LGNN: A Context-aware Line Segment Detector

Quan Meng
mengquan@shanghaitech.edu.cn
ShanghaiTech University
Shanghai, China

Jiakai Zhang
zhangjk@shanghaitech.edu.cn
ShanghaiTech University
Shanghai, China

Qiang Hu
huqiang@shanghaitech.edu.cn
ShanghaiTech University
Shanghai, China

Xuming He
hexm@shanghaitech.edu.cn
Shanghai Engineering Research Center of Intelligent Vision and Imaging, School of Information Science and Technology, ShanghaiTech University
Shanghai, China

Jingyi Yu
yujingyi@shanghaitech.edu.cn
Shanghai Engineering Research Center of Intelligent Vision and Imaging, School of Information Science and Technology, ShanghaiTech University
Shanghai, China

ABSTRACT

We present a novel real-time line segment detection scheme called Line Graph Neural Network (LGNN). Existing approaches require a computationally expensive verification or postprocessing step. Our LGNN employs a deep convolutional neural network (DCNN) for proposing line segment directly, with a graph neural network (GNN) module for reasoning their connectivities. Specifically, LGNN exploits a new quadruplet representation for each line segment where the GNN module takes the predicted candidates as vertexes and constructs a sparse graph to enforce structural context. Compared with the state-of-the-art, LGNN achieves near real-time performance without compromising accuracy. LGNN further enables time-sensitive 3D applications. When a 3D point cloud is accessible, we present a multi-modal line segment classification technique for extracting a 3D wireframe of the environment robustly and efficiently.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; **Object detection**; *Reconstruction.*

KEYWORDS

line segment detection; quadruplet; graph neural network; real-time

ACM Reference Format:
Quan Meng, Jiakai Zhang, Qiang Hu, Xuming He, and Jingyi Yu. 2020. LGNN: A Context-aware Line Segment Detector. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20), October 12–16, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394171.3413784

MM '20, October 12–16, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-7988-5/20/10…$15.00
https://doi.org/10.1145/3394171.3413784

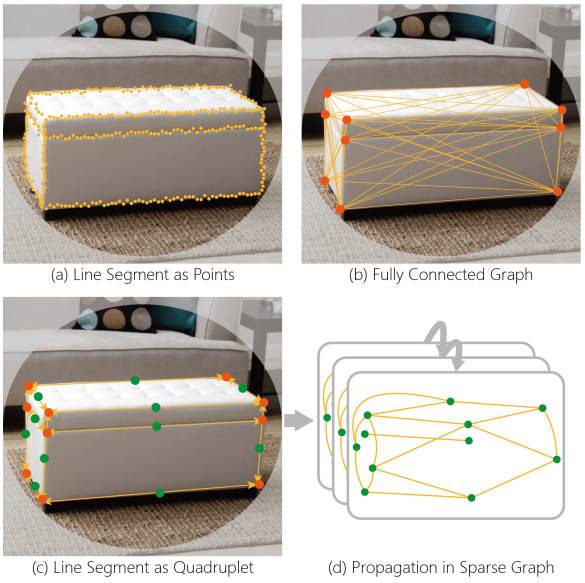

Figure 1: Different representations of 2D wireframe: (a) Grouping pixels to construct line segments [17, 41], which use sophisticated postprocessing and tend to produce short and overlapped line segments. (b) Modeling wireframe as a fully connected graph on line segment candidates [47, 49], which are time- and memory-consuming and prone to ineffective inference due to the noisy proposals. (c & d) Our proposed method of representing line segments as quadruplets: (start junction, end junction, line central point, line shift vector), which enables us to construct a sparse graph on line segment proposals and learn high-level semantic and geometric features during message-passing inference.

1 INTRODUCTION

Line segments provide rich information about a scene: creases are indications of foldings of pliable surfaces, occlusion boundary edges encode shape information, while textures manifest the appearance of regions. More importantly, they provide a more precise, compact, and structural representation of a 3D scene. The detected line

segments further benefit numerous computer vision tasks, ranging from stereo matching [45] and 3D reconstruction [7, 9, 16, 32, 46] to image stitching [38] and segmentation [2, 5]. Traditional techniques [1, 3, 11, 18, 37, 40] based on hand-crafted features are vulnerable to textureless regions, repetitive textures, illumination variations, occlusions, etc. More recent deep learning approaches [17, 41, 47, 49] attempt to explore semantic meanings of line segments to mitigate the problems.

Existing learning-based algorithms tackle the line detection problem via a predict-then-verify strategy. Pioneering approaches [17, 41] first adopt a deep convolution neural network (DCNN) to predict junctions as well as a line heat map or an attraction field map. They then apply sophisticated fusion algorithms for extracting line segments. Such approaches commonly produce crossing or fragmented line segments that are difficult to fix or even differentiate. More recent methods, including PPGNet [47] and L-CNN [49], first train a deep CNN to estimate a junction heatmap and then enumerate all junction pairs to verify their connectivities. The verification step greatly improves line detection quality but is time and memory consuming and scales poorly with the number of junctions in an image. For example, on a Tesla P40 GPU, verification over 512 junctions in PPGNet [47] requires about a second.

For many real-life line detection applications, it is critical to balance between speed and performance. In this paper, we propose a real-time line detector — Line Graph Neural Network (LGNN). LGNN can reliably handle a cluttered environment by exploiting a strong contextual structure between line segments. Specifically, LGNN employs two main modules: a DCNN module for generating line segment positions and features and a graph neural network for reasoning their connectivities. We propose a novel quadruplet representation - (start junction, end junction, line central point, line shift vector) - for each line segment, in place of the traditional junction-junction pairs. The DCNN sets out to predict a line central point heatmap along with a line shift vector map. We observe that, for cluttered scenes, the predicted line segments are less fragmented, where we can reliably map their endpoints to junctions. The GNN module then takes these line segment candidates as vertexes and construct a sparse graph to enforce structural constraints.

Our LGNN significantly accelerates the detection speed without compromising accuracy. We show that LGNN achieves near real-time performance. On the wireframe dataset [17], LGNN performs at 15.8 frames per second (FPS) with 62.3% structural AP (sAP) and a lightweight version achieves 34 FPS with 57.6% sAP. LGNN hence enables time-sensitive 3D applications: when a 3D point cloud is accessible and we can map the predicted 2D line segments onto 3D to determine their types - creases, occlusion edges or texture edges. We therefore further present a multi-modal edge classification technique for extracting a 3D wireframe of the environment robustly and efficiently.

2 RELATED WORKS

2D Line Segment Detection. Line segment detection has attracted a lot of research work. Classical approaches [1, 3, 11, 18, 37, 40] rely on low-level information, so are susceptible to external conditions. Recently, Wireframe [17] first adopts two independent networks to predict line and junction heatmaps parallelly, then combine junctions and lines to produce line segments. AFM [41] re-formulates it as a region coloring problem and leverage semantic segmentation networks to predict attraction field map, then group active pixels to construct line segments with region-growing algorithms similar to LSD [37]. Both of them need a sophisticated postprocessing method and tend to produce short line segments because they represent a line segment as a group of pixels. PPGNet [47] supplements the line segment dataset with outdoor scenes. L-CNN [49] proposes line sampling to overcome the data unbalance, and a more reasonable metric(sAP) to evaluate the structural quality of wireframes. Both PPGNet and L-CNN represent line segments with endpoints and enumerate all junction pairs, so they scale poorly with the time complexity of $O(n^2)$. In this work, by representing line segments as quadruplets, our method not only directly get accurate line segments but also run the fastest.

Objection Detection. Object detection approaches contain two types of pipelines, namely, region proposal based and regression based approaches. The former approaches like R-CNN [14], Fast R-CNN [13], Faster R-CNN [34], R-FCN [4], Mask R-CNN [15] and etc, generate region proposals at first and then classify each proposal into different object categories. The latter approaches like YOLO [33], SSD [26], CornerNet [21], CenterNet [8, 48] and etc, remove the RoI extraction process and directly classify and regress the candidate anchor boxes. While the performance of region proposal based approaches remain in a higher place, they need more computation and processing time. Our line segment detection approach inherits merits of both approaches, in which we first predict line central points and junctions and directly regress other properties, followed by a light-weight GNN refinement.

Graph Neural Network. Graph Neural Networks can effectively cope with non-Euclidean data, such as e-commerce [29, 44], citation network [20, 36], molecules [6, 12], scene relationship [42, 43], sketch recognition [39] and etc. Recently, Li et al. [22] build a very deep 56-layer Graph Convolutional Network(GCN) which significantly boosts performance in the task of point cloud semantic segmentation. Computer vision is one of the biggest application areas for graph neural networks. Yang et al. [42] propose a novel scene graph generation model called Graph R-CNN that reports state-of-the-art performance. Xu et al. [39] represent sketches as multiple sparsely connected graphs and designs the Multi-Graph Transformer that outperforms all RNN-based models. Lu et al. [28] first apply GNN in image segmentation and achieve about 1.34% improvement on the VOC dataset compared to the FCN model. We are first to formulate wireframe as a sparse graph and enable message passing. This view allows us to encode larger context information so that we can decide which line segment is globally significant.

3D Line Segment Detection. Existing 3D line segment detection methods extract 3D line segments from an unordered point cloud. These methods mainly include three categories: point based [24], plane based [35] and image based [25, 27]. Given a point cloud, point based method identifies the boundary points and fits for the 3D line segment. This method usually mistakes much noise for boundary points. Plane based method detects planes and intersects every two adjacent planes to calculate line segments. This method may fail to find the terminals of the intersection line. Image based

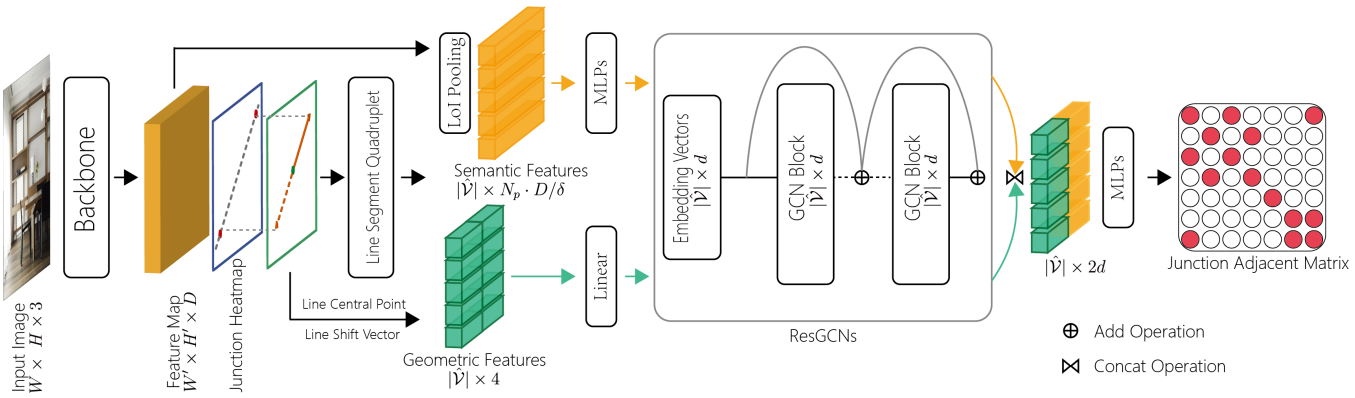

Figure 2: An overview of the proposed LGNN. Our model first uses a convolutional backbone network to produce a heatmap for line central point, a heatmap for junction, and a line shift vector map, which are assembled into a sparse graph of line segment quadruplets. We then introduce a deep ResGCNs to update the semantic and geometric features of line segments simultaneously. The resulting line representations are fed into an MLPs to score each line segment.

methods project point cloud into images and apply 2D line segment detector to extract line segments, finally re-project them to the point cloud. These methods only use hand-crafted features, so tend to produce short and crossed line segments, which are not good representations of the scene. In this work, we implement a structural 3D wireframe extraction algorithm based on LGNN. We also extract plane information to optimize the 3D line segment.

3 METHODS

3.1 Overview

In this section, we introduce our line segment detection framework, which aims to delineate salient 2D line boundaries in an image. Our goal is to achieve high efficiency by exploiting rich properties of line segments, which enable us to effectively reduce the search space in localization and simplify the model structure. To this end, we develop a novel line segment detection method that consists of two main modules: a Multitask Learner Module and a Relation Reasoning Module. Given an input RGB image, the first module (Multitask Learner) is a multi-head deep convolutional network that extracts several key properties of line segments, such as their endpoints and orientations. We then generate a set of line segment and junction proposals, and construct a *sparse graph* on line candidates with junctions as graph edges. The second module (Relation Reasoning) is a graph convolutional network that augments the line features with context cues, which are subsequently used for final line segment prediction. An overview of our framework is shown in Fig. 2.

The rest of this section first introduces our line representation in Section 3.2, followed by two model components with inference process in Section 3.3 and Section 3.4. Finally, we describe our training strategy and multi-task loss function in Section 3.5.

3.2 Line Segment Representation

Given an RGB image $I \in \mathbb{R}^{W \times H \times 3}$, we first denote the set of line endpoints or junctions as $\mathcal{J}$ and the set of line central/middle point as C. In order to generate line segment proposals efficiently, we represent a line segment as a quadruplet $v = (j^1, j^2, c, \mathbf{s})$, in which j^1 and $j^2 \in \mathcal{J}$ are the two endpoints of the line segment v, $c \in C$ is its central point, and $\mathbf{s} \in \mathbb{R}^2$ is a 2D shift vector indicating the line direction and segment length.

Concretely, we denote the 2D coordinates of the endpoints j^1, j^2 and the central point c as $\mathbf{p}_{j^1}, \mathbf{p}_{j^2}, \mathbf{p}_c \in \mathbb{R}^2$, respectively. The 2D shift vector $\mathbf{s}$ of the line central point c satisfies the following relations:

$$(\mathbf{p}_{j^1}, \mathbf{p}_{j^2}) = (\mathbf{p}_c - \mathbf{s}, \mathbf{p}_c + \mathbf{s}) \tag{1}$$

To avoid ambiguity of directions, we stipulate that $\mathbf{s}$ is always pointing to the endpoint closer to the right side of the image.

To capture relations between line segments, we further represent the entire set of line segments in an image as a graph $\mathcal{G} = (\mathcal{V}, \mathbf{A})$, where $\mathcal{V}$ stands for the set of the unordered line segments, $\mathbf{A}$ is the adjacent matrix representing the connectivity between these line segments. For two line segments, v_k and v_l, if they have a common endpoint, i.e., they are connected, then both $\mathbf{A}_{kl}$ and $\mathbf{A}_{lk}$ equal one; and otherwise, they are zero. Our goal is to develop a deep network to predict the graph $\mathcal{G}$ from the input image I, which will be described in detail below.

3.3 Mutlitask Learner Module

Our first module, the Multitask Learner, takes the image I as input and generates an initial estimation of line segment properties including their endpoints, central points and shift vectors. To achieve this, we develop a multi-head convolutional network with two components: a backbone ConvNet for feature extraction and a prediction module that outputs three properties of the line segments.

In this work, we adopt a stacked hourglass network [31] as our backbone. Using multiple bottom-up and top-down inference across scales, this backbone network produces a rich set of feature maps $\mathbf{F} \in \mathbb{R}^{W' \times H' \times D}$, where W', H' and D are the width, height and the number of channels of the feature maps.

The prediction module consists of three parallel convolutional heads, generating dense output maps for endpoints, central points and shift vectors, respectively. Each of the endpoint and central point heads produces an output map that indicates the confidence

scores of each location belonging to the junction set $\mathcal{J}$ or the line central point set C. We adopt a heatmap representation for those keypoints as in the human pose estimation. Besides, due to the discretization effect caused by the output stride, we also produce two offset maps to predict offsets for junctions and line central keypoints. The shift vector head simultaneously regresses the direction and (half) length of the line segments for each location.

We apply non-maximum suppression to remove duplicated keypoints and extract the local maximum in the line central point heatmap as candidates. To generate consistent line segment and endpoint/junction proposals, we introduce a simple nearest neighbor-based alignment procedure to match the predicted endpoints to the pairs of central point and shift vector and remove noisy center point candidates as they are less reliable.

Specifically, given a central point candidate $\hat{c}$ and its shift vector, $\hat{\mathbf{s}}$, we first generate a 2D line segment proposal $\hat{v}$ with endpoints $\hat{j}^1$ and $\hat{j}^2$ by shifting the line central point $\hat{c}$ in two opposite directions with the vector $\hat{\mathbf{s}}$ as in Eq. 1. We then match the endpoints of $\hat{v}$ to the predicted endpoint candidates. In particular, we find the closest endpoints to each generated line segment. If the total distance is below a threshold θ, we will replace the computed endpoints $\hat{j}^1$ and $\hat{j}^2$ by the matched endpoint candidates. If the distance exceeds θ, we will remove the line segment candidate. The isolated endpoint candidates will also be filtered out after matching. Finally, we build a candidate graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathbf{A}})$ using the remaining line segment and endpoint candidates, in which the vertex set $\hat{\mathcal{V}}$ comprises line segments and the adjacent matrix $\hat{\mathbf{A}}$ encodes the connectivity between lines due to shared endpoints.

3.4 Relation Reasoning Module

Given the candidate graph $\hat{\mathcal{G}}$, we now introduce our Relation Reasoning Module, which is a graph neural network defined on the top of $\hat{\mathcal{G}}$. Each vertex in the graph neural network is associated with a line segment proposal and is connected to other line proposals based on the adjacent matrix $\hat{\mathbf{A}}$. The graph neural network takes line segment features as input and conducts global reasoning through message passing, resulting in a context-aware representation for each line proposal. We then predict a binary label for each graph vertex to indicate whether it is a foreground line segment.

Concretely, we first extract two sets of line segment features to encode their semantic and geometric property. For semantic features, we adopt the LoI pooling [49], which max-pools and concatenates convolutional features from a set of uniformly sampled points on the line segments. Let the number of sampled points be N_p and the pooling stride be δ, we denote the semantic feature of vertex $v \in \hat{\mathcal{V}}$ as $\hat{\mathbf{x}}_v^s \in \mathbb{R}^{N_p \cdot D/\delta}$. For geometric features, we concatenate the line central point's coordinate $\mathbf{p}_{\hat{c}}$ and the shift vector $\hat{\mathbf{s}}$, which is denoted as $\hat{\mathbf{x}}_v^p$.

Given the input features, we initialize the vertex representations of graph neural network by computing an embedding of line segment features:

$$\mathbf{e}_v^{(0)} = \Phi(\hat{\mathbf{x}}_v^s), \quad \mathbf{g}_v^{(0)} = \Psi(\hat{\mathbf{x}}_v^p),\ v \in \hat{\mathcal{V}} \tag{2}$$

where $\mathbf{e}_v^{(0)}$ and $\mathbf{g}_v^{(0)} \in \mathbb{R}^d$ are the embedded representations of the semantic and geometric features of the vertex v. Here Φ is a two layer perceptron, and Ψ is simply a linear projection.

We update the semantic and geometric representations in parallel by running two separate message passing procedures in the graph in order to capture both semantic and geometric context. To achieve this, we first compile features from all the neighborhoods of vertices in our graph convolutional network and then perform a non-linear transformation on the aggregated features to update the representations of the vertices. To capture long-range context, we stack multiple layers of such graph convolutions.

Specifically, we adopt the residual GCN blocks [22] for message computation and vertex feature update within each layer. Let $\mathbf{E} = [\mathbf{e}_1, \cdots, \mathbf{e}_{|\hat{\mathcal{V}}|}]$ and $\mathbf{G} = [\mathbf{g}_1, \cdots, \mathbf{g}_{|\hat{\mathcal{V}}|}]$ be the collections of semantic and geometric representations of the vertices, we update their embedding in the $l+1$ layer as follows,

$$\mathbf{E}^{(l+1)} = \phi\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{E}^{(l)}\mathbf{W}_s^{(l)}\right) + \mathbf{E}^{(l)} \tag{3}$$

$$\mathbf{G}^{(l+1)} = \phi\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{G}^{(l)}\mathbf{W}_g^{(l)}\right) + \mathbf{G}^{(l)} \tag{4}$$

where $\widetilde{\mathbf{A}} = \hat{\mathbf{A}}+\mathbf{I}$, $\widetilde{\mathbf{D}}$ is a diagonal matrix with $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$. $\mathbf{W}_s^{(l)}$ and $\mathbf{W}_g^{(l)}$ are the weight parameters in the l-th layer. ϕ is an activation function, and we simply choose ReLU funcion.

We stack n residual GCN blocks to update the semantic and geometric features of vertices, which are concatenated to generate the final line segment representation: $\mathbf{H} = [\mathbf{E}^{(n)}, \mathbf{G}^{(n)}] \in \mathbb{R}^{|\hat{\mathcal{V}}| \times 2d}$. Finally, we adopt a multi-layer perceptron to classify line segments into foreground or background based on the updated line segment features $\mathbf{H}$. During inference, we also attach a sigmoid function to generate the final score for each line segment.

3.5 Model Training

To train our model, we develop a multi-task loss to supervise the learning of two model modules jointly. The loss function consists of two parts, one for the Multitask Learner Module and one for the entire network:

$$L = L_{ML} + L_{RR} \tag{5}$$

where L_{ML} denotes the loss terms for the first module and the L_{RR} is the loss terms imposed on the output of the second module.

The loss term L_{ML} includes the loss terms for the four outputs of the Multitask Learner Module as follows:

$$L_{ML} = \lambda_j L_j + \lambda_c L_c + \lambda_o L_o + \lambda_s L_s \tag{6}$$

where the loss item L_j is for junction keypoints, L_c is for line central keypoints, L_o is for offset of junction keypoints and line central keypoints, L_s is for line shift vector, and $\lambda_{j,c,o,s}$ are the weights of the corresponding loss item.

Specifically, we use the binary cross entropy loss L_j for junction/endpoint prediction, and

$$L_j = -\frac{1}{N_p}\sum_j (y_j \log(\hat{y}_j) + (1-y_j)\log(1-\hat{y}_j)) \tag{7}$$

where y_j is the binary junction indicator and $\hat{y}_j$ is the predicted junction probability. N_p is the number of pixels in the output heatmap.

For the line central point prediction, we use the focal loss [23] for line central point estimation due to unbalanced positive/negative

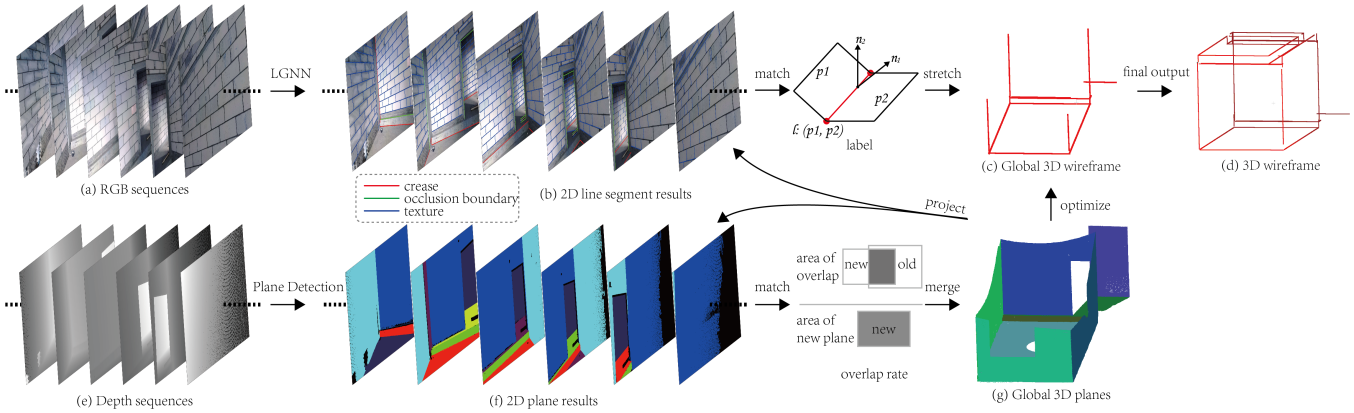

Figure 3: Flowchart of the 3D line segment detection. Given a sequence of RGBD images, we first detect 2D line segments and surface planes from RGB and depth, respectively. These results are subsequently fused into a 3D wireframe based on 2D-3D and line-plane consistency.

distribution:

$$L_c = -\frac{1}{N_p}\sum_c \begin{cases} (1-\hat{y}_c)^{\alpha}\log(\hat{y}_c) & \text{if } y_c = 1 \\ (1-y_c)^{\beta}\,\hat{y}_c^{\alpha}\log(1-\hat{y}_c) & \text{otherwise} \end{cases} \tag{8}$$

where y_c is ground truth scores of line central points and $\hat{y}_c$ is predicted line central point probability. α and β are hyper-parameters. We use $\alpha = 2$ and $\beta = 4$ as in [21].

We employ l1 loss for the junction and line central point offset regression loss L_o, and the shift vector prediction loss L_s.

The loss term L_{RR} is defined for the final outputs from the MLPs of the Relation Reasoning Module. Here we use the binary cross-entropy loss for line segments classification.

We train our network in a joint manner by computing an approximate gradient over the entire network, in which the gradient calculation treats the proposal generation is fixed in each iteration. This "end-to-end" training strategy works well in practice for our model and is simple to implement.

4 3D WIREFRAME EXTRACTION

A unique advantage of our LGNN-based line segment detector is its speed. Compared with the most accurate technique [49], LGNN slightly sacrifices the performance but nearly doubles the speed. The near real-time performance can benefit a number of applications.

When combined with 3D scanning, LGNN provides a viable solution for space measurement. Conceptually 3D scanning techniques such as LiDAR or time-of-flight can already produce 3D point cloud data amenable for analysis. In reality, the point cloud is generally of a low resolution and contains strong noise. We observe that walls, in particular, intersection between different walls, form line segments and as long as we can detect them, we can use the results for measuring the dimensionality of the space.

However, there are multiple challenges. Our line detector, same as any existing techniques, detects both geometric and texture edges. For the detected lines to be useful, it is essential to distinguish these lines: the key to space measurement is creases that correspond to junctions of walls. We employ a real-time line type classification algorithm on top of LGNN. Fig. 3 shows our processing pipeline.

We assume a moving 3D scanner whose position is calibrated in real-time using Visual SLAM techniques such as ORB-SLAM [30]. The point clouds are fused on the fly. Given each line segment provided by LGNN, we check if it corresponds to crease.

Specifically, we observe that it corresponds to crease when the two sides of the line segment correspond to two different planes (i.e., have very different normal directions). For texture lines, the two sides would correspond to the same plane. However, for occlusion boundaries, the two sides can also correspond to different planes as creases. Nonetheless, we can easily identify occlusion boundaries since the points on the two sides will have large depth disparity. Further, regular rooms have walls perpendicular to each other and therefore we can check if the normals are orthogonal to further improve accuracy on crease detection.

Two key steps in the approach above are 1) to group 3D points in terms of planes that they belong to and 2) to stretch line segments as the camera moves, to avoid fragmentation. Fig. 3 shows the complete pipeline of our technique: using a sequence of RGBD frames as input, we use LGNN to extract, for every single frame, line segments. We maintain a global 3D plane set and a global 3D line set. To maintain the 3D plane set, we implement a point-plane merging technique that adds newly detected 3D points to the set. To maintain the 3D line set, we use the plane set to classify the 2D lines detected by LGNN and refine the set on the fine. Both steps can be implemented in real time.

To elaborate, we apply LGNN on an RGB frame and the fast plane detection method [10] on the depth channel. The difficulty is that once we move the camera, we need to merge newly detected points with the existing ones. Further, the RGBD sensor is generally of a low resolution and the new points will slightly deviate from the actual plane. To handle that, we record the total number of already scanned 3D points N_i and label every point with a unique plane id i. When new 3D points become accessible, we add a plane Π_{new} into global plane set with a point count N_{new} (i.e., how many points in Π_{new}) and normal n_{new}. We set out to determine if Π_{new} should be merged with the existing set or should be added as a new one.

To do so, we project the global planes into the current RGB image via standard graphics rasterization. We then calculate its overlap count N_i of projected global plane Π_i with respect to the points in Π_{new}. We define overlap ratio $r_i = \frac{N_i}{N_{new}}$. Finally, we merge Π_{new} with Π_{max} that has the maximum overlap ratio with Π_{new}. If r_{max} exceeds a threshold and the normals of n_{new} and n_{max} are close enough, we will merge them into one plane. Otherwise, we will create a new plane and add it to the global plane set.

For line segment classification, we simply build a histogram in terms of the plane id of the points that lie within a range on both sides. If the histogram contains a very high peak that corresponds to a single id, the line is then deemed as a texture edge. If the histogram has two similar peaks of different ids and the depths of the points are similar, the line segment is then deemed as crease. Otherwise, it is deemed as an occlusion boundary. We further label the creases with the indexes of its two adjacent planes so that we can match these line segments in following frames.

The classification procedure avoids fragmentation of line segments that correspond to creases: we can merge creases that correspond to the same pair of junction planes (i.e., the same plane ids). In fact, we can obtain the final line segment by simply computing the intersection line of the two planes. Fig. 3(b) shows several typical examples of the line segment classification results. Finally, we can use the merged creases to obtain a wireframe model of the 3D environment and then measure the space, as shown in Fig. 3(d).

5 EXPERIMENTS

In this section, we introduce details of our implementation and evaluate the proposed line segment detector with existing state-of-the-art line segment detectors. Then, we visualize the results of our 3D wireframe detection system.

5.1 Implementation Details

We stack two hourglass networks as our backbone, for each input image, we first resize it to the size of $(512, 512, 3)$ and output a feature map with the size of $(128, 128, 256)$. Then, we feed the intermediate feature map into five network heads and produce junction heatmap, line central point heatmap, junction offset map, line central offset map, and line shift vector map. During the training phase, the number of proposals of junctions is two times the number of ground truth junctions and with the maximum value of 300. During the evaluating period, we set a threshold of 0.008 to choose the most likely junctions. To get line segment quadruplets, we set the threshold of θ of 15 to get matched line segments. For the LoIPooling, we first uniformly choose $N_p = 32$ middle point features along each line segments, then apply a max-pooling with the stride of 4 to get the flattened semantic line feature of size 2048. For graph neural network, we first embed the semantic feature and geometric feature to the same size of 256, then stack several residual GCN blocks to update the feature. Finally, we use a two-layer MLPs with hidden layers of the size of 32 to classify each line segment.

We train LGNN from scratch using ADAM optimizer [19], with an initial learning rate of $1.0e-3$ and weight decay of $1.0e-4$ on a single GPU P6000. We set batch size to 10 for the fastest training speed. To dynamically adjust the learning rate based on validation measurements, we adopt the ReduceLROnPlateau[1] scheduler with the patience of zero epoch and factor of 0.5. We augment the wireframe dataset with standard strategies including flipping vertically, horizontally, and centrally for images and annotations to overcome overfit.

Methods	sAP	FPS
Wireframe [17]	6.0	3.9
AFM [41]	27.5	12.0
L-CNN [49]	63.0	9.5
Ours-lite	57.6	34.0
Ours	62.3	15.8

Table 1: Performance comparison of line segment detection approaches on the wireframe dataset [17]. We adopt sAP [49] as our evaluation metric and report the average FPS on the test set of wireframe [17].

5.2 Performance Evaluations

We evaluate our line segment detection performance on the wireframe dataset [17], which contains 5,000 training images, 462 testing images. For faster training, we preprocess the wireframe dataset to generate ground-truth keypoint, offset and shift vector maps. Specially, we generate ground-truth line central point maps by using the 1D Gaussian kernel along each line segment. This step is crucial for our network to converge well.

LGNN vs. State-of-the-Art. We have compared our LGNN with the state-of-the-art line segment detection algorithms: AFM [41] and L-CNN [49], with the same wireframe dataset [17], the same training and testing split, and the same hardware environments.

We experiment with the sAP metric which is proposed by L-CNN [49] to evaluate the performance of these methods. The sAP metric for line segment detection properly penalizes for the overlapped and incorrectly connected line segments, so is a more reasonable metric for evaluating the structural quality of wireframes compared with the heat map-based metric $-AP^H$, which treats each pixel independently. As reported in Table 1, our method achieves comparable results while runs the fast.

We visualize results of the proposed LGNN and other methods in Fig. 4. We can see that our approach is capable of extracting complete and cleaner line segments compared with AFM [41] and L-CNN [49]. Treating each line segment as a quadruplet, we get an accurate description of each line segment. Conversely, AFM generates line segment by greedily grouping pixels, this way usually fails to guarantee a complete and accurate line segment. Compared with L-CNN [49], although it gets the best sAP performance, it produces more overlapped or close co-linear line segments. Our method suppresses most of these line segments so that our results look cleaner.

Ablation Studies. In this section, we run several ablation experiments to study the Relation Reasoning Module in our proposed method:

[1]For eg. https://pytorch.org/docs/stable/optim.html?highlight=reducelronplateau#torch.optim.lr_scheduler.ReduceLROnPlateau

Figure 4: Qualitative results on 2D line segment detection. First row: AFM [41]; Second row: L-CNN [49]; Third row: LGNN(ours); Fourth row: Ground truth. Our method achieves competitive results with real-time efficiency.

(a) Panorama (b) Plane results (c) 3D wireframe results

Figure 5: Given a sequence of RGBD images and camera poses as inputs, LGNN enables a 3D wireframe parsing algorithm to detect structural line segments in real-time, and to fuse with estimated planar surfaces for online 3D scene reconstruction.

(i) We experiment with several types of line segment features: only semantic feature; only geometric feature; both semantic and geometric feature; semantic feature, geometric feature, and key-point scores. We can see that effectively combining semantic and

semantic	geometric		scores	sAP
	coord	shift		
✓				61.4
✓	✓			61.7
✓	✓	✓		62.3
✓	✓	✓	✓	61.4

Table 2: Ablation study on multiple features in the Relaton Reasoning Module. 'coord' represents the line central point position, 'shift' represents the line shift vector, 'scores' represents the 3d concatenated keypoint scores of the line quadruplet.

geometric feature ensures the best line segment prediction performance. We also attempt to add the keypoint scores as an additional feature, which however can not further improve the performance. We conjecture that this might be caused by the redundancy and noisy nature of the keypoint scores.

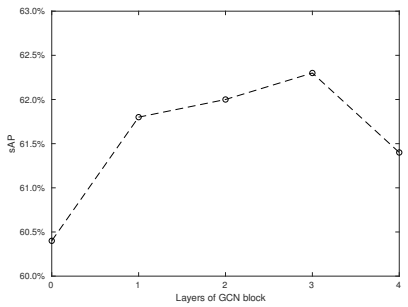

Figure 6: Model performance using different layers of ResGCNs in the Relation Reasoning Module. We show the sAP on the wireframe [17] test set for 0, 1, 2, 3, and 4 layer ResGCNs.

(ii) We also evaluate our Relation Reasoning Module with different layers of ResGCNs. In Fig. 6, we achieve the best performance with three layers ResGCNs. The performance of shallower ResGCNs shows a steady decrease. When the number of layers drops to zero, i.e., does not pass message, there is a sharp performance gap. Compared with MLPs, ResGCNs obtains an absolute gain of 1.9% in terms of sAP. The deeper ResGCNs also has a lower sAP partially due to the difficulty in model learning.

For a fair comparison among CNN, MLP, and GCN, we add CNN layers to the CNN backbone or substitute GCN with MLP in our proposed Relation Reasoning Module so as to keep the same layers and the same feature dimensions. We have experimented that GCN obtains an absolute gain of 1.9% in terms of sAP, while MLP and CNN only obtain 1.1% and 0.0%, respectively. We conclude that line segment detection benefits greatly from ResGCNs which can more effectively aggregate and cope with context information.

3D Wireframe Extraction. We have further tested our 3D wireframe extraction technique in multiple scenes. Fig. 5 and Fig. 7 show two typical examples, a medium sized roughcast room and

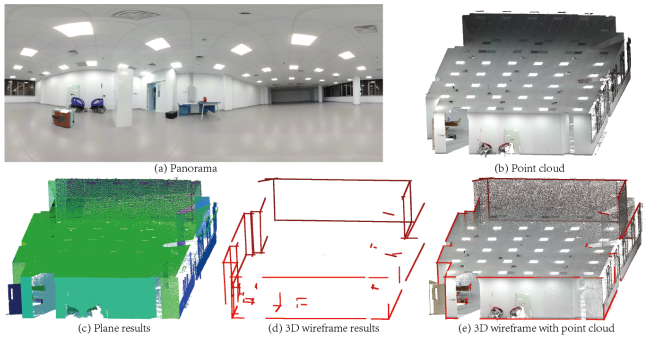

Figure 7: Wireframe parsing results on a large robotic lab. (a) Illustration of the overall scene as a panorama. (b & c) Point cloud and results of plane detection. (d & e) 3D wireframe result and its overlay with the point cloud.

a large robotic lab. Both rooms were pre-scanned using the Faro 3D sensor. We simulate procedure 3D scanning by position a virtual camera at the center of the room and the camera rotates to capture a sequence of RGBD images. Fig. 5 and Fig. 7 show that our system manages to extract accurate, complete and structural line segments in both cases, automatically forming high quality wireframe 3D models, a function largely missing in existing 3D scanning solutions. Recall that the Faro scanner still cannot recover regions occluded from the viewpoint of the scanning location, nor can it recover specular regions such as windows. The robotic lab scene is particularly challenging as direct mapping from 2D line segment to 3D introduces strong noise due to large depth range. Our technique, however, manages to not only reliably detect the structural and texture lines but also accurately determine their 3D locations. Fig. 7 (e) shows that our extracted 3D wireframe fits well with the point cloud.

6 CONCLUSIONS

We have introduced a novel yet effective line segment detection method based on graph neural network. By representing each line segment as a quadruplet and all line segments in an image as a sparse graph, our method manages to not only extract structural line segments but also greatly reduce the computational cost. Benefiting from deep residual graph neural network, our method can effectively incorporate both semantic and geometric features of line segments.

Our future work will extend the LGNN in several directions. Firstly, with additional semantic annotations, we can jointly infer the geometric attribute and semantic label of each line segment for more coherent wireframe reconstruction. In addition, it is desirable to integrate line and plane detection for more robust 3D scene parsing. Furthermore, we will go beyond straight lines and consider other types of curved object boundaries in complex scenes.

ACKNOWLEDGMENTS

This work was supported by NSFC programs (61976138, 61977047), STCSM (2015F0203-000-06), the National Key Research and Development Program (2018YFB2100500) and SHMEC (2019-01-07-00-01-E00003).

REFERENCES

[1] Emilio J Almazan, Ron Tal, Yiming Qian, and James H Elder. 2017. Mcmlsd: A dynamic programming approach to line segment detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2031–2039.

[2] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. 2010. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence* 33, 5 (2010), 898–916.

[3] Nam-Gyu Cho, Alan Yuille, and Seong-Whan Lee. 2017. A novel linelet-based representation for line segment detection. *IEEE transactions on pattern analysis and machine intelligence* 40, 5 (2017), 1195–1208.

[4] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*. 379–387.

[5] Johan De Bock and Wilfried Philips. 2007. Line segment based watershed segmentation. In *International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*. Springer, 579–586.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[7] Patrick Denis, James H Elder, and Francisco J Estrada. 2008. Efficient edge-based methods for estimating manhattan frames in urban imagery. In *European conference on computer vision*. Springer, 197–210.

[8] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 6569–6578.

[9] Olivier D Faugeras, Rachid Deriche, Hervé Mathieu, Nicholas Ayache, and Gregory Randall. 1992. The depth and motion analysis machine. In *Parallel image processing*. World Scientific, 143–175.

[10] Chen Feng, Yuichi Taguchi, and Vineet R Kamat. 2014. Fast plane extraction in organized point clouds using agglomerative hierarchical clustering. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 6218–6225.

[11] Yasutaka Furukawa and Yoshihisa Shinagawa. 2003. Accurate and robust line segment extraction by analyzing distribution around peaks in Hough space. *Computer Vision and Image Understanding* 92, 1 (2003), 1–25.

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1263–1272.

[13] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 1440–1448.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.

[16] Manuel Hofer, Michael Maurer, and Horst Bischof. 2017. Efficient 3D scene abstraction using line segments. *Computer Vision and Image Understanding* 157 (2017), 167–178.

[17] Kun Huang, Yifan Wang, Zihan Zhou, Tianjiao Ding, Shenghua Gao, and Yi Ma. 2018. Learning to parse wireframes in images of man-made environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 626–635.

[18] Varsha Kamat-Sadekar and Subramaniam Ganesan. 1998. Complete description of multiple line segments using the Hough transform. *Image and Vision Computing* 16, 9-10 (1998), 597–613.

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

[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[21] Hei Law and Jia Deng. 2018. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 734–750.

[22] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE International Conference on Computer Vision*. 9267–9276.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[24] Yangbin Lin, Cheng Wang, Bili Chen, Dawei Zai, and Jonathan Li. 2017. Facet segmentation-based line segment extraction for large-scale point clouds. *IEEE Transactions on Geoscience and Remote Sensing* 55, 9 (2017), 4839–4854.

[25] Yangbin Lin, Cheng Wang, Jun Cheng, Bili Chen, Fukai Jia, Zhonggui Chen, and Jonathan Li. 2015. Line segment extraction for large scale unorganized point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing* 102 (2015), 172–183.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[27] Xiaohu Lu, Yahui Liu, and Kai Li. 2019. Fast 3D Line Segment Detection From Unorganized Point Cloud. *arXiv preprint arXiv:1901.02532* (2019).

[28] Yi Lu, Yaran Chen, Dongbin Zhao, and Jianxin Chen. 2019. Graph-FCN for image semantic segmentation. In *International Symposium on Neural Networks*. Springer, 97–105.

[29] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*. 3697–3707.

[30] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE transactions on robotics* 31, 5 (2015), 1147–1163.

[31] Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*. Springer, 483–499.

[32] Pietro Parodi and Giulia Piccioli. 1996. 3D shape reconstruction by using vanishing points. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18, 2 (1996), 211–217.

[33] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[35] Aparajithan Sampath and Jie Shan. 2009. Segmentation and reconstruction of polyhedral building roofs from aerial lidar point clouds. *IEEE Transactions on geoscience and remote sensing* 48, 3 (2009), 1554–1567.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[37] Rafael Grompone Von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. 2012. LSD: a line segment detector. *Image Processing On Line* 2 (2012), 35–55.

[38] Tian-Zhu Xiang, Gui-Song Xia, Xiang Bai, and Liangpei Zhang. 2018. Image stitching by line-guided local warping with global similarity constraint. *Pattern Recognition* 83 (2018), 481–497.

[39] Peng Xu, Chaitanya K Joshi, and Xavier Bresson. 2019. Multi-graph transformer for free-hand sketch recognition. *arXiv preprint arXiv:1912.11258* (2019).

[40] Zezhong Xu, Bok-Suk Shin, and Reinhard Klette. 2015. A statistical method for line segment detection. *Computer Vision and Image Understanding* 138 (2015), 61–73.

[41] Nan Xue, Song Bai, Fudong Wang, Gui-Song Xia, Tianfu Wu, and Liangpei Zhang. 2019. Learning attraction field representation for robust line segment detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1595–1603.

[42] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)*. 670–685.

[43] Yibo Yang, Hongyang Li, Xia Li, Qijie Zhao, Jianlong Wu, and Zhouchen Lin. 2019. SOGNet: Scene Overlap Graph Network for Panoptic Segmentation. *arXiv preprint arXiv:1911.07527* (2019).

[44] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[45] Zhan Yu, Xinqing Guo, Haibing Lin, Andrew Lumsdaine, and Jingyi Yu. 2013. Line assisted light field triangulation and stereo matching. In *Proceedings of the IEEE International Conference on Computer Vision*. 2792–2799.

[46] Lilian Zhang and Reinhard Koch. 2014. Structure and motion from line correspondences: Representation, projection, initialization and sparse bundle adjustment. *Journal of Visual Communication and Image Representation* 25, 5 (2014), 904–915.

[47] Ziheng Zhang, Zhengxin Li, Ning Bi, Jia Zheng, Jinlei Wang, Kun Huang, Weixin Luo, Yanyu Xu, and Shenghua Gao. 2019. PPGNet: Learning point-pair graph for line segment detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7105–7114.

[48] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. 2019. Objects as points. *arXiv preprint arXiv:1904.07850* (2019).

[49] Yichao Zhou, Haozhi Qi, and Yi Ma. 2019. End-to-end wireframe parsing. In *Proceedings of the IEEE International Conference on Computer Vision*. 962–971.