

# Learning to Reconstruct 3D Manhattan Wireframes from a Single Image

Yichao Zhou<sup>1,2,\*</sup> Haozhi Qi<sup>1</sup> Yuexiang Zhai<sup>1</sup> Qi Sun<sup>2</sup> Zhili Chen<sup>2</sup> Li-Yi Wei<sup>2</sup> Yi Ma<sup>1</sup>

<sup>1</sup>UC Berkeley

<sup>2</sup>Adobe 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 depths and vanishing points. Compared with the state-of-the-art learning-based wireframe detection methods, our network is simpler and more unified, leading to better 2D wireframe detection. With global structural priors from parallelism, 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 have been made public at <https://github.com/zhou13/shapeunify>.

## 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 Toolkit) 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 [13] for better accuracy. The final scene

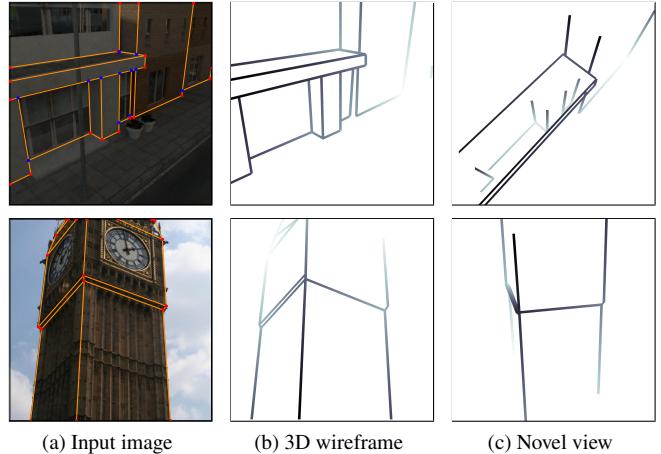


Figure 1. Column (a) shows the input images overlaid with the groundtruth wireframes, in which the red and blue dots represent the C- and T-type junctions, respectively. Column (b) shows the predicted 3D wireframe from our system, with grayscale visualizing depth. Column (c) shows alternative views of (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 [11] and mesh refinement [14, 19] 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 [1]. However, it remains challenging for vision algorithms to detect and utilize 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

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

detecting planes [29, 18], surfaces [9], 2D wireframes [12], room layouts [34], key points for mesh fitting [30, 28], and sparse scene representations from multiple images [5].

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 [7], ORB [21], or line segments [10, 4, 23] 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 (such as globally or locally Manhattan [2, 7, 23]), 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 [24, 31]. 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 [12], 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 lines. Specifically,  $E$  represents lines from physical intersections of two planes

while  $V$  represents (physical or projective) intersections of lines among  $E$ . Unlike [10, 12], our  $E$  totally 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.<sup>1</sup> By ruling out planar textural lines, we can group the junctions into two categories. Let  $J_v \in \{C, T\}$  be the junction type of  $v$ , in which each junction can either be a *C-junction* ( $J_v = C$ ) or a *T-junction* ( $J_v = T$ ). 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  $V_C = \{v \in V \mid J_v = C\}$  and  $V_T = \{v \in V \mid J_v = T\}$ . We note that the number of lines incident to a T-junction in  $E$  is always 1 rather than 3 because a T-junction do not connect to the two foreground vertices in 3D. Junction types are important for inferring 3D wireframe geometry, as different 3D priors will be applied to each type.<sup>2</sup> For each C-junction  $v_c \in V_C$ , define  $z_{v_c}$  as the depth of vertex  $v_c$ , i.e., the  $z$  coordinate of  $v_c$  in the *camera space*. For each occlusional T-junction  $v_t \in V_T$ , we define  $z_{v_t}$  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 a single image is our goal.

### 2.2. From a 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, respectively. The meaning of each symbol is detailed in Section 2.2.2.

<sup>1</sup>In urban scenes, lines from regular textures (such as windows on a facade) do encode accurate scene geometry [32]. 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.

<sup>2</sup>There is another type of junctions which are caused by lines intersecting with the image boundary. We treat them as C-junctions for simplicity.

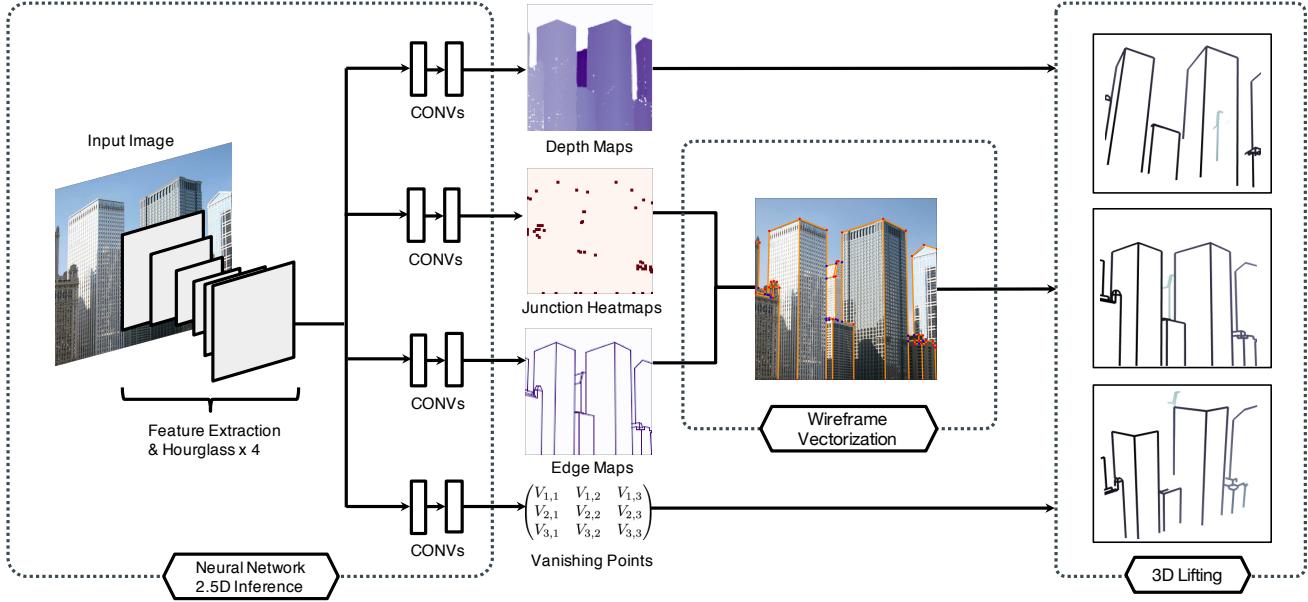


Figure 2. Overall pipeline of the proposed method.

### 2.2.1 Network Design

Our network structure is based on the stacked hourglass network [22]. The input images are cropped and re-scaled to  $512 \times 512$  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  $128 \times 128$ . 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:

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

The hyper-parameters  $\lambda_k$  represents the weight of each sub-loss. During experiments, we set  $\lambda$  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 junction heatmap

$$\hat{J}_t(\mathbf{p}) = \begin{cases} 1 & \exists \mathbf{v} \in V_t : \mathbf{p} = \lfloor \frac{\mathbf{v}}{4} \rfloor, \quad t \in \{C, T\}, \\ 0 & \text{otherwise} \end{cases}$$

where  $\mathbf{p}$  is the integer coordinate on the heatmap and  $\mathbf{v}$  is the coordinate of a junction with type  $t$  in the image space. Following [22], 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  $O$  and Loss  $L_O$ .** Comparing to the input image, the lower resolution of  $J$  might affect the precision of

junction positions. Inspired by [27], we use an offset map to store the difference vector from  $\hat{J}$  to its original position with sub-pixel accuracy:

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

We use the  $\ell_2$ -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

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

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  $\mathbf{O}_t(\mathbf{p}) \in [0, 1]^2$ . 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 [33] 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  $z_v$  for each junction  $v$ , we define the junction-wise depth map as

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

In many datasets with unknown depth units and camera intrinsic matrix  $K$ ,  $z_v$  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 the single image depth estimation literature [3]. It removes the influence of the global scale by summing the log difference between each pixel pair.

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

**Vanishing Point Map  $V$  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 prevent  $V_{i,x}$  or  $V_{i,y}$  from becoming too large, we normalize the vector so that

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

Because the two horizontal vanishing points  $V_1$  and  $V_2$  are order agnostic from a single RGB image, we use the Chamfer  $\ell_2$ -loss for  $V_1$  and  $V_2$ , and the  $\ell_2$ -loss for  $V_3$  (the vertical vanishing point):

$$\begin{aligned} L_V(V, \hat{V}) &\doteq \min(\|V_1 - \hat{V}_1\|, \|V_2 - \hat{V}_1\|) \\ &\quad + \min(\|V_1 - \hat{V}_2\|, \|V_2 - \hat{V}_2\|) + \|V_3 - \hat{V}_3\|_2^2. \end{aligned}$$

## 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  $V$  from the junction heatmaps  $J$  is straightforward. Let  $\vartheta_C$  and  $\vartheta_T$  be the thresholds for  $J_C$  and  $J_T$ . The junction candidate sets can be estimated as

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

**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  $\mathbf{u}, \mathbf{w} \in \mathcal{V}_C$ , connecting them, and testing if their line confidence score is greater than a threshold  $c(\mathbf{u}, \mathbf{w}) \geq \vartheta_E$ . The confidence score of a line with two endpoints  $\mathbf{u}$  and  $\mathbf{w}$  is given as  $c(\mathbf{u}, \mathbf{w}) = \frac{1}{|\mathbf{uw}|} \sum_{\mathbf{p} \in P(\mathbf{u}, \mathbf{w})} E(\mathbf{p})$  where  $P(\mathbf{u}, \mathbf{w})$  represents the set of pixels in the rasterized line  $\mathbf{uw}$ , and  $|\mathbf{uw}|$  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 [16], 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 [20], 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  $\tilde{z}_v \doteq \mathcal{D}_{J_v}(v)$  be the predicted depth for junction  $v$  from our neural network. We design the following convex objective function:

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

$$\text{subject to } z_v \geq 1, \quad \forall v \in V, \quad (5)$$

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

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

where  $A_i$  represents the set of lines corresponding to vanishing point  $i$ ;  $\alpha$  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  $(z_u \bar{\mathbf{u}}, z_w \bar{\mathbf{w}})$  parallel to vanishing point  $\bar{V}_i$  by penalizing over the parallelogram area

spanned by those two vectors. The second term regularizes  $z_v$  so that it is close to the network's estimation  $\tilde{z}_v$  up to a scale. Equation (5) prevents the degenerating solution  $z = \mathbf{0}$ . Equation (6) is a convex relaxation of  $\frac{\lambda}{z_u} + \frac{1-\lambda}{z_w} \geq \frac{1}{z_v}$ , the depth constraint for T-junctions.

## 3. Datasets and Annotation

One of the bottlenecks of supervised learning is inadequate dataset for training and testing. Previously, [12] 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.

**SceneCity Urban 3D Dataset (SU3).** To obtain a large number of images with accurate geometrical wireframes, we use a progressively generated 3D mesh repository, SceneCity<sup>3</sup>. 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  $8 \times 8$  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, 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 [17] 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 their wireframes, and register them with the rough 3D depth.

In our experiments, we pretrain our network on the SU3 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

<sup>3</sup><https://www.cgchan.com/>

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 [22]. 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  $3 \times 3$  convolutional layer to reduce the number of channels followed by a  $1 \times 1$  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 [15] optimizer is used. The learning rate and weight decay are set to  $8 \times 10^{-4}$  and  $1 \times 10^{-5}$ . All the experiments are conducted on four NVIDIA GTX 1080Ti GPUs, with each GPU holding 12 mini-batches. For the SceneCity Urban 3D 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 SU3 dataset 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  $512 \times 512$ . The final output is of stride 4, i.e., with size  $128 \times 128$ . During heatmap vectorization, we use the hyper-parameter  $\vartheta_C = 0.2$ ,  $\vartheta_T = 0.3$ , and  $\vartheta_E = 0.65$ .

#### 4.2. Evaluation Metrics

We use the standard AP (average precision) from object detection [6] to evaluate our junction prediction results. Our algorithm produces a set of junctions and their associated scores. The prediction is considered correct if its  $\ell^2$  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) [3, 8].

	supervisions				metrics							
	<b><math>J</math></b>	<b><math>O</math></b>	<b><math>E</math></b>	<b><math>D</math></b>	<b><math>J</math></b>	<b><math>E</math></b>	<b><math>D</math></b>					
	CE	$\ell_1$	$\ell_2$	CE	SILog	Ord	$AP^C$	$AP^T$	IoU <sub>E</sub>	SILog		
(a)	$\checkmark$						65.4	57.1	/	/		
(b)	$\checkmark$		$\checkmark$				69.3	55.8	/	/		
(c)	$\checkmark$		$\checkmark$				72.8	60.1	/	/		
(d)	$\checkmark$						/	/	73.3	/		
(e)	$\checkmark$		$\checkmark$				74.3	61.0	74.2	/		
(f)					$\checkmark$		/	/	/	3.59		
(g)					$\checkmark$		/	/	/	4.14		
(h)	$\checkmark$		$\checkmark$		$\checkmark$		<b>74.4</b>	<b>61.2</b>	<b>74.3</b>	<b>3.04</b>		

Table 1. 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 [3], and “Ord” represents the ordinary loss in [16]. “/” indicates that the maps are not generated and thus not evaluable.

#### 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  $\ell_1$ - and  $\ell_2$ -losses for offset regression and the ordinary loss [16] 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,  $\ell_2$  loss is better than  $\ell_1$  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 [16]. To our surprise, it does not improve the performance on our dataset (rows (f-g)). We hypothesis that this is because the relative orders of sparsely annotated junctions are harder to predict than the foreground/background relationship in [16].
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 [12], 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

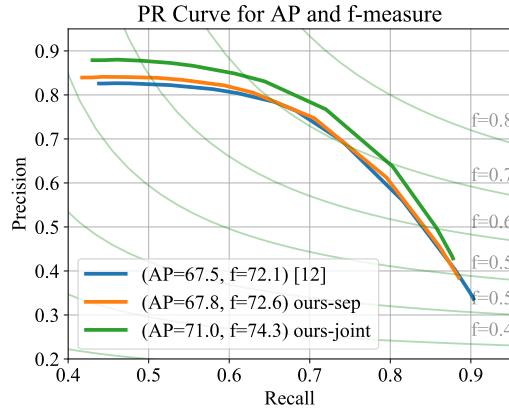


Figure 3. Comparison with [12] on 2D wireframe detection. We improve the baseline method by 4 points.

	avg[Ev]	med[Ev]	avg[Ef]	med[Ef]	failures
Ours	<b>2.69°</b>	1.55°	<b>4.02%</b>	1.38%	<b>2.3%</b>
[4, 26]	4.65°	<b>0.14°</b>	12.40%	<b>0.21%</b>	20.0%

Table 2. Performance comparison between our method and LSD/J-linkage [4, 26] 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  $E_V > 8^\circ$ .

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 [12]. 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 [12] except for the unified objective function and network structure. Figure 3 shows the comparison of precision and recall curves evaluated on the test images, using the same evaluation metrics as in [12]. Note that due to different network designs, their model has about 30M parameters while ours only has 19M. With fewer parameters, our system achieves 4-point AP improvement over [12] 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 3. The result is only slightly better than [12]. 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 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

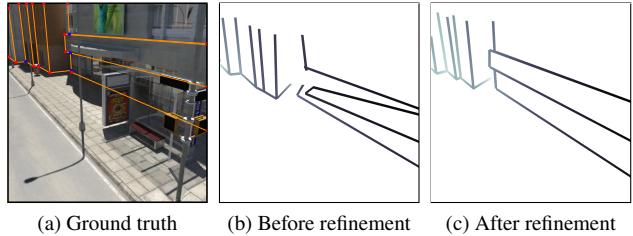


Figure 4. (b) shows a rendering of the wireframe from  $\tilde{z}_v$  from a slightly different view, while (c) shows the wireframe improved by the optimization in Section 2.4.2.

methods.

For vanishing point detection, Table 2 shows the performance comparison between our neural network-based method and the J-Linkage clustering algorithm [26, 25] with the LSD line detector [4] on the SU3 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. As shown in Figure 4, the depth refinement also improves the visual quality of the 3D wireframe. 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 found in <http://y2u.be/l3sUtddPJPY>. We do not show the ground truth 3D wireframes for the real landmark dataset due to its incomplete depth maps.

#### Acknowledgement

This work is partially supported by Sony US Research Center, Adobe Research, Berkeley BAIR, and Bytedance Research Lab.

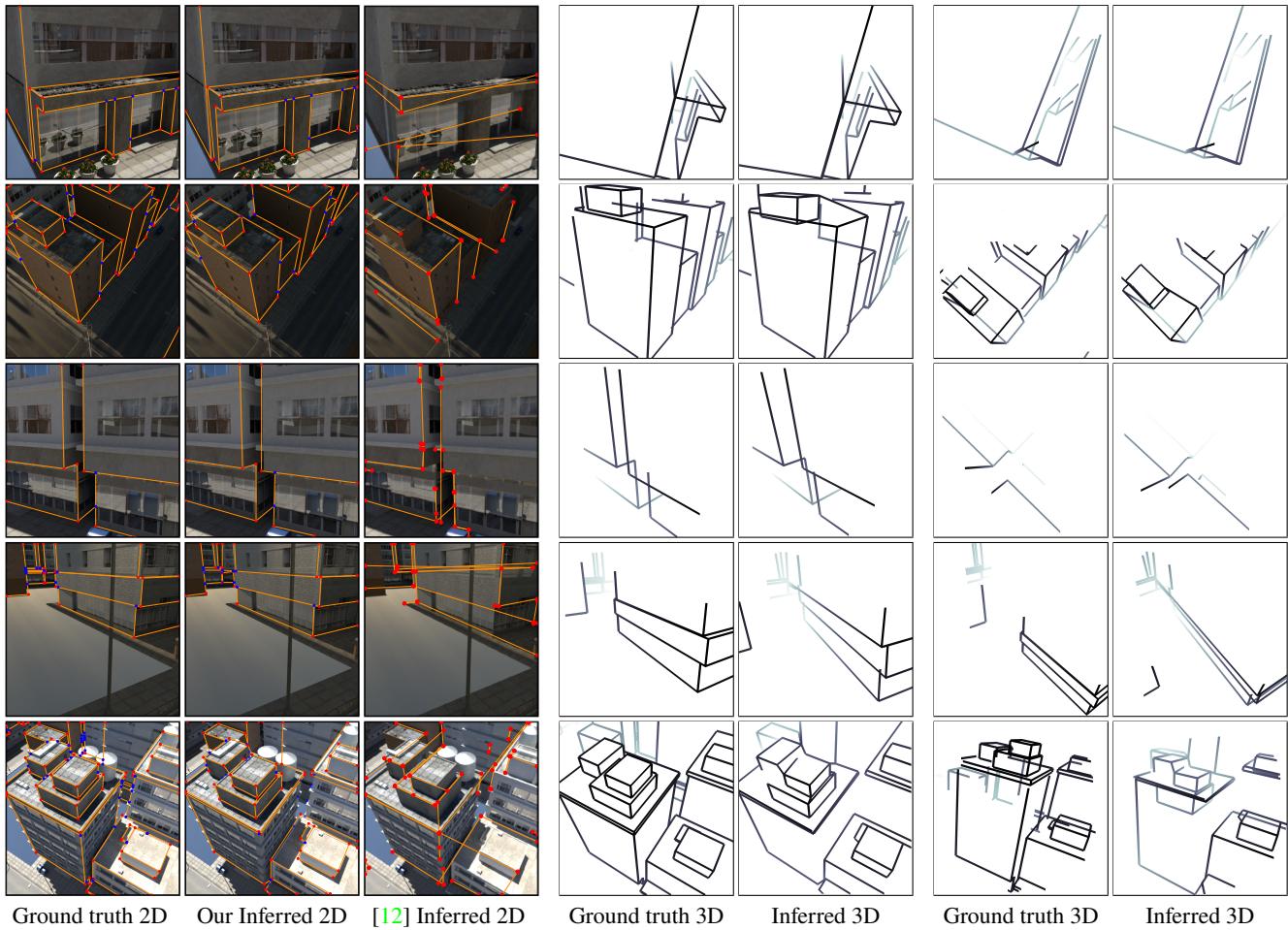


Figure 5. Left group: comparison of 2D results between the ground truth (column 1), our predictions (column 2), and the results from wireframe parser [12] (column 3). Middle (columns 4-5) and right groups (columns 6-7): novel views of the ground truths and our reconstructions to demonstrate the 3D representation of the scene. The color of the wireframes visualizes depth.

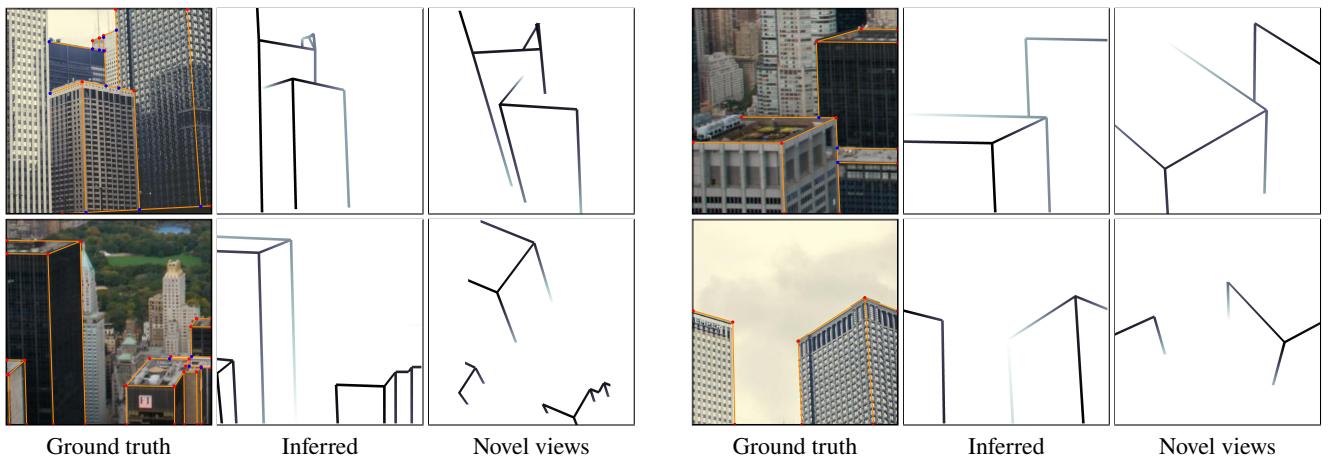


Figure 6. Test results on real images from MegaDepth.

## References

- [1] Marco Bertamini, Mai Helmy, and Daniel 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
- [2] James M Coughlan and Alan L Yuille. Manhattan world: Compass direction from a single image by bayesian inference. In *ICCV*, volume 2, pages 941–947, 1999. 2
- [3] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014. 4, 6
- [4] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular slam. In *ECCV*. 2014. 2, 7
- [5] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. Neural scene representation and rendering. *Science*, 2018. 2
- [6] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal visual object classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 6
- [7] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Manhattan-world stereo. In *CVPR*, 2009. 2
- [8] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 2013. 6
- [9] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A papier-mâché approach to learning 3D surface generation. In *CVPR*, 2018. 2
- [10] Manuel Hofer, Michael Maurer, and Horst Bischof. Efficient 3D scene abstraction using line segments. *Computer Vision and Image Understanding*, Apr. 2017. 2
- [11] Jingwei Huang, Angela Dai, Leonidas Guibas, and Matthias Niessner. 3Dlite: Towards commodity 3D scanning for content creation. *ACM Trans. Graph.*, 2017. 1
- [12] Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. Learning to parse wireframes in images of man-made environments. In *CVPR*, 2018. 2, 5, 6, 7, 8
- [13] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew 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
- [14] Michæl Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 1
- [15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6
- [16] Zhengqi Li and Noah Snavely. MegaDepth: Learning single-view depth prediction from internet photos. In *Computer Vision and Pattern Recognition (CVPR)*, 2018. 5, 6
- [17] Zhengqi Li and Noah Snavely. MegaDepth: Learning single-view depth prediction from internet photos. In *CVPR*, 2018. 5
- [18] Chen Liu, Jimei Yang, Duygu Ceylan, Ersin Yumer, and Yasutaka Furukawa. PlaneNet: Piece-wise planar reconstruction from a single RGB image. In *CVPR*, 2018. 2
- [19] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987. 1
- [20] Yi Ma, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. SpringerVerlag, 2003. 5
- [21] Raúl Mur-Artal, JMM Montiel, and Juan D Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 2015. 2
- [22] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016. 3, 6
- [23] Srikanth Ramalingam and Matthew Brand. Lifting 3D manhattan lines from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 497–504, 2013. 2
- [24] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2017. 2
- [25] Jean-Philippe 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
- [26] Roberto Toldo and Andrea Fusiello. Robust multiple structures estimation with J-linkage. In *European conference on computer vision*, pages 537–547. Springer, 2008. 7
- [27] Xinggang Wang, Kaibing Chen, Zilong Huang, Cong Yao, and Wenyu Liu. Point linking network for object detection. *arXiv*, 2017. 4
- [28] Jiajun Wu, Tianfan Xue, Joseph J Lim, Yuandong Tian, Joshua B Tenenbaum, Antonio Torralba, and William T Freeman. Single image 3D interpreter network. In *European Conference on Computer Vision*, pages 365–382. Springer, 2016. 2
- [29] Fengting Yang and Zihan Zhou. Recovering 3D planes from a single image via convolutional neural networks. In *ECCV*, 2018. 2
- [30] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016. 2
- [31] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *CVPR*, 2017. 2

- [32] Zhengdong Zhang, Arvind Ganesh, Xiao Liang, and Yi Ma. Tilt: Transform invariant low-rank textures. *International journal of computer vision*, 99(1):1–24, 2012. [2](#)
- [33] Alois Zingl. A rasterizing algorithm for drawing curves, 2012. [4](#)
- [34] Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. LayoutNet: Reconstructing the 3D room layout from a single RGB image. In *CVPR*, 2018. [2](#)