Learning to Detect Geometric Structures from Images for 3D Parsing

by

Yichao Zhou

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Yi Ma, Chair
Professor Jitendra Malik
Professor Yasutaka Furukawa
Professor Sara McMains

Fall 2020

Learning to Detect Geometric Structures from Images for 3D Parsing

Abstract

Learning to Detect Geometric Structures from Images for 3D Parsing

by

Yichao Zhou

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Yi Ma, Chair

Recovering 3D geometries of scenes from 2D images is one of the most fundamental and challenging problems in computer vision. On the one hand, traditional geometry-based algorithms such as SfM and SLAM are fragile in certain environments, and the resulting noisy point-clouds are hard to process and interpret. On the other hand, recent learning-based 3D-understanding neural networks parse scenes by extrapolating patterns seen in the training data, which often have limited generalizability and accuracy.

In my dissertation, I try to address these shortcomings and combine the advantage of geometry-based and data-driven approaches into an integrated framework. More specifically, I have applied learning-based methods to extract high-level geometric structures from images and use them for 3D parsing. To this end, I have designed specialized neural networks that understand geometric structures such as lines, junctions, planes, vanishing points, and symmetry, and detect them from images accurately; I have created large-scale 3D datasets with structural annotations to support data-driven approaches; and I have demonstrated how to use these high-level abstractions to parse and reconstruct scenes. By combining the power of data-driven approaches and geometric principles, future 3D systems are becoming more accurate, reliable, and easier to implement, resulting in clean, compact, and interpretable scene representations.

To my parents for their love and support

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude and thanks to my advisor, Professor Yi Ma. Unlike most Ph.D. students, I joined Yi's group at the end of the third year of my journey and became his first Ph.D. student at UC Berkeley. In this short period of time, Yi provided me not only with academic advice, but also numerous life suggestions. From Professor Ma, I learned how to think from a high-level and long-term perspective, instead of getting stuck on the nitty-gritty. I would not have been in this stage without his support and guidance.

Second, I want to thank Professor Leonidas J. Guibas from Stanford University, who was a member of my qualification exam committee. He drove all the way from Stanford to UC Berkeley to attend my qualification exam, which I feel really grateful for. He also provided me with helpful suggestions and we collaborated on several projects. Besides, I also want to thank all the other members of my dissertation committee: Professor Yasutaka Furukawa, Professor Jitendra Malik, and Professor Sara McMains, for their time of providing me guidance and attending my qualifying exam and/or dissertation talk.

During the pursuit of my Ph.D. degree, I have met many smart and friendly collaborators. There is one person who I want to specifically thank, Dr. Jingwei Huang from Stanford University. He is not only a reliable collaborator, but also a close friend of mine. He introduced me into the world of 3D reconstruction, and we collaborated on multiple projects related to traditional geometry processing techniques and more recently learning-based 3D vision. I also want to thank Shichen Liu from USC. He always responses to my random research ideas promptly and is a good Starcraft II player. Moreover, I want to thank Haozhi Qi from Yi Ma's group, who is truly an expert in instance segmentation. He inspired me to use techniques from object detection into the problem of wireframe detection. In addition, I want to thank all the other students in Berkeley that I have worked with: Xili Dai, Simon Zhai, Chong You, Yaodong Yu, Cecilia Zhang, Biye Jiang, and Hezheng Yin, just to name a few, for the insightful discussions and smooth collaboration.

Furthermore, I would like to thank my sponsor Sony Research, which funds most of the research projects in this dissertation. I want to specifically thank Kenji Tashiro from Sony for his helpful suggestions and resources. Besides, I want to thank all the mentors I met during my summer internship. Many of my research ideas will not come true without their kind help. Qi Sun, Zhili Chen and Li-Yi Wei were my mentors in Adobe Research, who provided me detailed guidance in the initial stage of the research covered in this dissertation. Linjie Luo was my mentor from Bytedance. I appreciate his help in the project of HoliCity. I had a great time working with Wei Chu from Snap Inc. and Sam Young at UCLA on the Beatles-style music composition project. I also want to thank Jing Liu, James Davis, and Eric Chen from Bellus3D, from whom I learned a lot in 3D vision, graphics, and entrepreneurship.

Last but not least, I feel grateful for the unconditional love from my parents all the time. My dad always encourages me to explore the outside world and step out of my comfort zone to be stronger, while my mom teaches me how to become a more mature person, and she always provides help to me whenever I need it. Finally, I sincerely want to thank my girlfriend for being so supportive of my Ph.D. career. You make me a better person.

Chapter 1

Introduction

1.1 Motivation

In the past decades, we have witnessed an increasing demand for 3D vision technologies. Traditional geometry-based reconstruction algorithms such as structure-from-motion (SfM) and simultaneous localization and mapping (SLAM) have been successfully applied to tasks such as autonomous driving, robotics, mapping, and augmented reality. Leading 3D vision products, such as Hololens and Apple ARkit, have been able to localize themselves and reconstruct the environment. The underlying localization and reconstruction algorithms might differ slightly depending on the configuration of input sensors, but they all rely on these three components: (1). Feature extraction & matching; (2). Camera pose estimation; and (3). Triangulation. From a stereo pair of images or videos, they first extract local point features such as SIFT [1], ORB [2], or Shi–Tomasi [3], and track them to build the inter-frame correspondence. Next, they apply multi-view geometry algorithms such as the 5-point algorithm [4] and P3P [5] along with RANSAC [6] and bundle adjustment [7] to estimate the relative camera poses. Finally, they triangulate detected feature points with the camera poses to determine the depth of each feature point and reconstruct sparse point-clouds.

Although the robustness of SfM has been improved over the last decades, it is known that these traditional geometric reconstruction systems are fragile under certain environments. Specifically, they are unreliable when scenes are textureless, reflective, repetitive, far away (small-baseline), or containing moving objects. Furthermore, the scene representation remains quasi-dense point-clouds, which are typically incomplete, noisy, hard to parse, and cumbersome to share. Intricate failure handling and post-processing procedures such as plane fitting [8] and Poisson surface reconstruction [9] are often necessary for downstream applications. Increasingly people have found that such a long pipeline of 3D reconstruction is difficult to implement correctly and efficiently, and results in in a low-level representation that are also unfriendly for parsing, editing, and sharing.

The recent success of deep convolutional neural networks (CNNs) in image classification [10] and object detection [11] have shown the potential of data-driven approaches, which

Figure 1.1: Examples of structures in man-made environments, including lines, smoothed curves, flat surfaces, parallelism, orthogonality, and symmetry.

inspires researchers that it is possible to directly predict the 3D information from images by interpolation from data with machine learning methods instead of relying on geometry principles. With an encoder-decoder structures [12], neural networks are able to predict a variety of 3D representations from single images, such as camera poses [13], 3D bounding boxes [14], depth maps [15], point-clouds [16], meshes [17], voxels [18], and implicit surfaces [19]. On some tasks, 3D deep learning-based neural networks have shown the state-of-the-art performance, such as shape reconstruction [17], multiview stereopsis [20], and single-view depth estimation [21].

However, unlike SfM and SLAM algorithms that use geometry cues to reconstruct 3D scenes, learning-based approaches predict 3D representations by extrapolating patterns seen during the training time. Therefore, the generalizability of these neural networks is severely limited, which leads to poor real-world performance. In fact, recent research indicates that all these existing single-view 3D inference networks with encoder-decoder structures do not perform better than a simple baseline which does image classification followed by 3D model retrieval [22]. This indicates that most 3D reconstruction neural networks can only "classify" rather than "reconstruct." Therefore, for tasks such as SfM and visual odometry that require high precision and good generalizability, neural network-based approaches still largely fall behind traditional geometry-based methods [23].

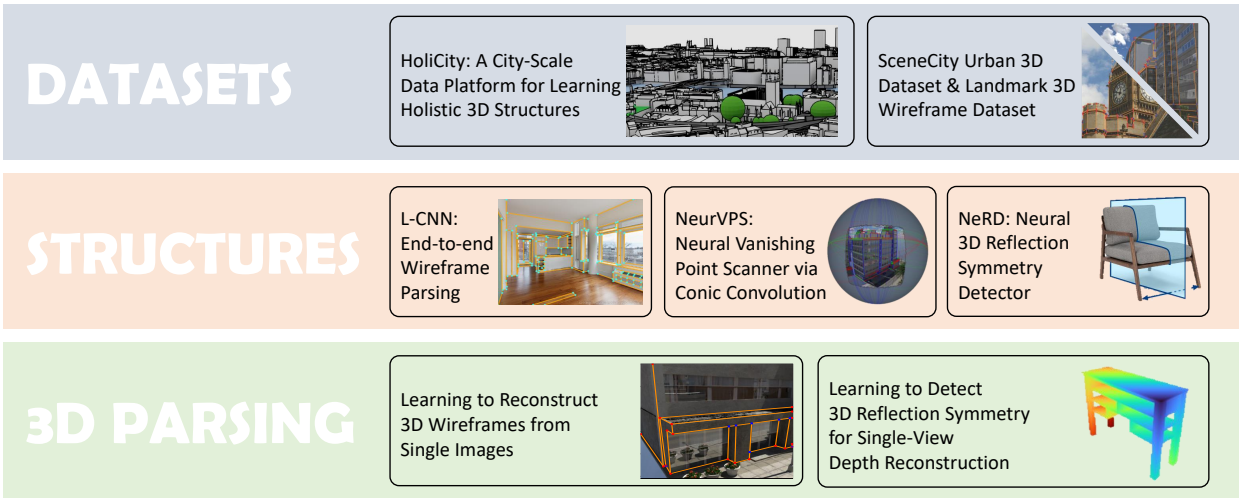


Figure 1.2: The structural diagram of the dissertation.

1.2 Contributions

One may wonder that under the setting of single-image 3D understanding, whether it is possible to do better than the data-driven neural networks in principle. After all, the problem itself seems to be fairly ill-posed from the geometric perspective, because there are no obvious constraints between the input RGB image and underlying 3D shapes from a single view. One way to tackle this problem is to take a look at the type of images that we normally deal with, as shown in Figure 1.1. In practice, most of the scenes that we interact with, indoor or outdoor, are man-made environments, which are rich in structural regularities, including straight lines, smooth curves, flat surfaces, parallelism, orthogonality, and different kinds of symmetries (reflectional, translational, and rotational). These structural regularities are prominent from a single image, which provides us strong priors for inferring 3D shapes.

Looking back at the origin of computer vision from the '70s, early vision scientists have already found that human beings abstract scenes with structural primitives, such as corners, line segments, and planes, to form our sense of 3D, navigate in cities and interact with environments [24]. This hints that instead of using low-level representations such as point-clouds, we can also use high-level geometric structures to represent scenes, which in many cases are more global, compact, intuitive, and easy to process. Early vision research does focus on understanding scenes with high-level abstractions, such as lines/wireframes [25], vanishing points [26], contours/boundaries [27], planes/surfaces [28], and cuboids/polyhedrons [29]. These high-level abstractions are often called *holistic structures*, as they tend to represent scenes globally, compared to the SIFT-like local features. However, the recognition of holistic structures from images seemed too challenging to be practical at that time. 3D reconstruction with high-level structures did not get enough attention despite its potential, until recently.

With the rise of deep convolutional neural networks [30], detecting high-level structures is becoming practical. Researchers have proposed a variety of neural networks to extract

high-level structures, such as wireframes [31], planes [32], cuboids [14], room layouts [33], and building layouts [34]. This hints that we may use learning-based approaches to detect geometric structures from images for 3D parsing, in which the high-level geometric structures serves as a bridge between the methods of "3D-from-geometry" and "3D-from-data."

The goal of my research is to develop principled algorithms that can accurately and robustly understand the 3D and reconstruct scenes with high-quality CAD-like models. To achieve that, I integrate geometry-based methods and data-driven approaches into a unified framework. More specifically, I apply learning-based methods to extract high-level geometric structures from images and use them for 3D parsing. To this end, I have designed specialized neural networks that understand geometric structures such as lines, junctions, planes, vanishing points, and bilateral symmetry, and accurately detect them from images (Chapter 2); I have created several 3D datasets with structural annotations to support data-driven approaches (Chapter 3); and I have demonstrated how to use these high-level abstractions to parse and reconstruct scenes (Chapter 4). With the advances in these tracks, future 3D systems will have better generalization capability and become more efficient, robust, and accurate. The output 3D representations will also be compact, semantically meaningful, and friendly for human perception, interpretation, and interaction, as well as easily and efficiently shared among autonomous agents.

Chapter 2

Detecting Geometric Structures from Images

In this chapter, we will introduce our data-driven approaches to extract high-level geometric structures from images, including wireframes (Section 2.1), vanishing points (Section 2.2), and reflectional symmetry (Section 2.3). The methods described in this chapter have been published in [35, 36, 37].

2.1 Learning to Detect Wireframes

2.1.1 Introduction

Recent progress in image classification [10] and large-scale datasets [30] has made it possible to recognize, extract, and utilize high-level geometric features or global structures of a scene for image-based 3D reconstruction. Unlike local features such as SIFT [38] and ORB [2] used in conventional 3D reconstruction systems such as structure from motion (SfM) and visual SLAM, high-level geometric features provide more salient and robust information about the global geometry of the scene. This line of research has drawn interests on the exploration of extracting structures such as lines and junctions (wireframes) [31], planes [32, 39], surfaces [40], and room layouts [33].

Among all the high-level geometric features, straight lines and their junctions (together called a wireframe [31]) are probably the most fundamental elements that can be used to assemble the 3D structures of a scene. Recently, works such as [31] encourages the research of wireframe parsing by providing a well-annotated dataset, a learning-based framework, as well as a set of evaluation metrics. Nevertheless, existing wireframe parsing systems are intricate and still inadequate for complex scenes with complicated line connectivity. The goal of this dissertation is to explore a clean and effective solution to this challenging problem.

Existing research [31, 41] addresses the wireframe parsing problem with two stages. First, an input image is passed through a deep convolutional neural network to generate pixel-wise

junction and line heat maps (or their variants [41]). After that, a heuristic algorithm is used to search through the generated heat map to find junction positions, vectorized line segments, and their connectivity. While these methods are intuitive and widely used in the current literature, their vectorization algorithms are often complex and rely on a set of heuristics, and thus sometimes lead to inferior solutions. Inspired by [11, 42, 43] in which the end-to-end pipelines outperform their stage-wise counterparts, we hypothesis that making wireframe parsing systems end-to-end trainable could also push the state-of-the-art. Therefore, in this section we address the following problem:

> *How to learn a vectorized representation of wireframes in an end-to-end trainable fashion?*

To this end, we propose a new network called *L-CNN*, an algorithm that performs end-to-end wireframe parsing using a single and unified neural network. Our network can be split into four parts: a feature extraction backbone, a junction proposal module, and a line verification module bridged by a line sampling module. Taken an RGB image as the input, the neural network directly generates a vectorized representation without using heuristics. Our system is fully differentiable and can be trained end-to-end through back-propagation, enabling us to fully exploit the power of the state-of-the-art neural network architectures to parse the scenes.

Besides, current wireframe evaluation metrics treat a line as a collection of independent pixels, so it cannot take the correctness of line connectivity into consideration, as discussed in Section 2.1.4.3. To evaluate such structural correctness of a wireframe, we introduce a new evaluation metric. Our new proposed metric uses line matching to calculate the precision and recall curves on vectorized wireframes. We perform extensive experiments on wireframe datasets [31] and carefully do the ablation study on the effects of different system design choices.

2.1.2 Related Work

Line Detection: Line detection is a widely studied problem in computer vision. It aims to produce vectorized line representation from images. Traditional methods such as [44, 45] detect lines based on local edge features. Recently, [41] combines the deep learning-based features with the line vectorization algorithm from [45]. Unlike the wireframe representation, traditional line detection algorithms do not provide the information about junctions and how lines and junctions are connected to each other, which limits its application in scene parsing and understanding.

Wireframe Parsing: [31] proposes the wireframe parsing task. The authors train two separate neural networks to predict junction and line heat maps from an input image. After that, the two predictions are combined using a heuristic wireframe fusion algorithm to produce the final vectorized output. Although it is intuitive and can produce reasonable

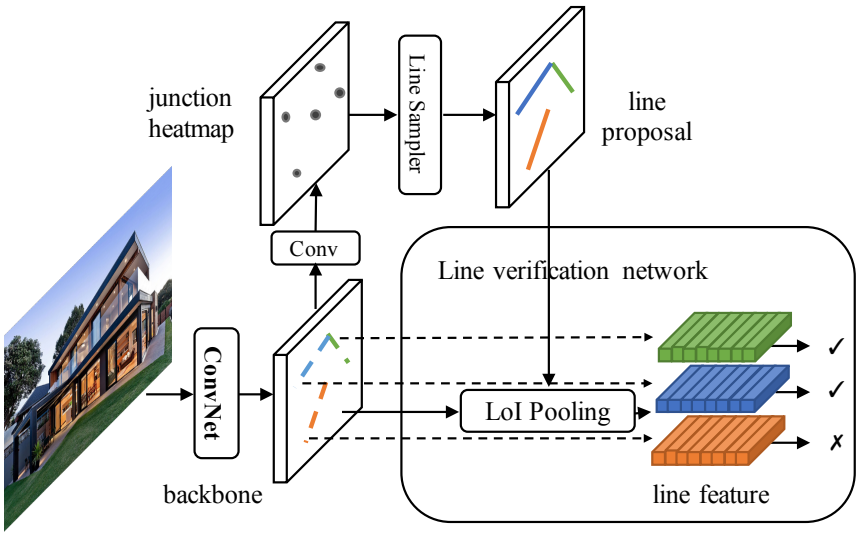


Figure 2.1: An overview of the network architecture of L-CNN.

results, such two-stage process prevents the benefits of end-to-end training. In contrast, our framework is based on a single end-to-end trainable neural network, which directly delivers a vectorized wireframe representation as the output.

Instance-level Recognition: At the technical level, our method is inspired by instance-level recognition frameworks such as Fast R-CNN [43], Faster R-CNN [46], and CornerNet [47]. Our pipeline and LoI pooling (Section 2.1.3.6) are conceptually similar to the RoI pooling in Faster R-CNN and Fast R-CNN. Both methods first generate a set of proposals and extract features to classify these proposes. The difference is that in [43, 46], the candidate proposals are generated by a sliding window fashion while our proposals are generated by connecting salient junctions (line sampler module in Section 2.1.3.5). In this sense, the proposal generation procedure is also similar to what is used in point-based object detection [47, 48]. The difference lies in how to discriminate between true lines and false positives. They use either similarity between points feature embedding [47] or the classification score in the geometric center of several salient points [48] while ours extracts features to feed into a small neural network (line verification network in Section 2.1.3.6).

2.1.3 Methods

2.1.3.1 Data Representation

Our representation of wireframes is based on the notation from graph theory. It can also be seen as a simplified version of the wireframe definition in [31]. Let $\mathsf{W} = (\mathsf{V}, \mathsf{E})$ be the wireframe of an image, in which the V is the set of junction indices and $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ is the set

of lines represented by the pair of junction endpoints in V. For each $i \in \mathsf{V}$, we use $\mathbf{p}_i \in \mathbb{R}^2$ to represent the (ground truth) coordinate of the junction i in the image space.

2.1.3.2 Overall Network Architecture

Figure 2.1 illustrates the L-CNN architecture. It contains four modules: 1) a feature extraction backbone (Section 2.1.3.3) that takes a single image as the input and provides shared intermediate feature maps for the successive modules; 2) a junction proposal module (Section 2.1.3.4) which outputs the candidate junctions; 3) a line sampling module (Section 2.1.3.5) that outputs line proposals based on the output junctions from the junction proposal module; 4) a line verification module (in Section 2.1.3.6) which classifies the proposed lines. The output of L-CNN are the positions of junctions and the connectivity matrix among those junctions. Our system is fully end-to-end trainable with stochastic gradient descent.

2.1.3.3 Backbone Network

The function of the backbone network is to extract semantically meaningful features for the successive modules of L-CNN. We choose stacked hourglass network [49] as our backbone for its efficiency and effectiveness. Input images are resized into squares. The stacked hourglass network first downsamples the input images twice in the spatial resolution via two 2-strided convolution layers. After that, learned feature maps are gradually refined by multiple U-Net-like modules [50] (the hourglass modules) with intermediate supervision imposed on the output of each module. The total loss of the network is the sum of the loss on those modules.

2.1.3.4 Junction Proposal Module

Junction Prediction. We use a simplified version of [31] to estimate the candidate junction locations in the wireframe. An input image with resolution $W \times H$ is first divided into $W_b \times H_b$ bins. For each bin, the neural network predicts whether there exists a junction inside it, and if yes, it also predicts the its relative location inside this bin. Mathematically, the neural network outputs a junction likelihood map J and an offset map $\mathbf{O}$. For each bin b, we have

$$J(b) = \begin{cases} 1 & \exists i \in \mathsf{V} : \mathbf{p}_i \in b, \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathbf{O}(b) = \begin{cases} (\mathbf{b} - \mathbf{p}_i)/W_b & \exists i \in \mathsf{V} : \mathbf{p}_i \in b, \\ 0 & \text{otherwise.} \end{cases}$$

where $\mathbf{b}$ represents the location of bin b's center and $\mathbf{p}$ represents the location of a vertex in V.

To predict J and $\mathbf{O}$, we design a network head that consists of two 1×1 convolution layers to transform the feature maps into J and $\mathbf{O}$. We treat the problem of prediction J as a classification problem and use the average binary cross entropy loss. We use ℓ_2 regression to predict the offset map $\mathbf{O}$. As the range of offset $\mathbf{O}(b)$ is bounded by $[-1/2, 1/2) \times [-1/2, 1/2)$, we append a sigmoid activation with offset -0.5 after the head to normalize the output. The loss on $\mathbf{O}$ is averaged over the bins that contain ground-truth junctions for each input image.

Non-Maximum Suppression. We apply non-maximum suppression (NMS) to remove duplicates around correct predictions. We use the same mechanism for remove blurred score map around correct predictions and get $J'(b)$ as:

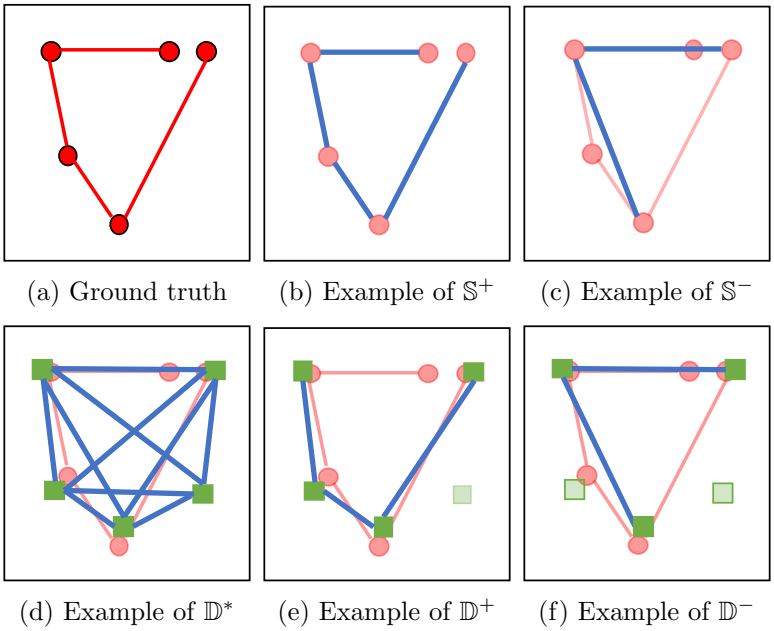
$$J'(b) = \begin{cases} J(b) & J(b) = \max_{b' \in \mathcal{N}(b)} J(b') \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathcal{N}(b)$ represents the 8 nearby bins around b. Here, we suppress the pixel values that are not the local maxima on the junction map. Such non-maximum suppression can be implemented with a max-pooling operator. The final output of the junction proposal network is the top K junction positions $\{\hat{\mathbf{p}}_i\}_{i=1}^K$ with the highest probabilities in J'.

2.1.3.5 Line Sampling Module

Given a list of K best candidate junctions $\{\hat{\mathbf{p}}_i\}_{i=1}^K$ from the junction proposal module, the purpose of the line sampling module is to generate a list of line candidates $\{L_j\}_{j=1}^M = \{(\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)\}_{j=1}^M$ during the training stage so that the line verification network can learn to predict the existence of a line. Here $\tilde{\mathbf{p}}_j^1$ and $\tilde{\mathbf{p}}_j^2$ represents the coordinates of two endpoints of the jth candidate line segment. In this task, the amount of positive samples and negatives samples are extremely unbalanced, we address this issue by carefully design the sampling mechanism as stated below.

Static Line Sampler. For each image, the static line sampler returns $N_{\mathbb{S}^+}$ positive samples and $N_{\mathbb{S}^-}$ negative samples that are directly derived from the ground truth labels. We call them static samples since they are irrelevant to the predicted candidate junction positions. Positive line samples are uniformly sampled from all the ground truth lines, denoted by $\mathbb{S}^+$, with the ground truth coordinate of the corresponding junctions. The number of total negative line samples is $O(|\mathsf{V}|^2)$, which is huge compared to the number of positive samples $O(|\mathsf{E}|)$. To alleviate the problem, we sample the negative lines from $\mathbb{S}^-$, a set of negative lines that are potentially hard to classify. We use the following heuristic to compute the $\mathbb{S}^-$: we first rasterize all the ground truth lines onto a 64×64 low-resolution bitmap. Then, for each possible connections formed by a pair of ground truth junctions that is not a ground truth line, we define its *hardness score* to be the average pixel density on the bitmap along this line. For each image, $\mathbb{S}^-$ is set to be the top 2000 lines with the highest hardness scores.



(a) Ground truth (b) Example of $\mathbb{S}^+$ (c) Example of $\mathbb{S}^-$

(d) Example of $\mathbb{D}^*$ (e) Example of $\mathbb{D}^+$ (f) Example of $\mathbb{D}^-$

Figure 2.2: Illustration of our sampling methods. Red circles represent the ground truth junctions, red lines represent the ground truth lines, green squares represent the predicted junctions, and blue lines represent the candidate lines in the static and dynamic samplers.

Dynamic Line Sampler. In contrast to the static line sampler, the dynamic line sampler samples the lines using the predicted junctions from the junction proposal module. The sampler first matches all the predicted junctions to the ground truth junction. Let $m_i := \arg\min_j \|\hat{\mathbf{p}}_i - \mathbf{p}_j\|_2$ be the index of the best matching ground truth junction for the ith junction candidates. If the ℓ_2-distance between $\hat{\mathbf{p}}_i$ and $\mathbf{p}_{m_i}$ is less than the threshold η, we say that the junction candidate $\hat{\mathbf{p}}_i$ is *matched*. For each line candidate line $(\hat{\mathbf{p}}_{i_1}, \hat{\mathbf{p}}_{i_2})$ in which $i_1, i_2 \in \{1, 2, \ldots, K\}$ and $i_1 \neq i_2$, we put it into line sets $\mathbb{D}^+$, $\mathbb{D}^-$, and $\mathbb{D}^*$ according to the following criteria:

- if both $\hat{\mathbf{p}}_{i_1}$ and $\hat{\mathbf{p}}_{i_2}$ are matched, and $(m_{i_1}, m_{i_2}) \in \mathsf{E}$, we add this line to the *positive sample set* $\mathbb{D}^+$;
- if both $\hat{\mathbf{p}}_{i_1}$ and $\hat{\mathbf{p}}_{i_2}$ are matched, and $(m_{i_1}, m_{i_2}) \in \mathbb{S}^-$, we add this line to the *hard negative sample set* $\mathbb{D}^-$;
- the *random sample set* $\mathbb{D}^*$ includes all the line candidates from the predicted junctions, regardless of their matching results.

Finally, we randomly choose $N_{\mathbb{D}^+}$ lines from the positive sample set, $N_{\mathbb{D}^-}$ lines from the hard negative sample set, $N_{\mathbb{D}^*}$ lines from the random line sample set, and return their union as the dynamic line samples.

On the one hand, the static line sampler helps cold-start the training at the beginning when there are few accurate positive samples from the dynamic sampler. It also complements the dynamic sampler by adding ground truth positive samples and hard negative samples

to help the joint training process. On the other hand, the dynamic line sampler improves the performance of line detection by adapting the line endpoints to the predicted junction locations.

2.1.3.6 Line Verification Network

The line verification network takes a list of candidate lines $\{L_j\}_{j=1}^{M} = \{(\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)\}_{j=1}^{M}$ along with the feature maps of the image from the backbone network as the input and predicts whether or not each line is in the wireframe of the scene. During training, L is computed by the line sampling modules, while during the evaluation, L is set to be every pair of the predicted junctions $\{\hat{\mathbf{p}}_i\}_{i=1}^{K}$.

For each candidate line segment $L_j = (\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)$, we feed the coordinates of its two endpoints into a *line of interest (LoI) pooling layer* (introduced below), which returns a fixed-length feature vector. Then, we pass the concatenated feature vector into a network head composed of two fully connected layers and get a logit. The loss of the line is the sigmoid binary cross entropy loss between the logit and the label of this line, i.e., a positive sample or a negative sample. To keep the loss balanced between positive and negative samples, the loss on each image for the line verification network is the sum of two separated loss, averaged over the positive lines and the negative lines, respectively.

LoI Pooling. To check whether a line segment exists in an image, we first turn the line into a feature vector. Inspired by the RoIPool and RoIAlign layers from the object detection community [11, 43, 46, 51], we propose the LoI pooling layer to extract line features while it can back-propagate the gradient to the backbone network.

Each LoI is defined by the coordinates of its two endpoints, i.e., $\tilde{\mathbf{p}}_j^1$ and $\tilde{\mathbf{p}}_j^2$. The LoI pooling layer first computes the coordinates of N_p uniform spaced middle points along the line with linear interpretation

$$\mathbf{q}_k = \frac{k}{N_p - 1}\tilde{\mathbf{p}}_j^1 + \frac{N_p - k}{N_p - 1}\tilde{\mathbf{p}}_j^2, \quad \forall k \in \{0, 1, \ldots, N_p - 1\}.$$

Then, it calculates the feature values at those N_p points in the backbone's feature map using bilinear interpretation to avoid quantization artifacts [11, 42, 52, 53]. The resulting feature vector has a spatial extent of $C \times N_p$, in which C is the channel dimension of the feature map from the backbone network. After that, the LoI Pooling layer reduces the size of the feature vector with a 1D max pooling layer. The result feature vector has shape $C \times \lceil \frac{N_p}{s} \rceil$, where s is the size of stride of the max pooling layer. This vector is then flattened and returned as the output of LoI pooling layer.

2.1.4 Experiments

2.1.4.1 Implementation Details

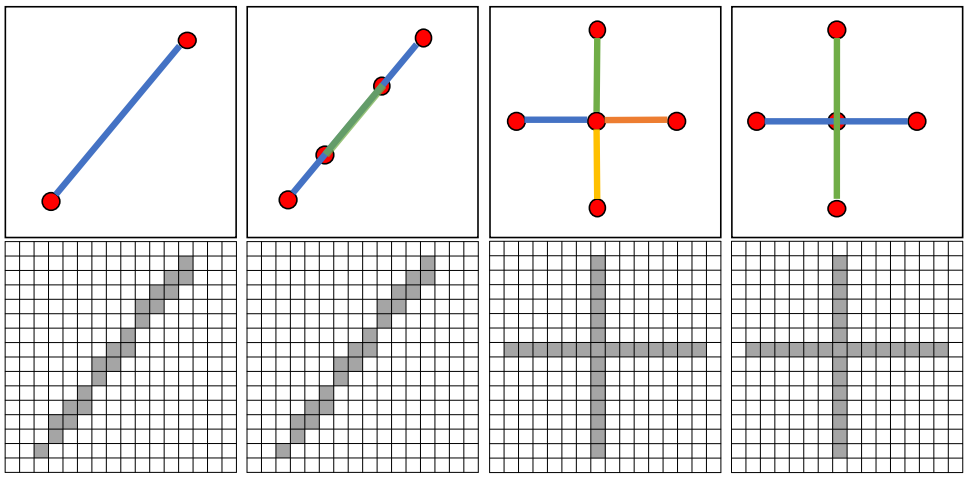
We use a stacked hourglass network [49] as our backbone. Given an input image, we first apply a 7×7 stride-2 convolution, three residual blocks with channel dimension 64, and append a stride-2 max pooling on it. Then this intermediate feature representation is fed into two stacked hourglass modules. In each hourglass, the feature maps are down-sampled with 4 stride-2 residual blocks and then up-sampled with nearest neighbour interpolation. The dimensions of both the input channel and the output channel of each residual block are 256. The network heads for J and $\mathbf{O}$ contain a 3×3 convolutional layer that reduces the number of channels to 128 with the ReLU non-linearity, followed by a 1×1 convolutional layer to match the output dimension.

We reduce the feature dimension from 256 to 128 using a 1×1 convolution kernels before feeding the feature map into the line verification network. For the LoIPool layer, we pick $N_p = 32$ points along each line as the features of the line, resulting a 128×32 feature for each line. After that, we apply a one-dimensional stride-4 max pooling to reduce the spatial dimension of line features from 32 to 8. Our final line feature has dimension 128×8. The head of the line verification network then takes the flattened feature vector and feeds it into two fully connected layers with ReLU non-linearity, in which the middle layer has 1024 neurons.

All the experiments are conducted on a single NVIDIA GTX 1080Ti GPU for neural network training. We use the ADAM optimizer [54]. The learning rate and weight decay are set to 4×10^{-4} and 1×10^{-4}, respectively. The batch size is set to 6 for maximizing the GPU memory occupancy. We train the network for 10 epochs and then decay the learning rate by 10. We stop the training at 16 epochs as the validation loss no longer decreases. The total training time is about 8 hours. All the input images are resized to $(H, W) = (512, 512)$ and we use $H_b \times W_b = 128 \times 128$ bins for J and $\mathbf{O}$. The junction proposal proposal network outputs the best $K = 300$ junctions. For the line sampling module, we use $N_{\mathbb{S}^+} = 300$, $N_{\mathbb{S}^-} = 40$, $N_{\mathbb{D}^+} = 300$, $N_{\mathbb{D}^-} = 80$, and $N_{\mathbb{D}^*} = 600$. The loss weights of multi-task learning for J, $\mathbf{O}$, and line verification network are set to 8, 0.25, and 1, respectively. Those weights are adjusted so that the magnitudes of the losses are in a similar scale.

2.1.4.2 Datasets

We conduct most of our experiments on the ShanghaiTech dataset [31]. It contains 5,462 images of man-made environments, in which 5000 images are used as the training set and 462 images are used as the testing set. The wireframe annotation of this dataset includes the positions of the salient junctions V and lines E. We also test the models trained with the ShanghaiTech dataset on the York Urban dataset [55] to evaluate the generalizability of all the methods.



(a) Overlapped lines (b) Incorrect connectivity

Figure 2.3: Demonstration of the problems in heat map-based metrics for wireframe quality evaluation. The upper part shows the detected lines and their heat maps, and the lower part shows the ground truth lines and their heat maps.

2.1.4.3 Evaluation Metric

Previously, researchers use two metrics to evaluate the quality of detected wireframes: the heat map-based AP^H to evaluate lines and junction AP to evaluate junctions. In this section, we first give a brief introduction to these metrics and discuss the reason why they are not proper for the wireframe parsing tasks. Then we give a new metric, named *structural AP*, a more reasonable way to evaluate the structural quality of wireframes.

Junction Accuracy. Junction mean average precision mAP^J evaluates the quality of junctions of a wireframe detection algorithm. For a given ranked list of predicted junction positions, a junction is considered to be correct if the ℓ^2 distance between this junction and its nearest ground-truth is within a threshold. Using this criteria, we can draw the precision recall curve by counting the number of true positive and false position. The AP is defined to be the area under this curve. The mean AP is defined to be the average of AP over difference distance thresholds. In our implementation, we choose the threshold to be 0.5, 1.0, and 2.0 threshold on 128×128 resolution.

Precision and Recall of Line Heat Maps. The precision and recall curve over line heap maps is often used to evaluate the performance of wireframe and line detection [31, 41]. Given a vectorized representation (lines or wireframes), it first generates a confidence heat map by rasterizing the lines. To compare it with the ground truth heat map, a bipartite matching that treats each pixel independently as a graph node is ran to match between two heat maps. Then precision and recall curve is computed according to the matching and

confidence of each pixel. In our experiment, we provide analysis of different methods using this metric. We show both the F-score (as in [41]) and the area under the PR curve (similar to [56]) as the quantitative measure, and write the them as F^{H} and AP^{H}, respectively.

These metrics were originally designed for evaluating boundary detection [57] and they work well for that purpose. However they are problematic in wireframe detection since

1. they do not penalize for overlapped lines (Figure 2.3a);
2. they do not properly evaluate the connectivity of the wireframe (Figure 2.3b).

For example, if a long line is broken into several short line segments, the resulted heat map is almost the same as the ground truth heat map, as shown in Figure 2.3. A good performance on the above two properties is vital for downstream tasks that rely on the correctness of line connectivity, such as inferring the 3D geometry through lines [58, 59].

Structural AP. To overcome those drawbacks, we propose a new evaluation metric defined on vectorized wireframes rather than on a heat map. We call our metric *structural average precision (sAP)*. This metric is inspired by the mean average precision commonly used in object detection [56]. Structural AP is defined to be the area under the precision recall curve computed from a scored list of the detected line segments on all test images. Recall is the proportion of the correctly detected line segments (up to a cutoff score) to all the ground truth line segments, while precision is the proportion of the correctly detected line segments above that cutoff to all the detected line segments.

A detected line segment $L_j = (\tilde{\mathbf{p}}_j^1, \tilde{\mathbf{p}}_j^2)$ is considered to be a true positive (correct) if and only if

$$\min_{(u,v)\in\mathsf{E}} \|\tilde{\mathbf{p}}_j^1 - \mathbf{p}_u\|_2^2 + \|\tilde{\mathbf{p}}_j^2 - \mathbf{p}_v\|_2^2 \le \vartheta,$$

where ϑ is a user-defined number represents the strictness of the metric. In this experiment section, we evaluate the structural AP at $\vartheta = 5$, $\vartheta = 10$, and $\vartheta = 15$ under the resolution of 128×128. We abbreviate them as sAP^5, sAP^{10}, and sAP^{15}, respectively. In addition, each ground truth line segment is not allowed to be matched more than once in order to penalize double-predicted lines. That is to say if there exists a line L_i that is ranked above the line L_j and

$$\arg\min_{(u,v)\in\mathsf{E}} \|\tilde{\mathbf{p}}_i^1 - \mathbf{p}_u\|_2^2 + \|\tilde{\mathbf{p}}_i^2 - \mathbf{p}_v\|_2^2 = \arg\min_{(u,v)\in\mathsf{E}} \|\tilde{\mathbf{p}}_j^1 - \mathbf{p}_u\|_2^2 + \|\tilde{\mathbf{p}}_j^2 - \mathbf{p}_v\|_2^2,$$

then the line L_j will always be marked as a false positive.

Junction Mean AP (mAP). The major difference between line detection and wireframe detection is that the wireframe representation encodes junction positions. Junctions have physical meaning in 3D (corners or occlusional points) and encodes the line connectivity information. Our junction mean AP ($\mathrm{mAP}^{\mathrm{J}}$) evaluates the quality of vectorized junctions of a wireframe detection algorithm without relying on heat maps as in [31]. To better understand the advantage of explicitly modeling junctions, we also evaluate our method using the junction mAP as described below: for a given ranked list of predicted junction

	sampler					head		metric		
	$\mathbb{D}^*$	$\mathbb{S}^+$	$\mathbb{S}^-$	$\mathbb{D}^+$	$\mathbb{D}^-$	fc+fc	conv+fc	sAP^5	sAP^{10}	sAP^{15}
(a)	✓					✓		43.7	48.2	50.2
(b)		✓	✓			✓		38.5	41.9	43.8
(c)	✓	✓	✓			✓		47.8	51.7	53.6
(d)	✓	✓	✓	✓	✓		✓	55.7	59.8	61.7
(e)	✓	✓		✓		✓		57.4	61.4	63.2
(f)	✓	✓	✓	✓	✓	✓		**58.9**	**62.9**	**64.7**

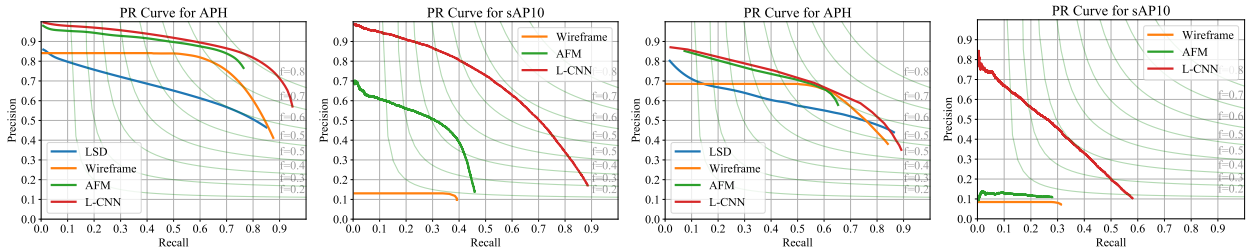
Table 2.1: Ablation study of L-CNN. The columns labeled with "sampler" represent whether a specific sampler is used during the training stage, as introduced in Section 2.1.3.5. The columns labeled with "head" represent the network head structured used in the line verification network. "fc + fc" is the network structure introduced in Section 2.1.3.6, while in "conv + fc" we replace the middle fully connected layer with a 1D Bottleneck layer [10].

positions, a junction is considered to be correct if the ℓ^2 distance between this junction and its nearest ground-truth is within a threshold. Each ground truth junction is only allowed to be matched once to penalize double-predicted junctions. Using this criteria, we can draw the precision recall curve by counting the number of true and false positives. The junction AP is defined to be the area under this curve. The mean junction AP is defined to be the average of junction AP over difference distance thresholds. In our implementation, we choose to average over 0.5, 1.0, and 2.0 thresholds under 128×128 resolution.

2.1.4.4 Ablation Study

In this section, we run a series of ablation experiments on the ShanghaiTech dataset [31] to study our proposed method. We use our structural average precision (sAP) as the evaluation metrics. The results are shown in Table 2.1.

Line Sampling Modules: We compare different design choices for line sampling modules, as shown in Table 2.1. (a) uses just the random pairs from the dynamic sampler. The sAP^5 is 43.7, which serves as the baseline. (b) only uses the sampled pairs from ground-truth junctions and get much worse performance. The performance gap is even larger when the evaluation criterion is loose. This is because (b) does not consider the imperfect of junction prediction map and cannot tackle when junction is slightly misaligned with the ground truth. After that, we combine the random dynamic sampling and the static sampling, which significantly improves the performance, as shown in Table 2.1 (c). Then we add dynamic sampler candidate $\mathbb{D}^+$ and $\mathbb{D}^-$, which leads to the best sAP^5 score 58.9 in (f). This experiment indicates that the carefully selected dynamic line candidates are vital to a good performance. Lastly, by comparing (e) and (f), we find that including hard examples $\mathbb{S}^-$ and $\mathbb{D}^-$ is indeed helpful compared to just doing the random sampling in $\mathbb{D}^*$.



(a) Heat map PR curves on ShanghaiTech [31].
(b) Structural PR curves on ShanghaiTech [31].
(c) Heat map PR curves on York Urban [55].
(d) Structural PR curves on York Urban [55].

Figure 2.4: Precision recall curves on wireframe parsing. Methods are trained on the ShanghaiTech dataset.

	ShanghaiTech [31]				York Urban [55]			
	sAP^{10}	mAP^{J}	AP^{H}	F^{H}	sAP^{10}	mAP^{J}	AP^{H}	F^{H}
LSD	/	/	52.0	61.0	/	/	51.0	60.0
Wireframe	5.1	40.9	67.8	72.6	2.6	13.4	53.4	63.7
AFM	24.4	23.3	69.5	77.2	3.5	6.9	48.4	63.1
L-CNN	**62.9**	**59.3**	**82.8**	**81.2**	**26.4**	**30.4**	**59.8**	**65.4**

Table 2.2: Performance comparison on wireframe parsing. All the models are trained on the ShanghaiTech dataset and evaluate on both datasets. The columns labelled with "sAP" show the line accuracy with respect to our structural metrics; the columns labelled with "mAP^{J}" shows the mean average precision of the predicted junctions; the columns labelled with "F^{H}" and "AP^{H}" shows the performance metrics related to heat map-based PR curves. Our method L-CNN has the state-of-the-art performance on all of the evaluation metrics.

Line Verification Networks: Table 2.1 also shows our ablation on how to design the line verification network. We tried two different designs: In Table 2.1 (e), we apply two fully-connected layers after the LoI Pooling feature to get the classification results, while in Table 2.1 (d) we firstly apply a 1D convolution on the features and then use the fully-connected layer on the flattened feature vector to get the final line classification. Experiments show that using convolution largely deteriorates the performance. We hypothesis that this is because line classification requires location sensitivity, which the translation-invariant convolution cannot provide [60, 61].

2.1.4.5 Comparison with Other Methods

Following the practice of [31, 41], we compare our method with LSD [45], deep learning-based line detectors [41], as well as wireframe parser from the ShanghaiTech dataset paper [31]. F^{H}, AP^{H}, and sAP with different thresholds are used to evaluate those methods quan-

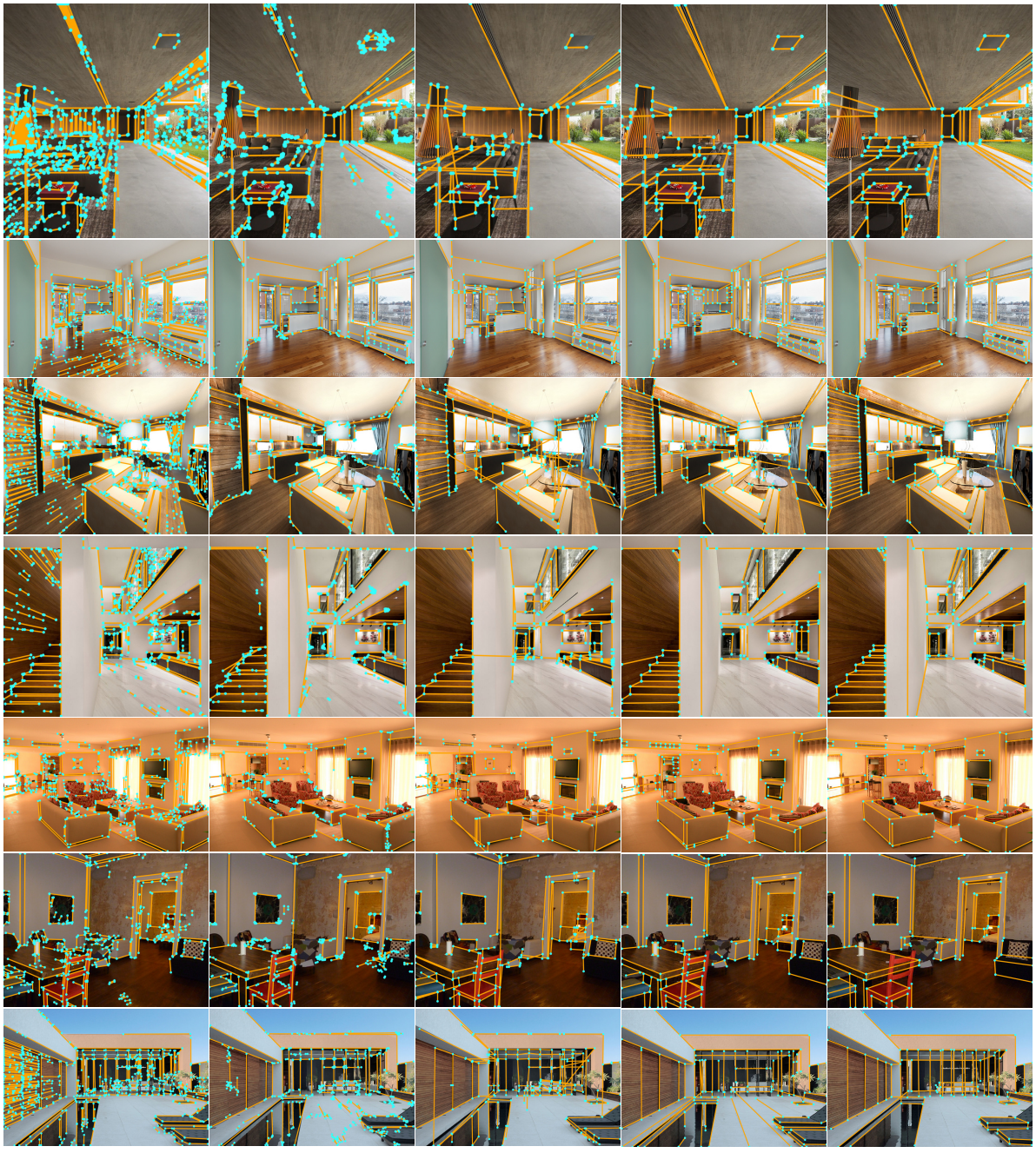
titatively. All the models are trained on the ShanghaiTech dataset and evaluate on both of the ShanghaiTech [31] and York Urban datasets [55]. The results are shown in Table 2.2 and Figure 2.4. We note that the difference of the numbers and curves for AP^H from [31, 41] is due to our more proper implementation of AP^H: 1) In the code provided by [31], they evaluate the precision and recall per image and average them together, while we first sum the number of true positives and false positives over the dataset and then compute the precision and recall. 2) Due to the insufficient number of thresholds, the PR curves in [31, 41] do not cover all the recall that an algorithm can achieve. We evaluate all the methods on more thresholds to extend the curves as long as possible.

Figure 2.4a shows that our algorithm is better than the state-of-the-art line detector methods under the PR curve of heat map-based line metrics, especially in the high-recall region. This indicates that our method can find more correct lines compared to other methods. We also quantitatively calculate the F-score and the average AP. Table 2.2 shows that our algorithm performs significantly better than previous state-of-the-art line detectors by 13.3 points in AP^H and 4.0 points in F^H [41]. We also want to emphasize that compared to line detection, it is conceptually harder for the wireframe detection methods to reach the same performance as the line detection methods in term of the heat map-based metrics. This is because a wireframe detection algorithm requires the positions of junctions, the endpoints of lines, to be correct, while a line detector can start and end a line arbitrarily to "fill" the line heat map. Before evaluating the heat map-based metrics, we post process the lines from L-CNN to remove the overlap.

We then evaluate all the methods with our proposed structural AP. The precision recall curve is shown in Figure 2.4b (LSD is missing here as its scores are too low to be drawn). The gap between our method and previous methods is even larger. Our method achieves 40-point sAP improvement over the previous state-of-the-art method. This is because our line verification network penalizes incorrect structures, while methods such as AFM and wireframe parser use a hand-craft algorithm to extract lines from heat maps, in which the information of junction connectivity gets lost. Furthermore, the authors of [31] mention that their vectorization algorithm will break lines and add junctions to better fit the predicted heat map. Such behaviors can worsen the structure correctness, which might explain its low sAP score.

The mAP^J results are shown in Table 2.2. For AFM, we treat the endpoints of each line as junctions and use the line NFA score as the score of its endpoints. We note that the inferior junction quality of AFM is not because their method is not well-designed but the end task is different. This shows that one cannot directly apply a line detection algorithm on the wireframe parsing task. In addition, our L-CNN outperforms the previous wireframe parser [31] by a large margin due to the joint training process of the pipeline.

Table 2.2 and Figures 2.4c and 2.4d show that L-CNN also performs the best among all the wireframe and line detection methods when testing on a different dataset [55] without finetune. This indicates that our method is able to generalize to novel scenes and data. We note that the relatively low sAP scores are due to the duplicated lines, texture lines, while missing many long lines in the annotation of the dataset.



(a) LSD (b) AFM (c) Wireframe (d) **L-CNN** (e) GT

Figure 2.5: Qualitative comparison of wireframe and line detection methods. From left to right, the columns shows the results from LSD [45], AFM [41], Wireframe [31], L-CNN (ours), and the ground truth. We also draw the detected junctions from Wireframe and L-CNN and the line endpoints from LSD and AFM.

2.1.4.6 Visualization

We visualize our algorithm's output in Figure 2.5. The junctions are marked cyan blue and lines are marked orange. Since LSD and AFM do not explicitly output junctions, we treat the endpoints of lines as junctions. As shown in Figure 2.5, LSD detects some high-frequency textures without semantic meaning. This is expected as LSD is not a data-driven method. By training a CNN to predict line heat maps, AFM [41] is able to suppress some noise. However, both LSD and AFM still produce a lot of short lines because they do not have an explicit notion of junctions. The wireframe parser [31] utilizes junctions to provide a relatively cleaner result, but their heuristic vectorization algorithm is sub-optimal and leads to crossing lines and incorrectly connected junctions. In contrast, our L-CNN uses powerful neural networks to classify whether a line indeed exists and thus provides the best performance.

2.2 Learning to Detect Vanishing Points

2.2.1 Introduction

Vanishing point detection is a classic and important problem in 3D vision. Given the camera calibration, vanishing points give us the direction of 3D lines, and thus let us infer 3D information of the scene from a single 2D image. A robust and accurate vanishing point detection algorithm enables and enhances applications such as camera calibration [62], 3D reconstruction [63], photo forensics [64], object detection [65], wireframe parsing [35, 59], and autonomous driving [66].

Although there has been a lot of work on this seemingly basic vision problem, no solution seems to be quite satisfactory yet. Traditional methods [67] usually first use edge/line detectors to extract straight lines and then cluster them into multiple groups. Many recent methods have proposed to improve the detection by training deep neural networks with labeled data. However, such neural networks often offer only a coarse estimate for the position of vanishing points [68] or horizontal lines [69]. The output is usually a component of a multi-stage system and used as an initialization to remove outliers from line clustering. Arguably the main reason for neural networks' poor precision in vanishing point detection (compared to line clustering-based methods) is likely because existing neural network architectures are not designed to represent or learn the special geometric properties of vanishing points and their relations to structural lines.

To address this issue, we propose a new convolutional neural network, called *Neural Vanishing Point Scanner (NeurVPS)*, that explicitly encodes and hence exploits the global geometric information of vanishing points and can be trained in an end-to-end manner to both robustly and accurately predict vanishing points. Our method samples a sufficient number of point candidates and the network then determines which of them are valid. A common criterion of a valid vanishing point is whether it lies on the intersection of a sufficient number of structural lines. Therefore, the role of our network is to measure the intensity of

the signals of the structural lines passing through the candidate point. Although this notion is simple and clear, it is a challenging task for neural networks to learn such geometric concept since the relationship between the candidate point and structural lines not only depend on global line orientations but also their pixel locations. In this work, we identify a *canonical conic space* in which this relationship only depends on local line orientations. For each pixel, we define this space as a local coordinate system in which the x-axis is chosen to be the direction from the pixel to the candidate point, so the associated structural lines in this space are always horizontal.

We propose a *conic convolution operator*, which applies regular convolution for each pixel in this conic space. This is similar to apply regular convolutions on a rectified image where the related structural lines are transformed into horizontal lines. Therefore the network can determine how to use the signals based on local orientations. In addition, feature aggregation in this rectified image also becomes geometrically meaningful, since horizontal aggregation in the rectified image is identical to feature aggregation along the structural lines.

Based on the canonical space and the conic convolution operator, we are able to design the convolutional neural network that accurately predicts the vanishing points. We conduct extensive experiments and show the improvement on both synthetic and real-world datasets. With the ablation studies, we verify the importance of the proposed conic convolution operator.

2.2.2 Related Work

Vanishing Point Detection. Vanishing point detection is a fundamental and yet surprisingly challenging problem in computer vision. Since initially proposed by [26], researchers have been trying to tackle this problem from different perspectives. Early researches estimate vanishing points using sphere geometry [26, 70, 71], hierarchical Hough transformation [72], or the EM algorithms [73, 74]. Researches such as [75, 76, 77, 78] use the Manhattan world assumptions [79] to improve the accuracy and the reliability of the detection. [80] extends the mutual orthogonality assumption to a set of mutual orthogonal vanishing point assumption (Atlanta world [81]).

The dominant approach is line-based vanishing point detection algorithms, which are often divided into several stages. Firstly, a set of lines are detected [45, 82]. Then a line clustering algorithm [83] are used to propose several guesses of target vanishing point position based on geometric cues. The clustering methods include RANSAC [84], J-linkage [67], Hough transform [85], or EM [73, 74]. [86] uses contour detection and J-linkage in natural scenes but only one dominate vanishing point can be detected. Our method does not rely on existing line detectors, and it can automatically learn the line features in the conic space to predict any number of vanishing points from an image.

Recently, with the help of convolutional neural networks, the vision community has tried to tackle the problem from a data-driven and supervised learning approach. [87, 88, 89] formulate the vanishing point detection as a patch classification problem. They can only detect vanishing points within the image frame. Our method does not have such limitation.

[69] detects vanishing points by first estimating horizontal vanishing line candidates and score them by the vanishing points they go through. They use an ImageNet pre-trained neural network that is fine-tuned on Google street images. [68] uses inverse gnomonic image and regresses the sphere image representation of vanishing point. Both work rely on traditional line detection algorithms while our method learns it implicitly in the conic space.

Structured Convolution Operators. Recently more and more operators are proposed to model spatial and geometric properties in images. For instance the wavelets based scattering networks (ScatNet) [90, 91] are introduced to ensure certain transform (say translational) invariance of the network. [52] first explores geometric deformation with modern neural networks. [53, 92] modify the parameterization of the global deformable transformation into local convolution operators to improve the performance on image classification, object detection, and semantic segmentation. While these methods allow the network to learn about the space where the convolution operates on, we here explicitly define the space from first principle and exploit its geometric information. Our method is similar to [52] in the sense that we both want to rectify input to a canonical space. The difference is that they learn a global rectification transformation while our transformation is local. Different from [53, 92], our convolutional kernel shape is not learned but designed according to the desired geometric property.

Guided design of convolution kernels in canonical space is well practiced for irregular data. For spherical images, [93] designs operators for rotation-invariant features, while [94] operates in the space defined by longitude and latitude, which is more meaningful for climate data. In 3D vision, geodesic CNN [95] adopts mesh convolution with the spherical coordinate, while TextureNet [96] operates in a canonical space defined by globally smoothed principal directions. Although we are dealing with regular images, we observe a strong correlation between the vanishing point and the conic space, where the conic operator is more effective than regular 2D convolution.

2.2.3 Methods

Figure 2.7 illustrates the overall structure of our NeurVPS network. Taken an image and a vanishing point as input, our network predicts the probability of a candidate being near a ground-truth vanishing point. Our network has two parts: a backbone feature extraction network and a conic convolution sub-network. The backbone is a conventional CNN that extracts semantic features from images. We use a single-stack hourglass network [49] for its ability to possess a large receptive field while maintaining fine spatial details. The conic convolutional network (Section 2.2.3.3) takes feature maps from the backbone as input and determines the existence of vanishing points around candidate positions (as a classification problem). The conic convolution operators (Section 2.2.3.2) exploit the geometric priors of vanishing points, and thus allow our algorithm to achieve superior performance without resorting to line detectors. Our system is end-to-end trainable.

Due to the classification nature of our model, we need to sample enough number of candidate points during inference. It is computationally infeasible to directly sample suffi-

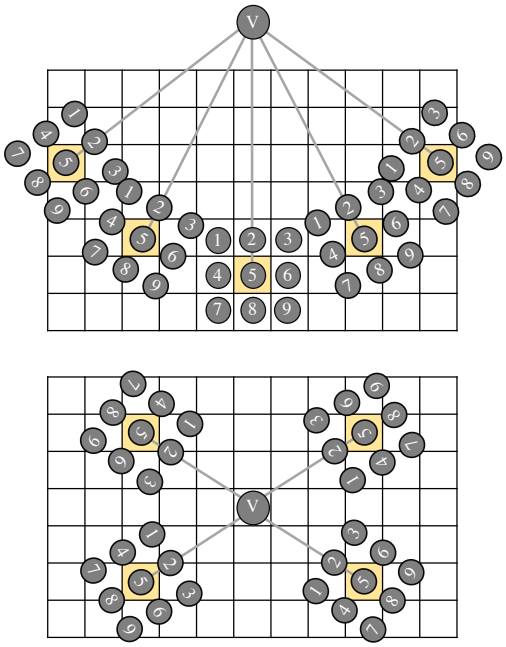


Figure 2.6: Illustration of sampled locations of 3×3 conic convolution. The bright yellow region is the output pixel and $\mathbf{v}$ stands for the vanishing point. Upper and lower figures illustrate the cases when the vanishing point is outside and inside the image, respectively.

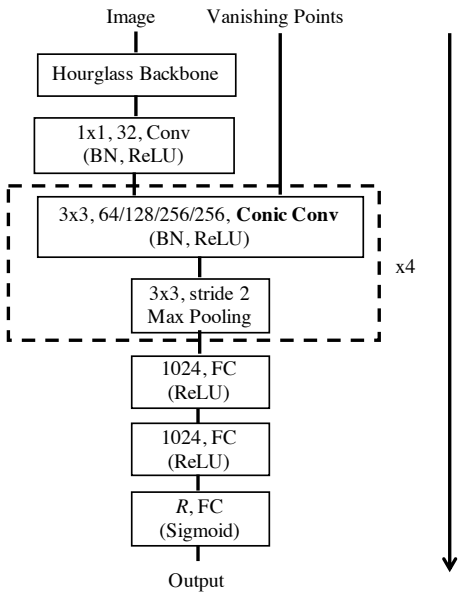


Figure 2.7: Illustration of the network structure of NeurVPS. The number of each convolutional block is the kernel size and output dimension respectively. The number of fully connected layer block is the output dimension. The kernel size of Max Pooling layer is 3 and stride is 2. Batch normalization and ReLU activation are appended after each conv and fc layer.

ciently dense candidates. Therefore, we use a coarse-to-fine approach (Section 2.2.3.4). We first sample $N_{\mathbf{d}}$ points on the unit sphere and calculate their likelihoods of being the line direction (Section 2.2.3.1) of a vanishing point using the trained neural network classifier. We then pick the top K candidates and sample another $N_{\mathbf{d}}$ points around each of their neighbours. This step is repeated until we reach the desired resolution.

2.2.3.1 Basic Geometry and Representations of Vanishing Points

The position of a vanishing point encodes the line 3D direction. For a 3D ray described by $\mathbf{o} + \lambda\mathbf{d}$ in which $\mathbf{o}$ is its origin and $\mathbf{d}$ is its unit direction vector, its 2D projection on the image is

$$z \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f & 0 & c_x \\ & f & c_y \\ & & 1 \end{bmatrix}}_{\mathbf{K}} \cdot (\mathbf{o} + \lambda\mathbf{d}), \tag{2.1}$$

where p_x and p_y are the coordinates in the image space, z is the depth in the camera space, $\mathbf{K}$ is the calibration matrix, f is the focal length, and $[c_x, c_y]^T \in \mathbb{R}^2$ is the optical center of the camera. The vanishing point is the point with $\lambda \to \infty$, whose image coordinate is $\mathbf{v} = [v_x, v_y]^T := \lim_{\lambda\to\infty}[p_x, p_y]^T \in \mathbb{R}^2$. We can then derive the 3D direction of a line in term of its vanishing point:

$$\mathbf{d} = \begin{bmatrix} v_x - c_x & v_y - c_y & f \end{bmatrix}^T \in \mathbb{R}^3. \tag{2.2}$$

In the literature, a normalized line direction vector $\mathbf{d}$ is also called the *Gaussian sphere representation* [26] of the vanishing point $\mathbf{v}$. The usage of $\mathbf{d}$ instead of $\mathbf{v}$ avoids the degenerated cases when $\mathbf{d}$ is parallel to the image plane. It also gives a natural metric that defines the distance between two vanishing points, the angle between their normalized line direction vectors: $\arccos|\mathbf{d}_i^T\mathbf{d}_j|$ for two unit line directions $\mathbf{d}_i, \mathbf{d}_j \in \mathbb{S}^2$. Finally, sampling vanishing points with the Gaussian sphere representation is easy, as it is equivalent to sampling on a unit sphere, while it remains ambiguous how to sample vanishing points directly in the image plane.

2.2.3.2 Conic Convolution Operators in Conic Space

In order for the network to effectively learn vanishing point related line features, we want to apply convolutions in the space where related lines can be determined locally. We define the *conic space* for each pixel in the image domain as a rotated regular local coordinate system where the x-axis is the direction from the pixel to the vanishing point. In this space, related lines can be identified locally by checking whether its orientation is horizontal. Accordingly, we propose a new convolution operator, named *conic convolution*, which applies the regular convolution in this conic space. This operator effectively encodes global geometric cues for classifying whether a candidate point (Section 2.2.3.5) is a valid vanishing point. Figure 2.6 illustrates how this operator works.

A 3×3 conic convolution takes the input feature map $\mathbf{x}$ and the coordinate of convolution center $\mathbf{v}$ (the position candidates of vanishing points) and outputs the feature map $\mathbf{y}$ with the same resolution. The output feature map $\mathbf{y}$ can be computed with

$$\mathbf{y}(\mathbf{p}) = \sum_{\delta x=-1}^{1}\sum_{\delta y=-1}^{1} \mathbf{w}(\delta x, \delta y)\cdot\mathbf{x}(\mathbf{p} + \delta x\cdot\mathbf{t} + \delta y\cdot\mathbf{R}_{\frac{\pi}{2}}\mathbf{t}), \text{ where } \mathbf{t} := \frac{\mathbf{v}-\mathbf{p}}{\|\mathbf{v}-\mathbf{p}\|_2} \in \mathbb{R}^2. \tag{2.3}$$

Here $\mathbf{p} \in \mathbb{R}^2$ is the coordinates of the output pixel, $\mathbf{w}$ is a 3×3 trainable convolution filter, $\mathbf{R}_{\frac{\pi}{2}} \in \mathbb{R}^{2\times2}$ is the rotational matrix that rotates a 2D vector by 90° counterclockwise, and $\mathbf{t}$ is the normalized direction vector that points from the output pixel $\mathbf{p}$ to the convolution center $\mathbf{v}$. We use bilinear interpolation to access values of $\mathbf{x}$ at non-integer coordinates.

Intuitively, conic convolution makes edge detection easier and more accurate. An ordinary convolution may need hundreds of filters to recognize edge with different orientations, while conic convolution requires much less filters to recognize edges aligning with the candidate vanishing point because filters are firstly rotated towards the vanishing point. The

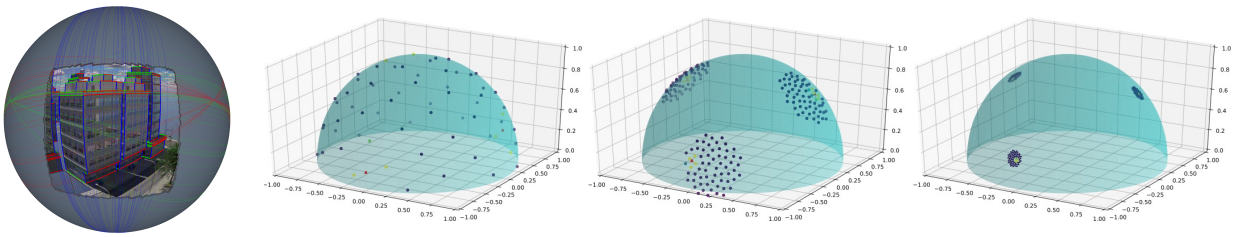


Figure 2.8: Illustration of vanishing points' Gaussian sphere representation of an image from the SU3 wireframe dataset [59] and our multi-resolution sampling procedure in the coarse-to-fine inference. In the right three figures, the red triangles represent the ground truth vanishing points and the dots represent the sampled locations.

strong/weak response (depends on the candidate is positive/negative) will then be aggregated by subsequent fully-connected layers.

2.2.3.3 Conic Convolutional Network

The conic convolutional network is a classifier that takes the image feature map $\mathbf{x}$ and a candidate vanishing point position $\hat{\mathbf{v}}$ as input. For each angle threshold $\gamma \in \Gamma$, the network predicts whether there exists a real vanishing point $\mathbf{v}$ in the image so that the angle between the 3D line directions $\mathbf{v}$ and $\hat{\mathbf{v}}$ is less than the threshold γ. The choice of Γ will be discussed in Section 2.2.3.4.

Figure 2.7 shows the structure diagram of the proposed conic convolutional network. We first reduce the dimension for the feature map from the backbone to save the GPU memory footprint with an 1×1 convolution layer. Then 4 consecutive conic convolution (with ReLU activation) and max-pooling layers are applied to capture the geometric information at different spatial resolutions. The channel dimension is increased by a factor of two in each layer to compensate the reduced spatial resolution. After that, we flatten the feature map and use two fully connected layers to aggregate the features. Finally, a sigmoid classifier with binary cross entropy loss is applied on top of the feature to discriminate positive and negative samples with respect to different thresholds from Γ.

2.2.3.4 Coarse-to-fine Inference

With the backbone and the conic convolutional network, we can compute the probability of vanishing point over the hemisphere of the unit line direction vector $\hat{\mathbf{d}} \in \mathbb{S}^2$, as shown in Figure 2.8. We utilize a multi-resolution strategy to quickly pinpoint the location of the vanishing points. We use R rounds to search for the vanishing points. In the r-th round, we uniformly sample $N_{\mathbf{d}}$ line direction vectors on the surface of the unit spherical cap

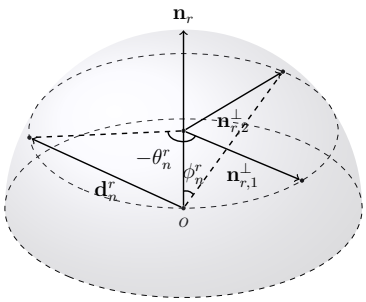


Figure 2.9: Illustration of the variables used in uniform spherical cap sampling.

with direction $\mathbf{n}_r$ and polar angle γ_r using the Fibonacci
lattice [97]. Mathematically, the n-th sampled line direction vector can be written as

$$\begin{aligned}
\mathbf{d}_n^r &= \cos\phi_n^r \mathbf{n}_r + \sin\phi_n^r(\cos\theta_n^r \mathbf{n}_{r,1}^\perp + \sin\theta_n^r \mathbf{n}_{r,2}^\perp),\\
\phi_n^r &:= \arccos\big(1 + (\cos\alpha_r - 1) * n/N_{\mathbf{d}}\big),\\
\theta_n^r &:= (1+\sqrt{5})\pi n,
\end{aligned}$$

in which $\mathbf{n}_{r,1}^\perp$ and $\mathbf{n}_{r,2}^\perp$ are two arbitrary orthogonal unit vectors that are perpendicular to $\mathbf{n}_r$, as shown in Figure 2.9. We initialize $\mathbf{n}_1 \leftarrow (0,0,1)$ and $\gamma_1 \leftarrow \pi$. For the round $r+1$, we set the threshold $\gamma_{r+1} \leftarrow \rho \max_{\mathbf{w}\in\mathbb{S}^2} \min_n \arccos|\langle \mathbf{w}, \mathbf{d}_n^r\rangle|$ and $\mathbf{n}_{r+1}$ to the $\mathbf{d}_n^r$ whose vanishing point obtains the best score from the conic convolutional network classifier with angle threshold $\gamma = \gamma_{r+1}$. Here, ρ is a hyperparameter controlling the distance between two nearby spherical caps. Therefore, we set the threshold set Γ in Section 2.2.3.2 to be $\{\gamma_{r+1} \mid r \in \{1,2,\ldots,R\}\}$ accordingly.

The above process detects a single dominant vanishing point in a given image. To search for more than one vanishing point, one can modify the first round to find the best K line directions and use the same process for each line direction in the remaining rounds.

2.2.3.5 Vanishing Point Sampling for Training

During training, we need to generate positive samples and negative samples. For each ground-truth vanishing point with line direction $\mathbf{d}$ and threshold γ, we sample N^+ positive vanishing points and N^- negative vanishing points. The positive vanishing points are uniformly sampled from $\mathcal{S}^+ = \{\mathbf{w} \mid \mathbf{w} \in \mathbb{S}^2 : \arccos|\langle \mathbf{w}, \mathbf{d}\rangle| < \gamma\}$ and the negative vanishing points are uniformly sampled from $\mathcal{S}^- = \{\mathbf{w} \mid \mathbf{w} \in \mathbb{S}^2 : \gamma < \arccos|\langle \mathbf{w}, \mathbf{d}\rangle| < 2\gamma\}$. In addition, we sample N^* random vanishing points for each image to reduce the sampling bias. The line directions of those vanishing points are uniformly sampled from the unit hemisphere.

2.2.4 Experiments

2.2.4.1 Datasets and Metrics

We conduct experiments on both synthetic [59] and real-world [86, 98] datasets.

Natural Scene [86]. This dataset contains images of natural scenes from AVA and Flickr. The authors pick the images that contain only one dominating vanishing point and label their locations. There are 2,275 images in the dataset. We divide them into 2,000 training images and 275 test images randomly. Because this dataset does not contain the camera calibration information, we set the focal length to the half of the sensor width for vanishing point sampling and evaluation. Such focal length simulates the wide-angle lens used in landscape photography.

ScanNet [98]. ScanNet is a 3D indoor environment dataset with reconstructed meshes and RGB images captured by mobile devices. For each scene, we find the three orthogonal principal directions for each scene which align with most of the surface normals and use them to compute the vanishing points for each RGB image. We split the dataset as suggested by ScanNet v2 tasks, and train the network to predict the three vanishing points given the RGB image. There are 266,844 training images. We randomly sample 500 images from the validation set as our test set.

SU3 Wireframe [59]. The "ground-truth" vanishing point positions in real world datasets are often inaccurate. To systematically evaluate the performance of our algorithm, we test our method on the recent synthetic SceneCity Urban 3D (SU3) wireframe dataset [59]. This photo-realistic dataset is created with a procedural building generator, in which the vanishing points are directly computed from the CAD models of the buildings. It contains 22,500 training images and 500 validation images.

Evaluation Metrics. Previous methods usually use horizon detection accuracy [69, 80, 99] or pixel consistency [86] to evaluate their method. These metrics are indirect for this task. To better understand the performance of our algorithm, we propose a new metric, called *angle accuracy* (AA). For each vanishing point from the predictions, we calculate the angle between the ground-truth and the predicted one. Then we count the percentage of predictions whose angle difference is within a pre-defined threshold. By varying different thresholds, we can plot the *angle accuracy curves*. AA^{θ} is defined as the area under the curve between $[0, \theta]$ divided by θ. In our experiments, the upper bound θ is set to be 0.2°, 0.5°, and 1.0° on the synthetic dataset and 1°, 2°, and 10° on the real world dataset. Two angle accuracy curves (coarse and fine level) are plotted for each dataset. Our metric is able to show the algorithm performance under different precision requirements. For a fair comparison, we also report the performance in pixel consistency as in the dataset paper [86].

2.2.4.2 Implementation Details

We implement the conic convolution operator in PyTorch by modifying the "im2col + GEMM" function according to Equation (2.3), similar to the method used in [53]. Input images are resized to 512×512. During training, the Adam optimizer [54] is used. Learning rate and weight decay are set to be 4×10^{-4} and 1×10^{-5}, respectively. All experiments are conducted on two NVIDIA RTX 2080Ti GPUs, with each GPU holding 6 mini-batches. For synthetic data [59], we train 30 epochs and reduce the learning rate by 10 at the 24-th epoch. We use $\rho = 1.2$, $N^{+} = N^{-} = 1$ and $N^{*} = 3$. For the Natural Scene dataset, we train the model for 100 epochs and decay the learning rate at 60-th epoch. For ScanNet [98], we train the model for 3 epochs. We augment the data with horizontal flip. We set $N_{\mathbf{d}} = 64$ and use $R_{\text{SU3}} = 5$, $R_{\text{NS}} = 4$, and $R_{\text{SN}} = 3$ in the coarse-to-fine inference for the SU3 dataset, the Natural Scene dataset, and the ScanNet dataset, respectively. During inference, the results from the backbone network can be shared so only the conic convolution layers need

	$AA^{0.2°}$	$AA^{0.5°}$	$AA^{1.0°}$	mean	median
LSD [100]	27.9	47.9	61.5	3.89°	0.21°
REG	2.2	6.5	15.0	2.07°	1.48°
CLS	2.2	9.1	23.7	1.77°	0.99°
Conic×2	10.5	28.9	50.3	0.78°	0.43°
Conic×4	47.5	**74.2**	**86.3**	0.15°	0.09°
Conic×6	**49.1**	74.0	86.2	**0.14°**	**0.09°**

Table 2.3: Ablation study of NeurVPS. "REG" denotes the baseline that directly regress line direction in the camera space. "CLS" denotes the baseline that do vanishing point classification using image feature and its coordinate. Conic×K denotes our methods with varying number of conic convolution layers.

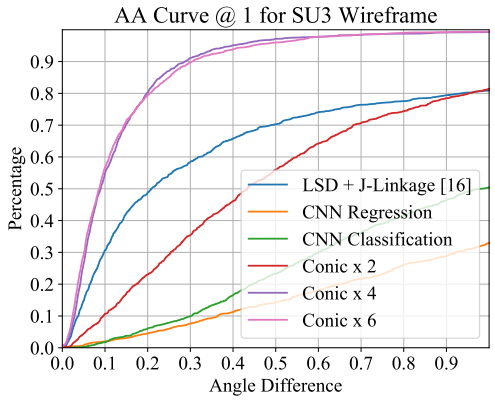
to be forwarded multiple times. Using the Nature Scene dataset as an example, we conduct 4 rounds of coarse-to-fine inference, in each of which we sample 64 vanishing points. So we forward the conic convolution part 256 times for each image during testing. The evaluation speed is about 1.5 vanishing points per second on a single GPU.

2.2.4.3 Ablation Studies on Synthetic Datasets

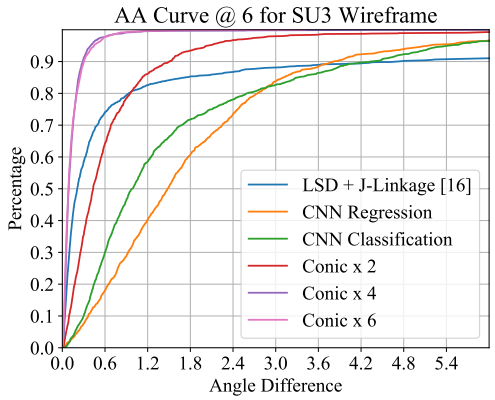
Comparison with Baseline Methods. We compare our method with both traditional line detection based methods and neural network based methods. The sample images and results can be found in Figure 2.8. For line-based algorithms, the LSD with J-linkage clustering [45, 67, 100] probably is the most widely used method for vanishing point detection. Note that LSD is a strong competitor on the synthetic SU3 dataset as the images contain many sharp edges and long lines.

We aim to compare pure neural network methods that only rely on raw pixels. Existing methods such as [55, 87, 88] can only detect vanishing points inside images. [68, 69] rely on an external line map as initial inputs. To the best of our knowledge, there is no existing pure neural network methods that are general enough to handle our case. Therefore, we propose two intuitive baselines. The first baseline, called REG, is a neural network that direct regresses value of $\mathbf{d}$ using chamfer-ℓ^2 loss, similar to [59]. We change all the conic convolutions to traditional 2D convolutions to make the numbers of parameters be the same. The second baseline, called CLS, uses our fine-to-coarse classification approach. We change all the conic convolutions to their traditional counterparts, and concatenate $\mathbf{d}$ to the feature map right before feeding it to the NeurVPS head to make the neural network aware of the position of vanishing points.

The results are shown in Table 2.3 and Figure 2.10. By utilizing the geometric priors and large-scale training data, our method significantly outperforms other baselines across all the metrics. We note that, compared to LSD, neural network baselines perform better in terms of mean error but much worse for AA. This is because line-based methods are generally

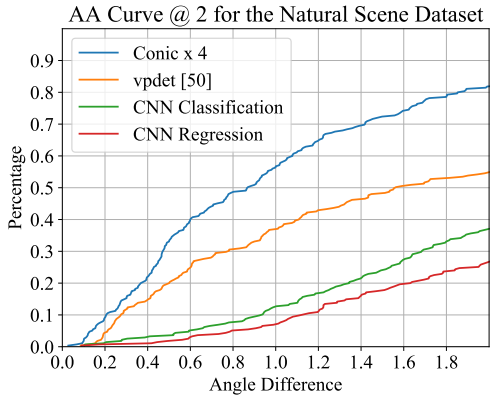


(a) Angle difference ranges from 0° to 1°.

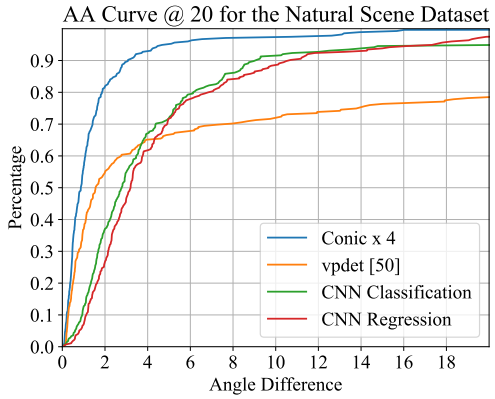


(b) Angle difference ranges from 0° to 6°.

Figure 2.10: Angle accuracy curves for different methods on the SU3 wireframe dataset [59].

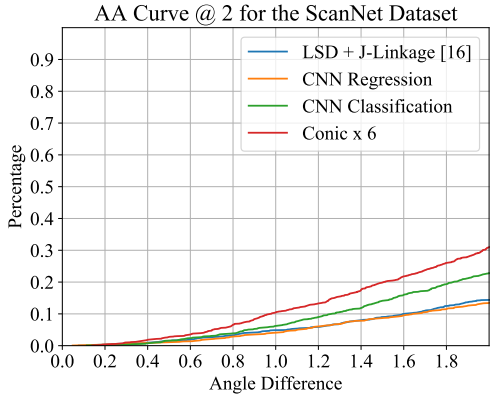


(a) Angle difference ranges from 0° to 2°.

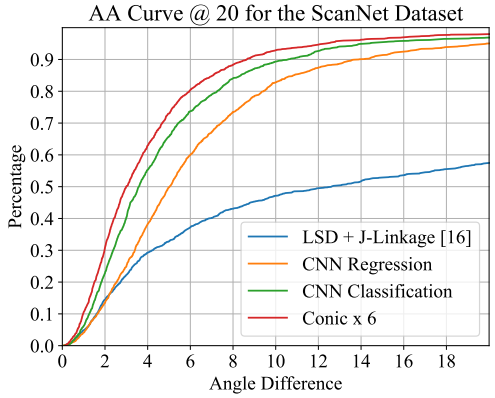


(b) Angle difference ranges from 0° to 20°.

Figure 2.11: Angle accuracy curves for different methods on the Natural Scene dataset [86].



(a) Angle difference ranges from 0° to 2°.



(b) Angle difference ranges from 0° to 20°.

Figure 2.12: Angle accuracy curves for different methods on the ScanNet dataset [98].

	$AA^{1°}$	$AA^{2°}$	$AA^{10°}$	mean	median
REG	2.4	9.9	58.8	5.09°	3.20°
CLS	4.4	14.5	62.4	5.80°	2.79°
vpdet [86]	18.5	33.0	60.0	12.6°	1.56°
Ours	**29.1**	**50.3**	**85.5**	**1.83°**	**0.87°**

Table 2.4: Performance of vanishing point detection algorithms on the Natural Scene dataset [86]. vpdet is the method from the dataset paper.

more accurate, while data-driven approaches are more unlikely to produce outliers. This phenomenon is also observed in Figure 2.10b, where neural network baselines achieve higher percentage when the angle error is larger than 4.5°.

Effect of Conic Convolution. We now examine the effect of different numbers of conic convolution layers. We test with 2/4/6 conic convolution layers, denoted as Conic×2/4/6, respectively. For Conic×2, we only keep the last two conic convolutions and replace others as their plain counterparts. For Conic×6, we add two more conic convolution layers at the finest level, without max pooling appended. The results are shown in Table 2.3 and Figure 2.10. We observe that the performance keeps increasing when adding more conic convolutions. We hypothesize that this is because stacking multiple conic convolutions enables our model to capture higher order edge information and thus significantly increase the performance. The performance improvement saturates at Conic×6.

2.2.4.4 NeurVPS on Real World Datasets

Natural Scene [86]. We validate our method on real world datasets to test its effectiveness and generalizability. The results of angle accuracy on the Natural Scene dataset [86] are shown in Table Table 2.4 and Figure 2.11. We also compare the performance in the *consistency measure*, a metric used by the baseline method (a contour-based clustering algorithm, labeled as vpdet) in the dataset paper [86] in Figure 2.13. Our method outperforms this strong baseline algorithm vpdet by a fairly large margin in term of all metrics. Our experiment also shows that the naive CNN baselines under-perform vpdet until the angle tolerance is around 4°. This is consistent with the results from [86], in which vpdet is better than the previous deep learning method [69] in the region that requires high precision. Such phenomena indicates that our geometry-aware network is able to accurately locate vanishing points in images, while naive CNNs can only roughly determine vanishing points' position.

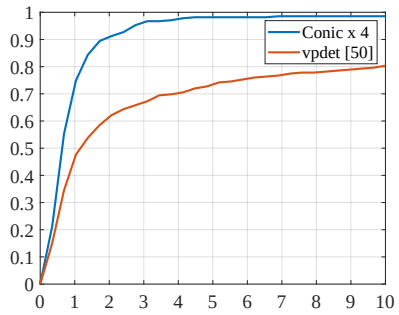


Figure 2.13: Consistency measure on the Nature Scene dataset [86].

	$AA^{1°}$	$AA^{2°}$	$AA^{10°}$	mean	median
LSD [100]	1.7	5.4	24.8	12.6°	11.8°
REG	1.5	5.1	45.1	6.9°	5.0°
CLS	2.0	8.1	55.9	5.3°	3.6°
Ours	**3.4**	**11.5**	**61.7**	**4.5°**	**3.0°**

Table 2.5: Performance of vanishing point detection algorithms on ScanNet [98].

ScanNet [98]. The results on the ScanNet dataset [98] are shown in Table 2.5 and Figure 2.12. For baseline of traditional methods, we only compare our method with LSD + J-linkage because other methods such as [86] are not directly applicable when there are three vanishing points in a scene. Our results reduced the mean and median error by 6 and 4 times, respectively. The angle accuracy also improves by a large margin. The ScanNet [98] is a large dataset, so both CLS and REG works reasonable good. However, because the traditional convolution cannot fully exploit the geometry structure of vanishing points, the performance of those baseline algorithms is worse than the performance of our conic convolutional neural network. It is also worth mentioning that errors of ground truth vanishing points of the ScanNet dataset are quite large due to the inaccurate 3D reconstruction and budget capture devices, which probably is the reason why the performance gap between conic convolutional networks and traditional 2D convolutional networks is not so significant.

One drawback of our data-driven method is the need of a large amount of training data. We do not evaluate our method on datasets such as YUD [55], ECD [80], and HLW [101] because there is no suitable public dataset for training. In the future, we will study how to exploit geometric information under unsupervised or semi-supervised settings hence to alleviate the data scarcity problem.

2.3 Learning to Detect Reflection Symmetry

2.3.1 Introduction

Recovering the 3D orientation of objects in an image is a fundamental problem in 3D vision, which plays a role in tasks such as robotics, autonomous driving, virtual reality (VR), augmented reality (AR), and 3D scene understanding. Traditionally, such problem is extremely hard to solve. Researchers often resort to RGB-D input captured with time-of-flight cameras or structured light [102, 103, 104]. Unfortunately, depth cameras often have limited range and can be interfered by other light sources. More importantly, the requirement of owning a depth camera is inconvenient for average users, which severely restricts its applications.

Recent advances in convolutional neural networks in object detection and instance segmentation have shown good potential in inferring object-level information from RGB images by leveraging supervised learning. Nowadays, single-view neural network-based methods are

able to predict the object pose under different settings. Some work explores the *instance-level 3D pose estimation* problem [17, 105, 106] in which the CAD models of the objects are known beforehand. However, these settings are rather limited because in practice objects are different and we do not have CAD models for most objects. Therefore, other work tries to tackle the *category-level 3D pose estimation* problem [14, 107, 108] without using the exact CAD models of objects. Unlike the cases where either depth maps or CAD models are available, previous single-view category-level 3D pose estimation methods hardly exploit the geometric constraints between the input RGB image and the 3D shape, and predict the pose solely by interpolating the training data. Hence, such formulation is ill-posed, which leads to inaccurate pose recovery [22].

We observe that the canonical space of objects is often determined by aligning the Y-Z plane to the symmetry planes of objects [109, 110], so the normal direction of the symmetry plane encodes most of the geometric information regarding the pose of the object. To this end, we propose the NeRD network to detect the reflection symmetry from RGB images. *NeRD* combines the strength of learning-based recognition and geometry-based reconstruction methods. Specifically, NeRD first detects the parameters of the mirror plane from the image with a coarse-to-fine strategy and then recovers the depth from a reflective stereopsis. We incorporate reflection symmetry as a prior into deep networks through plane-sweep cost volumes built from features of corresponding pixels, as shown in Figure 2.17a.

The network (see Figure 2.16) consists of a backbone feature extractor, a differentiable warping module for building the 3D cost volumes, and a cost volume network. This framework naturally enables neural networks to utilize the information from corresponding pixels of reflection symmetry inside a single image. We evaluate our method on the ShapeNet dataset [109] and Pix3D dataset [110]. Extensive comparisons and analysis show that by detecting and utilizing intra-image pixel correspondence from reflection symmetry, our method has better accuracy for recovering normal direction of symmetry plane and hence the object pose, even when the object is not perfectly symmetric.

Our main contributions are summarized as below:

- we identify the problem of learning neural 3D reflection symmetry detector, in which the intra-image pixel correspondence of symmetry can be utilized for accurate plane normal estimation;
- we propose a novel framework that leverages single-view dense feature matching to estimate symmetry planes, significantly outperforming previous methods;
- we show that the learned symmetry planes benefit tremendously a variety of downstream tasks, including single-view pose recovery and depth estimation.

2.3.2 Related Work

For many years, scientists from vision science and psychology have found that symmetry plays an important role in human vision system [111, 112]. People have exploited different kinds of symmetry for tasks such as texture impainting [113], unsupervised shape recovering [114], and image manipulation [115]. Researchers have utilized the correspondences of symmetry to

(a) 2D reflection symmetry (b) 3D reflection symmetry (c) 3D reflection symmetry

Figure 2.14: Examples of 2D and 3D reflection symmetry reconstruction, and demonstration of scale ambiguity in 3D reflection symmetries from 2D images. 2D symmetries are not helpful for 3D reconstruction due to lack of perspective distortion. Illustration of scale ambiguity. We display two scenes that only differ by a scale c. The images of the two scenes are exactly the same, but the distances between the origin and two symmetry planes are different, i.e., $\|\mathbf{w}\|_2 = c\|\mathbf{w}'\|_2$.

reconstruct shapes in different representations, such as points [116], curves [117], and recent deep implicit fields [118]. However, these methods either assume that the input camera pose or the symmetry plane is given, or require its correspondence points. This is because detecting 3D symmetry from a single view is challenging.

Symmetry Detection. On the one hand, most of geometry-based symmetry detection methods use handcrafted features and only work for 2D planar and front-facing objects [119, 120, 121], as shown in Figure 2.14a. The extracted 2D symmetry axes and correspondences cannot provide enough geometric cues for depth reconstruction. In order to make reflection symmetry useful for depth reconstruction, it is necessary to detect the 3D mirror plane and corresponding points of symmetric objects (Figure 2.14b) from perspective images. On the other hand, recent single-image processing neural networks [109, 122, 123, 124, 125] can approximately recover the camera orientation with respect to the canonical pose, which gives mirror plane of symmetry. However, the camera poses from those data-driven networks are not accurate enough, because they do not exploit the geometric constraints of symmetry. To remedy the above issues, our NeRD tries to takes the best of both worlds. The proposed method first detects the 3D mirror plane of a symmetric object from an image and then recovers the depth map by finding the pixel-wise correspondence with respect to the symmetry plane, all of which are supported with geometric principles. Our experiment (Section 2.3.4) shows that NeRD is indeed much more accurate for 3D symmetry plane detection, compared to previous learning-based methods [124, 126].

Learning-Based Single-Image 3D Understanding. Inspired by the success of CNNs in image classification and object detection, multiple single-view learning-based 3D understanding tasks have been explored, including depth estimation [20, 127], camera pose recovery, etc. Although these methods demonstrate promising results on benchmark datasets, the inferred results are not accurate enough for most subsequent 3D reconstruction purposes. To alleviate this issue, our method leverages the symmetry prior by matching pixel-level features for accurate single-view 3D understanding.

2.3.3 Methods

2.3.3.1 Camera Model and 3D Symmetry

Let $\mathbb{O} \subset \mathbb{R}^4$ be the set of points in the homogeneous coordinate that are on the surface of an object. If we say $\mathbb{O}$ admits the *symmetry*[1] with respect to a rigid transformation $\mathbf{M} \in \mathbb{R}^{4\times4}$, it means that

$$\forall \mathbf{X} \in \mathbb{O} : \mathbf{MX} \in \mathbb{O}, \quad \text{and} \quad \mathcal{F}(\mathbf{X}) = \mathcal{F}(\mathbf{MX}), \tag{2.4}$$

where $\mathbf{X}$ is homogeneous coordinates of a point on the surface of the object, $\mathbf{MX}$ is the corresponding point of $\mathbf{X}$ with respect to the symmetry, and $\mathcal{F}(\cdot)$ represents the surface properties at a given point, such as the surface material and texture. For example, if an object has reflection symmetry with respect to the Y-Z plane in the world coordinate, then we have its transformation $\mathbf{M}_x = \text{diag}(-1, 1, 1, 1)$. Figure 2.14 shows an example of 3D reflection symmetry.

Given two 3D points $\mathbf{X}, \mathbf{X}' \in \mathbb{O}$ in the homogeneous coordinate that are associated by the symmetry transform $\mathbf{X}' = \mathbf{MX}$, their 2D projections $\mathbf{x}$ and $\mathbf{x}'$ must satisfy the following conditions:

$$\mathbf{x} = \mathbf{K}\mathbf{R}_\mathbf{t}\mathbf{X}/d, \quad \text{and} \quad \mathbf{x}' = \mathbf{K}\mathbf{R}_\mathbf{t}\mathbf{X}'/d'. \tag{2.5}$$

Here, we keep all vectors in $\mathbb{R}^4$. $\mathbf{x} = [x, y, 1, 1/d]^T$ and $\mathbf{x}' = [x', y', 1, 1/d']^T$ represent the 2D coordinates of the points in the pixel space, d and d' are the depth in the camera space, $\mathbf{K} \in \mathbb{R}^{4\times4}$ is the camera intrinsic matrix, and $\mathbf{R}_\mathbf{t} = \left[\begin{smallmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{smallmatrix}\right]$ is the camera extrinsic matrix that rotates and translates the coordinate from the object space to the camera space.

From Equation (2.5), we can derive the following constraint for their 2D projections $\mathbf{x}$ and $\mathbf{x}'$:

$$\mathbf{x}' \propto \underbrace{\mathbf{K}\mathbf{R}_\mathbf{t}\mathbf{M}{\mathbf{R}_\mathbf{t}}^{-1}\mathbf{K}^{-1}}_{\mathbf{C}}\mathbf{x} \doteq \mathbf{C}\mathbf{x}. \tag{2.6}$$

We use the proportional symbol here as the 3rd dimension of $\mathbf{x}'$ can always be normalized to one so the scale factor does not matter. The constraint in Equation (2.6) is valuable to us because the neural network now has a geometrically meaningful way to check whether the estimated depth d is reasonable at (x, y) by comparing the image appearance at (x, y) and (x', y'), where (x', y') is computed from Equation (2.6) given x, y, and d. If d is a good

[1] An object might admit multiple symmetries. For example, a rectangle has two reflective symmetries and one rotational symmetry. We here describe one at a time.

estimation, the two corresponding image patches should be similar due to $\mathcal{F}(\mathbf{X}) = \mathcal{F}(\mathbf{X}')$ from the symmetry constraint in Equation (2.4). This is often called *photo-consistency* in the literature of multi-view steropsis [128].

An alternative way to understand Equation (2.6) is to substitute $\mathbf{X}' = \mathbf{M}\mathbf{X}$ into Equation (2.5) and treat the later equation as the projection from another view. By doing that, we reduce the problem of shape-from-symmetry to two-view steropsis, only that the stereo pair is in special positions.

Reflection Symmetry in 3D. Equation (2.6) gives us a generalized way to represent any types of symmetry with matrix $\mathbf{C} = \mathbf{K}\mathbf{R_t}\mathbf{M}\mathbf{R_t}^{-1}\mathbf{K}^{-1}$. For reflection symmetry, a more intuitive parametrization is to use the equation of the symmetry plane in the camera space. Let $\tilde{\mathbf{x}} \in \mathbb{R}^3$ be the coordinate of a point on the symmetry plane in the camera space. The equation of the symmetry plane can be written as

$$\mathbf{w}^T\tilde{\mathbf{x}} + 1 = 0, \tag{2.7}$$

where we use $\mathbf{w} \in \mathbb{R}^3$ as the parameterization of symmetry. *The goal of reflection symmetry detection is to recover* $\mathbf{w}$ *from images.*

On the first impression, one may wonder why $\mathbf{R_t}$ (i.e., camera poses) in Equation (2.6) has 6 degrees of freedoms (DoFs) while $\mathbf{w}$ only has 3. This is due to the specialty of reflection symmetry. Rotating the camera with respect to the normal of the symmetry plane (1 DoF) and translating the camera along the symmetry plane (2 DoFs) cannot change the relative pose of the camera with respect to the symmetry plane. Therefore the number of DoFs in reflection symmetry is indeed $6 - 1 - 2 = 3$.

To derive the relationship between $\mathbf{C}$ and $\mathbf{w}$, let $\tilde{\mathbf{x}}_0 \in \mathbb{R}^3$ be a point in the camera space and $\tilde{\mathbf{x}}_1$ be its mirror point with respect to the symmetry plane $\mathbf{w}^T\tilde{\mathbf{x}} + 1 = 0$. Figure 2.15 illustrates the process of miring a point in 2D, where the red dots are the pair of points $\tilde{\mathbf{x}}_0$ and $\tilde{\mathbf{x}}_1$ the line in the middle is the symmetry plane whose normal $\tilde{\mathbf{n}} = \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$. The distance between $\tilde{\mathbf{x}}_0$ and the symmetry plane is $\frac{\mathbf{w}^T\tilde{\mathbf{x}}_0+1}{\|\mathbf{w}\|_2}$, according to the formula of distance from a point to a plane. Therefore, we have

$$\tilde{\mathbf{x}}_1 = \tilde{\mathbf{x}}_0 - 2\frac{\mathbf{w}^T\tilde{\mathbf{x}}_0 + 1}{\|\mathbf{w}\|_2^2}\mathbf{w}.$$

We could also write it in matrix form:

$$\begin{bmatrix}\tilde{\mathbf{x}}_1 \\ 1\end{bmatrix} = \begin{bmatrix}\tilde{\mathbf{x}}_0 \\ 1\end{bmatrix}\begin{bmatrix}\mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^T}{\|\mathbf{w}\|_2^2} & -\frac{2\mathbf{w}}{\|\mathbf{w}\|_2^2} \\ \mathbf{0} & 1\end{bmatrix}.$$

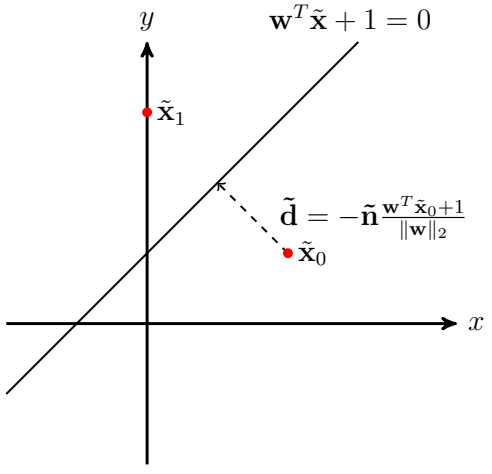


Figure 2.15: Illustration of reflection symmetry with two points.

Because the transformation between the camera space and the pixel space is given by

$$\mathbf{x} = \mathbf{K} \begin{bmatrix} \tilde{\mathbf{x}} \\ 1 \end{bmatrix},$$

we finally have

$$\begin{aligned} \mathbf{C}(\mathbf{w}) &= \mathbf{K} \begin{bmatrix} \mathbf{I} - \frac{2\mathbf{w}\mathbf{w}^T}{\|\mathbf{w}\|_2^2} & -\frac{2\mathbf{w}}{\|\mathbf{w}\|_2^2} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{K}^{-1} \\ &= \mathbf{K} \left(\mathbf{I} - \frac{2}{\|\mathbf{w}\|_2^2} \begin{bmatrix} \mathbf{w} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w}^T & \mathbf{1} \end{bmatrix} \right) \mathbf{K}^{-1}. \end{aligned} \tag{2.8}$$

Scale Ambiguity. Similar to structure-from-motion in which it is impossible to determine the absolute size of scenes [129], shape-from-symmetry also has a scale ambiguity. This is demonstrated in Figure 2.14c. In the case of reflection symmetry, we cannot determine the value of $\|\mathbf{w}\|_2$, i.e., the symmetry plane's distance from origin, from a single image without relying on size priors, as it is always possible to scale the scene by a constant (and thus scale $\|\mathbf{w}\|_2$) without affecting images. Therefore, we fix $\|\mathbf{w}\|_2$ to be a constant and leave the ambiguity as it is. In other words, NeRD is designed only to recover the normal direction of the symmetry plane. For real-world applications, this scale ambiguity can be resolved when the object size or the distance between the object and the camera is known.

2.3.3.2 Overall Pipeline of NeRD

Motivation. Section 2.3.3.1 provides us a geometric way to verify whether a given $\mathbf{w}$ is valid: For each pixel (x, y), we check if there exists a d so that the image feature at (x, y) and its mirror point (x', y') are similar, where (x', y') are computed with Equation (2.6). If $\mathbf{w}$ is correct, then for pixels whose mirror parts are not occluded, we should be able to find their corresponding points that are similar to themselves. To utilize such idea, we turn the problem of regressing $\mathbf{w}$ into a classification problem: We first enumerate possible plane normal directions and use a neural network to verify whether these directions are closed to the real symmetry planes or not. Figure 2.17a illustrates such process.

Methods. Figure 2.16 illustrates the overall pipeline of NeRD during inference. For each input image, we compute its 2D feature map (Section 2.3.3.3) and generate a list of candidate normal directions of its symmetry plane. For each candidate normal $\mathbf{w}$, we use it to warp the 2D feature map and construct a initial 3D cost volume (Section 2.3.3.4) for photo-consistency matching. After that, the cost volume network (Section 2.3.3.5) converts the cost volume tensor into a confidence value. We pick $\mathbf{w}$ with the highest confidence as the resulting normal direction of symmetry plane.

A brute-force enumeration of $\mathbf{w}$ is slow, especially when high precision is required. To accelerate it, NeRD uses a coarse-to-fine strategy, which we will describe in detail in Section 2.3.3.6. Figure 2.17b illustrates our method. In ith round of inference, the coarse-to-fine

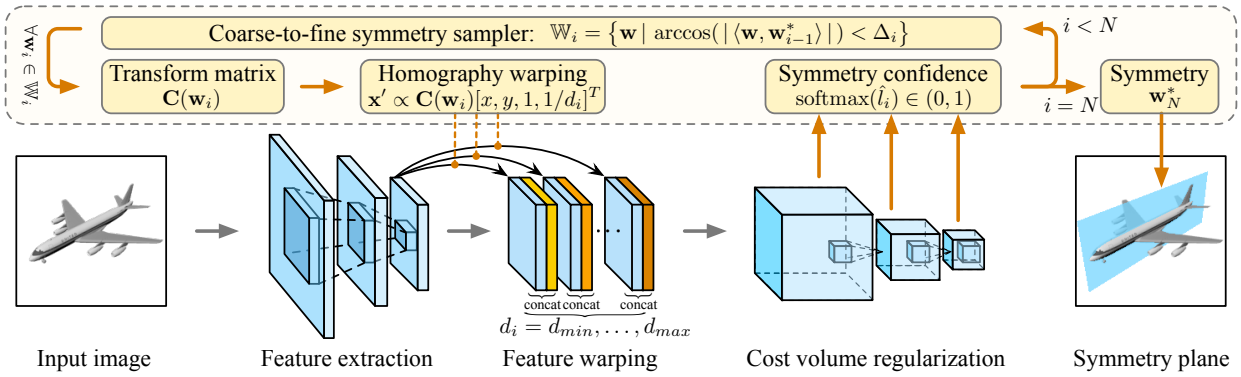


Figure 2.16: Pipeline of NeRD. During inference, the coarse-to-fine symmetry sampler gives a list of candidate normal directions of the symmetry plane. For each candidate symmetry plane, a warping transformation matrix $\mathbf{C}$ is computed according to Equation (2.8). Input images first go through the feature extraction (backbone) network. Features are then warped by a warping module based on the symmetry transformation $\mathbf{C}$ and depth d_i. A cost volume is constructed by fusing the warped features and feeding into a 3D convolutional neural network for refinement. The final confidence of each symmetry plane is predicted by aggregating the resulting depth probability tensor.

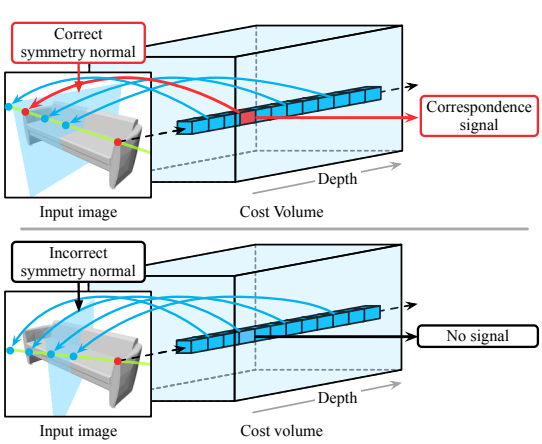
sampler samples N candidates symmetry plane $\{\mathbf{w}_i^k\}_{k=1}^K$ uniformly and evaluate their confidence with our neural network. Then, we find the pose $\mathbf{w}_i^*$ with the highest confidence score and limit the symmetry sampler to the nearby region around it. This process is repeated until an accurate symmetry plane is pinpointed.

2.3.3.3 Backbone Network

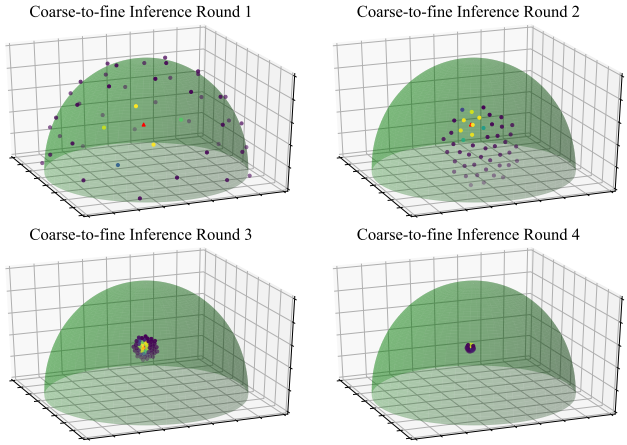
The goal of the *backbone network* is to extract 2D features from images. We use a modified ResNet-like network as our backbone. To reduce the memory footprint, we first down-sample the image with a stride-2 5×5 convolution. After that, the network has 8 *basic blocks* [10] with ReLU activation. The 5th basic block uses stride-2 convolution to further downsample the feature maps. The number of channels is 64. The output feature map $\mathbf{F}$ has dimension $\lfloor \frac{H}{4} \rfloor \times \lfloor \frac{W}{4} \rfloor \times 64$.

2.3.3.4 Feature Warping Module

The function of the *feature warping module* is to construct the initial 3D cost volume tensor $\mathbf{V}(x, y, d)$ for photo-consistency matching. We discretize d so that $d \in \mathcal{D} = \{d_{\min} + \frac{i}{D-1}(d_{\max} - d_{\min}) \mid i = 0, 1, \ldots, D-1\}$ to make the 3D cost volume homogeneous to 3D convolution, in which $d_{\min}$ and $d_{\max}$ is the minimal and maximal depth we want to predict and D is the number of sampling points for depth. As mentioned in Section 2.3.3.1, the correctness of d at (x, y) correlates with the appearance similarity of the image patch at pixels



(a) Symmetry detection process of NeRD.



(b) Coarse-to-fine inference of NeRD

.

Figure 2.17: Symmetry detection process and coarse-to-fine inference of NeRD. Left: for each pixel, we enumerate its depth and warp features along the line according to the symmetry plane hypothesis. If the hypothesis is correct, there should be matched features for most of the pixels. Right: We show the sampled normal direction in a 4-round coarse-to-fine inference. The color of points represents the scores from symmetry confidence network.

represented by $\mathbf{x}$ and $\mathbf{Cx}$. Therefore, we set $\mathbf{V}$ by concatenating the backbone features at these two locations, i.e.,

$$\mathbf{V}(x, y, d) = \Big[\mathbf{F}(x, y),\, \mathbf{F}(x', y')\Big], \tag{2.9}$$

where $[\,x', y', 1, 1/d'\,]^T \propto \mathbf{C}[\,x, y, 1, 1/d\,]^T$, i.e., (x', y') being the projection of the mirror point of the pixel (x, y) assuming its depth is d. Here $\mathbf{F}$ is the backbone feature, and $\mathbf{C}$ is computed from the sampled symmetry plane $\hat{\mathbf{w}}$. We apply bilinear interpolation to access the features at non-integer coordinates. The dimension of the cost volume tensor is $\lfloor \frac{H}{4} \rfloor \times \lfloor \frac{W}{4} \rfloor \times D \times 32$.

2.3.3.5 Cost Volume Network

The goal of the cost volume network is to turn the initial 3D cost volume tensor $\mathbf{V}$ from the feature warping module into a depth probability tensor $\mathbf{P}(x, y, d) := \Pr[\mathbf{D}(x, y) = d]$. This is possible as the matrix multiplication on the channel dimension can be used to check for the photo-consistency on $\mathbf{V}$. However, the initial cost volume aggregated from image features can be noisy. Thus, we use a network consists of multiple 3D convolution layers that is capable of efficiently regularize the cost volume information. We aggregate the multi-resolution encoder features with max-pool operators and then apply the sigmoid function to normalize the confidence values into $[0, 1]$.

2.3.3.6 Symmetry Sampler

Inference. As shown in Figure 2.17b, the symmetry sampler uniformly samples $\{\mathbf{w}_i^k\}_{k=1}^K$ from $\mathbb{W}_i \subset \mathbb{R}^3$ using Fibonacci lattice [36, 97], where $\mathbb{W}_i$ is the sampling space of the ith round of inference. In the first round, candidates are sampled from the surface of a unit hemisphere. For the following rounds, we set $\mathbb{W}_i = \{\mathbf{w} \in \mathbb{S}^2 \mid \arccos(|\langle \mathbf{w}, \mathbf{w}_{i-1}^* \rangle|) < \Delta_i\}$ to be a spherical cap, where $\mathbf{w}_{i-1}^*$ is the optimal $\mathbf{w}$ from the previous round and Δ_i is a hyper-parameter.

Training. During training, we sample symmetry planes for each image according to the hyper-parameter Δ_i. For the ith level, symmetry candidates are sampled from $\{\hat{\mathbf{w}} \in \mathbb{S}^2 \mid \arccos(|\langle \mathbf{w}, \hat{\mathbf{w}} \rangle|) \leq \Delta_i\}$, where $\mathbf{w}$ is the ground truth symmetry pose. We also add a random sample $\hat{\mathbf{w}} \in \mathbb{S}^2$ to reduce the sampling bias. For each sampled $\hat{\mathbf{w}}$, its confidence labels is $l_i = \mathbf{1}[\arccos(|\langle \mathbf{w}, \hat{\mathbf{w}} \rangle|) < \Delta_i]$ for the ith level. The training error could be written as

$$L_{\text{cls}} = \sum_i \text{BCE}(\hat{l}_i, l_i),$$

where BCE represents the binary cross entropy error, and $\hat{l}_i$ is predicted confidence of $\hat{w}$ for the ith level in the coarse-to-fine inference.

2.3.4 Experiments

2.3.4.1 Datasets

We conduct experiments on the synthetic ShapeNet dataset [109] and real-world Pix3D dataset [110], in which models have already been processed so that in their canonical poses the Y-Z plane is the plane of the reflection symmetry.

ShapeNet. We use the same camera pose, intrinsic, and train/validation/test split from a 13-category subset of the dataset as in R2N2 and others [130, 131, 132] to make the comparison easy and fair. We exclude the lamp category as it contains many asymmetric objects. We use Blender to render the images with resolution 256×256.

Pix3D. Pix3D [110] is a real-world dataset containing image-shape pairs with accurate 2D-3D registrations. To demonstrate the versatility of NeRD, we train and test NeRD on the Pix3D dataset. We assume that the bounding boxes of objects have been detected, and we use them to crop the images for removing the background while maintaining the aspect ratio. Then, we rescale the resulting images to 256×256 and adjust the camera intrinsic matrix $\mathbf{K}$ accordingly. Next, we reject images extraordinary with focal lengths and depth values. Finally, we randomly split the remaining data into train and test sets, which contains 5285 and 588 images, respectively.

	backbone (sec 2.3.3.3)	cost volume (sec 2.3.3.5)	feature warping var	avg	cat	error metrics avg	med	$<1°$	$<2°$
a		✓			✓	7.12°	**0.54°**	66.8%	77.2%
b	✓				✓	6.82°	0.99°	50.1%	70.1%
c	✓	✓	✓			6.33°	0.57°	68.1%	81.5%
d	✓	✓		✓		6.41°	0.66°	63.7%	77.7%
e	✓	✓			✓	**5.41°**	0.56°	**68.2%**	**81.5%**

Table 2.6: Ablation study of 3D reflection symmetry detection on ShapeNet.

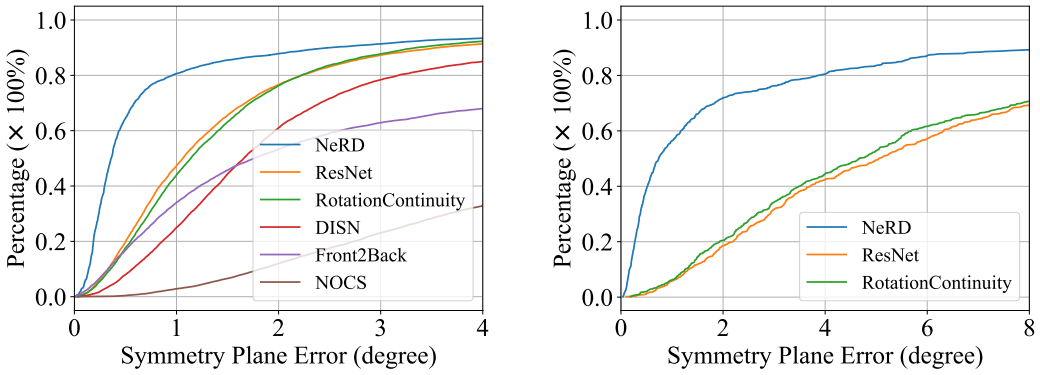
2.3.4.2 Implementation Details

We implement NeRD in PyTorch. We use the plane $x = 0$ in the object space as the ground truth symmetry plane because it is explicitly aligned for each model by authors of ShapeNet. We set $d_{\min}$ and $d_{\max}$ according to the depth distribution of the dataset, and use $D = 64$ for the depth of the cost volume. We use $N = 4$ rounds in the coarse-to-fine inference, in each of which $K = 32$ normal directions are sampled. We choice $\Delta = [20.7°, 6.44°, 1.99°, 0.61°]$ according to the gap between nearly directions. Our experiments are conducted on two NVIDIA RTX 2080Ti GPUs. We use the Adam optimizer [54] for training. Learning rate is set to 3×10^{-4} and batch size is set to 16 per GPU. We train the NeRD for 40 epochs and decay the learning rate by a factor of 10 at the 30th epoch. The overall inference speed is about 1 image per second per GPU.

Metrics. To better understand the performance of symmetry detection, we show two forms of metrics. We plot a performance curve for each detector-dataset pair, in which the x-axis represents the angle accuracy and the y-axis represents the proportion of the data whose error is less than that. We also report quantative metrics, including the median and mean of the angle difference, and the percentages of testing images whose error is smaller than 0.5°, 1.0°, 2.0°, and 4.0°, for ease of comparision.

2.3.4.3 Ablation Studies

We conduct ablation studies to justify each component in NeRD. In Table 2.6, we analyze the function of three main components of NeRD: the 2D backbone network (Section 2.3.3.3), feature warping module (Section 2.3.3.4), and the cost volume network (Section 2.3.3.5). The second column of Table 2.6 represents whether we use the feature from the 2D backbone or just RGB values to construct the cost volume. Comparing (a) and (e), we find that removing 2d backbone degrades the performance, especially at the region $> 2°$. We think this is because the 2D backbone network increases the receptive field, which make the network more robust. The third column represents whether we want to replace the cost volume network with a simple max-pool layer. Results in (b) and (e) show that the cost volume



(a) Curves on ShapeNet (synthetic)

(b) Curves on Pix3D (real-world)

Figure 2.18: Performance curves of symmetry detection algorithms.

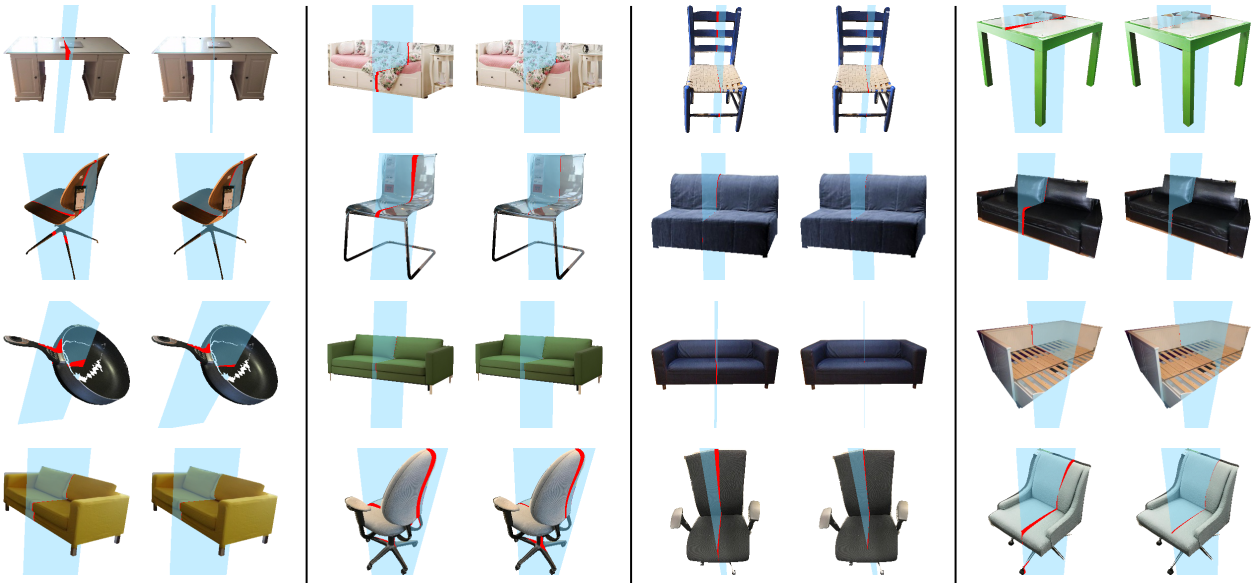
	avg	med	$<0.5°$	$<1.0°$	$<2.0°$
DISN [126]	2.80°	1.65°	7.96%	24.9%	61.0%
ResNet [10]	2.08°	1.06°	19.7%	47.3%	76.6%
RotationContinuity [124]	1.94°	1.14°	17.6%	43.9%	76.2%
Front2Back [125]	9.41°	1.76°	16.8%	34.0%	53.2%
NOCS [123]	9.95°	6.18°	0.39%	2.83%	11.9%
NeRD	**1.58°**	**0.36°**	**64.5%**	**80.6%**	**87.8%**

Table 2.7: Performance of symmetry detection and object pose recovery algorithms on ShapeNet. We report the normal direction error of the predicted symmetry planes. We note that NOCS [123] requires ground truth object shapes as input.

	avg	med	$<1.0°$	$<2.0°$	$<4.0°$
ResNet [10]	8.01°	5.06°	5.78%	18.5%	42.3%
RotationContinuity [124]	7.91°	4.67°	6.12%	20.4%	44.3%
NeRD	**3.37°**	**0.73°**	**56.3%**	**71.9%**	**80.4%**

Table 2.8: Performance of symmetry detection algorithms on the real-world Pix3D [110] dataset. We report the normal direction error of the predicted symmetry planes.

network is the key component for an accurate symmetry detector. Finally, we study the different pooling schemes in the feature warping module. From (c), (d), and (e), we find that the feature concatenation and variance pooling gives the best results, while the average pooling performs poorly in the high-precision region ($< 1°$). This matches our intuition in Section 2.3.3.1 that NeRD compares the feature to check photo-consistency.



(a) ResNet (b) NeRD (a) ResNet (b) NeRD (a) ResNet (b) NeRD (a) ResNet (b) NeRD

Figure 2.19: Qualitative results of symmetry detection on Pix3D. We show the detected symmetry planes from ResNet and our NeRD. Errors of symmetry planes (pixels between the predicted and ground truth planes) are highlighted.

2.3.4.4 Symmetry Detection on Synthetic Datasets

Baselines. We briefly introduce some state-of-the-art single-view symmetry detection and pose estimation baseline that we will compare against. Probably the plainest way to estimate the 3D symmetry plane $\mathbf{w}$ is direct regression. We implement it with ResNet-50 [10] and train it with L1 loss. RotationContinuity [124] identifies a 6D representation of rotation which they claim is more suitable for learning. We implement it and train with L1 loss. DISN [126] also implements its 6D representation for ShapeNet but is trained with L2 loss. We report the performance of their pre-trained model. Front2Back [125] is a recent work that detects the 3D symmetry plane, which first predicts a depth map and then fit the symmetry plane with a traditional method [133]. We report the performance of their results. NOCS [123] predicts a coordinate of normalized object coordinate space for each pixel and recovers the pose with Umeyama algorithm [134]. Following their paper, we train the NOCS estimator on ShapeNet and use their code to recover the orientation of objects from prediction.

Results. Table 2.7 and Figure 2.18a show the comparison on the ShapeNet dataset. By utilizing geometric cues from symmetry, our approach significantly outperforms previous state-of-the-art camera pose detection networks. The performance gap is much larger in the region of higher precision ($< 1°$). For example, NeRD can achieve an accuracy of $0.5°$ on about 70% of testing cases, while direct regression with ResNet and other baselines can only reach that on less than 20% of data. Such phenomena indicate that the intra-image

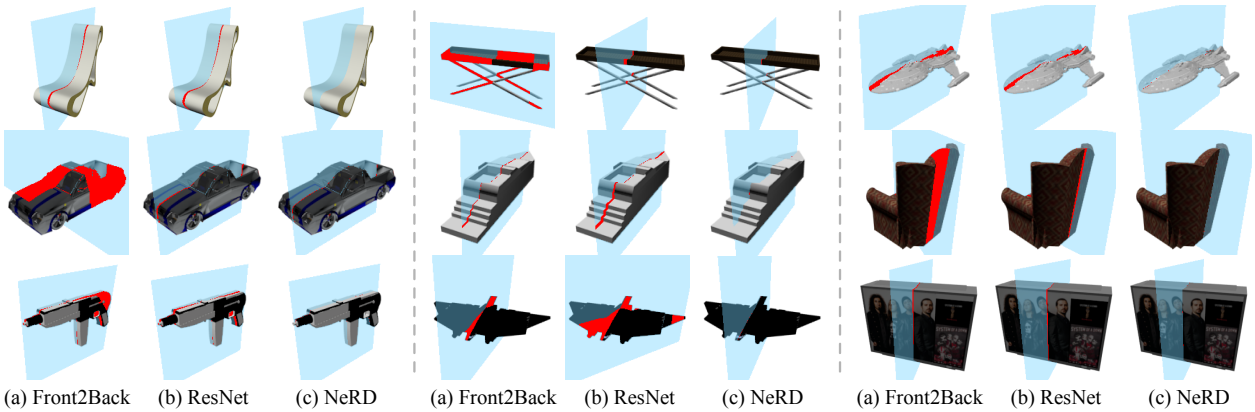


Figure 2.20: Qualitative results of symmetry detection on ShapeNet. Errors of symmetry planes (pixels between the predicted and ground truth planes) are highlighted.

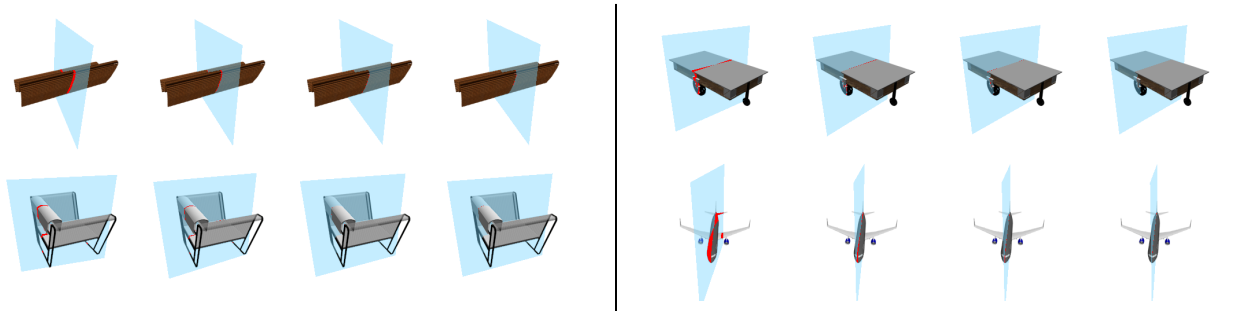
correspondence does help algorithms recover symmetry planes more accurately, while naive CNNs can only roughly predict the plane normal by interpolating from training data. We also find that end-to-end approaches that directly predict the symmetry plane (ResNet, DISN, NeRD, etc) performs better than the methods which require heavier post-processing (NOCS and Front2Back). This hints us that using a loss function that is more directly related to the goal has an advantage.

2.3.4.5 Symmetry Detection on Real-World Datasets

Table 2.8 and Figure 2.18b show the comparison on the real-world Pix3D dataset. NeRD outperforms the naive CNNs regression, and the margin is even bigger compared to the results on ShapeNet. We hypothesize that this is because images in Pix3D uses a larger number of camera configurations, including different focal lengths and object positions with respect to the focal center, while the dataset has fewer images. This requires more generalizability from the algorithms. Our geometry-based approach shines here because it can rely on the cues from correspondence to find the symmetry planes. Also, it is hard for naive convolutional neural networks to make use of the camera intrinsics, which varies from images to images, unlike ShapeNet. In contrast, NeRD uses camera intrinsic matrices in the feature warping module (Section 2.3.3.4) and thus generalizes better when dealing with different camera configurations.

2.3.4.6 Visualization

Qualitative Results. We visualize the detected symmetry in Figure 2.19 and Figure 2.20 on the Pix3D and ShapeNet datasets. We have the following observations: 1) our method outperforms previous methods on unusual objects, e.g. chairs in atypical shapes. This indicates that learning-based methods need to extrapolate from seen patterns and cannot generalize to unusual images well, while our method relies more on geometry cues from



(a) Round $i = 1$ (b) Round $i = 2$ (c) Round $i = 3$ (d) Round $i = 4$ (a) Round $i = 1$ (b) Round $i = 2$ (c) Round $i = 3$ (d) Round $i = 4$

Figure 2.21: Illustration of the coarse-to-fine inference on sampled images from ShapeNet. The symmetry plane with the highest confidence score in each round of coarse-to-fine inference is drawn. In the ith round, we determine the normal of symmetry plane to the accuracy of Δ_i, where $\Delta = [20.7°, 6.44°, 1.99°, 0.61°]$ are set according to the gap between nearly directions from the number of direction samples per round $K = 32$.

(a) Lack of correpondence (b) Multiple symmetry planes (c) Asymmetric object

Figure 2.22: Sampled failure cases of NeRD on ShapeNet.

symmetry, a more reliable source of information for 3D understanding. 2) NeRD gives accurate symmetry planes even on challenging camera poses such as the orientation from the back of chairs. We believe that this is because although visual cues along may not be remarkable enough in these cases, geometric information from correspondence helps to pinpoint the normal of symmetry planes.

Coarse-to-Fine Inference. Figure 2.21 shows the process of coarse-to-fine inference on sampled images from ShapeNet. We display the symmetry plane with the highest confidence score in each round of coarse-to-fine inference. In the ith round, we determine the normal of symmetry plane to the accuracy of Δ_i, where $\Delta = [20.7°, 6.44°, 1.99°, 0.61°]$ are set according to the gap between nearly directions from the number of direction samples per round $K = 32$. The coarse-to-fine inference dramatically reduces number of samples required to achieve a certain level of accuracy. As shown in the figure, the precision of the predicted plane increases with the number of rounds in the coarse-to-fine inference.

Failure Cases. Figure 2.22 shows sampled failure cases on ShapeNet. We categorize those cases into three classes: lack of correspondence, existence of multiple symmetry planes, and asymetric objects. For the first category, e.g., the firearm shown in Figure 2.22(a), it is hard to accurately find the symmetry plane from the geometry cues because for most pixels, their corresponding points are occluded and invisible in the picture. For the second category, objects in shapes such as squares and cylinders admit multiple reflection symmetry, and NeRD may return the reflection plane that differs from the symmetry plane of the ground truth. For the third category, some objects in ShapeNet are not symmetric. Thus, the detected symmetry plane might be different from the "ground truth symmetry plane" computed by applying $\mathbf{R_t}$ to the Y-Z plane in the world space.

Chapter 3

Datasets for Scene Abstraction

In this chapter, we will introduce three datasets that we have developed during my Ph.D. studies to support data-driven approaches for structure-based 3D parsing tasks: the SceneCity Urban 3D Synthetic dataset (SU3), the Landmark 3D Wireframe dataset (L3W), and the Holistic City-Scale data platform (HoliCity). The SU3 is a large-scale synthetic dataset and L3D is a small real-world dataset. Both of them have 3D wireframe annotation and they are developed along with the 3D wireframe projects and have been published in [59]. HoliCity is a larger full-featured data platform for research of learning abstracted high-level holistic 3D structures that can be derived from city CAD models, e.g., corners, lines, wireframes, planes, and cuboids, with the ultimate goal of supporting real-world applications including city-scale reconstruction, localization, mapping, and augmented reality. The accurate alignment of the 3D CAD models and panoramas also benefits low-level 3D vision tasks such as surface normal estimation, as the surface normal extracted from previous LiDAR-based datasets is often noisy. HoliCity has been published in [135].

3.1 SU3: The SceneCity Urban 3D Synthetic Dataset

3.1.1 Introduction

One of the bottlenecks of supervised learning methods from Chapter 2 is inadequate amounts of data for training and testing. As discussed in the section of 2D wireframe parsing (Section 2.1), [31] have developed a dataset for line and wireframe detection. However, their dataset does not contain any depth information for images, which is not sufficient for tasks of 3D parsing. To the best of our knowledge, there is no public image dataset that has both wireframe annotation and 3D information. To help the development of novel structure-based learning approaches and verify the concepts, we create a synthetic dataset containing a larger number of images of city scenes with automatically annotated 3D wireframes from CAD models.

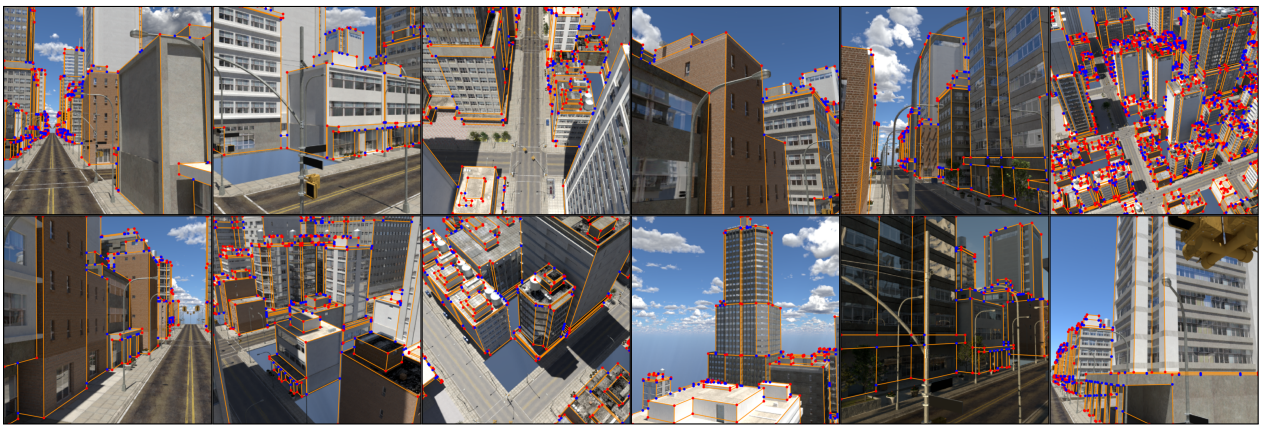


Figure 3.1: Sampled images in the SU3 dataset. Images are overlaid with the groundtruth wireframes, in which the red and blue dots represent the C- and T-type junctions, respectively.

3.1.2 Datasets

Overview. To obtain a large number of images with accurate geometrical wireframes annotated, we use a procedural 3D building mesh repository, SceneCity[1]. The renderings are shown in Figure 3.1. The 3D assets are made up of simple polygons with artist-tuned materials and textures. For each junction in the wireframe, we can group the junctions into two categories: C-junction or T-junction. Corner C-junctions are actual intersections of physical planes or edges, while T-junctions are generated by occlusion. We extract the C-junctions from the vertices of the mesh and compute T-junctions using computational geometry methods and OpenGL. Our dataset includes 230 cities, each containing 8×8 city blocks. Each city has different building arrangements and lighting conditions by varying the sky maps. We randomly generate 100 viewpoints for each city based on criteria such as the number of captured buildings to simulate both hand-held and drone cameras. The synthetic outdoor images are then rendered through global illumination by Blender, which provides $23,000$ images in total. The images are rendered with resolution 512×512 and 20 ray samples per pixel. We enabled Blender's geometry-based de-noise procedure, which significantly improves the quality of images and reduces the rendering time. We use the images of the first 227 cities for training and the remaining 3 cities for validation.

Camera Placement. In order to get a high-quality meaningful synthetic dataset, it is important to have a proper camera placement. For each view, we require that there are at least three visible building; all the objects are in front of the camera; the boundary of the city is in front of the camera; and the boundary of the photo is enclosed by the boundary of the city. If any of these conditions are not satisfied, we reject this placement and regenerate one.

[1]https://www.cgchan.com/

Figure 3.2: Sampled images in the L3W dataset. Images are overlaid with the groundtruth wireframes, in which the red and blue dots represent the C- and T-type junctions, respectively.

3.2 L3W: The Landmark 3D Wireframe Dataset

3.2.1 Introduction

The SU3 dataset in Section 3.1 provides us a large number of images with ground truth annotation. However, there might be a large domain gap between the virtual renderings of synthetic datasets (SU3) and our real-world images. Therefore, we collect the L3W landmark 3D wireframe dataset, which is smaller datasets with manual 3D wireframe annotation. This dataset, along with SU3, can support the learning-based 3D wireframe reconstruction with transfer-learning techniques, which first trains the models on a large-scale synthetic dataset and then fine-tune on a smaller real-world dataset with manual annotations.

3.2.2 Datasets

Overview. Manual annotating 3D wireframe on RGB images can be hard. Therefore, we resort to an existing dataset that provides depth annotation. The MegaDepth dataset [136] contains real images of 196 landmarks in the world with the depth maps reconstructed using structure-from-motion [137]. Due to the limitation of resources, we select about 200 images from it that approximately meet the Manhattan assumptions and can be represented by 3D wireframe, manually label their wireframes on images, and register them with the provided 3D depth maps. We assign 2/3 of the images for training and the remaining 1/3 for testing. The intended way to use this dataset is to first pre-train the neural network on a large synthetic dataset (e.g., SU3), and then use the real images from L3W to finetune the model. Sampled images from L3W are shown in Figure 3.2.

Annotation. We provide an annotation software so that labelers could annotate the wireframe efficiently. The UI of the software shows the image that requires annotation. Labelers could use the following keys to draw the wireframes on images and register the junctions with the provided 3D depth maps:

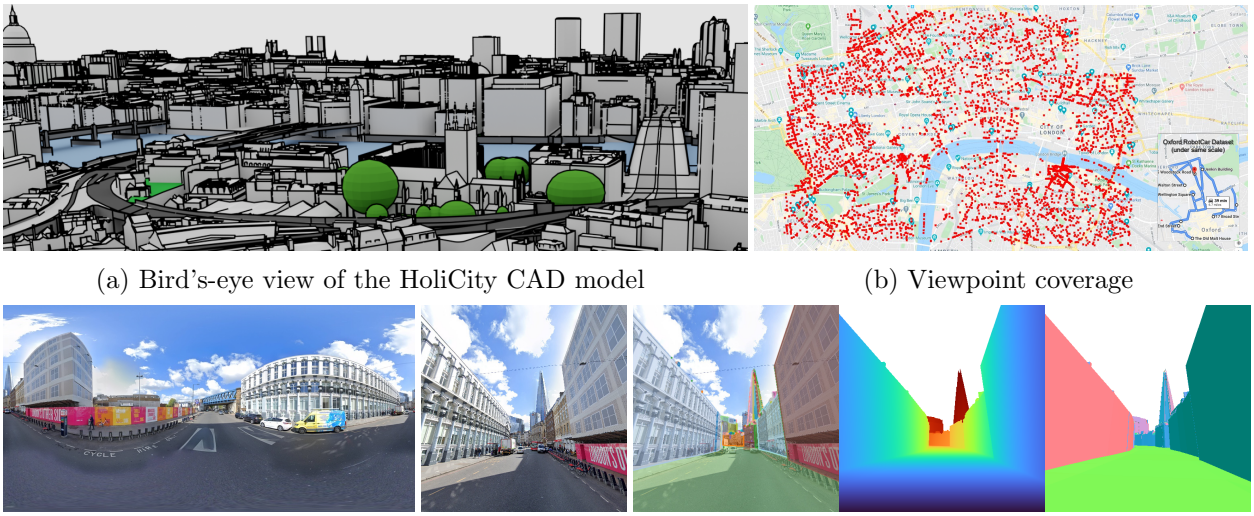
- Press `1` for plotting C-junctions (red dots) on following clicks;
- Press `2` for plotting T-junctions (blue dots) on following clicks(restricted to a line);
- Press `3` for selecting dots on following clicks;
- Press `4` for selecting lines on following clicks;
- Press `z` for rescaling the image to a full window size;
- Press `x` to delete a dot/line, after selecting them;
- Press `c` to move a dot, after selecting them;
- Press `d` to mark the current image as invalid;
- Press `e` to switch between depth maps and RGB images.

3.3 HoliCity: The Holistic City-Scale Data Platform

3.3.1 Introduction

With the development of point features such as ORB [2] and SIFT [1], structure-from-motion (SfM) and simultaneous localization and mapping (SLAM) have been successfully applied to tasks such as autonomous driving, robotics, and augmented reality. Although the robustness of SfM has been improved over the last decades, the resulting point clouds are still noisy, incomplete, and thus can hardly be directly used. Intricate post-processing procedures, such as plane fitting [8], Poisson surface reconstruction [9], and TSDF fusion [138] are necessary for downstream applications. Increasingly have people found that the long pipeline of 3D reconstruction is difficult to implement correctly and efficiently, and results in low-level representations such as point clouds are also unfriendly for parsing, editing, and sharing.

Looking back at the origin of computer vision from the '70s, scientists have found that human beings do not perceive the world with point-clouds. Instead, we abstract scenes with high-level geometry primitives, such as corners, line segments, and planes, to form our sense of 3D, navigate in cities and interact with environments [24]. This hints us that instead of point clouds, we can also use high-level structures as a representation for 3D reconstruction, which in many cases are more compact, intuitive, and easy to process. In fact, early vision research does focus on reconstructing with high-level abstractions, such as lines/wireframes [25], contours/boundaries [27], planes/surfaces [28], and cuboids/polyhedrons [29]. We name these high-level abstractions *holistic structures* in this dissertation, as they tend to represent scenes globally, comparing to the SIFT-like local features. However, recognition of holistic structures from images seems too challenging to be practical at that time. 3D reconstruction with high-level abstractions does not get enough attention despite its potentials, until recently.



(a) Bird's-eye view of the HoliCity CAD model (b) Viewpoint coverage

(c) Panorama (d) RGB (e) Renderings (surface segments, depth, normal)

Figure 3.3: Our HoliCity dataset consists of accurate city-scale CAD models and spatially-registered street view panoramas. HoliCity covers an area of more than 20 km^2 in London from 6,300 viewpoints, which dwarfs previous datasets such as Oxford RobotCar [139] (3.3b). From the CAD models (3.3a) and the panoramas (3.3c), it is possible to generate clean structured ground-truths for 3D understanding tasks, including perspective RGB images (3.3d), surface segments, and normal maps (3.3e).

Inspired by the recent success of deep convolutional neural networks, researchers have proposed a variety of neural networks to extract high-level structures from images, such as wireframes [35, 59], planes [32], cuboids [14], vanishing points [36], room layouts [33], and building layouts [34]. Most of them are supervised learning algorithms, which rely on annotated datasets for training. However, making a properly outdoor 3D dataset for a particular high-level representation is complex. The building process usually has 2 stages: (1) 3D data collection and (2) structure labeling. Collecting 3D data such as depth images is a cost- and labor-intensive process. This is especially true for outdoor scenes due to the lack of dense depth sensors. Even with expensive LiDAR systems, the point clouds from scans are still noisy and have lower spatial resolution compared to RGB images. Derived features such as surface normal are unsmooth, which might be the reason why previous normal estimation research [21, 140, 141, 142] only demonstrates their results on indoor scenes. These characteristics are unfavourable for extracting holistic structures. In addition, labeling high-level abstractions on the collected 3D data is also challenging. On the one hand, manually annotating high-level structures is time-consuming, as it requires researchers to design complicated 3D labeling software. On the other hand, the quality of automatically extracted structures from algorithms such as J-Linkage[143] might not be adequate. The results can be inaccurate, incomplete, and erroneous, especially when the quality of 3D data is not that good. To make the problem worse, frequently a dataset that is labeled for one particular structure cannot be easily reused for other structures. As a result, data

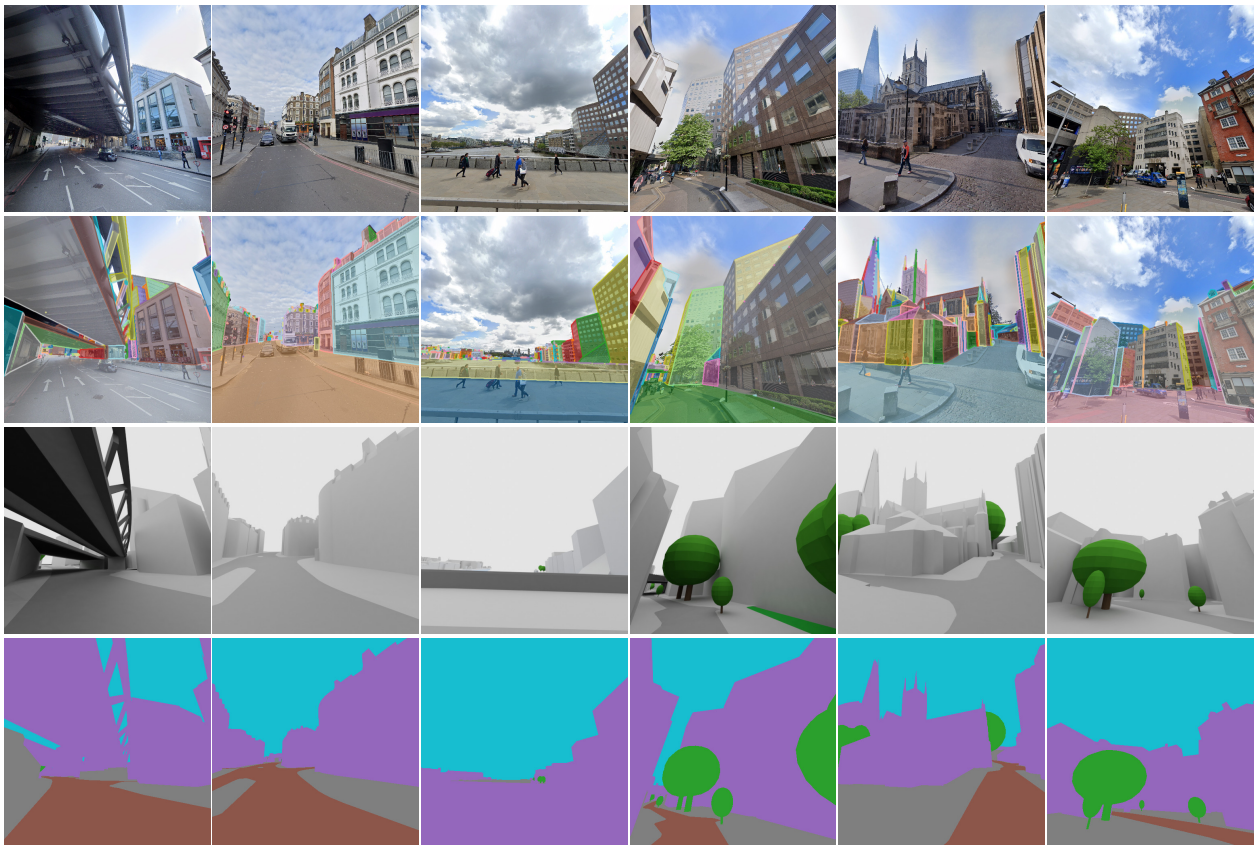


Figure 3.4: Images and generated 3D information from sampled viewpoints of HoliCity dataset. From top to bottom: perspective images rendered from panoramas, surface segments overlaid with images, CAD model renderings, and semantic segmentation.

preparation has become one of the major road blockers for structural 3D vision research.

To provide a high-quality multi-purpose dataset for the vision community, we develop "HoliCity," a data platform for learning holistic 3D structures in urban environments. Figure 3.3 shows an overview. HoliCity is composed of 6,300 high-resolution real-world panoramas that are accurately aligned with the 3D CAD model of downtown London with more than 20 km^2 of area (see Figures 3.3a to 3.3c). Instead of relying on expensive vehicle-mounted LiDAR scanners, HoliCity takes the advantage of existing high-quality 3D CAD city models from the GIS community. This way, we can collect a large area of 3D data with fine details, structure-level annotation, and semantic labels at relatively low cost, in which the CAD models are parametrized by corners, lines, and smooth surfaces so that it is friendly for researchers to extract holistic structures. In comparison, traditional LiDAR-based datasets such as KITTI [144] and RobotCar [139] cover a much smaller area (see Figure 3.3b for comparison), are more expensive to collect, and use noisy point clouds as their representation. The panorama photographs in HoliCity are sharp, professionally captured, and with resolution as high as 13312×6656. In contrast, images of LiDAR-based datasets are often from

	NYUv2	ScanNet	Stanford-2D-3D	SYNTHIA	MegaDepth	KITTI	RobotCar	**HoliCity**
type	real	real	real	synthetic	real	real	real	**real**
scene	indoor	indoor	indoor	driving	landmark	driving	driving	**city**
depth	RGBD	RGBD	RGBD	CAD	SfM	LIDAR	LIDAR	**CAD**
style	dense	dense	◯	dense	dense	quasi	quasi	**dense**
normal	◯	✓	✓	◯	◯	◯	◯	✓
plane	◯	✓	◯	✓	◯	◯	◯	✓
coverage	/	0.034 km^2	0.006 km^2	/	/	/	/	**20 km^2**
path length	/	/	/	/	/	39.2 km	10 km	/
time span	1 scan	1 scan	1 scan	/	unknown	5 scans	2014-2015	**2008-2019**
diversity	464 rooms	707 rooms	4 buildings	/	200 scenes	path	path	**city**
# of images	1.4k	2.5m	1.4k	50k	100k	93k	20m	**6.3k**
source	image	video	video	/	image	video	video	**image**
FoVs	$71° \times 60°$	$45° \times 34°$	panorama	$100° \times 84°$	random	$90° \times 35°$	multi-cam	**panorama**
sementics	2D	3D	3D	3D	N.A.	N.A.	N.A.	**3D**
max depth	(indoor)	(indoor)	(indoor)	∞	(relative)	80m	50m	∞

Table 3.1: Comparing HoliCity with existing 3D datasets. We list the features of NYUv2 [145], ScanNet[98], Stanford-2D-3D-Semantics [146], SYNTHIA [147], MegaDepth [136], KITTI [144], and RobotCar [139]. The ◯ in the normal and plane rows represents that they are not available but possible to use fitting algorithms such as J-Linkage [143] to get the annotations, but the quality might vary.

video recordings, so they can be blurry, low-resolution, and repetitive. Application-wise, traditional LiDAR-based datasets focus on tasks related to low-level representations, such as depth map prediction, reconstruction with point clouds, and camera relocalization, while HoliCity is designed to additionally support the research of 3D reconstruction with high-level holistic structures, such as junctions, lines, wireframes, planes, parameterized surfaces, and other geometry primitives that they can be derived from CAD models.

In summary, the main contributions of this work include:

1. we propose a novel pipeline for creating a city-scale 3D dataset by utilizing existing CAD models and street-view imagery at a relatively low cost;
2. we develop HoliCity as a data platform for learning holistic 3D structures in urban environments;
3. we accurately align the panorama images with the CAD models, in which the median reprojection error is less than half a degree for an average image;
4. we conduct experiments to justify the necessity of a CAD model-based data platform for 3D vision research, including demonstrating potential applications and testing its generalizability from/to other datasets.

3.3.2 Related Work

Synthetic 3D Datasets. Recently, object-level synthetic datasets such as ShapeNet [109] are popular for computer vision research, as people are free to convert 3D CAD models to any representations that their learning-based algorithms like, such as depth maps [20], meshes [148], voxels [149], point clouds [150], and signed distance fields [151]. With the availability

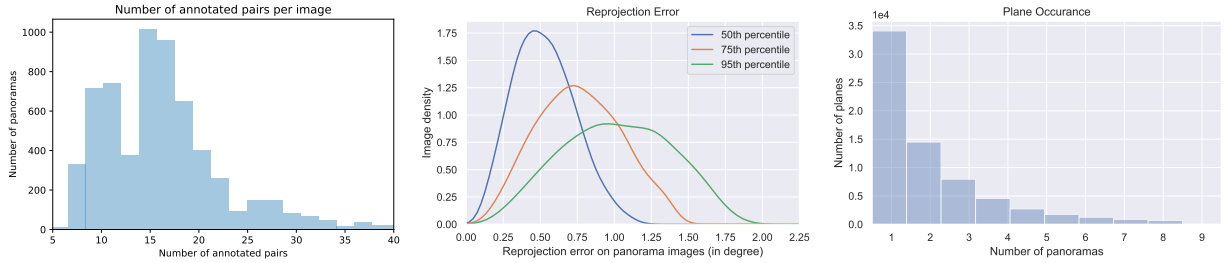
of CAD models, not only HoliCity shares similar freedom as these synthetic object-level datasets, but it also offers scene-level real-world images in urban environments. Additionally, synthetic approaches have also been used to create structured 3D scenes, as seen in SceneNet [152], SUNCG [18], SYNTHIA [147] and GTA5 [153] datasets. They provide perfect labels for depth information and semantic segmentation, and it is also possible to extract high-level structural information from them. Nevertheless, their images are still fake. In our experiments, we find that there exists a large domain gap between the virtual renderings of synthetic datasets and our real-world images.

Outdoor Datasets. Due to the high cost and the limitation of LiDAR systems, acquiring 3D measurements for outdoor scenes is difficult. Publicly available datasets created with LiDAR technology, such as KITTI [144] and RobotCar [139], are relatively small-scale and low-resolution, and mainly focus on the driving scenarios. Recently, outdoor datasets emerge by leveraging structure-from-motion (SfM) and multi-view stereo (MVS) on web imagery in-the-wild [136]. These datasets provide depth information at a low cost with the expense of quality, because visual 3D reconstruction is not really accurate or robust for random Internet images. In addition, previous 3D outdoor datasets mainly use point clouds as their representation, which are usually noisy. Hardly any of them provide structured annotations such as lines, wireframes, segmented 3D planes, and buildings instance. In comparison, HoliCity offers high-quality CAD models and ground truth of holistic 3D structures that cover an unprecedented range of areas and viewpoints at the scale of a city (Figures 3.3 and 3.4).

Indoor Datasets. Thanks to increasingly affordable indoor dense depth sensors such as Kinect and RealSense, high-quality real-world indoor 3D data can be produced on a massive scale. Datasets like NYUv2 [145] provide RGBD images for a variety of indoor scenes. Recent datasets such as SUN3D [154], ScanNet [98], Stanford-2D-3D-Semantics [146], and Matterport3D [155] provide surface reconstruction results and 3D sementics annotation in addition to depth maps. The quality of indoor datasets often varies from scenes to scenes, depending on how well the scene is scanned. Compared to HoliCity that provides accurate CAD models designed for learning holistic structures, the noises, holes, and misalignments in the point clouds of these indoor datasets make them not ideal for extracting high-level 3D abstractions. More importantly, our experiment shows that it is hard for a network to generalize from indoor training data to outdoor 3D tasks, due to the significant domain gaps.

3.3.3 Exploring HoliCity

Our goal is to develop a large-scale outdoor 3D dataset that is rich of holistic structural information. To this end, HoliCity uses commercially available CAD models provided by AccuCities, which are reconstructed and built using photogrammetry from high-resolution aerial imagery, each with accurately-recorded GPS position, height, tilt, pitch, and roll.



(a) Annotations per Image (b) Reprojection Error (c) Plane Occurrence

Figure 3.5: Statistics of HoliCity. We show the number of annotations per panorama for registration (3.5a); the reprojection errors of annotated 3D points on panoramas (3.5b) and the occurrence of planes on different panoramas (3.5c).

Aerial photogrammetry is a mature technique and it has been widely used to build models with different levels of details for city planning in the field of geographic information systems (GIS). As a result, we are able to get the CAD models that cover a wide range of city areas. The CAD models we use contain details of building features with up to 15 cm accuracy, according to the provider.

To make the CAD models useful for image-based tasks, we need to precisely align the CAD models with images taken from the ground. To do that, we collect the panorama images from Google Street View that cover the same area, along with their geotags. To increase the alignment accuracy between the panoramas and the CAD models, we implement an annotation software for precise localization (See Section 3.3.4 for details).

Table 3.1 summarizes the difference between our dataset and the existing ones. Compared to the previous 3D outdoor datasets, HoliCity has its advantage on the following aspects:

Holistic Structures. With CAD models, it is straightforward to extract high-quality holistic structures such as corners, lines, planes, and even curved surfaces from HoliCity compared to the point clouds, as shown in the second row of Figure 3.4. HoliCity also supports traditional low-level representations such as depth maps and normal maps (Figure 3.3e), as well as rendering the maps of semantics annotations of roads, buildings, curbs, sky, water, and others, which have already been annotated in the CAD model. In contrast, most existing outdoor datasets use point clouds as the representation. Due to the limitation of LiDAR technology and costs, point clouds are often too sparse and noisy for an algorithm to extract such high-level structures reliably. Hardly any of existing outdoor datasets provide high-level structure annotations such as lines, wireframes, segmented surfaces, and identified buildings. Ground truth semantic segmentation also needs to be labeled manually afterward [98, 155].

Coverage. Compared to other datasets, HoliCity is able to cover a much larger area of more than 20 km^2 in downtown London with more diverse urban scenes and viewpoints,

thanks to the existing CAD models and street-view panoramas. Figure 3.3b shows the coverage map that is aligned with Google Maps and compares it against the Oxford RobotCar dataset. HoliCity contains 6,300 panorama images from diverse viewpoints. We note that it might look like that our dataset has fewer images than other datasets, it is actually fairly large among panorama-based ones, such as Stanfold-2D-3D [146] (1,413 images), and SUN3D [154] (6,161 images). For the datasets in Table 3.1 with much higher image counts, their images are mostly extracted from videos, which are highly repetitive and blurry. Therefore, we think "coverage" is a more fair metric for evaluating the size and variety of a dataset, especially considering that our dataset already have had a reasonable density of viewpoints as seen in Figure 3.3b.

Accuracy. We carefully align the panoramas with the CAD model using a reasonable number of annotated correspondence points between them, as shown in Figure 3.5a. Because the original geolocation of Google Street View images is not precise, we re-estimate the camera pose by minimizing the reprojection error of our annotations. Figure 3.5b shows the reprojection error of annotated points between the images and the CAD model. We find that for an average image, the median reprojection error is less than half a degree and the 95th percentile does not exceed 1.2 degrees. Besides accurate camera registration, our CAD model-based dataset does not have constraints on maximum depth, unlike the depth obtained from LiDAR. Hence it is more suitable for evaluating image-based 3D reconstruction algorithms.

Panorama. HoliCity uses panorama images from Google Street View with resolution 13312×6656. This way, our dataset can capture the full view from each viewpoint and it is not biased towards any directions or landmarks. It also gives us extra flexibility to render many times more perspective images and emulate cameras of different types. In contrast, images from previous outdoor datasets are mainly captured by the front-facing cameras mainly towards roads. The field of views (FoVs) is limited and the area of interest is biased.

Multi-View. The number of occurrences of each 3D plane in our panorama database is shown in Figure 3.5c. More than half of the planes occur in more than one image and about a third of planes occur in more than two images. This means that our dataset can support the 3D vision research that requires multi-view correspondence between images, e.g., structure-from-motion, multi-view stereopsis, and neural renderings.

Time Span. Most of existing 3D outdoor datasets are collected in short periods of time, as shown in the row "time span" of Table 3.1. In contrast, HoliCity utilizes the panorama images from Google Street View, which are captured during a span of over 10 years. This greatly increases the variety of data, which can benefit learning-based methods and bring additional challenges to tasks.

3.3.4 Building HoliCity

3.3.4.1 City Data Collection

3D Models. Although there exist many public city CAD models from the GIS community [156] and municipality governments, determining their quality is hard as these datasets are built for different purposes. In this project, we use the commercially available CAD model from AccuCities. Their CAD model covers the area of downtown London and comes with two levels of details. The low-resolution version (cover 20 km^2) has details accurate to 2m, while the high-resolution version (covers 4 km^2) are accurate to 15cm in all three axes. The CAD model is stored in the mesh format and each surface is tagged with semantic types such as `BUILDING`, `TERRAIN`, `BRIDGE`, `TREE`, etc.

Street-View Images. We collect street-view panorama images from Google Street View. At each viewpoint, we have a 360° panorama image along with the geographic data of the camera from GPS and IMUs: 1) latitude and longitude in WGS84 coordinate; 2) azimuth, the angle between the forward-up plane of the camera and geographic north; 3) accelerating direction, which normally is the direction of gravity. The geographic information along from Google Street View is not sufficient for accurately registering the camera pose between the CAD model and the panorama images. First, we do not have the elevation of the camera. We estimate the initial z of the camera by adding 2.5m (the height of the camera) to the ground elevation, as terrains are provided in the CAD model. Second, the provided GPS and IMU data along are not accurate enough for an accurate alignment between the images and the CAD model. Therefore, we resort to human annotation for registration.

3.3.4.2 Annotation Pipeline

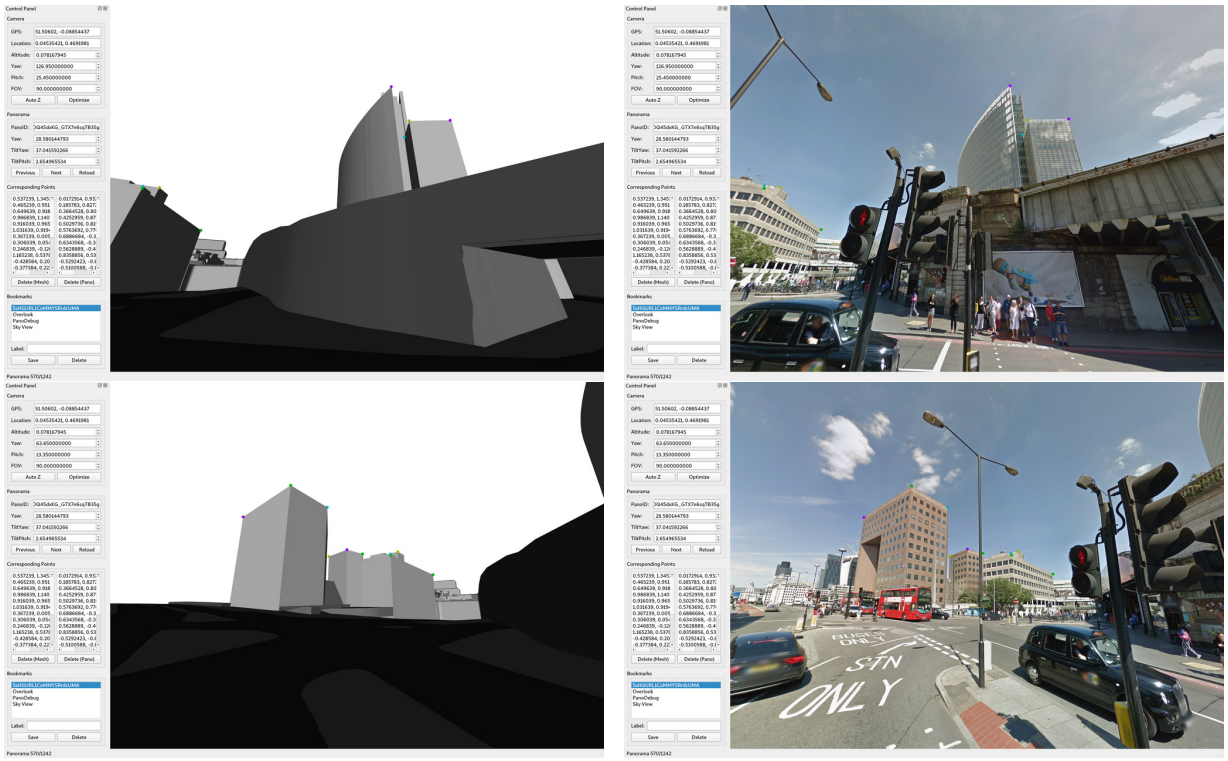
Geo-tagging the CAD Models. In the first step, we register the CAD model with the WGS84 coordinate used by Google Street View. To do that, we annotate 44 corresponding 2D locations on both Google Maps and our CAD model. We label most of the points on the inner corners of roof ridges to maximize the registration accuracy. We employ a nonlinear mesh deformation model for registration. Let $\mathbf{X}_{\text{WGS}}$ and $\mathbf{X}_{\text{CAD}}$ be the 2D coordinates of the points on Google Maps and our CAD models and Γ be the mapping from $\mathbf{X}_{\text{CAD}}$ to $\mathbf{X}_{\text{WGS}}$ parameterized by Ω. Mathematically, we have

$$\Gamma(\mathbf{X}_{\text{CAD}}, \Omega) = \mathbf{X}_{\text{WGS}} \tag{3.1}$$

Here, we use $\Omega[x, y] \in \mathbb{R}^2$ is a 2D lookup table and Γ simply bilinear interpolates Ω and returns $\Omega[\mathbf{X}_{\text{CAD}}]$. We can find the optimal $\hat{\Omega}$ by optimizing

$$\min \|\Gamma(\mathbf{X}_{\text{CAD}}, \Omega) - \mathbf{X}_{\text{WGS}}\|_2^2 + \lambda \|\Delta\Omega\|_F^2, \tag{3.2}$$

where $\Delta\Omega$ is the Laplacian of Ω. The Laplacian term serves as a regularization to keep the transformation smooth. The objective function is convex, so we can find the global



(a) UI when annotating the 3D model. (b) UI when annotating panorama images.

Figure 3.6: User interface of the HoliCity annotation tool.

optimal solution. We do a 44-fold cross-validation to determine the best λ. The final average and maximum errors are 39cm and 1.5m in the cross-validation, respectively. For reverse mapping from the WGS84 coordinate to the CAD model, we simply use the Newton-Gaussian algorithm .

Per-Image Fine-Tuning. In the second step, we fine-tune the camera pose. For each image, we first ask annotators whether it is indoor or outdoor. We discard all the indoor images. Next, we let annotators label pairs of corresponding points on the CAD model and the panoramas We provide a labeling software so that an annotator can switch between the 3D model and the images, click to add points, and optimize camera poses to minimize reprojection errors. We instruct the annotator to only put points on roof corners if possible, as our CAD model is made from aerial images so that the geometry of roofs are more reliable. We ask annotators to label at least 8 pairs of points for each view if possible.

Because we have a good initialization of the camera pose from IMU data, we apply Levenberg–Marquardt algorithm to compute the camera pose that minimizes the reprojection error of the corresponding points. Mathematically, let $\mathbf{x}_i \in \mathbb{S}^3$ be the unit vector of the ray direction of the ith labeled point on the panorama image and $\mathbf{X}_i \in \mathbb{R}^3$ be the coordinate of the corresponding labeled vertex in the CAD world space. The problem can be formulated

(a) RGB image of HoliCity (input) | (b) Ground truth of HoliCity (segment) | (c) [11] trained on HoliCity (segment) | (d) [11] trained on ScanNet (segment) | (e) [157] trained on HoliCity (segment) | (f) [157] trained on ScanNet (segment) | (g) [39] trained on SYNTHIA (segment) | (h) Ground truth of HoliCity (normal) | (i) [50] trained on HoliCity (normal) | (j) [50] trained on ScanNet (normal)

Figure 3.7: Qualitative results of models evaluated on HoliCity. We test models of MaskR-CNN [11], Associative Embedding [157], PlaneRecover [39], and UNet [50] that are trained on HoliCity, ScanNet [98], and SYNTHIA [147] on HoliCity.

(a) RGB image of MegaDepth (input) | (b) [11] trained on HoliCity (segment) | (c) [11] trained on ScanNet (segment) | (d) [157] trained on HoliCity (normal) | (d) [157] trained on ScanNet (normal) | (e) [39] trained on SYNTHIA (segment) | (f) [50] trained on HoliCity (normal) | (g) [50] trained on ScanNet (normal)

Figure 3.8: Qualitative results of models evaluated on images from the MegaDepth dataset [136]. We test models of MaskRCNN [11], Associative Embedding [157] and UNet [50] trained on HoliCity, ScanNet, and SYNTHIA. Models are *not* fine-tuned on the targeting dataset (MegaDepth).

as finding the best 6-DoF camera pose Θ (parameterized by its location, azimuth, and up direction) that minimizes the reprojection error:

$$\min_{\Theta} \sum_{i=1}^{n} \arccos^2 \left(\langle \mathbf{x}_i, \mathbf{P}_{\Theta}(\mathbf{X}_i) \rangle \right), \tag{3.3}$$

where $\mathbf{P}_{\Theta}$ projects the world-space coordinate to the panorama space $\mathbb{S}^3$ with respect to the camera pose Θ.

(a) RGB image of SYNTHIA (input) | (a) Ground truth of SYNTHIA (segment) | (b) [11] trained on HoliCity (segment) | (c) [39] trained on SYNTHIA (segment) | (a) RGB image of SYNTHIA (input) | (a) Ground truth of SYNTHIA (segment) | (b) [11] trained on HoliCity (segment) | (c) [39] trained on SYNTHIA (segment)

Figure 3.9: Qualitative results of models evaluated on the SYNTHIA dataset. We test our MaskRCNN model and PlaneRecover [39] trained on HoliCity and SYNTHIA, respectively.

3.3.5 Experiments

In this section, we will justify the necessity of data platforms based on CAD models, e.g., HoliCity, for 3D vision research. We conduct experiments on the tasks of *surface segmentation* (high-level representation) and *surface normal estimation* (low-level representation) to demonstrate the use of HoliCity and study of its generalizability from and to other datasets. The reason we choose surface segmentation and normal estimation is because previously researchers hardly test their methods on outdoor environments for these tasks. For example, existing works on surface normal estimation [21, 140, 141, 142] only demonstrate their results on indoor scenes. We hypothesize that this is because the quasi-dense and noisy points clouds from outdoor datasets cannot reliably provide the direction of surface normal.

For surface segmentation, algorithms take RGB images as input and predict regions that are considered as a continuous smooth surface, as shown in the second row of Figure 3.4. Surface segmentation is useful for applications in AR/VR such as object placement. It can be viewed as generalized plane detection [32], in which curved surfaces are also included in addition to flat planes. Prior to HoliCity, methods of plane detection are designed for indoor datasets [32, 157, 158] or synthetic urban scenes [39] only, probably because it is too hard to extract high-quality ground truth planes from noisy point clouds in real-world outdoor datasets (Section 3.3.2). We note that the uses of HoliCity are not limited to the aforementioned tasks. With its CAD models, researchers have the freedom to process and convert our data into a wide range of representations and extract the structures they need.

3.3.5.1 Data Processing

Splits. We provide two different splits of viewpoints as training, validation, and testing sets:

1. data are split randomly for tasks such as relocalization;

2. data are split according to x and y coordinates so that there is no spatial overlap between each set. We use it to study the generalizability on tasks such as normal estimation and surface segmentation.

Rendering. As most existing algorithms take perspective images as input, we provide the perspective renderings for all the viewpoints. For each panorama, we sample 8 views with evenly-spaced (45 degrees apart) yaw angles and randomly sampled pitch angles between 0 and 45 degrees. We use the camera with a 90-degree field of view and render the images with resolution 512×512. We render depth maps, normal maps, and semantic segmentation (Figures 3.3e and 3.4) from the CAD model with the same specifications using OpenGL.

Surface Segmentation. One advantage of HoliCity over traditional LiDAR-based outdoor datasets such as KITTI [144] and RobotCar [139] is that the CAD model from HoliCity could provide a structured and accurate representation of surfaces, which makes extracting high-level representations more reliable. Here, we briefly describe our algorithm of extracting the surface segmentation from HoliCity. The sampled results are shown in the second row of Figure 3.4.

The CAD model in our dataset is represented as a set of polygons of surfaces. We do a breadth-first-search (BFS) compute the surface segment of each polygon. For each nearby polygon visited during BFS, we add it into the current segments if the (approximated) curvature at the intersection line between the adjacent polygons is less than a threshold. This threshold controls the minimal curvature required for splitting a surface segment. Because the provided CAD model is not a perfect manifold, we treat two polygons as neighbors if there exists a vertex on each of them whose distance is smaller than a threshold distance. This distance also controls the granularity of the resulting segments. Increasing its value removes small segments.

3.3.5.2 Settings and Baselines

Although it is hard to directly extract high-quality surface segments and normal maps from traditional outdoor datasets, it is still possible to train models on an indoor or synthetic outdoor dataset and then apply them to a real-world outdoor dataset. Therefore, we design experiments to evaluate the feasibility of such an approach and justify the necessity of HoliCity. Besides, we test how well the model trained on HoliCity can generalize to other street-view datasets such as MegaDepth [136].

Datasets. We use HoliCity (ours), ScanNet [98] (indoor), SYNTHIA (synthetic outdoor) as the training datasets. We evaluate the trained models on images from HoliCity, MegaDepth [136], and SYNTHIA. We perform both qualitative and quantitative comparison on HoliCity and SYNTHIA, while we only perform the qualitative comparison on street-view images of MegaDepth because the ground truth surface segmentation and surface normal are not provided.

Methods	Training Datasets	Surface Segmentation			Normal Est.
		AP_{50}	AP_{75}	mAP	Mean Error
MaskRCNN [11]	HoliCity	42.0	19.8	21.9	
	ScanNet	5.0	0.6	1.7	
Associative Embedding [157]	HoliCity	20.2	8.5	9.9	
	ScanNet	3.3	0.6	1.1	
UNet [50]	HoliCity				22.6°
	ScanNet				46.3°

Table 3.2: Results of surface segmentation and normal estimation evaluated on the validation split of HoliCity. Methods are trained on HoliCity (our dataset), ScanNet (indoor dataset) [98], and SYNTHIA [147] (synthetic outdoor dataset) and tested on HoliCity *without fine-tuning*. We report the AP metrics for surface segmentation and mean angular error for normal estimation.

Surface Segmentation. We include three baseline methods: MaskRCNN [11], Associative Embedding [157], and PlaneRecover [39]. MaskRCNN is the most popular method for instance segmentation. We use the implementation from Detectron2 [159] and train the models by ourselves. Associative Embedding is a method for indoor plane detection. We use its official pre-trained model on ScanNet and retrain the Associative Embedding model on HoliCity from scratches. PlaneRecover is an approach designed for SYNTHIA [147]. We evaluate its official pre-trained model.

Normal Estimation. We report the performance of UNet [50]. We train the models on all datasets by ourselves.

3.3.5.3 Results and Discussions

We show the qualitative results evaluated on the HoliCity dataset of multiple methods in Figure 3.7, in which we trained the models of MaskRCNN [11], Associative Embedding [157], PlaneRecover [39], and UNet [50] on HoliCity (ours) ScanNet [98] (indoor dataset), and SYNTHIA [147] (synthetic outdoor dataset) on the task of surface segmentation and normal estimation. We find that for both tasks methods trained on ScanNet and SYNTHIA do not generalize well to HoliCity, which is probably due to the domain gap between training sets and testing sets. This can also be verified by the quantitative metrics in Table 3.2. We can see that the methods trained on indoor or synthetic outdoor datasets perform much worse on real-world outdoor scenes than the methods trained on HoliCity. We conclude that for existing methods such as MaskRCNN and Associative Embedding, a dataset such as HoliCity is necessary for the tasks of surface segmentation and normal estimation in outdoor environments.

Training Datasets (Methods)	Testing Datasets (AP_{50})	
	HoliCity	SYNTHIA
HoliCity (MaskRCNN [11])	42.0	36.1
SYNTHIA (PlaneRecover [39])	1.90	40.6

Table 3.3: Results of surface segmentation cross-trained and evaluated on the validation split of HoliCity and SYNTHIA [147]. We test our MaskRCNN model [11] trained on HoliCity and the official PlaneRecover model trained on SYNTHIA from [39]. Models are *not fine-tuned* on testing datasets.

We also conduct the cross-dataset experiment on HoliCity and synthetic SYNTHIA datasets for surface segmentation. In this experiment, we use the official plane detection model trained on SYNTHIA from [39] and train the MaskRCNN [11] model on HoliCity. Then, we evaluate both models on HoliCity and SYNTHIA. We show the quantitative results in Table 3.3. We find that the model trained on HoliCity can generalize to a synthetic outdoor dataset such as SYNTHIA well, while the model trained on SYNTHIA completely fails on HoliCity. Such observations also apply to the qualitative results in Figures 3.7 and 3.9, where the HoliCity-trained model recovers most of the building surfaces in SYNTHIA despite the differences between the definitions of surface segments in HoliCity and planes in [39]. We hypothesize that the causes of these phenomena are due to the wider variety of scenes covered by HoliCity, compared to the scenes from SYNTHIA.

In fact, methods trained on ScanNet and SYNTHIA do not generalize well to HoliCity, nor to other outdoor datasets such as MegaDepth [136], as shown in Figure 3.8. In comparison, methods trained on HoliCity produce much better surface segmentation and normal maps, which shows HoliCity's potential generalizability to general outdoor imagery. Finally, we summarize our observations as follows:

1. previous research of plane detection and normal estimation hardly experiments on outdoor datasets;
2. HoliCity can provide both high-quality holistic structures (e.g., surface segments) and low-level representations (e.g., normal maps) of urban environments;
3. models trained on indoor or synthetic outdoor datasets cannot generalize well to real-world outdoor datasets;
4. models trained on HoliCity can generalize to both synthetic outdoor scenes and real-world street-view imagery from different datasets.

Chapter 4

Structure-Based 3D Parsing

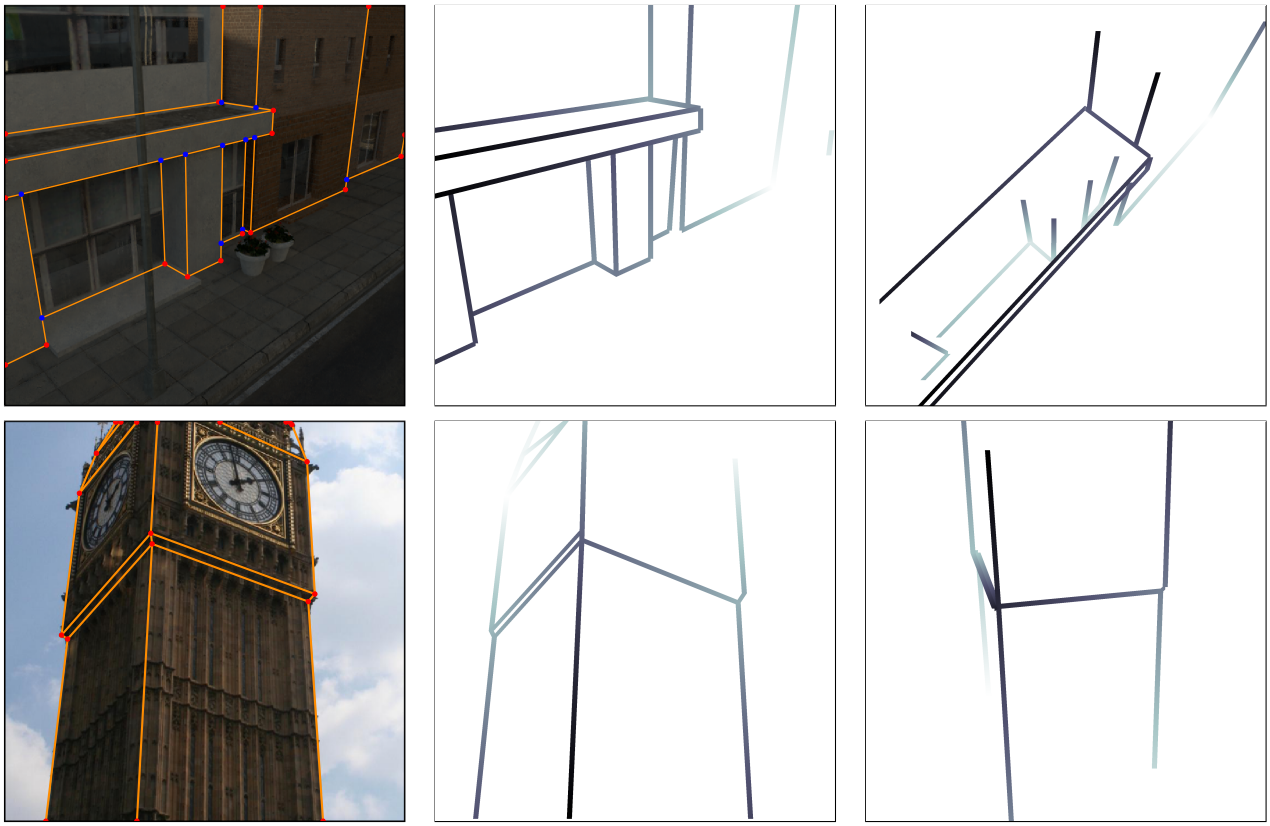
In this chapter, we will describe two works on applying detected geometric structures to 3D parsing. In the first work (Section 4.1), we train a convolutional neural network to detect junctions, straight lines, and vanishing points, and reconstruct the scene in a 3D CAD-quality wireframe representation. This method has been published in [59]. In the second work (Section 4.2), we apply the detected reflection symmetry (Section 2.3) to dense depth reconstruction, most of which has been published in [37].

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

4.1.1 Introduction

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

On the other hand, traditional image-based 3D reconstruction methods, such as Structure from Motion (SfM) and visual SLAM, often rely on local features. Although the efficiency and reliability have been improving (e.g., Microsoft Hololens, Magic Leap), they often need multiple cameras with depth sensors [160] for better accuracy. The final scene representation remains quasi-dense point clouds, which are typically incomplete, noisy, and cumbersome to store and share. Consequently, complex post-processing techniques such as plane-fitting [8] and mesh refinement [9, 161] are required. Such traditional representations can hardly meet



(a) Input image (b) 3D wireframe (c) Novel view

Figure 4.1: Results of our method tested on a synthetic image (top row) and a real image (bottom row). Column a shows the input images overlaid with the groundtruth wireframes, in which the red and blue dots represent the C- and T-type junctions, respectively. Column b shows the predicted 3D wireframe from our system, with grayscale visualizing depth. Column c shows alternative views of b. Note that our system recovers geometrically salient wireframes, without being affected by the textural lines, e.g., the vertical textural patterns on the Big Ben facade.

the increasing demand for high-level 3D modeling, content editing, and model sharing from hand-held cameras, mobile phones, and even drones.

Unlike conventional 3D geometry capturing systems, the human visual system does not perceive the world as uniformly distributed points. Instead, humans are remarkably effective, efficient, and robust in utilizing geometrically salient *global* structures such as lines, contours, planes, and smooth surfaces to perceive 3D scenes [24]. However, it remains challenging for vision algorithms to detect and utilize such global structures from local image features, until recent advances in deep learning which makes learning high-level features possible from labeled data. The examples include detecting planes [32], 2D wireframes [31], room layouts [31], and key points [162, 163].

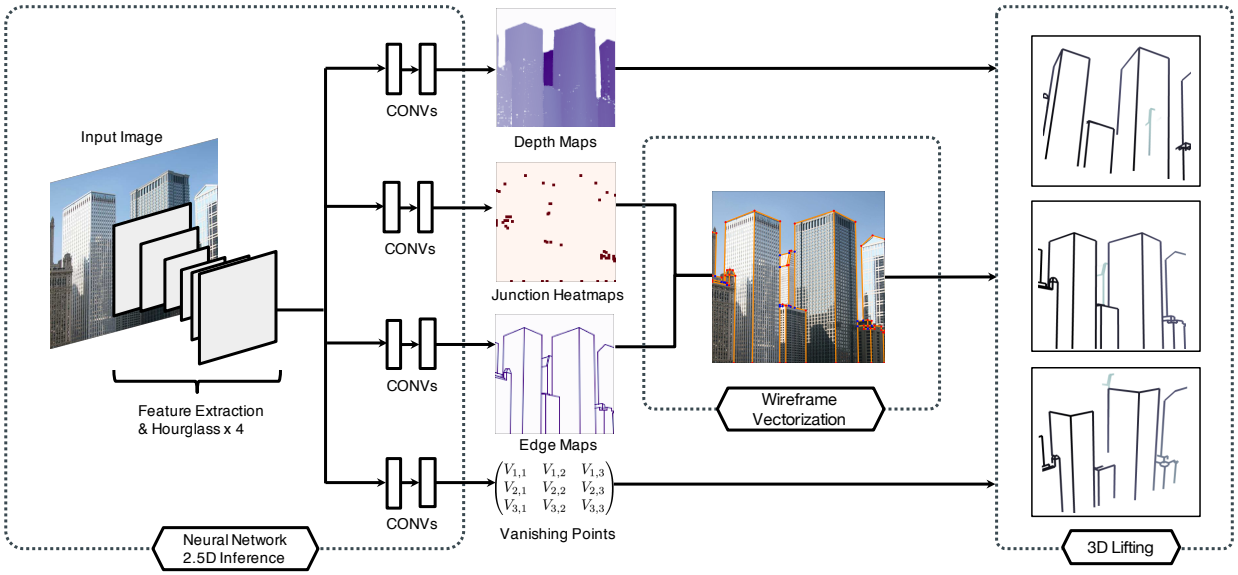


Figure 4.2: Overall pipeline of the proposed method.

In this work, we infer global 3D scene layouts from learned line and junction features, as opposed to local corner-like features such as SIFT [164], ORB [165], or line segments [58, 166] used in conventional SfM or visual SLAM systems. Our algorithm learns to detect a special type of wireframes that consist of junctions and lines representing the corners and edges of buildings. We call our representation the *geometric wireframe* and demonstrate that together with global priors, the wireframe representation allows effective and accurate recovery of the scene's 3D geometry, even from a single input image. Our method trains a neural network to estimate global lines and two types of junctions with depths, and constructs full 3D wireframes using the estimated depths and geometric constraints.

Previously, there have been efforts trying to understand the indoor scenes with the help of the 3D synthetic datasets such as the SUNCG [18]. Our work aims at natural urban environments with a variety of geometries and textures. To this end, we build two new datasets containing both synthetic and natural urban scenes. Figure 4.1 shows the sampled results of the reconstruction and Figure 4.2 shows the full pipeline of our system.

Contributions. Comparing to existing wireframe detection algorithms such as [31], our method

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

4.1.2 Methods

As depicted in Figure 4.2, our system starts with a neural network that takes a single image as input and jointly predicts multiple 2D heatmaps, from which we vectorize lines and junctions as well as estimate their initial depth values and vanishing points. We call this intermediate result a 2.5D wireframe. Using both the depth values and vanishing points estimated from the same network as the prior, we then lift the wireframe from the 2.5D *image-space* into the full 3D *world-space*.

4.1.2.1 Geometric Representations

In a geometric wireframe $\mathsf{W} = (\mathsf{V}, \mathsf{E})$ of the scene, V and $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ are the junctions and lines. Specifically, E represents lines from physical intersections of two planes while V represents (physical or projective) intersections of lines among E. Unlike [31], our E totally excludes planar textural lines, such as the vertical textures of Big Ben in Figure 4.1. The so-defined W aims to capture global scene geometry instead of local textural details.[1] By ruling out planar textural lines, we can group the junctions into two categories. Let $J_i \in \{C, T\}$ be the junction type of i, in which each junction can either be a *C-junction* ($J_i = C$) or a *T-junction* ($J_i = T$). Corner C-junctions are actual intersections of physical planes or edges, while T-junctions are generated by occlusion. Examples of T-junctions (in blue) and C-junctions (in red) can be found in Figure 4.1. We denote them as two disjoint sets $\mathsf{V} = \mathsf{V}_C \cup \mathsf{V}_T$, in which $\mathsf{V}_C = \{i \in \mathsf{V} \mid J_i = C\}$ and $\mathsf{V}_T = \{i \in \mathsf{V} \mid J_i = T\}$. We note that the number of lines incident to a T-junction in E is always 1 rather than 3 because a T-junction do not connect to the two foreground vertices in 3D. Junction types are important for inferring 3D wireframe geometry, as different 3D priors will be applied to each type.[2] For each C-junction $i_c \in \mathsf{V}_C$, define z_{i_c} as the depth of vertex i_c, i.e., the z coordinate of i_c in the *camera space*. For each occlusional T-junction $i_t \in \mathsf{V}_T$, we define z_{i_t} as the depth on the occluded line in the background because the foreground line depth can always be recovered from other junctions. With depth information, 3D wireframes that are made of C-junctions, T-junctions, and lines give a compact representation of the scene geometry. Reconstructing such 3D wireframes from a single image is our goal.

4.1.2.2 From a Single Image to 2.5D Representations

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

[1] In urban scenes, lines from regular textures (such as windows on a facade) do encode accurate scene geometry [167]. The neural network can still use them for inferring the wireframe but only not to keep them in the final output, which is designed to give a compact representation of the geometry only.

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

Given the image I of a scene, the pixel-wise outputs of our neural network consist of five outputs − junction probability J, junction offset $\mathbf{O}$, edge probability E, junction depth $\mathcal{D}$, and vanishing points $\mathbf{V}$:

$$Y \doteq (J, \mathbf{O}, E, \mathcal{D}, \mathbf{V}), \quad \hat{Y} \doteq (\hat{J}, \hat{\mathbf{O}}, \hat{E}, \hat{\mathcal{D}}, \hat{\mathbf{V}}), \tag{4.1}$$

where symbols with and without hats represent the ground truth and the prediction from the neural network, respectively.

Network Design. Our network structure is based on the stacked hourglass network [49]. The input images are cropped and re-scaled to 512×512 before entering the network. The feature-extracting module, the first part of the network, includes strided convolution layers and one max pooling layer to downsample the feature map to 128×128. The following part consists of S hourglass modules. Each module will gradually downsample then upsample the feature map. The stacked hourglass network will gradually refine the output map to match the supervision from the training data. Let the output of the jth hourglass module given the ith image be $F_j(I_i)$. During the training stage, the total loss to minimize is:

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

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

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

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

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

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

$$\hat{J}_t(\mathbf{b}) = \begin{cases} 1 & \exists i \in \mathsf{V}_t : \mathbf{b} = \lfloor \frac{i}{4} \rfloor \\ 0 & \text{otherwise} \end{cases}, \ t \in \{C, T\}.$$

where $\mathbf{b}$ is the integer coordinate on the heatmap and i is the coordinate of a junction with type t in the image space. Following [49], the resolution of the junction heatmap is 4 times less than the resolution of the input image.

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

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

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

Offset Map O and Loss $L_{\mathbf{O}}$. Comparing to the input image, the lower resolution of J might affect the precision of junction positions. We use an offset map to store the difference vector from $\hat{J}$ to its original position with sub-pixel accuracy:

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

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

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

where $\mathbf{O}_t(\mathbf{b})$ is computed by applying a sigmoid and constant translation function to the last layer of the offset branch in the neural network to enforce $\mathbf{O}_t(\mathbf{b}) \in [0, 1)^2$. We normalize $L_{\mathbf{O}}$ by the number of junctions of each type.

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

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

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

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

Junction Depth Maps $\mathcal{D}$ and Loss $L_{\mathcal{D}}$. To estimate the depth z_i for each junction i, we define the junction-wise depth map as

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

In many datasets with unknown depth units and camera intrinsic matrix K, z_i remains a relative scale instead of absolute depth. To remove the ambiguity from global scaling, we use scale-invariant loss (SILog) which has been introduced in the single image depth estimation literature [15]. It removes the influence of the global scale by summing the log difference between each pixel pair.

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

Vanishing Point Map V and Loss $L_{\mathbf{V}}$. Lines in man-made outdoor scenes often cluster around the three mutually orthogonal directions. Let $i \in \{1, 2, 3\}$ represent these three directions. In perspective geometry, parallel lines in direction i will intersect at the same vanishing point $(V_{i,x}, V_{i,y})$ in the image space, possibly at infinity. To prevent $V_{i,x}$ or $V_{i,y}$ from becoming too large, we normalize the vector so that

$$\mathbf{V}_i = \frac{1}{V_{i,x}^2 + V_{i,y}^2 + 1} \left[V_{i,x}, V_{i,y}, 1 \right]^T. \tag{4.2}$$

Because the two horizontal vanishing points $\mathbf{V}_1$ and $\mathbf{V}_2$ are order agnostic from a single RGB image, we use the Chamfer ℓ_2-loss for $\mathbf{V}_1$ and $\mathbf{V}_2$, and the ℓ_2-loss for $\mathbf{V}_3$ (the vertical vanishing point):

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

4.1.2.3 Heatmap Vectorization

As seen from Figure 4.2, the outputs of the neural network are essentially image-space 2.5D heatmaps of the desired wireframe. Vecterization is needed to obtain a compact wireframe representation.

Junction Vectorization. Recovering the junctions V from the junction heatmaps J is straightforward. Let γ_C and γ_T be the thresholds for J_C and J_T. The junction candidate sets can be estimated as

$$\mathsf{V}_t \leftarrow \{\mathbf{b} + \mathbf{O}_t(\mathbf{b}) \mid J_t(\mathbf{b}) \geq \gamma_t\}, \; t \in \{C, T\}. \tag{4.3}$$

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

In the second stage, we construct all the lines between "T-T" and "T-C" junction pairs. We repeatedly add a T-junction to the wireframe if it is tested to be close to a detected line. Unlike corner C-junctions, the degree of a T-junction is always one. So for each T-junction, we find the best edge associated with it. This process is repeated until no more lines could be added. Finally, we run a post-processing procedure to remove lines that are too close or cross each other. By handling C-junctions and T-junctions separately, our line vectorization algorithm is both efficient and robust for scenes with hundreds of lines.

4.1.2.4 Image-Space 2.5D to World-Space 3D

So far, we have obtained vectorized junctions and lines in 2.5D image space with depth in a relative scale. However, in scenarios such as AR and 3D design, absolute depth values are necessary for 6DoF manipulation of the 3D wireframe. In this section, we present the steps to estimate them with our network predicted vanishing points.

Calibration from Vanishing Points. In datasets such as MegaDepth [136], the camera calibration matrix $K \in \mathbb{R}^{3\times 3}$ of each image is unknown, although it is critical for a full 3D wireframe reconstruction. Fortunately, calibration matrices can be inferred from three mutually orthogonal vanishing points if the scenes are mostly Manhattan. According to [129], if we transform the orthogonal vanishing points $\mathbf{V}_i$ to the calibrated coordinates $\bar{\mathbf{V}}_i \doteq K^{-1}\mathbf{V}_i$, then $\bar{\mathbf{V}}_i$ should be mutually orthogonal, i.e.,

$$\mathbf{V}_i K^{-T} K^{-1} \mathbf{V}_j = 0, \quad \forall i, j \in \{1,2,3\},\ i \neq j.$$

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

Depth Refinement with Vanishing Points. Due to the estimation error, the predicted depth map may not be consistent with the detected vanishing points $\mathbf{V}_i$. In practice, we find the neural network performs better on estimating the vanishing points than predicting the 2.5D depth map. This is probably because there are more geometric cues for the vanishing points, while estimating depth requires priors from data. Furthermore, the unit of the depth map might be unknown due to the dataset (e.g., MegaDepth) and the usage of SILog loss. Therefore, we use the vanishing points to refine the junction depth and determine its absolute

value. Let $\tilde{z}_i \doteq \mathcal{D}_{J_i}(i)$ be the predicted depth for junction i from our neural network. We design the following convex objective function:

$$\begin{aligned}
\min_{z,\alpha} \quad & \sum_{i=1}^{3} \sum_{(j,i)\in \mathsf{A}_i} \left\| (z_j \bar{j} - z_i \bar{i}) \times \bar{\mathbf{V}}_i \right\|_2 \\
& \qquad + \lambda_R \sum_{i \in \mathsf{V}} \|z_i - \alpha \tilde{z}_i\|_2^2 && (4.4) \\
\text{subject to} \quad & z_i \geq 1, \quad \forall i \in \mathsf{V}, && (4.5) \\
& \lambda z_j + (1-\lambda) z_i \leq z_k, && (4.6) \\
& \forall k \in \mathsf{V}_T, (j,i) \in \mathsf{E} : k = \lambda j + (1-\lambda) i,
\end{aligned}$$

where A_i represents the set of lines corresponding to vanishing point i; α resolves the scale ambiguity in the depth dimension; $\bar{j} \doteq K^{-1}[u_x\, u_y\, 1]^T$ is the vertex position in the calibrated coordinate. The goal of the first term in Equation (4.4) is to encourage the line $(z_j \bar{j}, z_k \bar{k})$ parallel to vanishing point $\bar{\mathbf{V}}_i$ by penalizing over the parallelogram area spanned by those two vectors. The second term regularizes z_i so that it is close to the network's estimation $\tilde{z}_i$ up to a scale. Equation (4.5) prevents the degenerating solution $\mathbf{z} = \mathbf{0}$. Equation (4.6) is a convex relaxation of $\frac{\lambda}{z_j} + \frac{1-\lambda}{z_k} \geq \frac{1}{z_i}$, the depth constraint for T-junctions.

4.1.3 Experiments

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

4.1.3.1 Implementation Details

Our backbone is a two-stack hourglass network [49]. Each stack consists of 6 stride-2 residual blocks and 6 nearest neighbour upsamplers. After the stacked hourglass feature extractor, we insert different "head" modules for each map. Each head contains a 3×3 convolutional layer to reduce the number of channels followed by a 1×1 convolutional layer to compute the corresponding map. For vanishing point regression, we use a different head with two consecutive stride-2 convolution layers followed by a global average pooling layer and a fully-connected layer to regress the position of the vanishing points.

During the training, the ADAM [54] optimizer is used. The learning rate and weight decay are set to 8×10^{-4} and 1×10^{-5}. All the experiments are conducted on four NVIDIA GTX 1080Ti GPUs, with each GPU holding 12 mini-batches. For the SceneCity Urban 3D dataset, we train our network for 25 epochs. The loss weights are set as $\lambda_J = 2.0$, $\lambda_{\mathbf{O}} = 0.25$ $\lambda_E = 3.0$, and $\lambda_{\mathcal{D}} = 0.1$ so that all the loss terms are roughly equal. For the

real-world dataset, we initialize the network with the one trained on the SU3 dataset and use a 10^{-4} learning rate to train for 5 epochs. We horizontally flip the input image as data-augmentation. Unless otherwise stated, the input images are cropped to 512×512. The final output is of stride 4, i.e., with size 128×128. During heatmap vectorization, we use the hyper-parameter $\gamma_C = 0.2$, $\gamma_T = 0.3$, and $\gamma_E = 0.65$.

4.1.3.2 Evaluation Metrics

We use the standard AP (average precision) from object detection [56] to evaluate our junction prediction results. Our algorithm produces a set of junctions and their associated scores. The prediction is considered correct if its ℓ^2 distance to the nearest ground truth is within a threshold. By this criterion, we can draw the precision-recall curve and compute the *mean AP* (mAP) as the area under this curve averaging over several different thresholds of junction distance.

In our implementation, mAP is averaged over thresholds 0.5, 1.0, and 2.0. In practical applications, long edges between junctions are typically preferred over short ones. Therefore, we weight the mAP metric by the sum of the length of the lines connected to that junction. We use AP^C and AP^T to represent such weighted mAP metric for C-junctions and T-junctions, respectively. We use the intersection over union (IoU) metric to evaluate the quality of line heatmaps. For junction depth map, we evaluate it on the positions of the ground truth junctions with the scale invariant logarithmic error (SILog) [15, 144].

4.1.3.3 Ablation on Joint Training and Loss Functions

We run a series of experiments to investigate how different feature designs and multi-task learning strategies affect the wireframe detection accuracy. Table 4.1 presents our ablation studies with different combinations of tasks to research the effects of joint training. We also evaluate the choice of ℓ_1- and ℓ_2-losses for offset regression and the ordinary loss [136] for depth estimation. We conclude that:

1. Regressing offset is significantly important for localizing junctions (7.4 points for AP^C and 3 points for AP^T), by comparing rows (a-c). In addition, ℓ_2 loss is better than ℓ_1 loss, probably due to its smoothness.
2. Joint training junctions and lines improve in both tasks. Rows (c-e) show improvements with about 1.5 points in AP^C, and 0.9 point in AP^T and line IoU. This indicates the tight relation between junctions and lines.
3. For depth estimation, we test the ordinal loss from [136]. To our surprise, it does not improve the performance on our dataset (rows (f-g)). We hypothesis that this is because the relative orders of sparsely annotated junctions are harder to predict than the foreground/background relationship in [136].
4. According to rows (f) and (h), joint training with junctions and lines slightly improves the performance of depth estimation by 0.55 SILOG point.

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

Table 4.1: Ablation study of multi-task learning on 3D wireframe parsing. The columns under "supervisions" indicate what losses and supervisions are used during training; the columns under "metrics" indicate the performance given such supervision during evaluation. The second row shows the symbols of the feature maps; the third row shows the loss function names of the corresponding maps. "CE" stands for the cross entropy loss, "SILog" loss is proposed by [15], and "Ord" represents the ordinary loss in [136]. "/" indicates that the maps are not generated and thus not evaluable.

4.1.3.4 Comparison with 2D Wireframe Extraction

One recent work related to our system is [31], which extracts 2D wireframes from single RGB images. However, it has several fundamental differences from ours: 1) It does not differentiate between corner C-junctions and occluding T-junctions. 2) Its outputs are only 2D wireframes while ours are 3D. 3) It trains two separated networks for detecting junctions and lines. 4) It detects texture lines while ours only detects geometric wireframes.

In this experiment, we compare the performance with [31]. The goal of this experiment is to validate the importance of joint training. Therefore we follow the exact same training procedure and vectorization algorithms as in [31] except for the unified objective function and network structure. Figure 4.3 shows the comparison of precision and recall curves evaluated on the test images, using the same evaluation metrics as in [31]. Note that due to different network designs, their model has about 30M parameters while ours only has 19M. With fewer parameters, our system achieves 4-point AP improvement over [31] on the 2D wireframe detection task.

As a sanity check, we also train our network separately for lines and junctions, as shown by the green curve in Figure 4.3. The result is only slightly better than [31]. This experiment shows that our performance gain is from jointly trained objectives instead of neural network engineering.

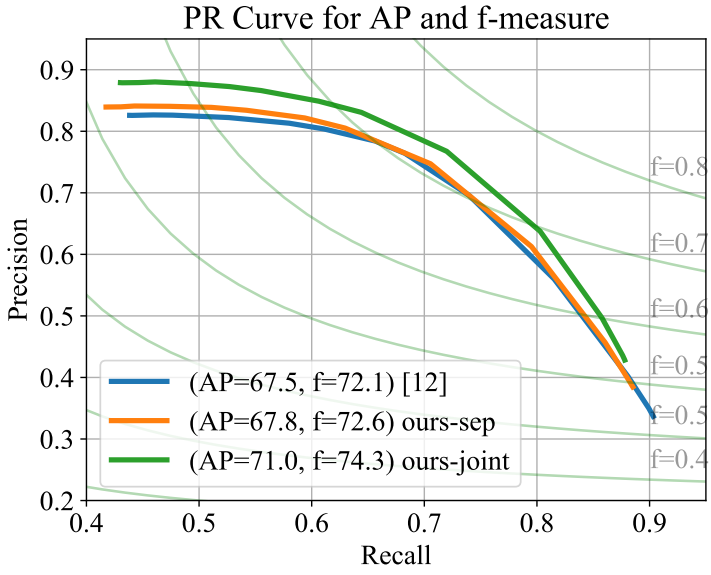


Figure 4.3: Comparison with [31] on 2D wireframe detection. We improve the baseline method by 4 points.

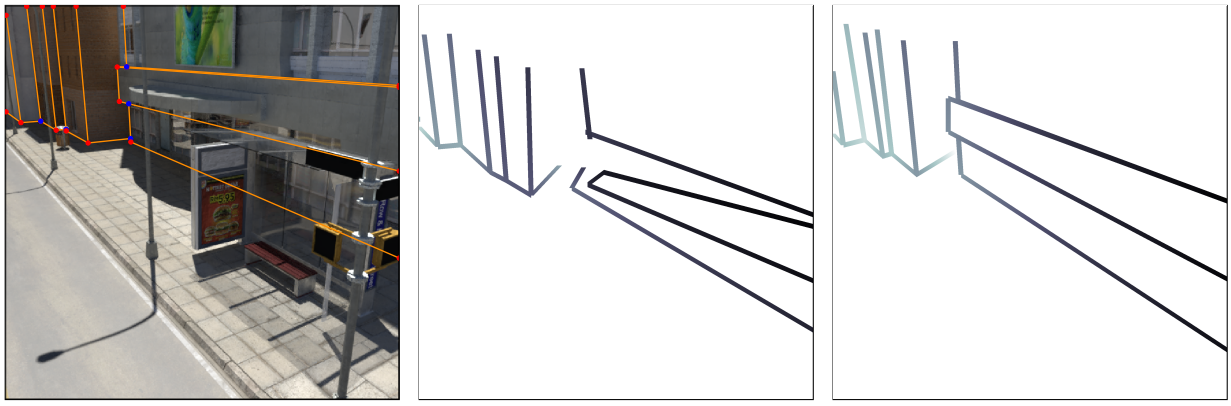
	avg[E_{V}]	med[E_{V}]	avg[E_{f}]	med[E_{f}]	failures
Ours	**2.69°**	1.55°	**4.02%**	1.38%	**2.3%**
[45, 143]	4.65°	**0.14°**	12.40%	**0.21%**	20.0%

Table 4.2: Performance comparison between our method and LSD/J-linkage [45, 143] for vanishing point detection. E_{V} represents the angular error of $\mathbf{V}_i$ in degree, E_{f} represents the relative error of the recovered camera focal lengths, and "failures" represents the percentage of cases whose $E_{\text{V}} > 8°$.

4.1.3.5 Vanishing Points and Depth Refinement

In Section 4.1.2.4, vanishing point estimation and depth refinement are used in the last stage of the 3D wireframe representation. Their robustness and precision are critical to the final quality of the system output. In this section, we conduct experiments to evaluate the performance of these methods.

For vanishing point detection, Table 4.2 shows the performance comparison between our neural network-based method and the J-Linkage clustering algorithm [67, 143] with the LSD line detector [45] on the SU3 dataset. We find that our method is more robust in term of the percentage of failures and average error, while the traditional line cluster algorithm is more accurate when it does not fail. This is because LSD/J-linkage applies a stronger geometric prior, while the neural network learns the concept from the data. We choose our method for its simplicity and robustness, as the focus of this project is more on the 3D wireframe



(a) Ground truth (b) Before refinement (c) After refinement

Figure 4.4: (b) shows a rendering of the wireframe from $\tilde{z}_i$ from a slightly different view, while (c) shows the wireframe improved by the optimization in Section 4.1.2.4.

representation side, but we believe the performance can be further improved by engineering a hybrid algorithm or designing a better network structure.

We also compare the error of the junction depth before and after depth refinement in term of SILog. We find that on 65% of the testing cases, the error is smaller after the refinement. This shows that the geometric constraints from vanishing points does help improve the accuracy of the junction depth in general. As shown in Figure 4.4, the depth refinement also improves the visual quality of the 3D wireframe. On the other hand, the depth refinement may not be as effective when the vanishing points are not precise enough, or the scene is too complex so that there are many erroneous lines in the wireframe.

4.1.3.6 3D Wireframe Reconstruction Results

We test our 3D wireframe reconstruction method on both the synthetic dataset and the real images. Examples illustrating the visual quality of the final reconstruction are shown in Figures 4.5 and 4.6. A video demonstration can be found in `http://y2u.be/l3sUtddPJPY`. We do not show the ground truth 3D wireframes for the real landmark dataset due to its incomplete depth maps.

4.2 Learning to Estimate Depth from Reflection Symmetry

4.2.1 Introduction

3D reconstruction is one of the most long-lasting problems in computer vision. Although commercially available solutions using time-of-flight cameras or structured lights may be able to meet the needs of specific purposes, they are often expensive, have limited range,

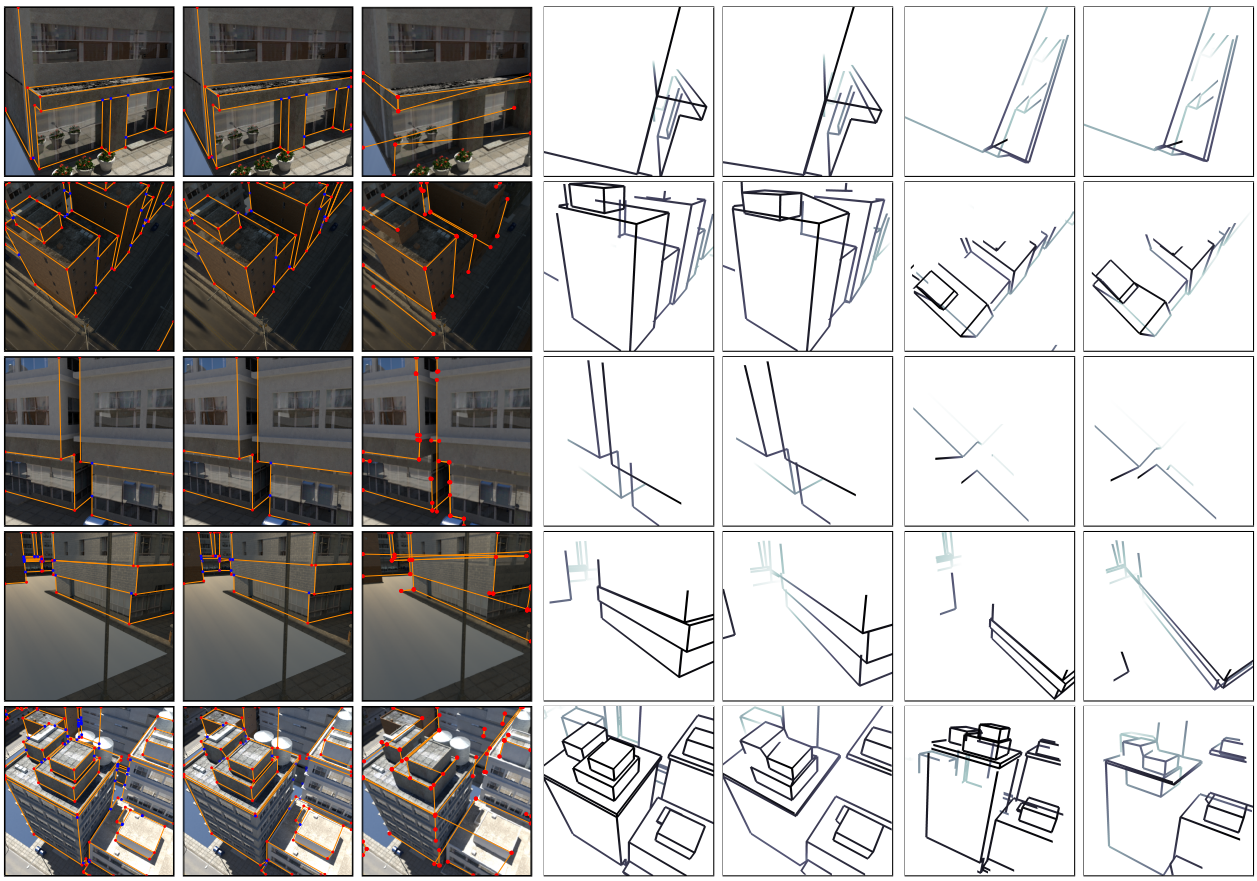


Figure 4.5: Left group: comparison of 2D results between the ground truth (column 1), our predictions (column 2), and the results from wireframe parser [31] (column 3). Middle (columns 4-5) and right groups (columns 6-7): novel views of the ground truths and our reconstructions to demonstrate the 3D representation of the scene. The color of the wireframes visualizes depth.

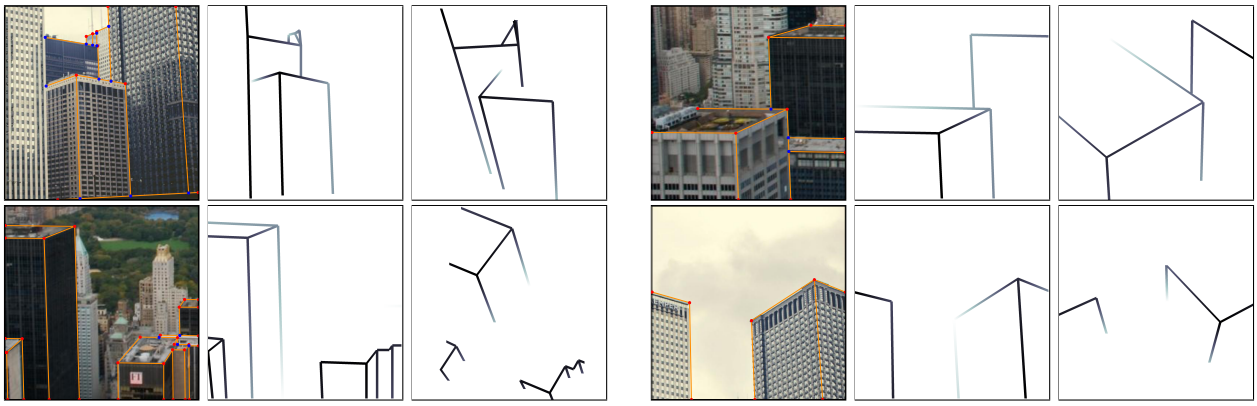


Figure 4.6: Results of 3D wireframe on real images from MegaDepth.

and can be interfered by other light sources. On the other hand, traditional image-based 3D reconstruction methods, such as structure-from-motion (SfM), visual SLAM, and multi-view stereopsis (MVS) use only RGB images and recover the underlying 3D geometry [168].

Recent advances in convolutional neural networks (CNNs) have shown good potential in inferring dense 3D depth maps from RGB images by leveraging supervised learning. Deep learning-based multi-view stereo and optical flow methods employ 3D CNNs to regularize the cost volumes that are built upon geometric structures to infer depth maps and have shown the state-of-the-art results on various benchmarks [169]. Due to the popularity of mobile apps, 3D inference from a single image starts to draw increasing attention. Nowadays, learning-based single-view reconstruction methods are able to predict 3D shapes in various representations with an encoder-decoder CNN that extrapolates objects from seen patterns during the test time. However, unlike multi-view stereopsis, previous single-view methods hardly exploit the geometric constraints between the input RGB image and the resulting 3D shape. Hence, the formulation is ill-posed, and leads to inaccurate 3D shape recovery and limited generalization capability [22].

To address the illness, we identify a structure that commonly exists in man-made objects, the *reflection symmetry*, as a geometric connection between the depth maps and the images, and incorporate it as a prior into deep networks through plane-sweep cost volumes built from features of corresponding pixels, aiming to faithfully recover 3D shapes from single-view images under the principle of shape-from-symmetry. To this end, our method combines the strength of learning-based recognition and geometry-based reconstruction methods. Specifically, we first detect the parameters of the mirror plane from the image with a coarse-to-fine strategy and then recovers the depth from reflective stereopsis. The network consists of a backbone feature extractor, a differentiable warping module for building the 3D cost volumes, and a cost volume network. This framework naturally enables neural networks to utilize the information from corresponding pixels of reflection symmetry inside a single image.

4.2.2 Related Work

Learning-Based Single-View 3D Reconstruction. Inspired by the success of CNNs in classification and detection, numerous 3D representations and associated learning schemes have been explored under the setting of single-view 3D reconstruction, including depth maps [20, 127], voxels [149, 170], point clouds [150], signed distance fields (SDF) [151], and meshes [131]. Although these methods demonstrate promising results on some datasets, single-view reconstruction is essentially an *ill-posed* problem. Without geometric constraints, inferred shapes *will not be accurate enough* by extrapolating from training data, especially for unseen objects. To alleviate this issue, our method leverages the symmetry prior for accurate single-view 3D reconstruction.

Multi-View Stereopsis. Traditional multi-view stereo methods build the cost volumes from photometric measures of images, regularize the cost volume, and post-process to recover the depth maps [171, 172, 173, 174]. Recent efforts leverage learning-based methods and have

shown promising results on benchmarks [144] with CNNs. Some directly learn patch correspondence between two [175] or more views [176]. Others build plane-sweep cost volumes from image features and employ 3D CNNs to regularize cost volumes, which then can be either transformed into 3D representations [130, 177] or aggregated into depth maps [169, 178, 179, 180]. Different from these methods, our approach builds the plane-sweep cost volume through a symmetry-based feature warping module, which makes such a powerful tool applicable to the single-image setting.

4.2.3 Methods

Our method is based on NeRD in Section 2.3. We modify it so that the neural network is able to fulfill two tasks simultaneously: *symmetry detection* and *depth estimation*. The algorithm takes a single image as input and outputs parameters of the symmetry plane and depth maps. We note that our method treats the symmetry as a prior rather than a hard constraint, so the pipeline still works for objects that are not perfectly symmetric.

As cost volumes (i.e., depth probability tensors) has already been used in the symmetry detection pipeline (Section 2.3.3.5), it is straightforward to use it for a geometry-based depth estimation. With the estimated $\mathbf{w}^*$, we compute the expectation of depth from the probability tensor $\mathbf{P}$ as the depth map prediction $\hat{\mathbf{D}}$. This is sometimes referred as *soft argmin* [181]. Mathematically, we have

$$\hat{\mathbf{D}}(x, y) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} d\mathbf{P}(x, y, d). \tag{4.7}$$

During training, we rescale the ground truth depth according to $\|\hat{\mathbf{w}}\|_2$ and add an additional ℓ_1 term to the training loss as the supervision of depth:

$$L_{\text{dpt}} = \frac{1}{n} \sum_{x,y} \left| \hat{\mathbf{D}}(x, y) - \mathbf{D}(x, y) \right|, \tag{4.8}$$

where n is the number of pixels.

4.2.4 Experiments

We conduct experiments on the ShapeNet dataset [109]. We use the same settings and hyper-parameter as in Section 2.3.4. Despite we use additional branches in the cost volume network, the total inference speed remains around 1 image per second per GPU.

4.2.4.1 Results

We compare our method with popular monocular depth estimation networks [49, 127, 182, 183] and shape reconstruction networks [126, 131]. The results on the task of *depth estimation* are shown in Table 4.3. NeRD outperforms both monocular depth estimation networks

	absRel	sqRel	rmse	mae	$< \delta^1$	$< \delta^2$	$< \delta^3$
DORN [127]	0.028	0.0014	0.026	0.020	30.8%	54.1%	69.0%
GeoNet [182]	0.028	0.0013	0.025	0.019	29.7%	53.4%	69.2%
Hourglass [49]	0.026	0.0012	0.024	0.018	33.0%	56.9%	71.5%
DenseDepth [183]	0.024	0.0011	0.022	0.017	36.3%	60.5%	74.6%
Pixel2Mesh [131]	0.102	0.0546	0.032	0.073	28.6%	49.2%	62.3%
DISN [126]	0.040	0.0030	0.038	0.028	24.0%	43.4%	57.8%
NeRD	**0.019**	**0.0009**	**0.021**	**0.011**	**49.5%**	**71.9%**	**82.3%**
NeRD*	0.015	0.0006	0.018	0.011	60.2%	78.7%	86.5%

Table 4.3: Quantitative results on the task of depth estimation on ShapeNet. Here, $\delta = 1.01$. NeRD* uses the ground truth symmetry plane as input.

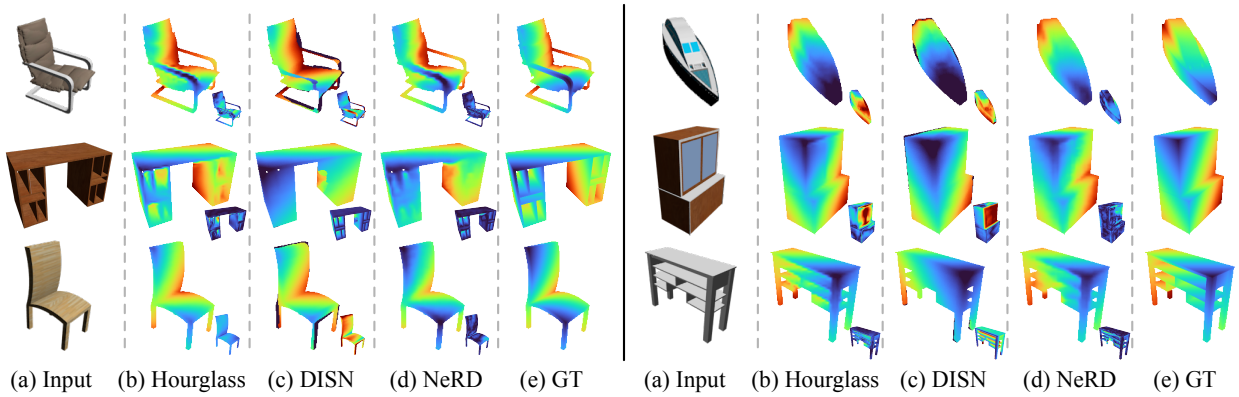


Figure 4.7: Qualitative results on the task of of depth estimation. We visualize the depth maps from Pixel2Mesh [131], DISN [126], and our NeRD on ShapeNet. The per-pixel ℓ_1 errors are plotted at the lower right corner of each depth map. Bluish color represents smaller values for error and depth.

and shape reconstruction networks. Besides, NeRD*, the variant of NeRD that uses the ground truth symmetry plane instead of the one predicted in coarse-to-fine inference, only slightly outperforms the standard NeRD. These behaviors indicate that detecting symmetry planes and incorporating photo-consistency priors of reflection symmetry into the neural network makes the task of single-view reconstruction less ill-posed and thus can improve the performance.

4.2.4.2 Visualization

In Figure 4.7, we show sampled results of depth maps. Visually, NeRD gives sharp and accurate results among all the tested methods. For example, it is able to capture the details

of desk frames and the shapes of ship cabins, while those details are more blurry in the results of other methods. Results from the hourglass network are also sharp but are less accurate, which shows the signs of overfitting. In the region such as the chair armrests and table legs, NeRD can recover the depth more accurate compared to the baseline methods. This is because for NeRD, pixel-matching based on photo-consistency in those areas is easy and can provide a strong signal, while other baseline methods need to extrapolate from the training data.

Chapter 5

Conclusion

In this dissertation, I have studied the problem of 3D parsing from both data-driven and geometric perspectives. I have presented a series of works on how to apply a geometry-based learning framework to detect high-level structures for 3D parsing. More specifically, I have designed novel neural networks that understand geometric structures such as lines, junctions, planes, vanishing points, and bilateral symmetry, and accurately detect them from images (Chapter 2); I have collected multiple 3D datasets to support future research of structure extraction and structure-based scene understanding (Chapter 3); and I have demonstrated how to use detected geometric structures to parse scenes (Chapter 4). I find that with proper datasets and carefully designed neural network architectures, exploiting the structural regularities that widely exist in man-made environments not only improves the performance of learning-based algorithms, but also brings compact 3D representations that are semantically meaningful, easy to interpret, and efficient to share.

5.1 Future Work

In this section, I will discuss some interesting topics that are relevant to my dissertation and worth exploring further.

Learning-Based Multi-View Geometry. In NeRD (Section 2.3), we have addressed the problem of detecting the camera pose with respect to the symmetry plane of an object. A highly related problem is determining the 6-DoF camera poses from a set of images, which traditionally is solved with point features [38] and bundle adjustment [7] since the late '90s, as mentioned in the introduction. The problem is especially hard when the baseline is large, which is often the case in SLAM systems to detect the loop [165]. A recent benchmark [184] has found that with deep learning-based point feature extractors, the overall performance of 6-DoF pose recovery barely improves when compared to the pipeline using traditional SIFT features [1], if not worsens. However, according to the evidence shown in Chapter 2, I believe that when doing correctly, a learning-based multi-view geometry algorithm should

bring considerable performance improvement. There might be two routes down the road. We can use neural networks to extract better low-level or even high-level geometric features (e.g., points, lines, vanishing points, and symmetries) and match them to get the association, similar to the methods in the survey [184]. It is also possible to design neural networks that take two images and directly output a 6-DoF camera pose, which might result in better accuracy due to a more end-to-end pipeline. For methods in both categories, I think the important thing is to properly incorporate geometric structures into the neural network to achieve an efficient, robust, and accurate algorithm.

Image-Based CAD Model Reconstruction. In the wireframe project (Sections 2.1 and 4.1), we have reconstructed the scene in a compact wireframe representation. The ultimate goal here is to reconstruct the scene in a high-level editable form, such as the boundary representation (B-Rep) [185] that is commonly used in CAD solid modeling. One thing that is currently missing is planar structures, which are vital for real-world applications. We have shown that neural networks are able to detect planar information in the form of a pixel-wise mask in HoliCity (Section 3.3). It is worthwhile to explore how to turn such pixel-wise representations into or directly predict a vectorized planar representation, and merge it with the 3D wireframe to get an integrated CAD representation. Moreover, man-made objects often consist of curved lines and curved surfaces. It is necessary to sort out the geometric relationship between 3D curved objects and their 2D projection and have the ability to detect 2D and 3D curved structures.

SLAM/SfM in Deep Learning Software Systems. In Chapter 2, I have worked on multiple problems related to detecting geometric structures from images. It is possible to apply detected structures to help SfM/SLAM systems after building the correspondences. For example, researchers have used vanishing points as a visual compass for navigation [74]. During my research, I found it time-consuming to design and test neural networks under the framework of SLAM/SfM, as the pipeline of SLAM is often quite complex and written in C++ [137]. Currently, we already have several high-quality tensor algebra frameworks for deep learning, such as TensorFlow, PyTorch, and recently JAX. On the one hand, operators in deep-learning algebraic frameworks are often differentiable, so it is convenient to use gradient descent to fine-tune hyper-parameters in SLAM/SfM systems and add learning-based ingredients. On the other hand, modern deep-learning algebraic frameworks use a scripting language such as Python as their interface, which significantly reduces the time for modifying existing code and implementing new ideas. Because we have witnessed the power of data-driven methods, and the current trend is to replace more and more components in the traditional vision system with their learning-based counterparts for performance improvements, I believe such a system can benefit research across multiple topics in 3D vision, such as learning-based point features extraction, outlier rejection, view-geometry, visual odometry, and multiview stereopsis.

Bibliography

[1] Relja Arandjelović and Andrew Zisserman. "Three Things Everyone Should Know to Improve Object Retrieval". In: *CVPR*. 2012.

[2] Ethan Rublee et al. "ORB: An Efficient Alternative to SIFT or SURF". In: *ICCV*. 2011.

[3] Jianbo Shi and Carlo Tomasi. "Good Features to Track". In: *CVPR*. 1994.

[4] David Nistér. "An Efficient Solution to the Five-Point Relative Pose Problem". In: *PAMI* (2004).

[5] Xiao-Shan Gao et al. "Complete Solution Classification for the Perspective-Three-Point Problem". In: *PAMI* (2003).

[6] Ondřej Chum, Jiří Matas, and Josef Kittler. "Locally Optimized RANSAC". In: *Joint Pattern Recognition Symposium*. 2003.

[7] Bill Triggs et al. "Bundle Adjustment — A Modern Synthesis". In: *International Workshop On Vision Algorithms*. 1999.

[8] Jingwei Huang et al. "3Dlite: Towards Commodity 3D Scanning for Content Creation". In: *ToG* (2017).

[9] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson Surface Reconstruction". In: *SGP*. 2006.

[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CVPR*. 2016.

[11] Kaiming He et al. "Mask R-CNN". In: *ICCV*. 2017.

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CVPR*. 2015.

[13] Yu Xiang et al. "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes". In: *CVPR*. 2019.

[14] Arsalan Mousavian et al. "3D Bounding Box Estimation Using Deep Learning and Geometry". In: *CVPR*. 2017.

[15] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth Map Prediction From a Single Image Using a Multi-Scale Deep Network". In: *NIPS*. 2014.

[16] Charles Ruizhongtai Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *NIPS*. 2017.

[17] Shichen Liu et al. "Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning". In: *ICCV*. 2019.

[18] Shuran Song et al. "Semantic Scene Completion From a Single Depth Image". In: *CVPR*. 2017.

[19] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *CVPR*. 2020.

[20] Jia-Ren Chang and Yong-Sheng Chen. "Pyramid Stereo Matching Network". In: *CVPR*. 2018.

[21] David Eigen and Rob Fergus. "Predicting Depth, Surface Normals and Semantic Labels With a Common Multi-Scale Convolutional Architecture". In: *CVPR*. 2015.

[22] Maxim Tatarchenko et al. "What Do Single-View 3D Reconstruction Networks Learn?" In: *CVPR*. 2019.

[23] Sen Wang et al. "DeepVO: Towards End-to-End Visual Odometry With Deep Recurrent Convolutional Neural Networks". In: *ICRA*. 2017.

[24] Maxwell B Clowes. "On Seeing Things". In: *Artificial intelligence* (1971).

[25] Gilbert Falk. "Interpretation of Imperfect Line Data as a Three-Dimensional Scene". In: *Artificial intelligence* (1972).

[26] Stephen T Barnard. "Interpreting Perspective Images". In: *Artificial intelligence* (1983).

[27] Kent A Stevens. "The Visual Interpretation of Surface Contours". In: *Artificial Intelligence* (1981).

[28] Takeo Kanade. "Recovery of the Three-Dimensional Shape of an Object From a Single View". In: *Artificial intelligence* (1981).

[29] Lawrence G Roberts. "Machine Perception of Three-Dimensional Solids". PhD thesis. Massachusetts Institute of Technology, 1963.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification With Deep Convolutional Neural Networks". In: *NIPS*. 2012, pp. 1097–1105.

[31] Kun Huang et al. "Learning to Parse Wireframes in Images of Man-Made Environments". In: *CVPR*. 2018.

[32] Chen Liu et al. "PlaneNet: Piece-Wise Planar Reconstruction From a Single RGB Image". In: *CVPR*. 2018.

[33] Chuhang Zou et al. "LayoutNet: Reconstructing the 3D Room Layout From a Single RGB Image". In: *CVPR*. 2018.

[34] Huayi Zeng et al. "Bundle Pooling for Polygonal Architecture Segmentation Problem". In: *CVPR*. 2020.

[35] Yichao Zhou, Haozhi Qi, and Yi Ma. "End-to-End Wireframe Parsing". In: *ICCV*. 2019.

[36] Yichao Zhou et al. "NeurVPS: Neural Vanishing Point Scanning via Conic Convolution". In: *NeurIPS*. 2019.

[37] Yichao Zhou, Shichen Liu, and Yi Ma. "Learning to Detect 3D Reflection Symmetry for Single-View Reconstruction". In: (2020). arXiv:2006.10042 [cs.CV].

[38] David G Lowe. "Object Recognition From Local Scale-Invariant Features". In: *ICCV*. 1999.

[39] Fengting Yang and Zihan Zhou. "Recovering 3D Planes From a Single Image via Convolutional Neural Networks". In: *ECCV*. 2018.

[40] Thibault Groueix et al. "A Papier-MâChÉ Approach to Learning 3D Surface Generation". In: *CVPR*. 2018.

[41] Nan Xue et al. "Learning Attraction Field Representation for Robust Line Segment Detection". In: *CVPR*. 2019.

[42] Jifeng Dai, Kaiming He, and Jian Sun. "Instance-Aware Semantic Segmentation via Multi-Task Network Cascades". In: *CVPR*. 2016.

[43] Ross Girshick. "Fast R-CNN". In: *Proceedings Of The IEEE International Conference On Computer Vision*. 2015.

[44] Richard S Stephens. "Probabilistic Approach to the Hough Transform". In: *Image and vision computing* (1991).

[45] Rafael Grompone Von Gioi et al. "LSD: A Fast Line Segment Detector With a False Detection Control". In: *PAMI* (2010).

[46] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks". In: *NIPS*. 2015.

[47] Hei Law and Jia Deng. "CornerNet: Detecting Objects as Paired Keypoints". In: *ECCV*. 2018.

[48] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. "Bottom-up Object Detection by Grouping Extreme and Center Points". In: *CVPR*. 2019.

[49] Alejandro Newell, Kaiyu Yang, and Jia Deng. "Stacked Hourglass Networks for Human Pose Estimation". In: *ECCV*. 2016.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *International Conference On Medical Image Computing And Computer-Assisted Intervention*. 2015.

[51] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *CVPR*. 2014.

[52] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial Transformer Networks". In: *NIPS*. 2015.

[53] Jifeng Dai et al. "Deformable Convolutional Networks". In: *ICCV*. 2017.

[54] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *ICLR* (2015).

[55] Patrick Denis, James H Elder, and Francisco J Estrada. "Efficient Edge-Based Methods for Estimating Manhattan Frames in Urban Imagery". In: *ECCV*. 2008.

[56] Mark Everingham et al. "The PASCAL Visual Object Classes (VOC) Challenge". In: *IJCV* (2010).

[57] David R Martin, Charless C Fowlkes, and Jitendra Malik. "Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues". In: *PAMI* (2004).

[58] Srikumar Ramalingam and Matthew Brand. "Lifting 3D Manhattan Lines From a Single Image". In: *ICCV*. 2013.

[59] Yichao Zhou et al. "Learning to Reconstruct 3D Manhattan Wireframes From a Single Image". In: *ICCV*. 2019.

[60] Jifeng Dai et al. "R-FCN: Object Detection via Region-Based Fully Convolutional Networks". In: *NIPS*. 2016.

[61] Yi Li et al. "Fully Convolutional Instance-Aware Semantic Segmentation". In: *CVPR*. 2017.

[62] Roberto Cipolla, Tom Drummond, and Duncan P Robertson. "Camera Calibration From Vanishing Points in Image of Architectural Scenes." In: *BMVC*. 1999.

[63] Erwan Guillou et al. "Using Vanishing Points for Camera Calibration and Coarse 3D Reconstruction From a Single Image". In: *The Visual Computer* (2000).

[64] James F O'Brien and Hany Farid. "Exposing Photo Manipulation With Inconsistent Reflections." In: *ToG* (2012).

[65] Derek Hoiem, Alexei A Efros, and Martial Hebert. "Putting Objects in Perspective". In: *IJCV* (2008).

[66] Seokju Lee et al. "VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition". In: *ICCV*. 2017.

[67] Jean-Philippe Tardif. "Non-Iterative Approach for Fast and Accurate Vanishing Point Detection". In: *ICCV*. 2009.

[68] Florian Kluger et al. "Deep Learning for Vanishing Point Detection Using an Inverse Gnomonic Projection". In: *GCPR*. 2017.

[69] Menghua Zhai, Scott Workman, and Nathan Jacobs. "Detecting Vanishing Points Using Global Image Context in a Non-Manhattan World". In: *CVPR*. 2016.

[70] Michael J Magee and Jake K Aggarwal. "Determining Vanishing Points From Perspective Images". In: *Computer Vision, Graphics, and Image Processing* (1984).

[71] Marco Straforini, C Coelho, and Marco Campani. "Extraction of Vanishing Points From Images of Indoor and Outdoor Scenes". In: *Image and Vision Computing* (1993).

[72] Long Quan and Roger Mohr. "Determining Perspective Structures Using Hierarchical Hough Transform". In: *Pattern Recognition Letters* (1989), pp. 279–286.

[73] Jana Košecká and Wei Zhang. "Efficient Computation of Vanishing Points". In: *ICRA*. 2002.

[74] Jana Košecká and Wei Zhang. "Video Compass". In: *ECCV*. Springer. 2002.

[75] Horst Wildenauer and Allan Hanbury. "Robust Camera Self-Calibration From Monocular Images of Manhattan Worlds". In: *CVPR*. 2012.

[76] Faraz M Mirzaei and Stergios I Roumeliotis. "Optimal Estimation of Vanishing Points in a Manhattan World". In: *ICCV*. 2011.

[77] Jean-Charles Bazin et al. "Globally Optimal Line Clustering and Vanishing Point Estimation in Manhattan World". In: *CVPR*. 2012.

[78] Michel Antunes and Joao P Barreto. "A Global Approach for the Detection of Vanishing Points and Mutually Orthogonal Vanishing Directions". In: *CVPR*. 2013.

[79] James M Coughlan and Alan L Yuille. "Manhattan World: Compass Direction From a Single Image by Bayesian Inference". In: *ICCV*. 1999.

[80] Olga Barinova et al. "Geometric Image Parsing in Man-Made Environments". In: *ECCV*. 2010.

[81] Grant Schindler and Frank Dellaert. "Atlanta World: An Expectation Maximization Framework for Simultaneous Low-Level Edge Grouping and Camera Calibration in Complex Man-Made Environments". In: *CVPR*. 2004.

[82] John Canny. "A Computational Approach to Edge Detection". In: *PAMI* (1986).

[83] GF McLean and D Kotturi. "Vanishing Point Detection by Line Clustering". In: *PAMI* (1995).

[84] Robert C Bolles and Martin A Fischler. "A RANSAC-Based Approach to Model Fitting and Its Application to Finding Cylinders in Range Data". In: *IJCAI*. 1981.

[85] Paul VC Hough. "Machine Analysis of Bubble Chamber Pictures". In: *International Conference On High Energy Accelerators And Instrumentation*. 1959.

[86] Zihan Zhou, Farshid Farhat, and James Z Wang. "Detecting Dominant Vanishing Points in Natural Scenes With Application to Composition-Sensitive Image Retrieval". In: *IEEE Transactions on Multimedia* (2017).

[87] Chin-Kai Chang, Jiaping Zhao, and Laurent Itti. "DeepVP: Deep Learning for Vanishing Point Detection on 1 Million Street View Images". In: *ICRA*. 2018.

[88] Ali Borji. "Vanishing Point Detection With Convolutional Neural Networks". In: *arXiv preprint* (2016).

[89] Xiaodan Zhang et al. "Dominant Vanishing Point Detection in the Wild With Application in Composition Analysis". In: *Neurocomputing* (2018).

[90] Laurent Sifre and Stéphane Mallat. "Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination". In: *CVPR*. 2013.

[91] Joan Bruna and Stéphane Mallat. "Invariant Scattering Convolution Networks". In: *PAMI* (2013).

[92] Yunho Jeon and Junmo Kim. "Active Convolution: Learning the Shape of Convolution for Image Classification". In: *CVPR*. 2017.

[93] Taco S Cohen et al. "Spherical CNNs". In: *ICLR*. 2018.

[94] Chiyu Jiang et al. "Spherical CNNs on Unstructured Grids". In: *ICLR*. 2019.

[95] Jonathan Masci et al. "Geodesic Convolutional Neural Networks on Riemannian Manifolds". In: *ICCV Workshop*. 2015.

[96] Jingwei Huang et al. "TextureNet: Consistent Local Parametrizations for Learning From High-Resolution Signals on Meshes". In: *CVPR*. 2019.

[97] Álvaro González. "Measurement of Areas on a Sphere Using Fibonacci and Latitude–Longitude Lattices". In: *Mathematical Geosciences* (2010).

[98] Angela Dai et al. "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes". In: *CVPR*. 2017.

[99] José Lezama et al. "Finding Vanishing Points via Point Alignments in Image Primal and Dual Domains". In: *CVPR*. 2014.

[100] Chen Feng, Fei Deng, and Vineet R Kamat. "Semi-Automatic 3D Reconstruction of Piecewise Planar Building Models From Single Image". In: *CONVR* (2010).

[101] Scott Workman, Menghua Zhai, and Nathan Jacobs. "Horizon Lines in the Wild". In: *BMVC*. 2016.

[102] Changhyun Choi and Henrik I Christensen. "3D Pose Estimation of Daily Objects Using an RGB-D Camera". In: *IROS*. 2012.

[103] Shuran Song and Jianxiong Xiao. "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images". In: *CVPR*. 2016.

[104] Charles R Qi et al. "Frustum Pointnets for 3D Object Detection From RGB-D Data". In: *CVPR*. 2018.

[105] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. "Real-Time Seamless Single Shot 6D Object Pose Prediction". In: *CVPR*. 2018.

[106] Mahdi Rad and Vincent Lepetit. "BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects Without Using Depth". In: *ICCV*. 2017.

[107] Yu Xiang et al. "Data-Driven 3D Voxel Patterns for Object Category Recognition". In: *CVPR*. 2015.

[108] Xiaozhi Chen et al. "Monocular 3D Object Detection for Autonomous Driving". In: *CVPR*. 2016.

[109] Angel X Chang et al. "ShapeNet: An Information-Rich 3D Model Repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[110] Xingyuan Sun et al. "Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling". In: *CVPR*. 2018.

[111] N.F. Troje and H.H. Bulthoff. "How Is Bilateral Symmetry of Human Faces Used for Recognition of Novel Views". In: *Vision Research* (1998).

[112] T. Vetter, T. Poggio, and H. H. Bulthoff. "The Importance of Symmetry and Virtual Views in Three-Dimensional Object Recognition". In: *Current Biology* (1994).

[113] Thommen Korah and Christopher Rasmussen. "Analysis of Building Textures for Reconstructing Partially Occluded Facades". In: *ECCV*. 2008.

[114] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. "Unsupervised Learning of Probably Symmetric Deformable 3D Objects From Images in the Wild". In: *CVPR*. 2020.

[115] Xuaner Cecilia Zhang et al. "Portrait Shadow Manipulation". In: *ToG* (2020).

[116] W. Hong, A. Y. Yang, and Y. Ma. "On Group Symmetry in Multiple View Geometry: Structure, Pose and Calibration From Single Images". In: *IJCV* (2004).

[117] W. Hong, Y. Yu, and Y. Ma. "Reconstruction of 3D Symmetric Curves From Perspective Images Without Discrete Features". In: *ECCV*. 2004.

[118] Yifan Xu et al. "Ladybird: Quasi-Monte Carlo Sampling for Deep Implicit Field Based 3D Reconstruction With Symmetry". In: *ECCV*. 2020.

[119] Gareth Loy and Jan-Olof Eklundh. "Detecting Symmetry and Symmetric Constellations of Features". In: *ECCV*. 2006.

[120] H. Zabrodsky, S. Peleg, and D. Avnir. "Symmetry as a Continuous Feature". In: *PAMI* (1995).

[121] N. Kiryati and Y. Gofman. "Detecting Symmetry in Grey Level Images: The Global Optimization Approach". In: *IJCV* (1998).

[122] Eldar Insafutdinov and Alexey Dosovitskiy. "Unsupervised Learning of Shape and Pose With Differentiable Point Clouds". In: *NIPS*. 2018.

[123] He Wang et al. "Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation". In: *CVPR*. 2019.

[124] Yi Zhou et al. "On the Continuity of Rotation Representations in Neural Networks". In: *CVPR*. 2019.

[125] Yuan Yao et al. "Front2Back: Single View 3D Shape Reconstruction via Front to Back Prediction". In: *CVPR*. 2020.

[126] Qiangeng Xu et al. "DISN: Deep Implicit Surface Network for High-Quality Single-View 3D Reconstruction". In: *NIPS*. 2019.

[127] Huan Fu et al. "Deep Ordinal Regression Network for Monocular Depth Estimation". In: *CVPR*. 2018.

[128] Yasutaka Furukawa and Jean Ponce. "Accurate, Dense, and Robust Multiview Stereopsis". In: *PAMI* (2009).

[129] Yi Ma et al. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer Science & Business Media, 2012.

[130] Abhishek Kar, Christian Häne, and Jitendra Malik. "Learning a Multi-View Stereo Machine". In: *NIPS*. 2017.

[131] Nanyang Wang et al. "Pixel2mesh: Generating 3D Mesh Models From Single RGB Images". In: *ECCV*. 2018.

[132] Christopher B Choy et al. "3D-R2N2: A Unified Approach for Single and Multi-View 3D Object Reconstruction". In: *ECCV*. 2016.

[133] Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. "Partial and Approximate Symmetry Detection for 3D Geometry". In: *ToG* (2006).

[134] Shinji Umeyama. "Least-Squares Estimation of Transformation Parameters Between Two Point Patterns". In: *PAMI* (1991).

[135] Yichao Zhou et al. "HoliCity: A City-Scale Data Platform for Learning Holistic 3D Structures". In: (2020). arXiv:2008.03286 [cs.CV].

[136] Zhengqi Li and Noah Snavely. "MegaDepth: Learning Single-View Depth Prediction From Internet Photos". In: *Computer Vision And Pattern Recognition (CVPR)*. 2018.

[137] Johannes L Schonberger and Jan-Michael Frahm. "Structure-From-Motion Revisited". In: *CVPR*. 2016.

[138] Brian Curless and Marc Levoy. "A Volumetric Method for Building Complex Models From Range Images". In: *ToG*. 1996.

[139] Will Maddern et al. "1 Year, 1000 Km: The Oxford RobotCar Dataset". In: *IJRR* (2017).

[140] Aayush Bansal, Bryan Russell, and Abhinav Gupta. "Marr Revisited: 2D-3D Alignment via Surface Normal Prediction". In: *CVPR*. 2016.

[141] Xiaolong Wang, David Fouhey, and Abhinav Gupta. "Designing Deep Networks for Surface Normal Estimation". In: *CVPR*. 2015.

[142] Jingwei Huang et al. "FrameNet: Learning Local Canonical Frames of 3D Surfaces From a Single RGB Image". In: *ICCV*. 2019.

[143] Roberto Toldo and Andrea Fusiello. "Robust Multiple Structures Estimation With J-Linkage". In: *European Conference On Computer Vision*. 2008.

[144] Andreas Geiger et al. "Vision Meets Robotics: The KITTI Dataset". In: *IJRR* (2013).

[145] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. "Indoor Segmentation and Support Inference From RGBD Images". In: *ECCV*. 2012.

[146] Iro Armeni et al. "Joint 2D-3D-Semantic Data for Indoor Scene Understanding". In: *arXiv* (2017).

[147] German Ros et al. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *CVPR*. 2016.

[148] Lars Mescheder et al. "Occupancy Networks: Learning 3D Reconstruction in Function Space". In: *CVPR*. 2019.

[149] Xinchen Yan et al. "Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction Without 3D Supervision". In: *NIPS*. 2016.

[150] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A Point Set Generation Network for 3D Object Reconstruction From a Single Image". In: *CVPR*. 2017.

[151] Jeong Joon Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *CVPR* (2019).

[152] John McCormac et al. "SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-Training on Indoor Segmentation?" In: *ICCV*. 2017.

[153] Stephan R Richter et al. "Playing for Data: Ground Truth From Computer Games". In: *ECCV*. 2016.

[154] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. "SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels". In: *ICCV*. 2013.

[155] Angel Chang et al. "Matterport3D: Learning From RGB-D Data in Indoor Environments". In: *3DV* (2017).

[156] Thomas H Kolbe, Gerhard Gröger, and Lutz Plümer. "CityGML: Interoperable Access to 3D City Models". In: *Geo-Information For Disaster Management*. Springer, 2005.

[157] Zehao Yu et al. "Single-Image Piece-Wise Planar 3D Reconstruction via Associative Embedding". In: *CVPR*. 2019.

[158] Chen Liu et al. "PlaneRCNN: 3D Plane Detection and Reconstruction From a Single Image". In: *CVPR*. 2019.

[159] Yuxin Wu et al. *Detectron2*. `https://github.com/facebookresearch/detectron2`. 2019.

[160] Shahram Izadi et al. "KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *ISMAR*. 2011.

[161] William E Lorensen and Harvey E Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *ToG*. 1987.

[162] Kaipeng Zhang et al. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". In: *IEEE Signal Processing Letters* (2016).

[163] Jiajun Wu et al. "Single Image 3D Interpreter Network". In: *ECCV*. 2016.

[164] Yasutaka Furukawa et al. "Manhattan-World Stereo". In: *CVPR*. 2009.

[165] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE transactions on robotics* (2015).

[166] Manuel Hofer, Michael Maurer, and Horst Bischof. "Efficient 3D Scene Abstraction Using Line Segments". In: *Computer Vision and Image Understanding* (2017).

[167] Zhengdong Zhang et al. "TILT: TRansform Invariant Low-Rank Textures". In: *IJCV* (2012).

[168] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2003.

[169] Yao Yao et al. "MVSNet: Depth Inference for Unstructured Multi-View Stereo". In: *ECCV*. 2018.

[170] Jiajun Wu et al. "Learning Shape Priors for Single-View 3D Completion and Reconstruction". In: *ECCV*. 2018.

[171] Sing Bing Kang, Richard Szeliski, and Jinxiang Chai. "Handling Occlusions in Dense Multi-View Stereo". In: *CVPR*. 2001.

[172] Yasutaka Furukawa et al. "Towards Internet-Scale Multi-View Stereo". In: *CVPR*. 2010.

[173] Yasutaka Furukawa, Carlos Hernández, et al. "Multi-View Stereo: A Tutorial". In: *Foundations and Trends® in Computer Graphics and Vision* (2015).

[174] Johannes L Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *ECCV*. 2016.

[175] Jure Zbontar, Yann LeCun, et al. "Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches." In: *JMLR* (2016).

[176] Wilfried Hartmann et al. "Learned Multi-Patch Similarity". In: *ICCV*. 2017.

[177] Mengqi Ji et al. "SurfaceNet: An End-to-End 3D Neural Network for Multiview Stereopsis". In: *ICCV*. 2017.

[178] Po-Han Huang et al. "DeepMVS: Learning Multi-View Stereopsis". In: *CVPR*. 2018.

[179] Vincent Leroy, Jean-Sébastien Franco, and Edmond Boyer. In: *ECCV*. 2018.

[180] Sungil Choi et al. "Learning Descriptor, Confidence, and Depth Estimation in Multi-View Stereo". In: *CVPR Workshops*. 2018.

[181] Alex Kendall et al. "End-to-End Learning of Geometry and Context for Deep Stereo Regression". In: *ICCV*. 2017.

[182] Zhichao Yin and Jianping Shi. "GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose". In: *CVPR*. 2018.

[183] Ibraheem Alhashim and Peter Wonka. "High Quality Monocular Depth Estimation via Transfer Learning". In: *arXiv preprint arXiv:1812.11941* (2018).

[184] Yuhe Jin et al. "Image Matching across Wide Baselines: From Paper to Practice". In: *IJCV* (2020).

[185] Martti Mantyla. *Introduction to solid modeling*. WH Freeman & Co., 1988.