3	/
(e)	✓		✓	✓			74.3	61.0	74.2	/
(f)					✓		/	/	/	3.59
(g)					✓	✓	/	/	/	4.14
(h)	✓	✓	✓	✓			74.4	61.2	74.3	3.04

Table 1: Ablation study of multi-task learning on 3D wireframe parsing. The columns under “supervisions” indicate what losses and supervisions are used during training; the columns under “metrics” indicate the performance given such supervision during evaluation. The second row shows the symbols of the feature maps; the third row shows the loss function names of the corresponding maps. “CE” stands for the cross entropy loss, “SILog” loss is proposed by [4], and “Ord” represents the ordinary loss in [17]. “/” indicates that the maps are not generated and thus not evaluable.

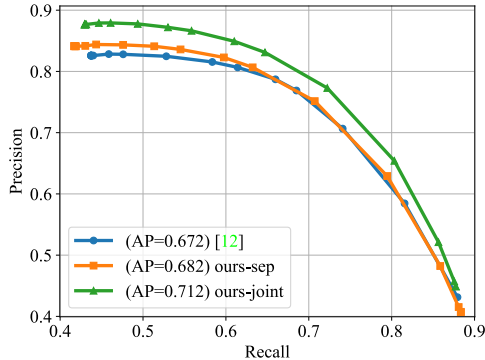


Figure 4: Comparison with [13] on 2D wireframe detection. We improve the baseline method by 4 points.

designs, their model has about 30M parameters while ours only has 19M. With fewer parameters, our system achieves 4-point AP improvement over [13] on the 2D wireframe detection task.

As a sanity check, we also train our network separately for lines and junctions, as shown by the green curve in Figure 4. The result is only slightly better than [13]. This experiment shows that our performance gain is from jointly trained objectives instead of neural network engineering.

4.5. Vanishing Points and Depth Refinement

In Section 2.4, vanishing point estimation and depth refinement are used in the last stage of the 3D wireframe

	avg[E_V]	med[E_V]	avg[E_f]	med[E_f]	failures
Ours	2.69°	1.55°	4.02%	1.38%	2.3%
[5, 27]	4.65°	0.14°	12.40%	0.21%	20.0%

Table 2: Performance comparison between our method and LSD/J-linkage [5, 27] for vanishing point detection. E_V represents the angular error of V_i in degree, E_f represents the relative error of the recovered camera focal lengths, and “failures” represents the percentage of cases whose $E_V > 8^\circ$.

representation. Their robustness and precision are critical to the final quality of the system output. In this section, we conduct experiments to evaluate the performance of these methods.

For vanishing point detection, Table 2 shows the performance comparison between our neural network-based method and the J-Linkage clustering algorithm [27, 26] with the LSD line detector [5] on the synthetic dataset. We find that our method is more robust in term of the percentage of failures and average error, while the traditional line cluster algorithm is more accurate when it does not fail. This is because LSD/J-linkage applies a stronger geometric prior, while the neural network learns the concept from the data. We choose our method for its simplicity and robustness, as the focus of this project is more on the 3D wireframe representation side, but we believe the performance can be further improved by engineering a hybrid algorithm or designing a better network structure.

We also compare the error of the junction depth before and after depth refinement in term of SILog. We find that on 65% of the testing cases, the error is smaller after the refinement. This shows that the geometric constraints from vanishing points does help improve the accuracy of the junction depth in general. On the other hand, the depth refinement may not be as effective when the vanishing points are not precise enough, or the scene is too complex so that there are many erroneous lines in the wireframe. Some failure cases can be found in the supplementary material.

4.6. 3D Wireframe Reconstruction Results

We test our 3D wireframe reconstruction method on both the synthetic dataset and the real images. Examples illustrating the visual quality of the final reconstruction are shown in Figures 5 and 6. A video demonstration can be find in <https://youtu.be/l3sUtdPJPY>. We cannot show the ground truth 3D wireframes for the real landmark dataset due to its incomplete depth maps and lack of ground truth camera calibrations.

5. Future Work

We have shown the feasibility of learning and detecting high-level geometric features such as a wireframe of lines and junctions for the recovery of compact 3D geometry of an

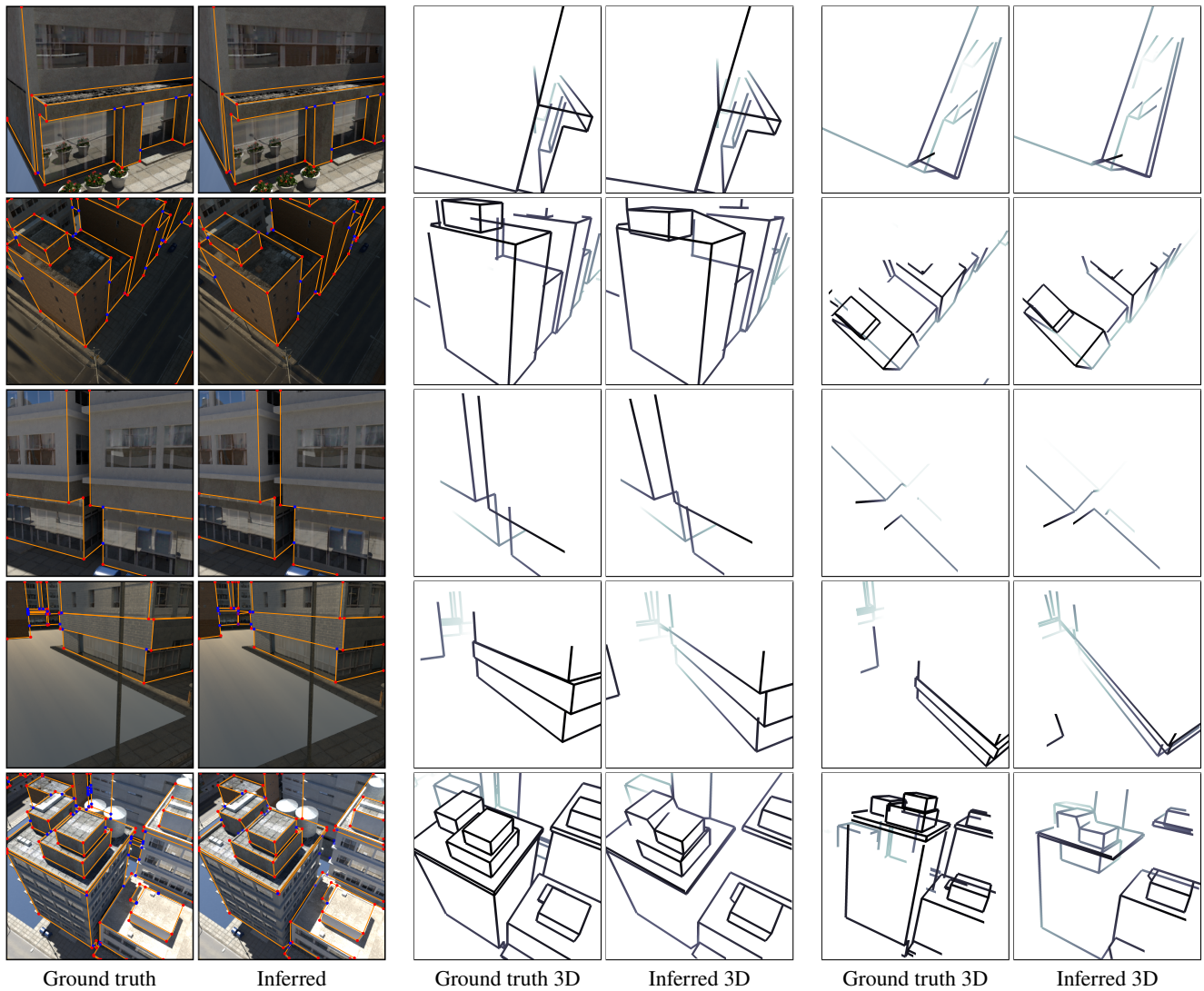


Figure 5: Test results on the synthetic SceneCity image dataset. Left group: comparison between the ground truth (column 1) and our predictions (column 2). Middle (columns 3-4) and right groups (columns 5-6): novel views of the ground truths and our reconstructions to demonstrate the 3D representation of the scene. The color of the wireframes visualizes depth.

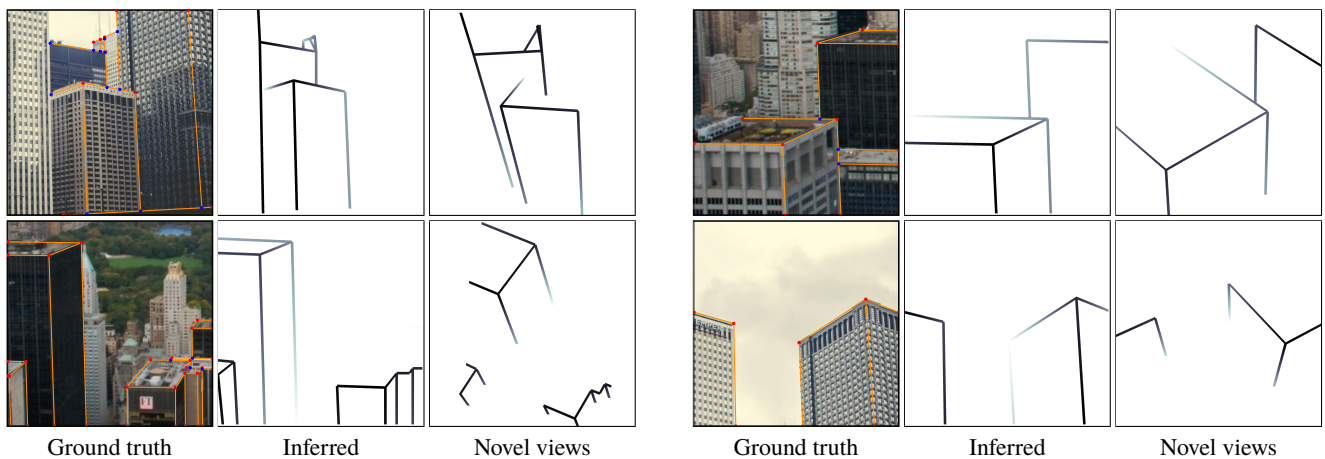


Figure 6: Test results on real images from MegaDepth.

urban scene, even from a single view. To further improve the performance of our system and apply to diverse real-world scenes (both indoors and outdoors), we need new datasets with high-quality images and high-level 3D models of such scenes. We also plan to extend our wireframe representations to accommodate curves, surfaces, and more complex geometric primitives, and generalize the 3D reconstruction methods to handle scenes that are not entirely Manhattan.

References

- [1] S. T. Barnard. Interpreting perspective images. *Artificial Intelligence*, 21(4):435–462, Nov. 1983. [4](#)
- [2] M. Bertamini, M. Helmy, and D. Bates. The visual system prioritizes locations near corners of surfaces (not just locations near a corner). *Attention, Perception, & Psychophysics*, 75(8):1748–1760, Nov 2013. [1](#)
- [3] J. M. Coughlan and A. L. Yuille. Manhattan world: Compass direction from a single image by bayesian inference. In *ICCV*, 1999. [2](#)
- [4] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014. [4](#), [6](#), [7](#)
- [5] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular slam. In *ECCV*. 2014. [2](#), [7](#)
- [6] S. M. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu, and D. Hassabis. Neural scene representation and rendering. *Science*, 2018. [2](#)
- [7] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. [6](#)
- [8] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Manhattan-world stereo. In *CVPR*, 2009. [2](#)
- [9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 2013. [6](#)
- [10] T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry. AtlasNet: A papier-mâché approach to learning 3D surface generation. In *CVPR*, 2018. [2](#)
- [11] M. Hofer, M. Maurer, and H. Bischof. Efficient 3D scene abstraction using line segments. *Computer Vision and Image Understanding*, Apr. 2017. [2](#)
- [12] J. Huang, A. Dai, L. Guibas, and M. Niessner. 3Dlite: Towards commodity 3D scanning for content creation. *ACM Trans. Graph.*, 2017. [1](#)
- [13] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, and Y. Ma. Learning to parse wireframes in images of man-made environments. In *CVPR*, 2018. [2](#), [5](#), [6](#), [7](#), [11](#)
- [14] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011. [1](#)
- [15] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. [1](#)
- [16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. [6](#)
- [17] Z. Li and N. Snavely. MegaDepth: Learning single-view depth prediction from internet photos. In *Computer Vision and Pattern Recognition (CVPR)*, 2018. [5](#), [6](#), [7](#)
- [18] Z. Li and N. Snavely. MegaDepth: Learning single-view depth prediction from internet photos. In *CVPR*, 2018. [5](#)
- [19] C. Liu, J. Yang, D. Ceylan, E. Yumer, and Y. Furukawa. PlaneNet: Piece-wise planar reconstruction from a single RGB image. In *CVPR*, 2018. [2](#)
- [20] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH computer graphics*, volume 21, pages 163–169. ACM, 1987. [1](#)
- [21] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. SpringerVerlag, 2003. [5](#)
- [22] R. Mur-Artal, J. Montiel, and J. D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 2015. [2](#)
- [23] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016. [3](#), [6](#), [11](#)
- [24] S. Ramalingam and M. Brand. Lifting 3D manhattan lines from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 497–504, 2013. [2](#)
- [25] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2017. [2](#)
- [26] J.-P. Tardif. Non-iterative approach for fast and accurate vanishing point detection. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1250–1257. IEEE, 2009. [7](#)
- [27] R. Toldo and A. Fusiello. Robust multiple structures estimation with J-linkage. In *European conference on computer vision*, pages 537–547. Springer, 2008. [7](#)
- [28] X. Wang, K. Chen, Z. Huang, C. Yao, and W. Liu. Point linking network for object detection. *arXiv*, 2017. [3](#)
- [29] J. Wu, T. Xue, J. J. Lim, Y. Tian, J. B. Tenenbaum, A. Torralba, and W. T. Freeman. Single image 3D interpreter network. In *European Conference on Computer Vision*, pages 365–382. Springer, 2016. [2](#)
- [30] F. Yang and Z. Zhou. Recovering 3D planes from a single image via convolutional neural networks. In *ECCV*, 2018. [2](#)
- [31] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016. [2](#)
- [32] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *CVPR*, 2017. [2](#)

- [33] Z. Zhang, A. Ganesh, X. Liang, and Y. Ma. Tilt: Transform invariant low-rank textures. *International journal of computer vision*, 99(1):1–24, 2012. 2
- [34] A. Zingl. A rasterizing algorithm for drawing curves, 2012. 4
- [35] C. Zou, A. Colburn, Q. Shan, and D. Hoiem. LayoutNet: Reconstructing the 3D room layout from a single RGB image. In *CVPR*, 2018. 2

A. Supplementary Materials

A.1. Pseudocode for Line Vectorization

Algorithm 1 gives a more detailed description for line vectorization. The algorithm takes the C-junction set V_C and T-junction set V_T as the input and outputs a vectorized wireframe (V, E) . In the first stage (Lines 2-3), we find the lines among C-junctions according to the line confidence function. The procedure PRUNE-LINES greedily removes the lines with the lowest confidence that either intersect with other lines (Line 32) or are too close to other lines in term of the polar angle (Line 35). In the second stage (Lines 4-25), we add the T-junctions into the wireframe. From Lines 6-14, we find the T-junctions that are on the existing wireframe. We first adjust the positions of those T-junctions by projecting them onto the line (Line 9) and then add them to the candidate T-junction set V' (Line 10). Because the degree of a T-junction is always one, we try to find the connection with the highest confidence for those candidates T-junctions (Lines 15-25). We repeat the this process until V , V' , and E remain the same in the last iteration.

A.2. Line Assignments for Vanishing Points

In Equation (4), we need to find the set of lines $A_i \subseteq E$ corresponding to the vanishing point i . Mathematically, we define the objective function

$$\min_A \sum_i \sum_{(u,v) \in A_i} \|(u - V_i) \times (u - v)\|_2,$$

where $\|(\cdot) \times (\cdot)\|_2$ can be understood as the parallelogram area formed by two vectors. Since each line in this equation is mutually independent, we can solve this optimization problem by greedily assigning each line to the best vanishing point i to minimize the objective function.

A.3. Sampled Failure Cases and Discussions

Figure 7 demonstrates some failure cases in our SceneCity dataset. We found that our pipeline might not work well on the scenes in which there are many lines and junctions that are close to each other. This is because the resolution of the output heat map is 128×128 , so any detail whose size is below two or three pixels might get lost during the vectorization stage. Therefore, one of our future work is to explore the possibility of using high-resolution

Algorithm 1 Edge Vectorization Algorithm

Require: Candidate C-junction set V_C , T-junction set V_T .

Require: Hyper-parameters η_c and η_o .

Ensure: Wireframe (V, E) .

```

1: procedure VECTORIZE( $V_C, V_T$ )
2:    $V \leftarrow V_C$ 
3:    $E \leftarrow \text{PRUNE-LINES}(\{(u, v) | u, v \in V, c(u, v) > \eta_c\})$ 
4:    $V' \leftarrow \emptyset$ 
5:   while  $V, V'$ , or  $E$  change in the last iteration do
6:     for all  $w \in V_T$  do
7:       for all  $e = (u, v) \in E$  do
8:         if  $w$  is near  $e$  then
9:           project  $w$  to the line  $e$ 
10:           $V' \leftarrow V' \cup \{w\}$ 
11:          break
12:        end if
13:      end for
14:    end for
15:    for all  $u \in V'$  do
16:       $v \leftarrow \arg\max_{v \in V \cup V'} c(u, v)$ 
17:      if  $c(u, v) \geq \eta_c$  then
18:         $V' \leftarrow V' \setminus \{u\}$ 
19:         $V \leftarrow V \cup \{u\}$ 
20:         $E \leftarrow E \cup \{(u, v)\}$ 
21:      end if
22:    end for
23:     $V_T \leftarrow V_T \setminus (V \cup V')$ 
24:  end while
25:   $E \leftarrow \text{PRUNE-LINES}(E)$ 
26:  return  $(V, E)$ 
27: end procedure
28: procedure PRUNE-LINES( $E$ )
29:  sort  $E$  w.r.t confidence values in descending order
30:   $E' \leftarrow \emptyset$ 
31:  for all  $e \in E$  do
32:    if  $\exists e' \in E' : e$  intersects with  $e'$  then
33:      continue
34:    end if
35:    if  $\exists e' \in E' : e' \cap e \neq \emptyset$  and  $\angle(e, e') < \eta_o$  then
36:      continue
37:    end if
38:     $E' \leftarrow E' \cup \{e\}$ 
39:  end for
40:  return  $E'$ 
41: end procedure

```

input and output images. There are also issues in the 3D depth refinement stage. When the scene is complex, finding the assignment A_i for each line can be hard, due to the error in the junction position and line direction. In addition, the term contributed by erroneous lines in Equation (4) can make the depth of some junctions inaccurate. Such problem might potentially be alleviated by increasing the resolution

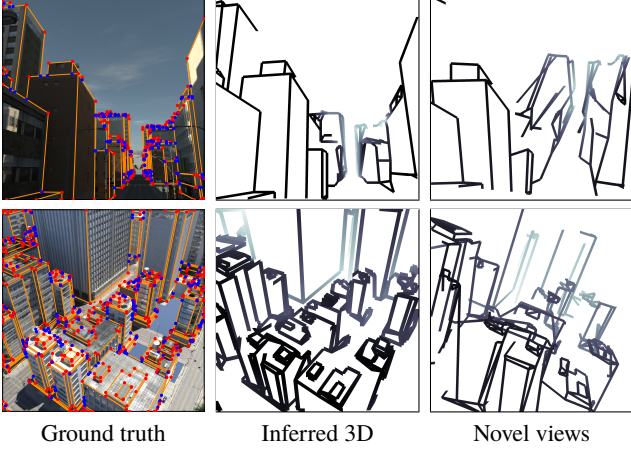


Figure 7: Failure cases on the SceneCity dataset.

of the input and output images, using a more data-driven method, designing a better objective function, or employing a RANSIC approach in those two stages.

A.4. Network Comparison

In this section, we discuss the difference between our network and the one in [13]. Our network is similar as their line detection network with the difference in the following perspectives:

1. In each hourglass module, they use two consecutive residual modules (RM) at each spatial resolution while we only use one RM, resulting less parameters in each hourglass module. Note that our design is the same as the original hourglass paper [23], which enables us to use more RM in each hourglass module and reduce the computational complexity since more computation is allocated in lower resolution stages. We adopt such design since we find using two RM gives negligible gains to the performance compared with only one.
2. We apply the intermediate supervision to the stacked hourglass network. For each hourglass modules, the loss term associated with the predicted heat maps is added to the final loss. [13] does not such intermediate supervision in their method. We find such intermediate supervision vital in both our synthetic dataset as well as their 2D dataset in terms of both accuracy and robustness.
3. We observe that using 2 stacked hourglass modules gives a similar performance as the 5 stacked hourglass modules in [13]. On the other hand, using 2 stacks consumes less memory, which gives us more design flexibility. For example, we are able to utilize a larger batch size to make the gradients more stable.