# Learning to Parse Wireframes in Images of Man-Made Environments

Kun Huang[1], Yifan Wang[1], Zihan Zhou[2], Tianjiao Ding[1], Shenghua Gao[1], and Yi Ma[3]

[1]ShanghaiTech University `{huangkun, wangyf, dingtj, gaoshh}@shanghaitech.edu.cn`
[2]The Pennsylvania State University `zzhou@ist.psu.edu`
[3]University of California, Berkeley `yima@eecs.berkeley.edu`

## Abstract

*In this paper, we propose a learning-based approach to the task of automatically extracting a "wireframe" representation for images of cluttered man-made environments. The wireframe (see Fig. 1) contains all salient straight lines and their junctions of the scene that encode efficiently and accurately large-scale geometry and object shapes. To this end, we have built a very large new dataset of over 5,000 images with wireframes thoroughly labelled by humans. We have proposed two convolutional neural networks that are suitable for extracting junctions and lines with large spatial support, respectively. The networks trained on our dataset have achieved significantly better performance than state-of-the-art methods for junction detection and line segment detection, respectively. We have conducted extensive experiments to evaluate quantitatively and qualitatively the wireframes obtained by our method, and have convincingly shown that effectively and efficiently parsing wireframes for images of man-made environments is a feasible goal within reach. Such wireframes could benefit many important visual tasks such as feature correspondence, 3D reconstruction, vision-based mapping, localization, and navigation. The data and source code are available at* `https://github.com/huangkuns/wireframe`.

## 1. Introduction

How to infer 3D geometric information of a scene from 2D images has been a fundamental problem in computer vision. Conventional approaches to build a 3D model typically rely on detecting, matching, and triangulating local image features (e.g. corners, edges, SIFT features, and patches). One great advantage of working with local features is that the system can be somewhat oblivious to the scene, as long as it contains sufficient distinguishable features. Meanwhile, modern applications of computer vision systems often require an autonomous agent (e.g., a car, a robot, or a UAV) to efficiently and effectively negotiate with a physical space in cluttered *man-made* (indoor or outdoor) environments. Such scenarios present significant
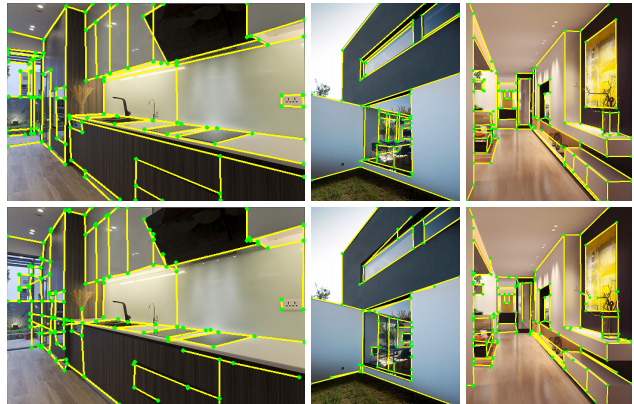


Figure 1. **First row:** Examples of typical indoor or outdoor scenes with geometrically meaningful wireframes labelled by humans; **Second row:** Wireframes automatically extracted by our method.

challenges to the current local-feature based approaches: Man-made environments typically consist of large texture-less surfaces (e.g. white walls or the ground); or they may be full of repetitive patterns hence local features are ambiguous to match; and the visual localization system is required to work robustly and accurately over extended routes and sometimes across very large baseline between views.

Nevertheless, the human vision system seems capable of effortlessly localizing or navigating among such environments arguably by exploiting larger-scale (global or semi-global) structural features or regularities of the scene. For instance, many works [6, 21, 14, 11, 35] have demonstrated that prior knowledge about the scene such as a Manhattan world could significantly benefit the 3D reconstruction tasks. The Manhattan assumption can often be violated in cluttered man-made environments, but it is rather safe to assume that man-made environments are dominantly piece-wise planar hence rich of visually salient lines (intersection of planes) and junctions (intersection of lines). Conceptually, such junctions or lines could just be a very small "sub-set" among the local corner features (or SIFTs) and edge features detected by conventional methods, but they already encode most information about larger-scale geometry of the scene. For simplicity, we refer to such a set of lines and their

1

intersected junctions collectively as a "wireframe".[1]

The goal of this work is to *study the feasibility of developing a vision system that could efficiently and effectively extract the wireframe of a man-made scene*. Intuitively, we wish such a system could emulate the level of human perception of the scene geometry, even from a single image. To this end, we have collected a database of over 5,000 images of typical indoor and outdoor environments and asked human subjects to manually label out all line segments and junctions that they believe to be important for understanding shape of regular objects or global geometric layout of the scene.[2] Fig. 1 (first row) shows some representative examples of labelled wireframes.

In the literature, several methods have been proposed to detect line segments [45] or junctions [35, 47] in the image, and to further reason about the 3D scene geometry using the detected features [21, 11, 10, 34, 51]. These methods typically take a bottom-up approach: First, line segments are detected in the image. Then, two or more segments are grouped to form candidate junctions. However, there are several inherent difficulties with this approach. *First*, by enumerating all pairs of line segments, a large number of intersections are created. But only a very small subset of them are true junctions in the image. To retrieve the true junctions, various heuristics as well as RANSAC-based verification techniques have been previously proposed. As result, such methods are often time consuming and even break down when the scene geometry and texture become complex. *Second*, detecting line segments itself is a difficult problem in computer vision. If one fails to detect certain line segments in the image, then it would be impossible for the method to find the associated junctions. *Third*, since all existing methods rely on low-level cues such as image gradients and edge features to detect line segments and junctions, they are generally unable to distinguish junctions and line segments that are of global geometric importance with those produced by local textures or irregular shapes.

In view of the fundamental difficulties of existing methods, we propose a complementary approach to wireframe (junctions and line segments) detection in this paper. Our method does not rely on grouping low-level features such as image gradients and edges. Instead, we directly learn detectors for junctions and lines of large spatial support from the above large-scale dataset of manually labeled junctions and lines. In particular, inspired by the recent success of convolutional neural networks in object detection, we design novel network architectures for junction and line detection, respectively. We then give a simple but effective method to

establish incidence relationships among the detected junctions and lines and produce a complete wireframe for the scene. Fig. 1 (second row) shows typical results of the proposed method. As one can see, our method is able to detect junctions formed by long line segments with weak gradient while significantly reducing the number of false detections. In addition, as the labelled junctions and line segments are primarily associated with salient, large-scale geometric structures of the scene, the resulting wireframe is geometrically more meaningful, emulating human perception of the scene's geometry.

**Contributions of this work** include: (i) the establishment of a large dataset for learning-based wireframe detection of man-made environments, and (ii) the development of effective, end-to-end trainable CNNs for detecting geometrically informative junctions and line segments. Comparing with existing methods on junction and line segment detection, our learning-based method has achieved, both quantitatively and qualitatively, superior performances on both tasks, hence convincingly verified the feasibility of wireframe parsing. Furthermore, both junction and line detection achieves almost real-time performance at the testing time, thus is suitable for a wide range of real-world applications such as feature correspondence, 3D reconstruction, vision-based mapping, localization and navigation.

## Related Work

**Edge and line segment detection.** Much work has been done to extract line segments from images. Existing methods are typically based on perceptual grouping of low-level cues (i.e., image gradients) [33, 45, 2, 23, 24]. A key challenge of these local approaches is the choice of some appropriate threshold to discriminate true line segments from false conjunctions. Another line of work extends Hough transform to line segment detection [30, 15, 50]. While Hough transform has the ability to accumulate information over the entire image to determine the presence of a line structure, identifying endpoints of the line segment in the image remains a challenge [1]. Recently, machine learning based approaches have been shown to produce the state-of-the-art results in generating pixel-wise edge maps [8, 48, 28]. But these methods do not attempt to extract straight line segments from the image.

**Junction detection.** Detecting and analyzing junctions in real-world images remains a challenging problem due to a large number of fragmented, spurious, and missing line segments. In the literature, there are typically two ways to tackle this problem. The first group of methods focuses on operators based on local image cues, such as the Harris corner detector [18]. However, local junction detection is known to be difficult, even for humans [31]. More recent methods detect junctions by first locating contours (in natural images) [25] or straight line segments (in man-made environments) [21, 35, 10, 47] and then grouping them to

---

[1]In architecture design, a wireframe is often referred to a line drawing of a building or a scene on paper. Interpretation of such line drawings of 3D objects has a long history in computer vision dated back to the '70s and '80s [19, 3, 42, 26].

[2]For simplicity, this work is limited to wireframes consisting of straight lines. But the idea and method obviously apply to wireframes with curves.

Figure 2. Example images of our wireframe dataset, which covers a wide range of man-made scenes with different viewpoints, lighting conditions, and styles. For each image, we show the manually labelled line segments (first row) and the ground truth junctions derived from the line segments (second row).

form junctions. As we discussed before, such bottom-up methods are (i) sensitive to scene complexity, and (ii) vulnerable to imperfect line segment detection results.

**Line- and junction-based geometric reasoning.** Knowledge about junctions and the associated line structures is known to benefit many real-world 3D vision tasks. From a single image, a series of recent work use these features to recover the 3D layout of the scene [21, 35, 34, 51]. Meanwhile, observing that the junctions impose incident constraints on the adjacent line segments, [20] devises a method for 3D reconstruction of lines without explicitly matching them across views, whereas [46] proposes a surface scaffold structure that consists of sets of connected edges to regularize stereo-based methods for building reconstruction. Furthermore, [10] uses line segments and junctions to develop a robust and efficient method for two-view pose estimation, and [49] systematically studies how knowledge about junctions can affect the complexity and number of solutions to the Perspective-n-Line (PnL) problem.

**Machine learning and geometry.** There is a large body of work on machine learning based approach to inferring pixel-level geometric properties of the scene, such as the depth [40, 9], and the surface normal [12, 13]. But few work has been done on detecting mid/high-level geometric primitives with supervised training data. Recently, [17] proposes a method to recognize planes in a single image, [16] uses SVM to classify indoor planes (e.g., walls and floors), and [27, 39, 5] train fully convolutional networks (FCNs) to predict "informative" edges formed by the pairwise intersections of room faces. However, none of the work aims to detect highly compressive vectorized junctions or line segments in the image, let alone a complete wireframe.

## 2. A New Dataset for Wireframe Detection

As part of our learning-based framework to wireframe detection, we have collected 5,462 images of man-made environments. Some examples are shown in Fig. 2. The scenes include both indoor environments such as bedroom, living room, and kitchen, and outdoor scenes, such as house and yard. For each image, we manually labelled all the line segments associated with the scene structures. Here, our focus is on the *structural elements* in the image, that is, elements (i.e., line segments) from which meaningful geometric information of the scene can be extracted. As a result, we do not label line segments that are associated with texture (e.g., curtains, tree leaves), irregular or curved objects (e.g., sofa, humans, plants), shadows etc.

With the labelled line segments, ground truth junction locations and their branches can be easily obtained from the intersection or incidence relationships among two or more line segments in the image (Fig. 2, second row). Note that, unlike previous works [35, 34], we do not restrict ourselves to *Manhattan junctions*, which are formed by line segments aligned with one of three principal and mutually orthogonal directions in the scene. In fact, many scenes in our dataset do not satisfy the Manhattan world assumption [4]. For example, the scene depicted in the last column of Fig. 2 has more than two horizontal directions.

In summary, our annotation in each image includes a set of *junction points* $P = \{p_n\}_{n=1}^N$ and a set of *line segments* $L = \{l_m\}_{m=1}^M$. Each junction $p$ is the intersection of several, say $R$, line segments, called its branches. The coordinates of $p$ are denoted as $x \in \mathbb{R}^2$ and its line branches are recorded by their angles $\{\theta_r\}_{r=1}^R$. The number $R$ is known as the order of the junction, and the typical "$L$", "$Y$", and "$X$"-type junctions have orders $R = 2, 3$, and $4$, respectively. Each line segment is represented by its two end points: $l = (p_1, p_2)$. Hence, the *wireframe*, denoted as $W$, records all incidence and intersection relationships between junctions in $P$ and lines in $L$. It can be represented by an $N \times M$ matrix $W$ whose $nm$-th entry is 1 if $p_n$ is on $l_m$, and 0 otherwise. Notice that two line segments are intersected at some junction if and only if the corresponding entry in $W^T W$ is nonzero; and similarly $W W^T$ for connected junctions.

## 3. Wireframe Detection Method

Recently, deep convolutional neural networks (CNNs) such as [41, 38, 37] have shown impressive performance in object detection tasks. Utilizing the dataset we have, we here design new, end-to-end trainable CNNs for detecting junctions and lines, respectively, and then merge them into a complete wireframe. Fig. 3 shows the overall architecture of our proposed networks and method. Note that we choose different network architectures for junctions and lines due to the nature of their geometric properties, which we will elaborate below.
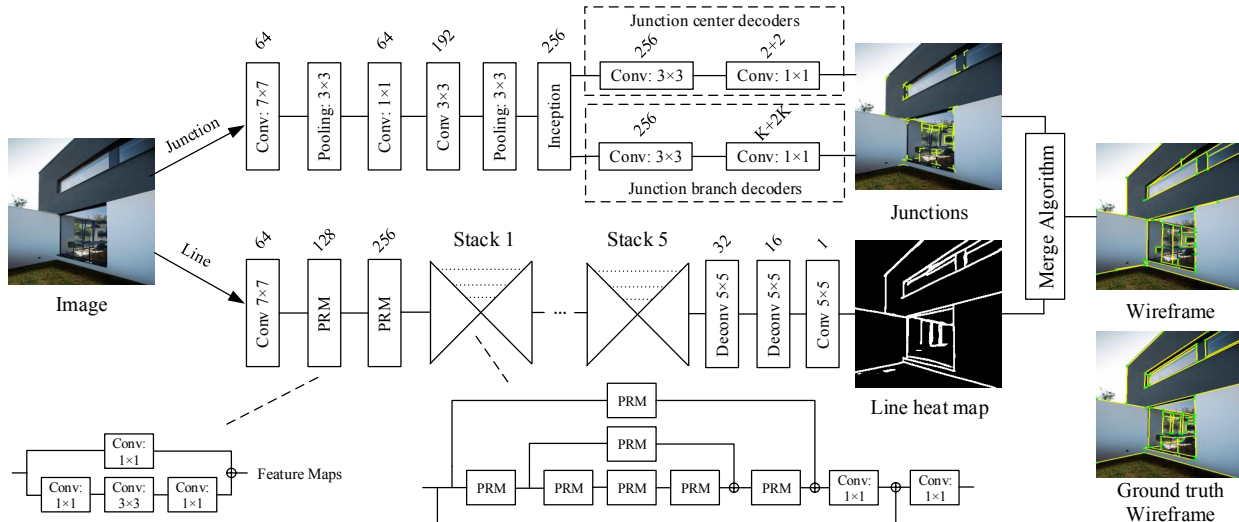
Figure 3. Architecture of the overall system. **Top:** junction detection network. **Bottom:** line detection network.
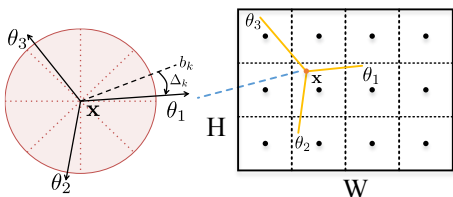


Figure 4. Representation of a junction with three branches.

## 3.1. Junction Detection

### 3.1.1 Design Rationale

Our design of the network architecture is guided by several important observations about junctions.

**Fully convolutional network for global detection.** As we mentioned before, *local* junction detection is a difficult task, which often leads to spurious detections. Therefore, it is important to enable the network to reason globally when making predictions. This motivates us to choose a *fully convolutional network* (FCN), following its recent success in object detection [37]. Unlike other popular object detection techniques that are based on sliding windows [41] or region proposals [38], FCN sees the entire image so it implicitly captures the contextual information about the junctions. Similar to [36, 37], our network divides the input image into an $H \times W$ mesh grid, see Fig. 4 right. If the center of a junction falls into a grid cell, that cell is responsible for detecting it. Thus, each $ij$-th cell predicts a confidence score $c_{ij}$ reflecting how confident the model thinks there exists a junction in that cell. To further locate the junction, each $ij$-th cell also predicts its relative displacement $\boldsymbol{x}_{ij}$ w.r.t. the center of the cell. Note that the behavior of the grid cells resembles the so-called "anchors", which serve as regression references in the latest object detection pipelines [38, 36, 22].

**Multi-bin representation for junction branches.** Unlike traditional object detection tasks, each cell in our network needs to make different numbers of predictions due to the varying number of branches in a junction. To address this issue, we borrow the idea of spatial grid and propose a new multi-bin representation for the branches, as shown in Fig. 4 left. We divide the circle (i.e., from 0 to 360 degrees) into $K$ equal bins, with each bin spanning $\frac{360}{K}$ degrees. Let the center of the $k$-th bin be $b_k$, we then represent an angle $\theta$ as $(k, \Delta_k)$, if $\theta$ fall into the $k$-th bin, where $\Delta_k$ is the angle residual from the center $b_k$ in the clockwise direction. Thus, for each bin we regress to this local orientation $\Delta_k$.

As a result, our network architecture consists of an encoder and two sets of decoders. The encoder takes the whole image as input and produces an $H \times W$ grid of high-level descriptors via a convolutional network. The decoders then use the feature descriptors to make junction predictions. Each junction is represented by $\boldsymbol{p}_{ij} = \left( \boldsymbol{x}_{ij}, c_{ij}, \{\theta_{ijk}, c_{ijk}^{\theta}\}_{k=1}^{K} \right)$, where $\boldsymbol{x}_{ij}$ is the coordinates of the junction center, $c_{ij} \in [0,1]$ is the confidence score for the presence of a junction in the $ij$-th grid cell, $\theta_{ijk}$ is the angle for the branch in the $k$-th bin, and $c_{ijk}$ is the confidence score for the bin. The two sets of decoders predict the junction center and the branches respectively. Each FCN decoder is simply a convolutional layer followed by a regressor, as shown in Fig. 3 top.

Unlike local junctions, the junctions we aim to detect each is formed by the intersection of two or more long line segments (the branches). While the junction detection does not explicitly rely on edge/line detection as an intermediate step, the knowledge about the associated edges is indirectly learned by enforcing the network to make correct detection of the branches and their directions.

### 3.1.2 Loss Function

To guide the learning process towards the desired output, our loss function consists of four modules. Given a set of ground truth junctions $\boldsymbol{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}$ in an image, we

write the loss function as follows:

$$L = \lambda_{conf}^c L_{conf}^c + \lambda_{loc}^c L_{loc}^c + \lambda_{conf}^b L_{conf}^b + \lambda_{loc}^b L_{loc}^b. \quad (1)$$

In the following, we explain each term in more detail.

**Junction center confidence loss $L_{conf}^c$.** The junction center confidence decoder predicts a score $\hat{c}_{ij}$, indicating the probability of a junction for each grid cell. Let $c_{ij}$ be the ground truth binary class label, we use the cross-entropy loss:

$$L_{conf}^c = -\frac{1}{H \times W} \sum_{i,j} E(\hat{c}_{ij}, c_{ij}). \quad (2)$$

**Junction center location loss $L_{loc}^c$.** The junction center location decoder predicts the relative position $\hat{x}_{ij}$ of a junction for each grid cell. We compare the prediction with each ground truth junction using the $\ell_2$ loss:

$$L_{loc}^c = -\frac{1}{N} \sum_{n=1}^{N} \|\hat{x}_{f(n)} - x_{f(n)}\|_2^2, \quad (3)$$

where $f(n)$ returns the index of the grid cell that the $n$-th ground truth junction falls into, and $x_{f(n)}$ is the relative position of the ground truth junction w.r.t. that cell center.

**Junction branch confidence loss $L_{conf}^b$.** The junction branch confidence decoder predicts a score $\hat{c}_{ijk}^\theta$ for each bin in each grid cell, indicating the probability of a junction branch in that bin. Similar to the junction center confidence loss above, we use the cross-entropy loss to compare the predictions with the ground truth labels. The only difference is that we only consider those grid cells in which a ground truth junction is present:

$$L_{conf}^b = -\frac{1}{N \times K} \sum_{n=1}^{N} \sum_{k=1}^{K} E(\hat{c}_{f(n),k}^\theta, c_{f(n),k}^\theta). \quad (4)$$

**Junction branch location loss $L_{loc}^b$.** Similar to the junction center location loss, we first decide, for each ground truth junction, the indices of the bins that its branches fall into, denoted as $g(r), r = 1, \ldots, R_n$, where $R_n$ is the order of $p_n$. Then, we compare our predictions with the ground truth using the $\ell_2$ loss:

$$L_{loc}^b = -\frac{1}{N} \sum_{n=1}^{N} \left( \frac{1}{R_n} \sum_{r=1}^{R_n} \|\hat{\theta}_{f(n),g(r)} - \theta_{f(n),g(r)}\|_2^2 \right). \quad (5)$$

**Implementation details.** We construct our model to encode an image into $60 \times 60$ grid of 256-dimensional features. Each cell in the grid is responsible for predicting if a junction is present in the corresponding image region. Our encoder is based on Google's Inception-v2 model [43], which extracts multi-scale features and is well-suited for our problem. For our problem, we only use the early layers in the Inception network, i.e., the first layer to "Mixed_3b". Each decoder consists of a $3 \times 3 \times 256$ convolutional layer, followed by a ReLU layer and a regressor. Note that the regressor is conveniently implemented as $1 \times 1 \times d$ convolutional layer, where $d$ is the dimension of the output.

The default values for the weights in Eq. (1) are set to the following: $\lambda_{conf}^c = \lambda_{conf}^b = 1, \lambda_{loc}^c = \lambda_{loc}^b = 0.1$.

We choose the number of bins $K = 15$. Our network is trained from scratch with the Stochastic Gradient Descent (SGD) method. The momentum parameter is set to 0.9, and the batch size is set to 1. We follow the standard practice in training deep neural networks to augment the data with image domain operations including mirroring, flipping upside-down, and cropping. The initial learning rate is set to 0.01. We decrease it by a multiple of 0.1 after every 100,000 iterations. Convergence is reached at 300,000 iterations.

### 3.2. Line Detection

Next we design and train a convolutional neural network (Fig. 3 bottom) to infer line information from RGB images. The network predicts for each pixel $p$ whether it falls on a (long) line $l$. To suppress local edges, short lines, and curves, the predicted value $h(p)$ (of the heat map) at pixel $p$ is set to be the length of the line it belongs to. Given an image with ground truth lines $L$, the target value for $h(p)$ is defined to be:

$$h(p) = \begin{cases} d(l) & p \text{ is on a line } l \text{ in } L, \\ 0 & p \text{ is not on any line in } L, \end{cases} \quad (6)$$

where $d(l)$ is the length of the line $l$. Let $\hat{h}(p)$ be the estimated heatmap value, then the loss function we try to minimize the $\ell_2$ loss:

$$L = \sum_{i,j} \|\hat{h}(p_{ij}) - h(p_{ij})\|_2^2. \quad (7)$$

where the sum is over all pixels of the image.

**Implementation details.** The network architecture is inspired by the Stacked Hourglass network [32]. It takes a $320 \times 320 \times 3$ RGB image as input, extracts a $80 \times 80 \times 256$ feature maps via three Pyramid Residual Modules (PRM), see Fig. 3 bottom. The feature maps then go through five stacked hourglass modules, followed by two fully convolutional and ReLU layers ($5 \times 5 \times 32$ and $5 \times 5 \times 16$) and a $5 \times 5 \times 1$ convolutional layer to output a $320 \times 320 \times 1$ pixel-wise heat map. The detailed pyramid residual module and stacked hourglass module can be found in [32].

During training, we adopt the Stochastic Gradient Descent (SGD) method. The momentum parameter is set to 0.9, and the batch size is set to 4. Again, we augment the data with image domain operations including mirroring and flipping upside-down. The initial learning rate is set to 0.001. We decrease it by a multiple of 0.1 after 100 epochs. Convergence is reached at 120 epochs.

Notice that we have used an Inception network for junction detection whereas an hourglass network for line detection. In junction detection, we are not interested in the entire support of the line, hence the receptive field of an Inception network is adequate for such tasks. However, we find that for accurately detecting lines with large spatial support, the Stacked Hourglass network works much better due to its large (effective) receptive field. In addition, our experiment

also shows that above length-dependent $\ell_2$ loss is more effective than the cross-entropy cost often used in learning-based edge detection.

### 3.3. Combine Junctions and Lines for Wireframe

The final step of the system is to combine the results from junction detection and line detection to generate a wireframe $W$ for the image, which, as mentioned before, consists of a set of junction points $P$ connected by a set of line segments $L$.

Specifically, given a set of detected junctions $\{p_i\}_{i=1}^{N}$ and a line heat map $h$, we first apply a threshold $w$ to convert $h$ into a binary map $\mathcal{M}$. Then, we construct the wireframe based on the following rules and procedure:

1. The set $P$ is initialized with the output from the junction detector. A pair of detected junctions $p$ and $q \in P$ are connected by a line segment $l = (p, q)$ if they are on (or close to be on) each other's branches, and we add this segment $l$ to $L$. If there are multiple detected junctions on the same branch of a junction point $p$, we only keep the shortest segment to avoid overlap.[3]

2. For any branch of a junction $p$ that is not connected to any other junction, we attempt to recover additional line segment using $\mathcal{M}$. We first find the farthest line pixel $q_{\mathcal{M}}$ (pixel $p$ is a line pixel if $\mathcal{M}(p) = 1$) that is also on the ray starting at $p$ along the branch. Then, we find all the intersection points $\{q_1, \ldots, q_S\}$ of line segment $(p, q_{\mathcal{M}})$ with existing segments in $L$. Let $q_0 = p_i$ and $q_{S+1} = q_{\mathcal{M}}$, we calculate the line support ratio $\kappa(q_{s-1}, q_s), s = \{1, \ldots, S, S+1\}$, for each segment. Here, $\kappa$ is defined as the ratio of the number of line pixels to the total length of the segment. If $\kappa$ is above a threshold, say $0.6$, we add the segment to $L$ and its endpoints to $P$.

Notice that both the sets $P$ and $L$ may have *two* sources of candidates. For the junction set $P$, besides those directly detected by the junction detection, the line segments could also produce new intersections or endpoints that were missed by the junction detection. For the line segment set $L$, it could come from branches of the detected junctions and the line detection.

We leave more detailed description of the algorithm to the supplementary material. Of course, there could be more advanced ways to merge the detected junctions and line heat map which we will explore in future work. Nevertheless, from our experiments (see next section), we find that the results from junction detection and line detection are rather complementary to each other and the above simple procedure already produces rather decent results.

---

[3]Hence we are less interested in detecting a straight line with the longest possible support, instead, we are interested in its incidence relationship with other lines and junctions.

## 4. Experiments

In this section, we conduct extensive experiments to evaluate the quality of junctions and final wireframes generated by our method, and compare it to the state-of-the-art. All experiments are conducted on one NVIDIA Titan X GPU device. In testing phase, our method runs at about two frames per second, thus our method is potentially suitable for applications which require real-time processing.

### 4.1. Datasets and Evaluation Metrics

For performance evaluation, we split our wireframe dataset into a training set and a testing set. Among the 5,462 images in the dataset, 5,000 images are randomly selected for training and validation, and the remaining 462 images are used for testing. For junction detection (Section 4.2), we compare the junctions detected by any method with the ground truth junctions (Fig. 2, second row). For wireframe construction (Section 4.3), we compare the line segments detected by any method with the ground truth line segments labeled by human subjects (Fig. 2, first row).

For both junction detection and wireframe construction experiments, all methods are the evaluated quantitatively by means of the *recall* and *precision* as described in [29, 25, 47]. In the context of junction detection, recall is the fraction of true junctions that are detected, whereas precision is the fraction of junction detections that are indeed true positives. In the context of wireframe construction, recall is the fraction of line segment pixels that are detected, whereas precision is the fraction of line segment pixels that are indeed true positives.

Specifically, let $G$ denote the set of ground truth junctions (or line segment pixels), and $Q$ denote the set of junctions (or line segment pixels) detected by any method, the precision and recall are defined as follows:

$$\text{Precision} \doteq |G \cap Q|/|Q|, \quad \text{Recall} \doteq |G \cap Q|/|G|. \quad (8)$$

Note that, following the protocols of previous work [29, 25, 47], the particular measures of recall and precision allow for some small tolerance in the localization of the junctions (or line segment pixels). In this paper, we set the tolerance to be 0.01 of the image diagonal.

### 4.2. Junction Detection Comparison

We compare our junction detection method with two recent methods, namely Manhattan junction detection (MJ) [35] and *a contrario* junction detection (ACJ) [47]. **MJ [35]**: This method detects Manhattan junctions formed by line segments in three principal orthogonal directions using a simple voting-based scheme. As the authors did not release their code, we use our own implementation of the method. Line segments are first detected using LSD [45], and then clustered using J-Linkage [44] to obtain the vanishing points. Note that this method only ap-
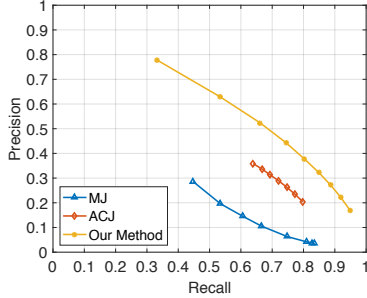
Figure 5. The precision-recall curves of different junction detection methods on our test dataset.
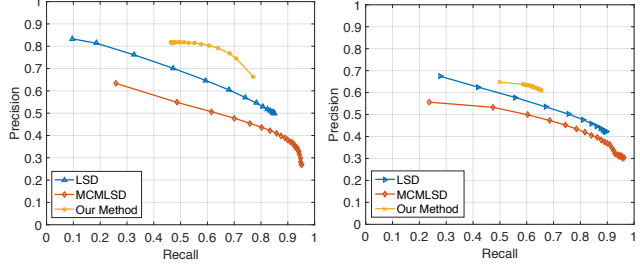


Figure 6. The precision-recall curves of different line segment detection methods. **Left:** on our test dataset. **Right:** on the York Urban dataset [7].

plies to scenes that satisfy the Manhattan world assumption. For fair comparison, we only keep the images in which three principal vanishing points are detected. An important parameter in our implementation is the maximum distance $d_{max}$ between a line segment and a point $p$ for that line segment to vote for $p$. We vary the value $d_{max} \in \{10, 20, 30, 50, 100, 200, 300, 500\}$ pixels.

**ACJ [47]**: This method relies on statistical modeling of image gradients and an *a contrario* approach to detect junctions. Specifically, meaningful junctions are detected as those which are very unlikely under a null hypothesis $\mathcal{H}_0$, which is defined based on the distribution of gradients of arbitrary natural images. In the method, each candidate junction is associated with a strength value depending on the image gradients around it. Then, the candidate junction is validated with a threshold, which is derived by controlling the number of false detections, $\epsilon$, in an image following $\mathcal{H}_0$. For the experiments, we use the implementation provided by the authors of [47] and vary the value $\epsilon \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\}$.

**Performance comparison.** Fig. 5 shows the precision-recall curves of all methods on our new dataset. For our method, we vary the junction confidence threshold $\tau$ from 0.1 to 0.9. As one can see, our method outperforms the other methods by a large margin. Fig. 7 compares qualitatively the results of all methods on our test data. Compared to the other two methods, MJ tends to miss important junctions due to the imperfect line segment detection results. Moreover, since MJ relies on local image features, it noticeably produces quite a few repetitive detections around some junctions. By directly modeling the image gradients, ACJ is able to find most junctions on the scene structures. However, as a local method, ACJ makes a lot of false predictions on textured regions (e.g., floor of the first, sky of the fourth image). In contrast, our method is able to detect most junctions intersected by salient lines, while minimizing the number of false detections. This is no surprise because our supervised framework implicitly encodes high-level structural and semantic information of the scene as it learns from the labeled data provided by humans.

## 4.3. Line Segment Detection Comparison

In this section, we compare the wireframe results of our method with two state-of-the-art line segment detection methods, namely the Line Segment Detector (LSD) [45] and the Markov Chain Marginal Line Segment Detector (MCMLSD) [1]. We test and compare with these methods on both our new dataset and the York Urban dataset [7] used in the work of MCMLSD [1].

**LSD [45]**: This method is a linear-time line segment detector that requires no parameter tuning. It also uses an *a contrario* approach to control the number of false detections. In this experiment, we use the code released by the authors[4] and vary the threshold for $-\log(\text{NFA})$ (NFA is the number of false alarms) in $0.01 \times \{1.75^0, 1.75^1, 1.75^2, ..., 1.75^{19}\}$.

**MCMLSD [1]**: This method proposes a two-stage algorithm to find line segments. In the first stage, it uses the probabilistic Hough transform to identify globally optimal lines. In the second stage, it searches each of these lines for their supports (segments) in the image, which can be modeled as labeling hidden states in a linear Markov chain. In this experiment, we use the code released by the authors.[5] Be aware that authors of [1] have introduced a different metric than ours that tends to penalize over-segmentation. Hence our metric can be unfair to their method. Nevertheless, our metric is more appropriate for wireframe detection as we prefer to interpret a long line as several segments between junctions if it intersects with other lines.

**Performance comparison.** Fig. 6 shows the precision-recall curves of all methods on our dataset and the York Urban dataset, respectively. As one can see, our method outperforms the other methods by a significant margin on our dataset. The margin on the York Urban dataset is decent but not so large. According to [1], the labeling of the York Urban dataset is not as complete for salient line segments, hence it is not entirely suitable for the wireframe detection task here. Fig. 8 compares qualitatively the results of all methods on our test data. Since the other two methods rely on local measurements, they tend to produce many line seg-

---

[4]http://www.ipol.im/pub/art/2012/gjmr-lsd/
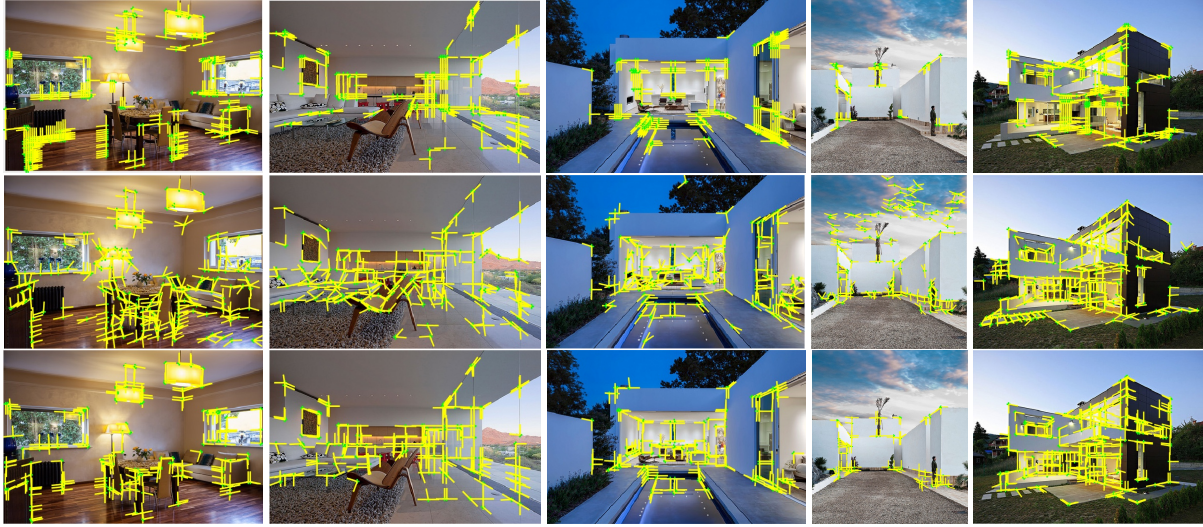[5]http://www.elderlab.yorku.ca/resources/

Figure 7. Junction detection results. **First row:** MJ ($d_{max} = 20$). **Second row:** ACJ ($\epsilon = 1$). **Third row:** Our method ($\tau = 0.5$).
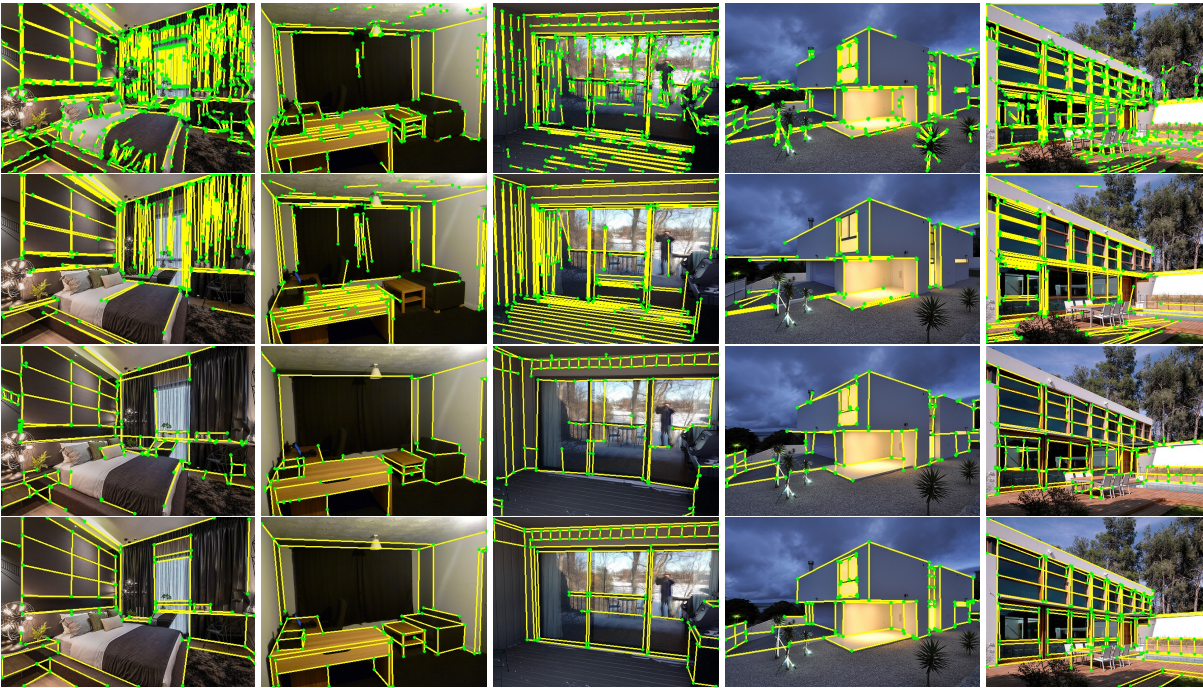


Figure 8. Line/wireframe detection results. **First row:** LSD (-log(NFA) $> 0.01 \times 1.75^8$). **Second row:** MCMLSD (confidence top 100). **Third row:** Our method (line heat map $h(p) > 10$). **Fourth row:** Ground truth.

ments on textured regions (e.g. curtain of the first image) which do not correspond to scene structures.

## 5. Conclusion

This paper has demonstrated the feasibility of parsing wireframes in images of man-made environments. The proposed method is based on combining junctions and lines detected from respective neural networks trained on a new large-scale dataset. Both quantitatively and qualitatively, the results of our method approximately emulate those labelled by humans. The junctions and line segments in a wireframe and their incidence relationships encode rich and accurate large-scale geometry of the scene and shape of regular objects therein, in a highly compressive and efficient manner. Hence results of this work can significantly facilitate and benefit visual tasks such as feature correspondence, 3D reconstruction, vision-based mapping, localization, and navigation in man-made environments.

# References

[1] E. J. Almazan, R. Tal, Y. Qian, and J. H. Elder. Mcmlsd: A dynamic programming approach to line segment detection. In *CVPR*, July 2017. 2, 7

[2] M. Brown, D. Windridge, and J. Guillemaut. A generalisable framework for saliency-based line segment detection. *Pattern Recognition*, 48(12):3993–4011, 2015. 2

[3] M. B. Clowes. On seeing things. *Artif. Intell.*, 2(1):79–116, 1971. 2

[4] J. M. Coughlan and A. L. Yuille. Manhattan world: Orientation and outlier detection by bayesian inference. *Neural Computation*, 15(5):1063–1088, 2003. 3

[5] S. Dasgupta, K. Fang, K. Chen, and S. Savarese. Delay: Robust spatial layout estimation for cluttered indoor scenes. In *CVPR*, pages 616–624, 2016. 3

[6] E. Delage, H. Lee, and A. Y. Ng. Automatic single-image 3D reconstructions of indoor manhattan world scenes. In *International Symposium on Robotics Research*, pages 305–321, 2005. 1

[7] P. Denis, J. H. Elder, and F. J. Estrada. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *ECCV*, pages 197–210, 2008. 7

[8] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, pages 1841–1848, 2013. 2

[9] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, pages 2366–2374, 2014. 3

[10] A. Elqursh and A. M. Elgammal. Line-based relative pose estimation. In *CVPR*, pages 3049–3056, 2011. 2, 3

[11] A. Flint, D. W. Murray, and I. Reid. Manhattan scene understanding using monocular, stereo, and 3D features. In *ICCV*, pages 2228–2235, 2011. 1, 2

[12] D. F. Fouhey, A. Gupta, and M. Hebert. Data-driven 3d primitives for single image understanding. In *ICCV*, pages 3392–3399, 2013. 3

[13] D. F. Fouhey, A. Gupta, and M. Hebert. Unfolding an indoor origami world. In *ECCV*, pages 687–702, 2014. 3

[14] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Manhattan-world stereo. In *CVPR*, pages 1422–1429, 2009. 1

[15] Y. Furukawa and Y. Shinagawa. Accurate and robust line segment extraction by analyzing distribution around peaks in hough space. *CVIU*, 92(1):1–25, 2003. 2

[16] R. Guo, C. Zou, and D. Hoiem. Predicting complete 3d models of indoor scenes. *CoRR*, abs/1504.02437, 2015. 3

[17] O. Haines and A. Calway. Recognising planes in a single image. *IEEE TPAMI*, 37(9):1849–1861, 2015. 3

[18] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 1–6, 1988. 2

[19] D. Huffman. Impossible objects as nonsense sentences. *Machine Intelligence*, 6, 1971. 2

[20] A. Jain, C. Kurz, T. Thormählen, and H. Seidel. Exploiting global connectivity constraints for reconstruction of 3d line segments from images. In *CVPR*, pages 1586–1593, 2010. 3

[21] D. C. Lee, M. Hebert, and T. Kanade. Geometric reasoning for single image structure recovery. In *CVPR*, pages 2136–2143, 2009. 1, 2, 3

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 4

[23] X. Liu, Z. Cao, N. Gu, S. Nahavandi, C. Zhou, and M. Tan. Intelligent line segment perception with cortex-like mechanisms. *IEEE Trans. Systems, Man, and Cybernetics*, 45(12):1522–1534, 2015. 2

[24] X. Lu, J. Yao, K. Li, and L. Li. Cannylines: A parameter-free line segment detector. In *ICIP*, pages 507–511, 2015. 2

[25] M. Maire, P. Arbelaez, C. C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. In *CVPR*, 2008. 2, 6

[26] J. Malik. Interpreting line drawings of curved objects. *IJCV*, 1(1):73–103, 1987. 2

[27] A. Mallya and S. Lazebnik. Learning informative edge maps for indoor scene layout prediction. In *ICCV*, pages 936–944, 2015. 3

[28] K. Maninis, J. Pont-Tuset, P. A. Arbeláez, and L. J. V. Gool. Convolutional oriented boundaries. In *ECCV*, pages 580–596, 2016. 2

[29] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE TPAMI*, 26(5):530–549, 2004. 6

[30] J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *CVIU*, 78(1):119–137, 2000. 2

[31] J. McDermott. Psychophysics with junctions in real images. *Perception*, 33:1101–1127, 2004. 2

[32] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, pages 483–499, 2016. 5

[33] M. Nieto, C. Cuevas, L. Salgado, and N. N. García. Line segment detection using weighted mean shift procedures on a 2d slice sampling strategy. *Pattern Anal. Appl.*, 14(2):149–163, 2011. 2

[34] S. Ramalingam and M. Brand. Lifting 3D manhattan lines from a single image. In *ICCV*, pages 497–504, 2013. 2, 3

[35] S. Ramalingam, J. K. Pillai, A. Jain, and Y. Taguchi. Manhattan junction catalogue for spatial reasoning of indoor scenes. In *CVPR*, pages 3065–3072, 2013. 1, 2, 3, 6

[36] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788, 2016. 4

[37] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. 3, 4

[38] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015. 3, 4

[39] Y. Ren, C. Chen, S. Li, and C. J. Kuo. A coarse-to-fine indoor layout estimation (CFILE) method. *CoRR*, abs/1607.00598, 2016. 3

[40] A. Saxena, S. H. Chung, and A. Y. Ng. 3-D depth reconstruction from a single still image. *IJCV*, 76(1):53–69, 2008. 3

[41] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. 3, 4

[42] K. Sugihara. Mathematical structures of line drawings of polyhedrons-toward man-machine communication by means of line drawings. *IEEE TPAMI*, 4(5):458–469, 1982. 2

[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016. 5

[44] J.-P. Tardif. Non-iterative approach for fast and accurate vanishing point detection. In *ICCV*, pages 1250–1257, 2009. 6

[45] R. G. von Gioi, J. Jakubowicz, J. Morel, and G. Randall. LSD: A fast line segment detector with a false detection control. *IEEE TPAMI*, 32(4):722–732, 2010. 2, 6, 7

[46] J. Wang, T. Fang, Q. Su, S. Zhu, J. Liu, S. Cai, C. Tai, and L. Quan. Image-based building regularization using structural linear features. *IEEE TVCG*, 22(6):1760–1772, 2016. 3

[47] G. Xia, J. Delon, and Y. Gousseau. Accurate junction detection and characterization in natural images. *IJCV*, 106(1):31–56, 2014. 2, 6, 7

[48] S. Xie and Z. Tu. Holistically-nested edge detection. In *ICCV*, pages 1395–1403, 2015. 2

[49] C. Xu, L. Zhang, L. Cheng, and R. Koch. Pose estimation from line correspondences: A complete analysis and a series of solutions. *IEEE TPAMI*, 39(6):1209–1222, 2017. 3

[50] Z. Xu, B. Shin, and R. Klette. Accurate and robust line segment extraction using minimum entropy with hough transform. *IEEE TIP*, 24(3):813–822, 2015. 2

[51] H. Yang and H. Zhang. Efficient 3D room shape recovery from a single panorama. In *CVPR*, 2016. 2, 3