

Washington University in St. Louis

Washington University Open Scholarship

McKelvey School of Engineering Theses & Dissertations

McKelvey School of Engineering

Spring 5-15-2019

Structured Indoor Modeling

Chen Liu

Washington University in St. Louis

Follow this and additional works at: https://openscholarship.wustl.edu/eng_etds



Part of the [Computer Sciences Commons](#)

Recommended Citation

Liu, Chen, "Structured Indoor Modeling" (2019). *McKelvey School of Engineering Theses & Dissertations*. 454.

https://openscholarship.wustl.edu/eng_etds/454

This Dissertation is brought to you for free and open access by the McKelvey School of Engineering at Washington University Open Scholarship. It has been accepted for inclusion in McKelvey School of Engineering Theses & Dissertations by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.

WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Computer Science and Engineering

Dissertation Examination Committee:

Tao Ju, Chair
Ayan Chakrabarti
Yasutaka Furukawa
Brendan Juba
Ulugbek Kamilov

Structured Indoor Modeling
by
Chen Liu

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

May 2019
St. Louis, Missouri

© 2019, Chen Liu

Table of Contents

List of Figures	vi
List of Tables	xii
Acknowledgments	xiii
Abstract	xv
Chapter 1: Introduction	1
1.1 Background.....	3
1.1.1 Piece-wise planar reconstruction	3
1.1.2 Floorplan reconstruction	5
1.2 Organization	8
Chapter 2: PlaneNet: Piece-wise Planar Reconstruction from a Single RGB Image	9
2.1 Introduction.....	9
2.2 PlaneNet	10
2.2.1 Plane parameter branch	11
2.2.2 Plane segmentation branch	12
2.2.3 Non-planar depth branch.....	13
2.3 Datasets and implemenation details	13
2.4 Experimental results	14
2.4.1 Plane segmentation accuracy.....	16
2.4.2 Depth reconstruction accuracy	18
2.4.3 Plane ordering consistency.....	19
2.4.4 Failure modes	22
2.5 Applications.....	24
2.6 Discussions	26

Chapter 3: PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image	27
3.1 Introduction.....	27
3.2 Approach.....	29
3.2.1 Plane Detection Network.....	29
3.2.2 Segmentation Refinement Network	32
3.2.3 Warping Loss Module	33
3.3 Benchmark construction	34
3.4 Experimental results	34
3.4.1 Qualitative evaluations	35
3.4.2 Plane reconstruction accuracy	35
3.4.3 Geometric accuracy.....	38
3.4.4 Ablation studies	38
3.4.5 Occlusion reasoning.....	39
3.5 Discussions	47
Chapter 4: Raster-to-Vector: Revisiting Floorplan Transformation	48
4.1 Introduction.....	48
4.2 Stratified floorplan representation	50
4.2.1 Junction layer	50
4.2.2 Primitive layer	51
4.3 Raster to vector conversion	52
4.3.1 Junction layer conversion via a CNN	53
4.3.2 Primitive layer conversion via IP.....	54
4.3.3 Final data conversion	57
4.4 Evaluations.....	58
4.4.1 Annotating a Floorplan Dataset	58
4.4.2 Metrics.....	60
4.4.3 Quantitative evaluations.....	60
4.4.4 Qualitative evaluations	61
4.5 Discussions	65

Chapter 5: FloorNet: A Unified Framework for Floorplan Reconstruction from 3D Scans	66
5.1 Introduction.....	66
5.2 FloorNet	68
5.2.1 Preliminaries	68
5.2.2 Triple-branch hybrid design	68
5.2.3 Inter-branch feature sharing.....	70
5.2.4 Loss functions.....	72
5.3 Floorplan reconstruction benchmark.....	72
5.4 Implementation details	74
5.4.1 DNN Training.....	74
5.4.2 Enhancement heuristics.....	75
5.5 Experiments.....	76
5.6 Discussions	81
Chapter 6: FloorPlotter: Towards Ultimate Inverse CAD System for Floorplans	83
6.1 Introduction.....	83
6.2 FloorPlotter: System Overview	85
6.3 Room-aware floorplan reconstruction.....	86
6.4 Room-wise coordinate descent.....	90
6.5 System Details.....	92
6.6 Experiments.....	94
6.6.1 Qualitative evaluations	95
6.6.2 Quantitative evaluations.....	96
6.7 Discussions	100
Chapter 7: Conclusion and Future Works	102
References	105
Appendix A: Baselines for Piece-wise Planar Reconstruction	[116]
Appendix B: Algorithm Details for Floorplan Transformation	[118]
B.1 Pre-processing Details	[118]

B.2 Post-processing Details [119]

List of Figures

Figure 1.1:	A piece-wise planar reconstruction example. From left to right, an input image, a piece-wise planar segmentation, a reconstructed depthmap, and a texture-mapped 3D model.	4
Figure 1.2:	An example of converting a raster floorplan image to the vector-graphics format. From left to right, an input floorplan image, reconstructed vector-graphics representation visualized by our custom renderer, and a popup 3D model.	6
Figure 2.1:	PlaneNet predicts plane parameters, their probabilistic segmentation masks, and a non-planar depthmap from a single RGB image.	11
Figure 2.2:	Piece-wise planar depthmap reconstruction results by PlaneNet. From left to right: input image, plane segmentation, depthmap reconstruction, and 3D rendering of our depthmap. In the plane segmentation results, the black color shows non-planar surface regions.	15
Figure 2.3:	Plane segmentation accuracy against competing baselines that use 3D points as input [65, 23, 66]. Either ground-truth depthmaps or inferred depthmaps (by a DNN-based system) are used as their inputs. PlaneNet outperforms all the other methods that use inferred depthmaps. Surprisingly, PlaneNet is even better than many other methods that use ground-truth depthmaps.	16
Figure 2.4:	Qualitative comparisons between PlaneNet and existing methods that use inferred depthmaps as the inputs. From left to right: an input image, plane segmentation results for [65], [23], [66], and PlaneNet, respectively, and the ground-truth.	18
Figure 2.5:	Room layout estimations. We have exploited the ordering consistency in the predicted planes to infer room layouts.	21
Figure 2.6:	Typical failure modes occur in the absence of enough image texture cues or at the presence of small objects and clutter.	23

Figure 2.7:	Texture editing applications. From top to bottom, an input image, a plane segmentation result, and an edited image.	25
Figure 3.2:	Our framework consists of three building blocks: 1) a plane detection network based on Mask R-CNN [28], 2) a segmentation refinement network that jointly optimizes extracted segmentation masks, and 3) a warping loss module that enforces the consistency of reconstructions with a nearby view during training.	29
Figure 3.3:	We estimate a plane normal by first picking one of the 7 anchor normals and then regressing the residual 3D vector. Anchor normals are defined by running the K-means clustering algorithm on the ground-truth plane normal vectors.	31
Figure 3.4:	Refinement network architecture. The network takes both global information (i.e., the input image, the reconstructed depthmap and the pixel-wise depthmap) and instance-specific information (i.e., the instance mask, the union of other masks, and the coordinate map of the instance) as input and refines instance mask with a U-Net architecture [61]. Each convolution in the encoder is replaced by a ConvAccu module to accumulate features from other masks.	36
Figure 3.5:	Plane-wise accuracy against baselines. PlaneRCNN outperforms all the competing methods except when the depth threshold is very small and MWS-G can fit 3D planes extremely accurately by utilizing the ground-truth depth values.	37
Figure 3.6:	Piece-wise planar reconstruction results by PlaneRCNN. From left to right: input image, plane segmentation, depthmap reconstruction, and 3D rendering of our depthmap (rendered from a new view with -0.4m and 0.3m translation along x-axis and z-axis respectively and 10 rotation along both x-axis and z-axis).....	42
Figure 3.7:	Plane segmentation results on unseen datasets without fine-tuning. From left to right: input image, PlaneNet [48] results, PlaneRecover [78] results, and ours. From top to the bottom, we show two examples from each dataset in the order of NYUv2 [65], 7-scenes [64], KITTI [26], SYNTHIA [62], Tank and Temple [38], and PhotoPopup [33].	43
Figure 3.8:	Plane segmentation comparisons. From left to right: 1) input image, 2) MWS with inferred depths, 3) MWS with ground-truth depths, 4) PlaneNet, 5) Ours, and 6) ground-truth.	44

Figure 3.9:	Effects of the surface refinement network and the warping loss module. Top: the segmentation refinement network narrows the gap between adjacent planes. Bottom: the warping loss helps to correct erroneous plane geometries from the second view.....	45
Figure 3.10:	New view synthesis results with the layered depthmap models. A simple modification allows PlaneRCNN to also infer occluded surfaces and reconstruct layered depthmap models.	46
Figure 4.1:	Our approach converts a floorplan image through two intermediate representation layers. A neural architecture first converts a floorplan image into a junction layer, where data is represented by a set of junctions (<i>i.e.</i> , points with types) or per-pixel semantic classification scores. An integer programming aggregates junctions into a set of primitives (<i>i.e.</i> , lines or boxes), while ensuring a topologically and geometrically consistent result. A simple post-processing can be used to produce the final vector format.....	49
Figure 4.2:	There are four wall junction types: I-, L-, T-, and X-shaped, depending on the degrees of incident wall segments. Considering orientations, we have in total 13 ($= 4 + 4 + 4 + 1$) types.	52
Figure 4.3:	Loop constraints can be enforced locally at each junction. The room types must be the same for each pair of walls marked with the same color.	57
Figure 4.4:	Floorplan vectorization results. From left to right, an input floorplan image, reconstructed vector-graphics representation visualized by our custom renderer, and the corresponding popup 3D model. We have manually checked the correctness of each junction and primitive estimation, whose statistics are shown under each example.	59
Figure 4.5:	Typical failure cases. Relying on Manhattan assumption, our method is unable to detect walls which are neither horizontal nor vertical as shown in the first example. In the second example, our method puts a wall which does not exist.	62
Figure 4.6:	Floorplan vectorization results on an image in Rent3D (top) and an image from the web (bottom).	64

Figure 5.1:	FloorNet consists of three DNN branches. The first branch uses PointNet [56] to directly consume 3D information. The second branch takes a top-down point density image in a floorplan domain with a fully convolutional network [53], and produces pixel-wise geometry and semantics information. The third branch produces deep image features by a dilated residual network trained on the semantic segmentation task [79] as well as a stacked hourglass CNN trained on the room layout estimation [55]. The PointNet branch and the floorplan branch exchanges intermediate features at every layer, while the image branch contributes deep image features into the decoding part of the floorplan branch. This hybrid DNN architecture effectively processes an input RGBD video with camera poses, covering a large 3D space.	69
Figure 5.2:	FloorNet shares features across branches to exploit the best of all the architectures. PointNet features at 3D points are pooled into corresponding 2D cells in the floorplan branch. Floorplan features at 2D cells are unpooled to the corresponding 3D points in the PointNet branch. Deep image features are pooled into corresponding 2D cells in the floorplan branch, based on the depthmap and the camera pose information.	71
Figure 5.3:	Floorplan reconstruction benchmark. From left to right: Subsampled video frames, colored 3D point clouds, and ground-truth floorplan data. The floorplan data is stored in a vector-graphics representation, which is visualized with a simple rendering engine (e.g., rooms are assigned different colors based on their types, and objects are shown as canonical icons).	73
Figure 5.4:	Floorplan reconstruction results.	77
Figure 5.5:	Intermediate results. For each example, we show raw outputs of the networks (room corners, icon corners, opening corners, and icon types) compared against the ground-truth. In the second example, we produce a fake room (blue color) at the top due to poor quality 3D points. In the third example, reconstructed rooms have inaccurate shapes near the bottom left again due to noisy 3D points, illustrating the challenge of our problem.	78
Figure 5.6:	Qualitative comparisons against competing methods. The top is Oct-Net [60], a state-of-the-art 3D CNN architecture. The next three rows show variants of our FloorNet, where only one branch is enabled. FloorNet with all the branches overall produce more complete and accurate floorplans.	79

Figure 5.7:	Comparison against a commercial floorplan generator, Tango Navigator App. Top: Floorplan image from Tango. Bottom: Our results.	81
Figure 6.1:	System overview: (Left) Mask-RCNN finds room segments from a top-down projection image consisting of point density and mean surface normal, allowing us to reconstruct a floorplan as multiple loops. (Middle) Room-wise coordinate descent optimizes room structures one by one by minimizing the sum of data, consistency, and model complexity terms. (Right) Simple graph merging operations combine loops into a floorplan graph structure.	86
Figure 6.2:	Illustration of data and consistency terms. E_{data}^C and E_{data}^E are defined based on corner and edge likelihood maps. Blue pixels indicate lower costs in these toy examples. E_{consis} counts the number of pixels used by room corners and room edges. When neighboring rooms share corners and edges as shown in (c), E_{consis} goes down.	89
Figure 6.3:	We solve a shortest path problem for each room, where cost functions are encoded into edge weights. In order to avoid a trivial solution (i.e., an empty graph) and enforce the path to go around the room segment (R_i), we first identify a start-edge that is a part of a room shape with high-confidence. Next, we draw a (red) start-line perpendicularly to split the domain. We prohibit crossing the start-line, assign a very high penalty for going through R_i , then solve for a shortest path that starts and ends at the two end-points of the start-edge.	91
Figure 6.4:	Our dataset offers production-level panorama RGBD scans for 527 houses/apartments. We convert each scan into a point density/normal image from a top-down view, which is the input to our system. We annotated floorplan structure as a 2D polygonal graph. Note that for point-density/normal images in the middle column, the intensity encodes the point density, and the hue/saturation encodes the 2D horizontal component of the mean surface normal.	95
Figure 6.5:	Qualitative comparisons against FloorNet [47] and the variants of our approach. We select hard non-Manhattan examples here to illustrate the reconstruction challenges in our dataset. For reconstructions by FloorPlotter variants, room colors are determined by corresponding room segments from Mask R-CNN. For the ground-truth and the FloorNet, colors are based on the room types.	98

Figure 6.6:	Multiple rounds of the coordinate descent fix mistakes at challenging floorplan structure. The top row shows the results after the first round of the coordinate descent optimization, and the bottom shows the results after the second round. We also show the total amount of energy after each round. Corresponding ground-truth annotations are found in Figure 6.5.	99
Figure 6.7:	Typical failure modes. The left is the ground-truth annotation and the right is our result. Our system still makes mistakes for challenging non-Manhattan structures.	101
Figure B.1:	A floorplan example where kitchen and living room appear in the same closed polygon. A pair of facing walls (red line and purple line) with the same room label (e.g., living room) determines that pixels colored with green belong to living room.	[120]

List of Tables

Table 2.1:	Depth accuracy comparisons over the NYUv2 dataset.....	20
Table 2.2:	Room layout estimations. Quantitative evaluations against the top-performers over the NYUv2 303 dataset.	22
Table 3.1:	Geometric accuracy comparison over the NYUv2 dataset.	39
Table 3.2:	Ablation studies on the contributions of the four components in PlaneR-CNN. Plane segmentation and detection metrics are calculated over the ScanNet dataset. PlaneNet represents the competing state-of-the-art.	40
Table 4.1:	Quantitative evaluations based on our benchmark.	61
Table 5.1:	Dataset statistics. From left to right: the number of rooms, the number of icons, the number of openings (i.e., doors or windows), the number of room-corners, and the total area. The average and the standard deviation are reported for each entry.	73
Table 5.2:	Quantitative evaluations on low-, mid-, and high-level metrics against competing methods and our variants. The orange and cyan color indicates the best and the second best result for each entry.	80
Table 5.3:	In the PointNet to floorplan inter-branch pooling, we use a mix of sum and max pooling in projecting 3D points onto the 2D floorplan domain. To validate this hybrid scheme, we have also evaluated the performance when only the max-pooling or the sum-pooling is used at all the layers.	80
Table 6.1:	The main quantitative evaluation results. The colors cyan , orange, magenta represent the top three entries.....	95

Acknowledgments

Foremost, I would like to sincerely thank my advisor, Prof. Yasutaka Furukawa, for his support and guidance during my PhD study. His incomparable enthusiasm, immense experience, and rigorous attitude towards research have deeply shaped my understanding of scholarship. Moreover, I sincerely thank him for putting my interests first throughout my journey to the PhD degree and beyond. Besides my advisor, I would also like to thank my committee chair, Prof. Tao Ju, and the rest of my thesis committee, Prof. Ayan Chakrabarti, Prof. Ulugbek Kamilov, and Prof. Brendan Juba, for their insightful comments and challenging questions.

Many thanks are due to Dr. Jimei Yang of Adobe Systems Inc. and Dr. Kihwan Kim of Nvidia Inc., who offered me the internship opportunities and much fun. Without them, much of this work would not be possible.

I thank my excellent collaborators, Pushmeet Kohli, Jiajun Wu, Jimei Yang, Kihwan Kim, Jinwei Gu, Duygu Ceylan, Ersin Yumer, Jan Kautz, Vijay Badrinarayanan, Zhao Chen, Khushi Gupta, Xinchun Yan, for the inspiring discussions and for all the fun time we worked together. I would also like to thank my fellow labmates and my friends, Hang Yan, Huayi Zeng, Zhiyang Huang, Zhicheng Cui, Yiming Kang, Hao Yan, Dan Zeng, Satoshi Ikehata, Jiacheng Chen, without whom I would have much less fun in the past five years.

Last but not the least, I would like to thank my beloved wife, Yinghong Zhang, for her endless love and her encouragement through every stage of the journey. Finally, I would like to thank my parents, Jianbing Liu and Lixia Wang, for their unconditional support and understanding. My mother, Lixia Wang, helped me annotate a lot of data, which greatly contributes to my research.

Chen Liu

Washington University in Saint Louis

May 2019

ABSTRACT OF THE DISSERTATION

Structured Indoor Modeling

by

Chen Liu

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2019

Professor Tao Ju, Chair

Professor Yasutaka Furukawa, Co-Chair

In this dissertation, we propose data-driven approaches to reconstruct 3D models for indoor scenes which are represented in a structured way (e.g., a wall is represented by a planar surface and two rooms are connected via the wall). The structured representation of models is more application ready than dense representations (e.g., a point cloud), but poses additional challenges for reconstruction since extracting structures requires high-level understanding about geometries. To address this challenging problem, we explore two common structural regularities of indoor scenes: 1) most indoor structures consist of planar surfaces (planarity), and 2) structural surfaces (e.g., walls and floor) can be represented by a 2D floorplan as a top-down view projection (orthogonality). With breakthroughs in data capturing techniques, we develop automated systems to tackle structured modeling problems, namely piece-wise planar reconstruction and floorplan reconstruction, by learning shape priors (i.e., planarity and orthogonality) from data. With structured representations and production-level quality, the reconstructed models have an immediate impact on many industrial applications.

Chapter 1

Introduction

A structured model contains structural elements constituting an indoor scene, such as walls, doors, and furniture, as well as their relationships. Such 3D models could enable novel applications in science, engineering and commerce. For example, people would never get lost in shopping malls since their mobile phone could find the route based on the structured model of the shopping mall; A new homeowner can use our system to reconstruct his or her house, and explores customized design ideas by placing virtual objects in the reconstructed 3D model; A home seller can provide 3D models of their properties so that potential buyers can have a virtual walk-in tour without the trouble of scheduling a physical one; Home assistant robots can do more housework given accurate modeling of geometric structures.

The last decades have witnessed dramatic improvements in outdoor 3D modeling, thanks to the advance in Computer Vision techniques. Major technology companies are seeking to digitally recreate the entire globe and provide outdoor mapping services. However, outdoor modeling is merely a reconstruction of building facades, where existing methods produce only a "polygon soup" as raw geometry. Indoor spaces are more challenging to reconstruct for three

main reasons: 1) textureless surfaces, such as walls and floor, fail most existing multi-view geometry reconstruction systems which requires feature extraction from textures, 2) we only have partial observations of objects due to occlusion, and 3) diverse objects are placed in a complicated way in the real world, making the solution space for a reconstruction system huge. To reconstruct 3D models for an indoor scene robustly, a reconstruction system needs to incorporate room layout estimation, occlusion reasoning, object geometry and relationship modeling, which are all open research questions.

To address these challenges, we develop novel data-driven approaches to learn common shape priors from data, from which we can derive structured models. While complex, most indoor structures are governed by the following two shape priors:

Planarity. Most indoor structural surfaces are planes. Based on this property, we build learning-based frameworks to reconstruct the piece-wise planar representation which represents an indoor scene using a set of planar surfaces. We further extend the piece-wise planar representation to multiple layers to model occlusion explicitly.

Orthogonality. Most walls are orthogonal to the ground and the ceiling, and two adjacent walls are often orthogonal. This property indicates that a floorplan, which is the top-down view projection of wall structures, is informative to describe the indoor space. We tackle floorplan reconstruction from either a rasterized image or an RGBD image sequence.

Both the above shape priors pose geometrical constraints to most indoor structures, and we work on automated systems for extracting such constrained structures from image observations, namely piece-wise planar reconstruction and floorplan reconstruction. Besides pushing the frontiers of each individual line of research, we plan to combine them to build a unified framework for indoor scene reconstruction with a structured representation. We start with

background for piece-wise planar reconstruction and floorplan reconstruction (Section 1.1), and then explain the organization of this thesis (Section 1.2).

1.1 Background

1.1.1 Piece-wise planar reconstruction

Human vision has a remarkable perceptual capability in understanding high-level scene structures. Observing a typical indoor scene, we can instantly parse a room into a few number of dominant planes (e.g., a floor, walls, and a ceiling), perceive major surfaces for a furniture, or recognize a horizontal surface at the table-top. Developing a computational algorithm that masters similar perceptual capability would be a key for many applications in the emerging domains such as robotics or augmented reality (AR). For instance, a robot needs to identify the extent of a floor to plan a movement, or a table-top segmentation to place objects. In AR applications, planar surface detection is becoming a fundamental building block for placing virtual objects on a desk [36], replacing floor textures, or hanging artworks on walls for interior remodeling.

A difficult yet fundamental task is the inference of piece-wise planar structures from a single RGB image, which has two key challenges. First, 3D plane reconstruction from a single image is an ill-posed problem, requiring rich scene priors. Second, planar structures abundant in man-made environments often lack textures, requiring global image understanding as opposed to local texture analysis.

While most existing systems rely on a sequence of heuristics, we propose a learning-based framework, PlaneNet [48], to directly infer a set of plane parameters and corresponding plane segmentation masks from a single RGB image as shown in Fig. 1.1. We have generated

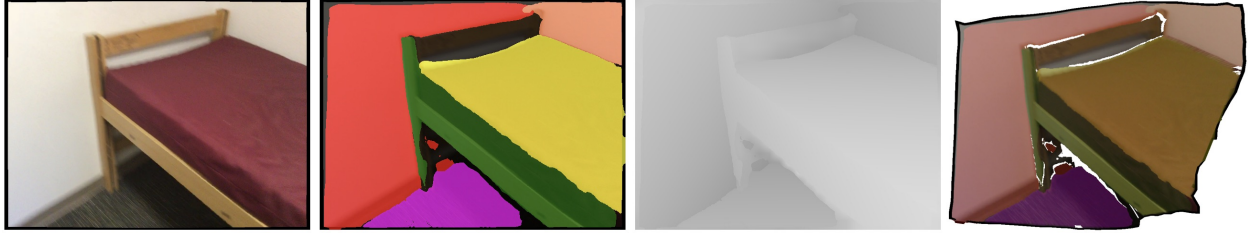


Figure 1.1: A piece-wise planar reconstruction example. From left to right, an input image, a piece-wise planar segmentation, a reconstructed depthmap, and a texture-mapped 3D model.

more than 50,000 piece-wise planar depthmaps for training and testing from ScanNet, a large-scale RGBD video database. Qualitative and quantitative evaluations demonstrate that the proposed approach outperforms baseline methods in terms of both plane segmentation and depth estimation accuracy. To the best of our knowledge, PlaneNet is the first end-to-end neural architecture for piece-wise planar reconstruction from a single RGB image.

While producing impressive results for most indoor scenes, PlaneNet has two major limitations: 1) the number of planes generated is constrained to be no larger than 10 and thus many small planes are discarded, and 2) the trained model has poor generalization ability. To address these issues, we present PlaneRCNN, a detection-based piece-wise planar reconstruction framework, which is able to recover an arbitrary number of planes robustly even for images from unseen datasets [44]. PlaneRCNN further refines all the segmentation masks with a novel loss function that enforces the consistency with a nearby view during training. Besides proposing the detection-based framework, we improve the benchmark proposed in PlaneNet by adding more fine-grained plane segmentations in the ground-truth. PlaneRCNN makes an important step towards robust plane extraction, which would have an immediate impact on a wide range of applications including robotics, augmented reality, and virtual reality.

1.1.2 Floorplan reconstruction

Architectural floorplans play a crucial role in designing, understanding, or remodeling indoor spaces. Their drawings are very effective in conveying geometric and semantic information of a scene. For instance, we can quickly identify room extents, the locations of doors, or object arrangements (geometry). We can also recognize the types of rooms, doors, or objects easily through texts or icon styles (semantics). Given a floorplan in vector-graphics format, a computer can analyze the indoor space, provide designing guidance, and help robots to navigate. However, most properties do not come with a floorplan in vector-graphics format and it is expensive to draw the floorplan manually by an architect. In our research, we seek to reconstruct floorplans automatically from sources which either commonly exist or can be easily obtained.

While professional architects or designers draw floorplans in a vector-graphics representation using software such as AutoCAD [1], HomeStyler [3], or Sketchup [6], the final use of the floorplans is often simply a visualization for clients (e.g., home buyers or renters). As a result, floorplans are rasterized to print or publish digitally. This process discards all the structured geometric and semantic information, limiting human post-processing or further computing capabilities such as model analysis, synthesis, and modification. Recovering the lost information from a rasterized floorplan image is a surprisingly hard task and has been a long-standing open problem. The problem poses two fundamental challenges. First, floorplan structure must satisfy high-level geometric and semantic constraints. For instance, walls corresponding to an external boundary or certain rooms must form a closed 1D loop. Second, this high-level model structure varies across examples (e.g., different houses have different numbers of bedrooms). Computer vision has recently witnessed dramatic progresses on similar high-level model prediction tasks, such as human hand or body pose estimation. In

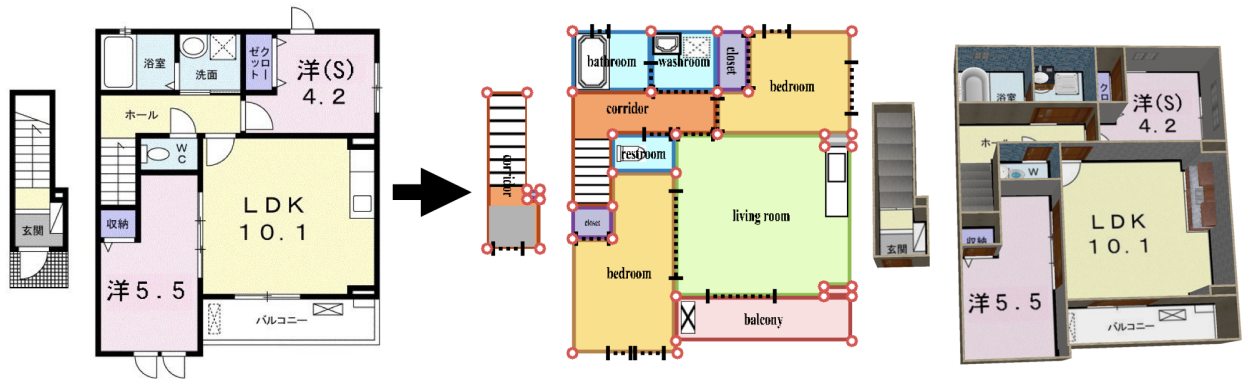


Figure 1.2: An example of converting a raster floorplan image to the vector-graphics format. From left to right, an input floorplan image, reconstructed vector-graphics representation visualized by our custom renderer, and a popup 3D model.

those problems, however, the model structure is fixed: Human hands have five fingers. In our problem, both the model structure as well as parameters need to be inferred.

To recover the structure information from rasterized floorplans, we propose an automated system to convert a rasterized floorplan image into a vector-graphics representation [46]. Unlike the existing approaches that rely on a sequence of low-level image processing heuristics, we adopt a learning-based approach. A neural architecture first transforms a rasterized floorplan image to a set of junctions that represent low-level geometric and semantic information (e.g., wall corners or door end-points). Integer programming is then formulated to aggregate junctions into a set of simple primitives (e.g., wall lines, door lines, or icon boxes) to produce a vectorized floorplan, while ensuring a topologically and geometrically consistent result. Our algorithm significantly outperforms existing methods, and achieves around 90% precision and recall, approaching the production-ready performance. The vector representation allows 3D model popup for better indoor scene visualization, direct model manipulation for architectural remodeling, and computational applications such as data analysis. Fig. 1.2 shows one example of conversion.

While the rasterized floorplan images are widely available and can be robustly converted to vector-graphics format by our method, many properties (e.g., 90% of houses in North America) do not have rasterized floorplans. The ultimate goal of floorplan reconstruction is to automatically reconstruct a vectorized floorplan simply by walking around a house with a smartphone. The consumer-grade depth sensors have revolutionized indoor 3D scanning with successful products. Matterport [5] produces detailed texture mapped models of indoor spaces by acquiring a set of panorama RGBD images with a specialized hardware. Google Project Tango phones [42] convert RGBD image streams into 3D or 2D models. These systems produce detailed geometry, but unlike floorplans or architectural blue-prints, they are not concise or informative about the underlying scene segmentation and semantic. To tackle this problem, we propose FloorNet, a novel deep neural architecture, to reconstruct floorplans from RGBD streams captured by a cheap mobile device [47]. The challenge lies in the processing of 3D data in a large space. FloorNet effectively processes the data through three network branches: 1) PointNet with 3D points, exploiting the 3D information; 2) CNN with a 2D point density image in a top-down view, enhancing the local spatial reasoning, and 3) CNN with RGB images, utilizing the full image information. FloorNet exchanges intermediate features across the branches to exploit the best of all the architectures. We have built a benchmark for floorplan reconstruction by acquiring RGBD video streams for 155 residential houses or apartments with Google Tango phones and annotating complete floorplan information. Our qualitative and quantitative evaluations demonstrate that the fusion of three branches effectively improves the reconstruction quality.

1.2 Organization

In this dissertation, we develop systems for structured indoor modeling, namely piece-wise reconstruction (Chapter 2 and 3) and floorplan reconstruction (Chapter 4, 5, and 6). Conclusion and future works are discussed in 7.

In Chapter 2, we describe PlaneNet, the first end-to-end trainable framework for piece-wise planar reconstruction, which popularizes learning-based frameworks for piece-wise planar reconstruction. We then present PlaneRCNN in Chapter 3, which addresses the limitation of PlaneNet and serves as the current state-of-the-art piece-wise planar reconstruction system.

In Chapter 4, we explain Floorplan Raster2Vector, the automated system we proposed for transforming rasterized floorplan images to vector-graphics format. We then elaborate FloorNet in Chapter 5, the system to reconstruct vector-graphics floorplan from a RGBD stream. We then discuss FloorPlotter, which extends FloorNet to handle non-Manhattan structures, in Chapter 6.

Chapter 2

PlaneNet: Piece-wise Planar

Reconstruction from a Single RGB

Image

2.1 Introduction

With the surge of deep neural networks, single image depthmap inference [18, 17, 40, 73, 77] and room layout estimation [41] have been active areas of research. However, to our surprise, little attention has been given to the study of *piece-wise planar depthmap reconstruction*, mimicking this remarkable human perception in a general form. The main challenge is that the piece-wise planar depthmap requires structured geometry representation (i.e., a set of plane parameters and their segmentation masks). In particular, we do not know the number of planes to be inferred, and the order of planes to be regressed in the output feature vector, making the task challenging even for deep neural networks.

This paper proposes a novel deep neural architecture “PlaneNet” that learns to directly produce a set of plane parameters and probabilistic plane segmentation masks from a single RGB image. Following a recent work on point-set generation [20], we define a loss function that is agnostic to the order of planes. We further control the number of planes by allowing probabilistic plane segmentation masks to be all 0 [71]. The network also predicts a depthmap at non-planar surfaces, whose loss is defined through the probabilistic segmentation masks to allow back-propagation. We have generated more than 50,000 piece-wise planar depthmaps from ScanNet [16] as ground-truth by fitting planes to 3D points and projecting them to images. Qualitative and quantitative evaluations show that our algorithm produces significantly better plane segmentation results than the current state-of-the-art. Furthermore, our depth prediction accuracy is on-par or even superior to the existing single image depth inference techniques that are specifically trained for this task.

2.2 PlaneNet

We build our network upon Dilated Residual Networks (DRNs) [79, 15] (See Fig. 5.1), which is a flexible framework for both global tasks (e.g., image classification) and pixel-wise prediction tasks (e.g., semantic segmentation). Given the high-resolution final feature maps from DRN, we compose three output branches for the three prediction tasks.

Plane parameters: For each scene, we predict a fixed number (K) of planar surfaces $\mathcal{S} = \{S_1, \dots, S_K\}$. Each surface S_i is specified by the three plane parameters P_i (i.e., encoding a normal and an offset). We use D_i to denote a depth image, which can be inferred from the parameters P_i *.

*The depth value calculation requires camera intrinsic parameters, which can be estimated via vanishing point analysis, for example. In our experiments, intrinsics are given for each image through the database information.

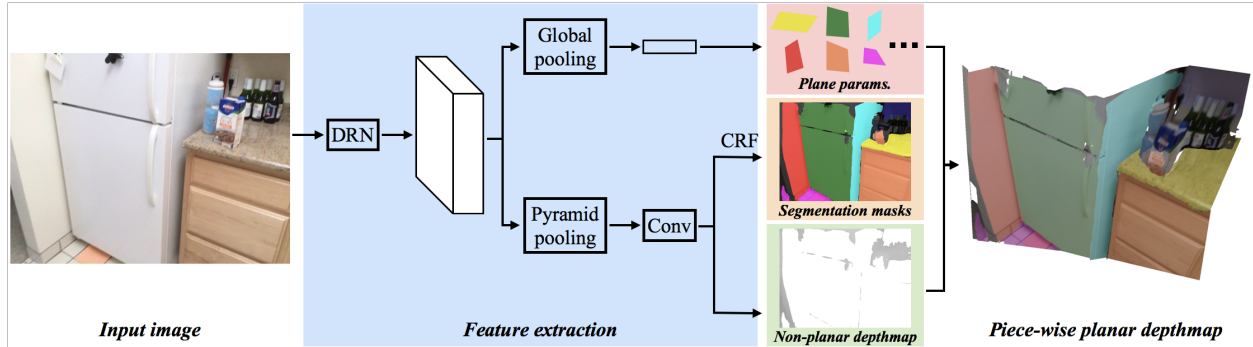


Figure 2.1: PlaneNet predicts plane parameters, their probabilistic segmentation masks, and a non-planar depthmap from a single RGB image.

Non-planar depthmap: We model non-planar structures and infer its geometry as a standard depthmap. With abuse of notation, we treat it as the $(K+1)^{th}$ surface and denote the depthmap as D_{K+1} . This does not explain planar surfaces.

Segmentation masks: The last output is the probabilistic segmentation masks for the K planes (M_1, \dots, M_K) and the non-planar depthmap (M_{K+1}) .

To summarize, the network predicts 1) plane parameters (P_1, \dots, P_K) , 2) a non-planar depthmap (D_{K+1}) , and 3) probabilistic segmentation masks (M_1, \dots, M_{K+1}) . We now explain more details and the loss function for each task.

2.2.1 Plane parameter branch

The plane parameter branch starts with a global average-pooling to reduce the feature map size to 1×1 [79], followed by a fully connected layer to produce $K \times 3$ plane parameters. We do not know the number of planes as well as their order in this prediction task. By following prior works [20, 71], we predict a constant number (K) of planes, then allow some predictions to be invalid by letting the corresponding probabilistic segmentation masks to be 0. Our ground-truth generation process (See Sect. 2.3) produces at most 10 planes for most examples,

thus we set $K = 10$ in our experiments. We define an order-agnostic loss function based on the Chamfer distance metric for the regressed plane parameters:

$$\mathcal{L}^P = \sum_{i=1}^{K^*} \min_{j \in [1, K]} \|P_i^* - P_j\|_2^2. \quad (2.1)$$

The parameterization P_i is given by the 3D coordinate of the point that is closest to the camera center on the plane. P_i^* is the ground-truth. K^* is the number of ground-truth planes.

2.2.2 Plane segmentation branch

The branch starts with a pyramid pooling module [85], followed by a convolutional layer to produce $K + 1$ channel likelihood maps for planar and non-planar surfaces. We append a dense conditional random field (DCRF) module based on the fast inference algorithm proposed by Krahenbuhl and Koltun [39], and jointly train the DCRF module with the precedent layers as in Zheng *et al.* [86]. We set the number of meanfield iterations to 5 during training and to 10 during testing. Bandwidths of bilateral filters are fixed for simplicity. We use a standard softmax cross entropy loss to supervise the segmentation training:

$$\mathcal{L}^M = \sum_{i=1}^{K+1} \sum_{p \in I} (\mathbf{1}(M^{*(p)} = i) \log(1 - M_i^{(p)})) \quad (2.2)$$

The internal summation is over the image pixels (I), where $M_i^{(p)}$ denotes the probability of pixel p belonging to the i^{th} plane. $M^{*(p)}$ is the ground-truth plane-id for the pixel.

2.2.3 Non-planar depth branch

The branch shares the same pyramid pooling module, followed by a convolution layer to produce a 1-channel depthmap. Instead of defining a loss specifically for non-planar regions, we found that exploiting the entire ground-truth depthmap makes the overall training more effective. Specifically, we define the loss as the sum of squared depth differences between the ground-truth and either a predicted plane or a non-planar depthmap, weighted by probabilities:

$$\mathcal{L}^D = \sum_{i=1}^{K+1} \sum_{p \in I} (M_i^{(p)} (D_i^{(p)} - D^{*(p)})^2) \quad (2.3)$$

$D_i^{(p)}$ denotes the depth value at pixel p , while $D^{*(p)}$ is the ground truth depth value.

2.3 Datasets and implementation details

We have generated 51,000 ground-truth piece-wise planar depthmaps (50,000 training and 1,000 testing) from ScanNet [16], a large-scale indoor RGB-D video database. A depthmap in a single RGB-D frame contains holes and the quality deteriorates at far distances. Our approach for ground-truth generation is to directly fit planes to a consolidated mesh and project them back to individual frames, while also exploiting the associated semantic annotations.

Specifically, for each sub mesh-models of the same semantic label, we treat mesh-vertices as points and repeat extracting planes by RANSAC with replacement. The inlier distance threshold is $5cm$, and the process continues until 90% of the points are covered. We merge two (not necessarily adjacent) planes that span different semantic labels if the plane normal difference is below 20, and if the larger plane fits the smaller one with the mean distance error below $5cm$. We project each triangle to individual frames if the three vertices are fitted by the same plane. After projecting all the triangles, we keep only the planes whose projected

area is larger than 1% of an image. We discard entire frames if the ratio of pixels covered by the planes is below 50%. For training samples, we randomly choose 90% of the scenes from ScanNet, subsample every 10 frames, compute piece-wise planar depthmaps with the above procedure, then use the final random sampling to produce 50,000 examples. The same procedure generates 1,000 testing examples from the remaining 10% of the scenes.

We have implemented PlaneNet using TensorFlow [7] based on DeepLab [15]. Our system is a 101-layer ResNet [31] with Dilated Convolution, while we have followed a prior work and modified the first few layers to deal with the degriding issue [79]. The final feature map of the DRN contains 2096 channels. We use the Adam optimizer [37] with the initial learning rate set to 0.0003. The input image, the output plane segmentation masks, and the non-planar depthmap have a resolution of 256x192. We train our network for 50 epochs on the 50,000 training samples.

2.4 Experimental results

Figure 5.4 shows our reconstruction results for a variety of scenes. Our end-to-end learning framework has successfully recovered piece-wise planar and semantically meaningful structures, such as floors, walls, table-tops, or a computer screen, from a single RGB image. We now provide quantitative evaluations on the plane segmentation accuracy and the depth reconstruction accuracy against the competing baselines, followed by more analyses of our results.

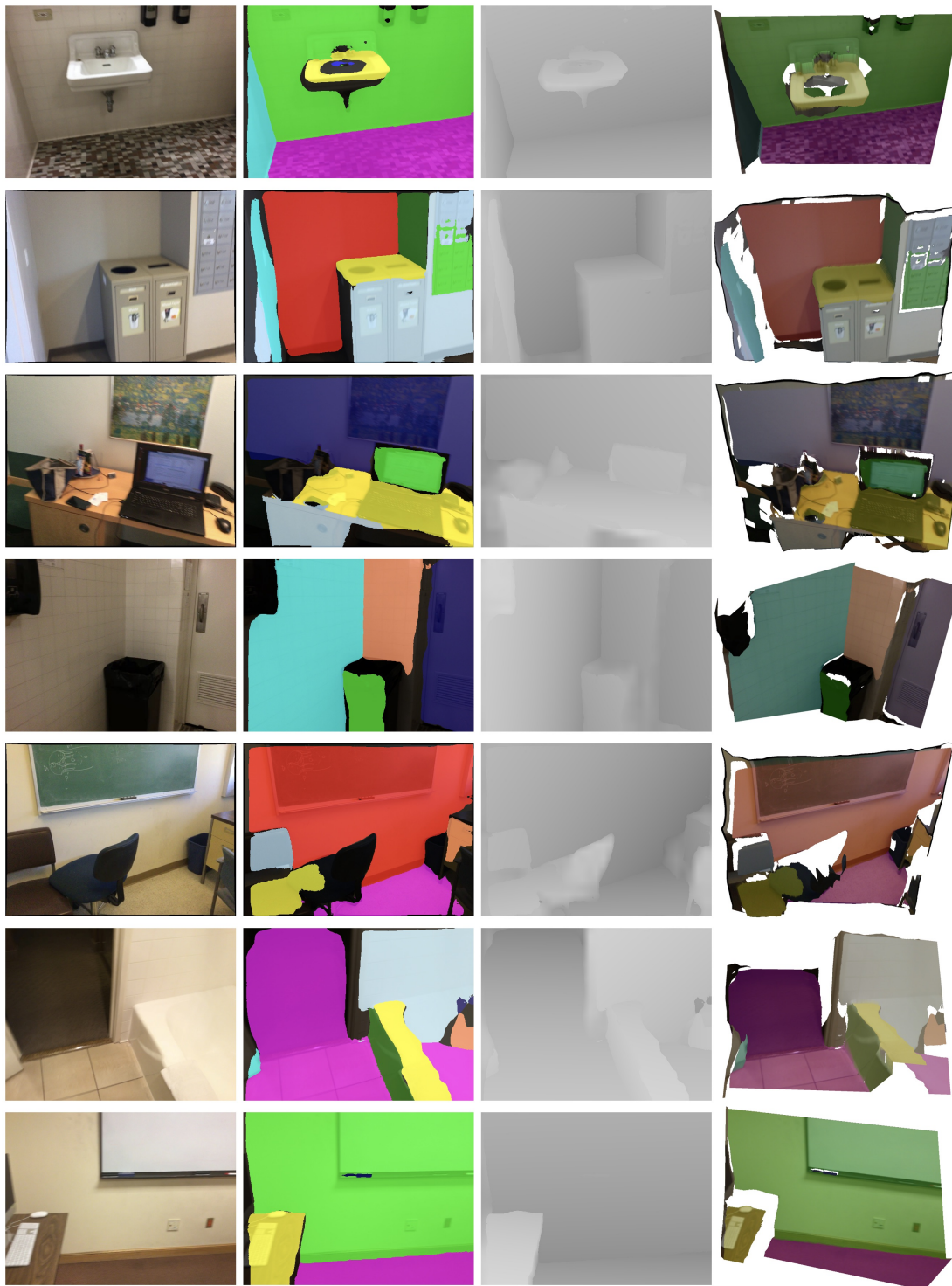


Figure 2.2: Piece-wise planar depthmap reconstruction results by PlaneNet. From left to right: input image, plane segmentation, depthmap reconstruction, and 3D rendering of our depthmap. In the plane segmentation results, the black color shows non-planar surface regions.

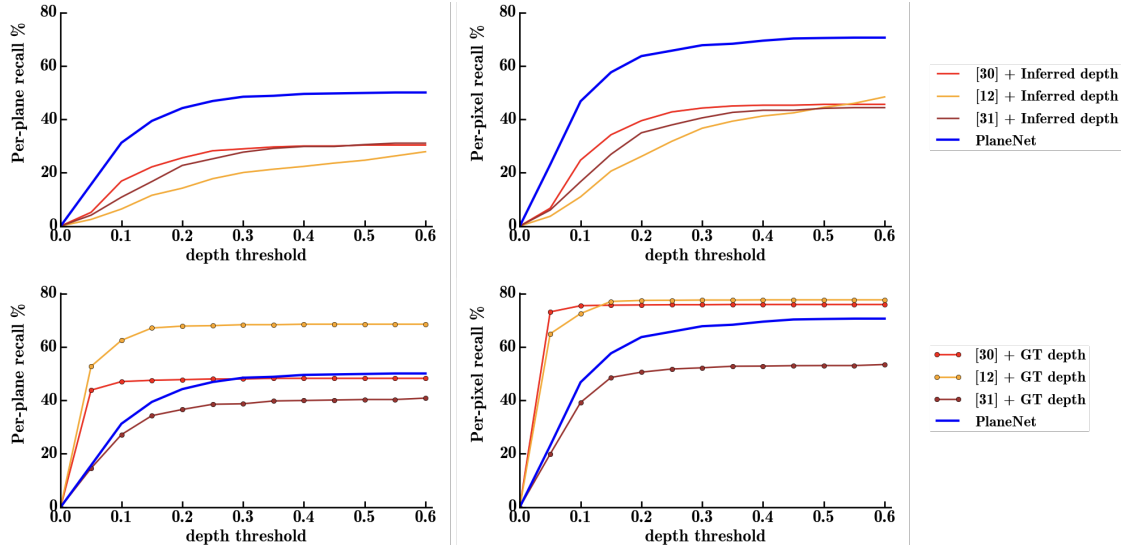


Figure 2.3: Plane segmentation accuracy against competing baselines that use 3D points as input [65, 23, 66]. Either ground-truth depthmaps or inferred depthmaps (by a DNN-based system) are used as their inputs. PlaneNet outperforms all the other methods that use inferred depthmaps. Surprisingly, PlaneNet is even better than many other methods that use ground-truth depthmaps.

2.4.1 Plane segmentation accuracy

Piece-wise planar reconstruction from a single RGB image is a challenging problem. While existing approaches have produced encouraging results [21, 52, 59], they are based on hand-crafted features and algorithmic designs, and may not match against big-data and deep neural network (DNN) based systems. Much better baselines would then be piece-wise planar depthmap reconstruction techniques from 3D points [23, 66, 25, 82], where input 3D points are either given by the ground-truth depthmaps or inferred by a state-of-the-art DNN-based system [40].

In particular, to infer depthmaps, we have used a variant of PlaneNet which only has the pixel-wise depthmap branch, while following Eigen *et al.* [17] to change the loss. Table 3.1 shows that this network, PlaneNet (Depth rep.), outperforms the current top-performers on the NYU benchmark [65].

For piece-wise planar depthmap reconstruction, we have used the following three baselines from the literature.

- “NYU-Toolbox” is a plane extraction algorithm from the official NYU toolbox [65] that extracts plane hypotheses using RANSAC, and optimizes the plane segmentation via a Markov Random Field (MRF) optimization.
- Manhattan World Stereo (MWS) [23] is very similar to NYU-Toolbox except that MWS employs the Manhattan World assumption in extracting planes and exploits vanishing lines in the pairwise terms to improve results.
- Piecewise Planar Stereo (PPS) [66] relaxes the Manhattan World assumption of MWS, and uses vanishing lines to generate better plane proposals. Please see Appendix A for more algorithmic details on the baselines.

Figure 2.3 shows the evaluation results on two recall metrics. The first metric is the percentage of correctly predicted ground-truth planes. We consider a ground-truth plane being correctly predicted, if one of the inferred planes has 1) more than 0.5 Intersection over Union (IOU) score and 2) the mean depth difference over the overlapping region is less than a threshold. We vary this threshold from 0 to 0.6m with an increment of 0.05m to plot graphs. The second recall metric is simply the percentage of pixels that are in such overlapping regions where planes are correctly predicted. The figure shows that PlaneNet is significantly better than all the competing methods when inferred depthmaps are used. PlaneNet is even better than some competing methods that use ground-truth depthmaps. This demonstrates the effectiveness of our approach, learning to infer piece-wise planar structures from many examples.

Figure 3.8 shows qualitative comparisons against existing methods with inferred depthmaps. PlaneNet produces significantly better plane segmentation results, while existing methods

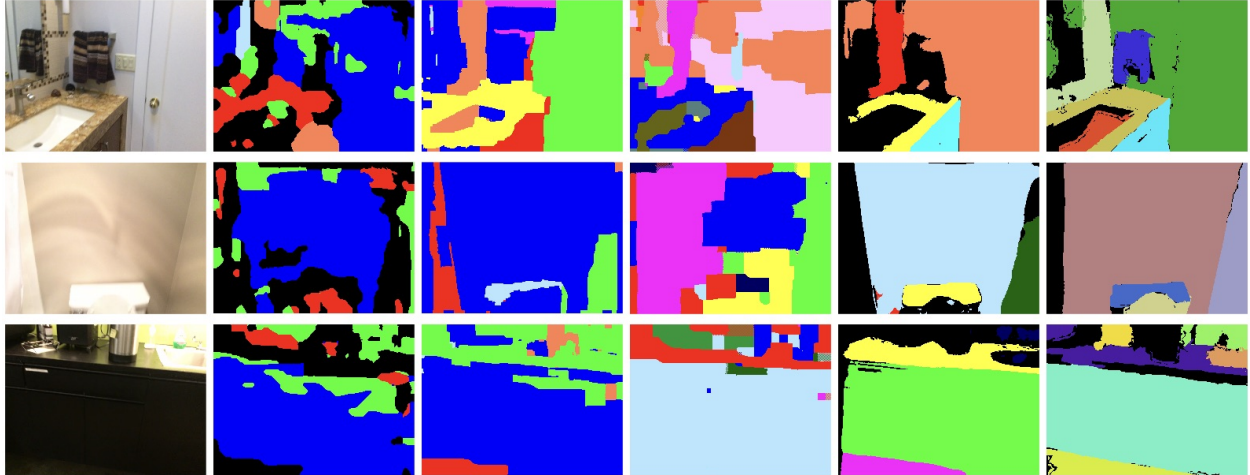


Figure 2.4: Qualitative comparisons between PlaneNet and existing methods that use inferred depthmaps as the inputs. From left to right: an input image, plane segmentation results for [65], [23], [66], and PlaneNet, respectively, and the ground-truth.

often generate many redundant planes where depthmaps are noisy, and fail to capture precise boundaries where the intensity edges are weak.

2.4.2 Depth reconstruction accuracy

While the capability to infer a plane segmentation mask and precise plane parameters is the key contribution of the work, it is also interesting to compare against depth prediction methods. This is to ensure that our structured depth prediction does not compromise per-pixel depth prediction accuracy. PlaneNet makes $(K+1)$ depth value predictions at each pixel. We pick the depth value with the maximum probability in the segmentation mask to define our depthmap.

Depth accuracies are evaluated on the NYUv2 dataset at 1) planar regions, 2) boundary regions, and 3) the entire image, against three competing baselines. [†] Eigen-VGG [17] is a convolutional architecture to predict both depths and surface normals. SURGE [73] is a more

[†]Following existing works, we choose NYUv2 to evaluate depth accuracy and consider only the valid 561x427 area as the entire image evaluation.

recent depth inference network that optimizes planarity. FCRN is the current state-of-the-art single-image depth inference network [40] [‡].

Depthmaps in NYUv2 are very noisy and ground-truth plane extraction does not work well. Thus, we fine-tune our network using only the depth loss (2.3). Note that the key factor in this training is that the network is trained to generate a depthmap through our piece-wise planar depthmap representation. To further verify the effects of this representation, we have also fine-tuned our network in the standard per-pixel depthmap representation by disabling the plane parameter and the plane segmentation branches. In this version, denoted as “PlaneNet (Depth rep.)”, the entire depthmap is predicted in the $(K + 1)^{th}$ depthmap (D_{K+1}).

Table 3.1 shows the depth prediction accuracy on various metrics introduced in the prior work [18]. The left five metrics provide different error statistics such as relative difference (Rel) or rooted-mean-square-error (RMSE) on the average per-pixel depth errors. The right three metrics provide the ratio of pixels, for which the relative difference between the predicted and the ground-truth depths is below a threshold. The table demonstrates that PlaneNet outperforms the state-of-the-art of single-image depth inference techniques. As observed in prior works [73, 13], the planarity constraint makes differences in the depth prediction task, and the improvements are more significant when our piece-wise planar representation is enforced by our network.

2.4.3 Plane ordering consistency

The ordering ambiguity is a challenge for piece-wise depthmap inference. We found that PlaneNet automatically learns a consistent ordering without supervision, for example, the

[‡]The numbers are different from the numbers reported in [40] since [40] evaluate on the original resolution, 640x480, and their numbers are influenced by the issue reported at <https://github.com/iro-cp/FCRN-DepthPrediction/issues/42>.

Table 2.1: Depth accuracy comparisons over the NYUv2 dataset.

Method	Lower the better					Higher the better		
	Rel	Rel(sqr)	\log_{10}	$RMSE$	$RMSE_{log}$	1.25	1.25^2	1.25^3
Evaluation over planar regions								
Eigen-VGG [17]	0.143	0.088	0.061	0.490	0.193	80.1	96.4	99.3
SURGE [73]	0.142	0.087	0.061	0.487	0.192	80.2	96.6	99.3
FCRN [40]	0.140	0.087	0.065	0.460	0.183	79.2	95.6	99.0
PlaneNet (Depth rep.)	0.130	0.080	0.054	0.399	0.156	84.4	96.7	99.2
PlaneNet	0.129	0.079	0.054	0.397	0.155	84.2	96.8	99.2
Evaluation over edge areas								
Eigen-VGG [17]	0.165	0.137	0.073	0.727	0.228	72.9	74.3	98.7
SURGE [73]	0.162	0.133	0.071	0.697	0.221	74.7	94.7	98.7
FCRN [40]	0.154	0.111	0.073	0.548	0.208	74.7	94.1	98.5
PlaneNet (Depth rep.)	0.145	0.099	0.061	0.480	0.179	80.9	95.9	99.0
PlaneNet	0.145	0.099	0.061	0.479	0.178	80.7	96.1	99.1
Evaluation over the entire image								
Eigen-VGG [17]	0.158	0.121	0.067	0.639	0.215	77.1	95.0	98.8
SURGE [73]	0.156	0.118	0.067	0.643	0.214	76.8	95.1	98.9
FCRN [40]	0.152	0.119	0.072	0.581	0.207	75.6	93.9	98.4
PlaneNet (Depth rep.)	0.143	0.107	0.060	0.518	0.180	81.3	95.5	98.7
PlaneNet	0.142	0.107	0.060	0.514	0.179	81.2	95.7	98.9

floor is always regressed as the second plane. In Fig. 5.4, colors in the plane segmentation results are defined by the order of the planes in the network output. Although the ordering loses consistency for small objects or extreme camera angles, major common surfaces such as the floor and walls have a consistent ordering in most cases.

We have exploited this property and implemented a simple room layout estimation algorithm. More specifically, we look at reconstruction examples and manually select the entries of planes that correspond to the ceiling, the floor, and the left/middle/right walls. For each possible room layout configuration [41], (e.g., a configuration with a floor, a left wall, and a middle wall visible), we construct a 3D concave hull based on the plane parameters and project it



Figure 2.5: Room layout estimations. We have exploited the ordering consistency in the predicted planes to infer room layouts.

back to the image to generate a room-layout. We measure the score of the configuration by the number of pixels, where the constructed room layout and the inferred plane segmentation (determined by the winner-takes-all) agree. We pick the constructed room layout with the best score as our prediction. Figure 2.5 shows that our algorithm is able to generate reasonable room layout estimations even when the scene is cluttered and contain many occluding objects. Table 2.2 shows the quantitative evaluations on the NYUv2 303 dataset [83], where our method is comparable to existing techniques which are designed specifically for this task. [§]

[§]RoomNet paper [41] does not provide code or evaluation numbers for the NYUv2 benchmark. We have implemented their system using Torch7 and trained on LSUN dataset as described in their paper.

Table 2.2: Room layout estimations. Quantitative evaluations against the top-performers over the NYUv2 303 dataset.

	Input	Layout error
Schwing <i>et al.</i> [63]	RGB	13.66%
Zhang <i>et al.</i> [83]	RGB	13.94%
Zhang <i>et al.</i> [83]	RGB+D	8.04%
RoomNet [41]	RGB	12.96%
PlaneNet	RGB	12.64%

2.4.4 Failure modes

While achieving promising results on most images, PlaneNet has some failure modes as shown in Fig. 2.6. In the first example, PlaneNet generates two nearly co-planar vertical surfaces in the low-light region below the sink. In the second example, it cannot distinguish a white object on the floor from a white wall. In the third example, it misses a column structure on a wall due to the presence of object clutter. While the capability to infer precise plane parameters is already super-human, there is a lot of room for improvement on the planar segmentation, especially in the absence of texture information or at the presence of clutter.



Figure 2.6: Typical failure modes occur in the absence of enough image texture cues or at the presence of small objects and clutter.

2.5 Applications

Structured geometry reconstruction is important for many application in Augmented Reality. We demonstrate two image editing applications enabled by our piece-wise planar representation: texture insertion and replacement (see Fig. 2.7). We first extract Manhattan directions by using the predicted plane normals through a standard voting scheme [23]. Given a piece-wise planar region, we define an axis of its UV coordinate by the Manhattan direction that is the most parallel to the plane, while the other axis is simply the cross product of the first axis and the plane normal. Given a UV coordinate, we insert a new texture by alpha-blending or completely replace a texture with a new one.



Figure 2.7: Texture editing applications. From top to bottom, an input image, a plane segmentation result, and an edited image.

2.6 Discussions

This paper proposes PlaneNet, the first deep neural architecture for piece-wise planar depthmap reconstruction from a single RGB image. PlaneNet learns to directly infer a set of plane parameters and their probabilistic segmentation masks. The proposed approach significantly outperforms competing baselines in the plane segmentation task. It also advances the state-of-the-art in the single image depth prediction task.

Chapter 3

PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image

3.1 Introduction

To address the limitation of PlaneNet discussed in Chapter 2, we propose a novel deep neural architecture, PlaneRCNN, to more effectively infer piecewise planar structure from a single RGB image. PlaneRCNN consists of three components.

The first component is a plane detection network built upon Mask R-CNN [28]. Besides an instance mask for each planar region, we also estimate the plane normal and per-pixel depth values. With known camera intrinsics, we can further reconstruct the 3D planes from the detected planar regions. This detection framework is more flexible and can handle an arbitrary number of planar regions in an image. To the best of our knowledge, this paper is the first to introduce a detection network, common in object recognition, to the depthmap reconstruction task. The second component is a segmentation refinement network that jointly

optimizes extracted segmentation masks to more coherently explain a scene as a whole. The refinement network is designed to handle an arbitrary number of regions via a simple yet effective neural module. The third component, the warping-loss module, enforces the consistency of reconstructions with another view observing the same scene *during training* and improves the plane parameter and depthmap accuracy in the detection network via end-to-end training.

The paper also presents a new benchmark for the piecewise planar depthmap reconstruction task. We collected 100,000 images from ScanNet [16] and generated the corresponding ground-truth by utilizing the associated 3D scans. The new benchmark offers 14.7 plane instances per image on the average, in contrast to roughly 6 instances per image in the existing benchmark [48].

The performance is evaluated via plane detection, segmentation, and reconstruction metrics, in which PlaneRCNN outperforms the current state-of-the-art with significant margins. Especially, PlaneRCNN is able to detect small planar surfaces and generalize well to new scene types.

The contributions of the paper are two-fold:

Technical Contribution: The paper proposes a novel neural architecture PlaneRCNN, where 1) a detection network extracts an arbitrary number of planar regions; 2) a refinement network jointly improves all the segmentation masks; and 3) a warping loss improves plane-parameter and depthmap accuracy via end-to-end training.

System Contribution: The paper provides a new benchmark for the piecewise planar depthmap reconstruction task with much more fine-grained annotations than before, in which PlaneRCNN makes significant improvements over the current state-of-the-art.

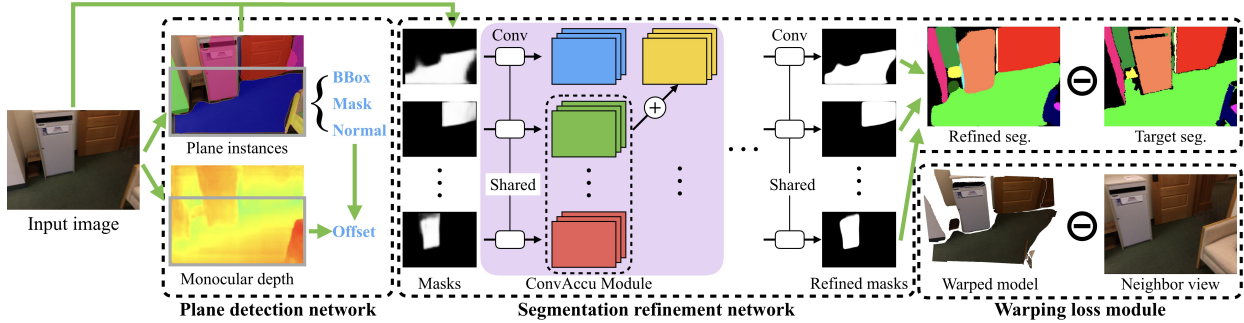


Figure 3.2: Our framework consists of three building blocks: 1) a plane detection network based on Mask R-CNN [28], 2) a segmentation refinement network that jointly optimizes extracted segmentation masks, and 3) a warping loss module that enforces the consistency of reconstructions with a nearby view during training.

3.2 Approach

PlaneRCNN consists of three main components (See Fig. 5.1): a plane detection network, a segmentation refinement network, and a warping loss module. Built upon Mask R-CNN [28], the plane proposal network (Sec. 3.2.1) detects planar regions given a single RGB image and predicts 3D plane parameters together with a segmentation mask for each planar region. The refinement network (Sec. 3.2.2) takes all detected planar regions and jointly optimizes their masks. The warping loss module (Sec. 3.2.3) enforces the consistency of reconstructed planes with another view observing the same scene to further improve the accuracy of plane parameters and depthmap during training.

3.2.1 Plane Detection Network

Mask R-CNN was originally designed for semantic segmentation, where images contain instances of varying categories (e.g., person, car, train, bicycle and more). Our problem has only two categories "planar" or "non-planar", defined in a geometric sense. Nonetheless, Mask R-CNN works surprisingly well in detecting planes in our experiments. It also enables

us to handle an arbitrary number of planes, where existing approaches need the maximum number of planes in an image a priori (i.e., 10 for PlaneNet [48] and 5 for PlaneRecover [78]).

We treat each planar region as an object instance and let Mask R-CNN detect such instances and estimate their segmentation masks. The remaining task is to infer 3D plane parameters, which consists of the normal and the offset information. While CNNs have been successful for depthmap [50] and surface normal [74] estimation, direct regression of plane offset turns out to be a challenge (even with the use of CoordConv [51]). Instead of direct regression, we solve it in three steps: (1) predict a normal per planar instance, (2) estimate a depthmap for an entire image, and (3) use a simple algebraic formula (Eq. 3.1) to calculate the plane offset (which is differentiable for end-to-end training). We now explain how we modify Mask-RCNN to perform these three steps.

Plane normal estimation: Directly attaching a parameter regression module after the ROI pooling produces reasonable results, but we borrow the idea of 2D anchor boxes for bounding box regression [28] to further improve accuracy. More precisely, we consider *anchor normals* and estimate a plane normal in the local camera coordinate frame by 1) picking an anchor normal, 2) regressing the residual 3D vector, and 3) normalizing the sum to a unit-length vector.

Anchor normals are defined by running the K-means clustering algorithm on the plane normals in 10,000 randomly sampled training images. We use $k = 7$ and the cluster centers become anchor normals, which are up-facing, down-facing, and horizontal vectors roughly separated by 45° in our experiments (See Fig. 3.3).

We replace the object category prediction in the original Mask R-CNN with the anchor ID prediction, and append one separate fully-connected layer to regress the 3D residual vector for each anchor normal (i.e., $21 = 3 \times 7$ output values). To generate supervision for each

ground-truth plane normal, we find the closest anchor normal and compute the residual vector. We use the cross-entropy loss for the anchor normal selection, and the smooth L1 loss for the residual vector regression as in the bounding box regression of Mask R-CNN.

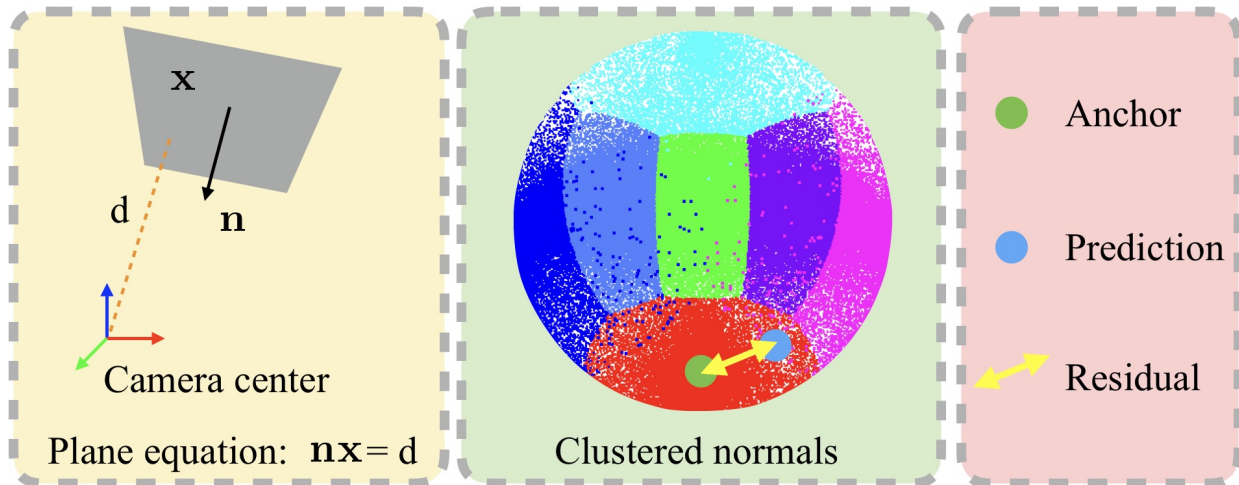


Figure 3.3: We estimate a plane normal by first picking one of the 7 anchor normals and then regressing the residual 3D vector. Anchor normals are defined by running the K-means clustering algorithm on the ground-truth plane normal vectors.

Depthmap estimation: While local image analysis per region suffices for surface normal prediction, global image analysis is crucial for depthmap inference. We add a decoder after the feature pyramid network (FPN) [43] in Mask R-CNN to estimate the depthmap D for an entire image. For the depthmap decoder, we use a block of 3×3 convolution with stride 1 and 4×4 deconvolution with stride 2 at each level. Lastly, bilinear upsampling is used to generate a depthmap in the same resolution as the input image (640×640).

Plane offset estimation: Given a plane normal \mathbf{n} , it is straightforward to estimate the plane offset d :

$$d = \frac{\sum_i m_i (\mathbf{n} \cdot (z_i K^{-1} \mathbf{x}_i))}{\sum_i m_i} \quad (3.1)$$

where K is the 3×3 camera intrinsic matrix, \mathbf{x}_i is the i_{th} pixel coordinate in a homogeneous representation, z_i is its predicted depth value, and m_i is an indicator variable, which becomes 1 if the pixel belongs to the plane. The summation is over all the pixels in the image. Note that we do not have a loss on the plane offset parameter, which did not make differences in the results. However, the plane offset influences the warping loss module below.

3.2.2 Segmentation Refinement Network

The plane detection network predicts segmentation masks independently. The segmentation refinement network jointly optimizes all the masks, where the major challenge is in the varying number of detected planes. One solution is to assume the maximum number of planes in an image, concatenate all the masks, and pad zero in the missing entries. However, this does not scale to large number of planes, and is prone to missing small planes.

Instead, we propose a simple yet effective module, ConvAccu, by combining a U-Net [61] and the idea of non-local module [75]. ConvAccu processes each plane segmentation mask represented in the entire image window with a convolution layer. We then calculate and concatenate the mean feature volumes over all the other planes at the same level before passing to the next level (See Fig. 5.1). This resembles the non-local module and can effectively aggregate information from all the masks.

Refined plane masks are concatenated at the end and compared against ground-truth with a cross-entropy loss. Note that besides the plane mask, the refinement network also takes the original image, the union of all the other plane masks, the depthmap (for planar and non-planar regions), and a 3D coordinate map for the specific plane as input. The target segmentation mask is generated on the fly during training by assigning a ground-truth mask

with the largest overlap. Planes without any assigned ground-truth masks do not receive supervision.

3.2.3 Warping Loss Module

The warping loss module enforces the consistency of reconstructed 3D planes with a nearby view during training. Specifically, our training samples come from RGB-D videos in ScanNet [16], and the nearby view is defined to be the one 20 frames ahead from the current. The module first builds a depthmap for each frame by 1) computing depth values from the plane equations for planar regions and 2) using pixel-wise depth values predicted inside the plane detection network for the remaining pixels. Depthmaps are converted to 3D coordinate maps in the local camera coordinate frames (i.e., a 3D coordinate instead of a depth value per pixel) by using the camera intrinsic information.

The warping loss is then computed as follows. Let M_c and M_n denote the 3D coordinate maps of the current and the nearby frames, respectively. For every 3D point $\mathbf{p}_n (\in M_n)$ in the nearby view, we use the camera pose information to project to the current frame, and use a bilinear interpolation to read the 3D coordinate \mathbf{p}_c from M_c . We then transform \mathbf{p}_c to the coordinate frame of the nearby view based on the camera pose and compute the 3D distance between the transformed coordinate \mathbf{p}_c^t and \mathbf{p}_n . L2 norm of all such 3D distances divided by the number of pixels is the loss. We ignore pixels that project outside the current image frame during bilinear interpolation.

The projection, un-projection, and coordinate frame transformation are all simple algebraic operations, whose gradients can be passed for training. Note that the warping loss module and the nearby view is utilized only during training to boost geometric reconstruction accuracy, and the system runs on a single image at test time.

3.3 Benchmark construction

Following steps described in PlaneNet [48], we build a new benchmark from RGB-D videos in ScanNet [16]. We add the following three modifications to recover more fine-grained planar regions, yielding 14.7 plane instances per image on the average, which is more than double the PlaneNet dataset containing 6.0 plane instances per image.

- First, we keep more small planar regions by reducing the plane area threshold from 1% of the image size to 0.16% (i.e., 500 pixels) and not dropping small planes when the total number is larger than 10.
- Second, PlaneNet merges co-planar planes into a single region as they share the same plane label. The merging of two co-planar planes from different objects causes loss of semantics. We skip the merging process and keep all instance segmentation masks.
- Third, the camera pose quality in ScanNet degrades in facing 3D tracking failures, which causes misalignment between image and the projected ground-truth planes. Since we use camera poses and aligned 3D models to generate ground-truth planes, we detect such failures by the discrepancy between our ground-truth 3D planes and the raw depthmap from a sensor. More precisely, we do not use images if the average depth discrepancy over planar regions is larger than 0.1m. This simple strategy removes approximately 10% of the images.

3.4 Experimental results

We have implemented our network in PyTorch. We use pre-trained Mask R-CNN [28] and initialize the segmentation refinement network with the existing model [30]. We train the network end-to-end on an NVIDIA TitanX GPU for 10 epochs with 100,000 randomly sampled

images from training scenes in ScanNet. We use the same scale factor for all losses. For the detection network, we scale the image to 640×480 and pad zero values to get a 640×640 input image. For the refinement network, we scale the image to 256×192 and align the detected instance masks with the image based on the predicted bounding boxes. The detailed architecture of the refinement network is illustrated in Fig. 3.4.

3.4.1 Qualitative evaluations

Fig. 5.4 demonstrates our reconstructions results for ScanNet testing scenes. PlaneRCNN is able to recover planar surfaces even for small objects.

Fig. 3.7 compares PlaneRCNN against two competing methods, PlaneNet [48] and PlaneRecover [78], on a variety of scene types from unseen datasets (except the SYNTHIA dataset is used for training by PlaneRecover). Note that PlaneRCNN and PlaneNet are trained on the ScanNet which contains indoor scenes, while PlaneRecover is trained on the SYNTHIA dataset (i.e., the 7th and 8th rows in the figure) which consist of synthetic outdoor scenes. The figure shows that PlaneRCNN is able to reconstruct most planes in varying scene types from unseen datasets regardless of their sizes, shapes, and textures. In particular, our results on the KITTI dataset are surprisingly better than PlaneRecover for planes close to the camera. In indoor scenes, our results are consistently better than both PlaneNet and PlaneRecover.

3.4.2 Plane reconstruction accuracy

Following PlaneNet [48], we evaluate plane detection accuracy by measuring the plane recall with a fixed Intersection over Union (IOU) threshold 0.5 and a varying depth error threshold (from 0 to $1m$ with an increment of $0.05m$). The accuracy is measured inside the overlapping regions between the ground-truth and inferred planes. Besides PlaneNet, we compare against

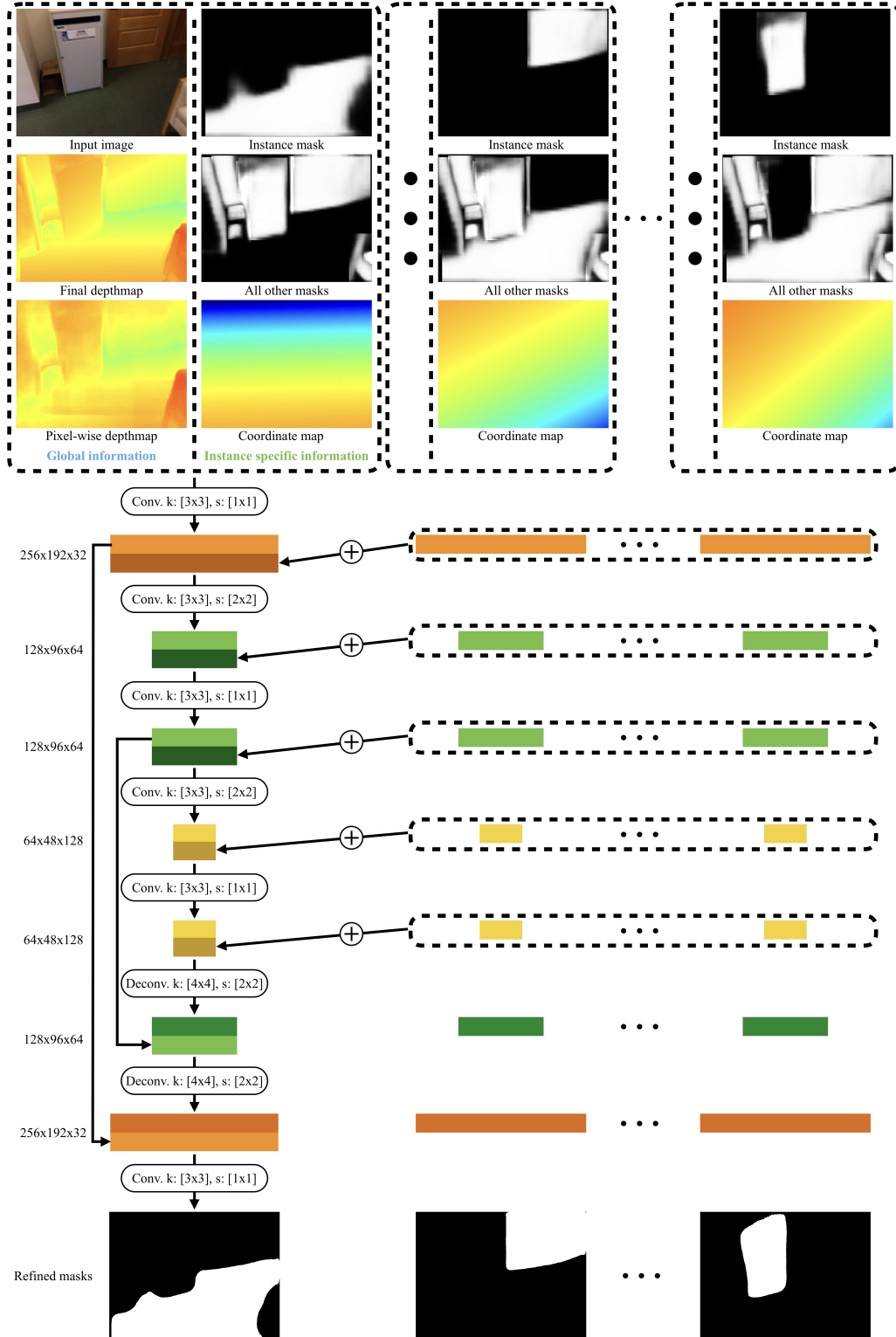


Figure 3.4: Refinement network architecture. The network takes both global information (i.e., the input image, the reconstructed depthmap and the pixel-wise depthmap) and instance-specific information (i.e., the instance mask, the union of other masks, and the coordinate map of the instance) as input and refines instance mask with a U-Net architecture [61]. Each convolution in the encoder is replaced by a ConvAccu module to accumulate features from other masks.

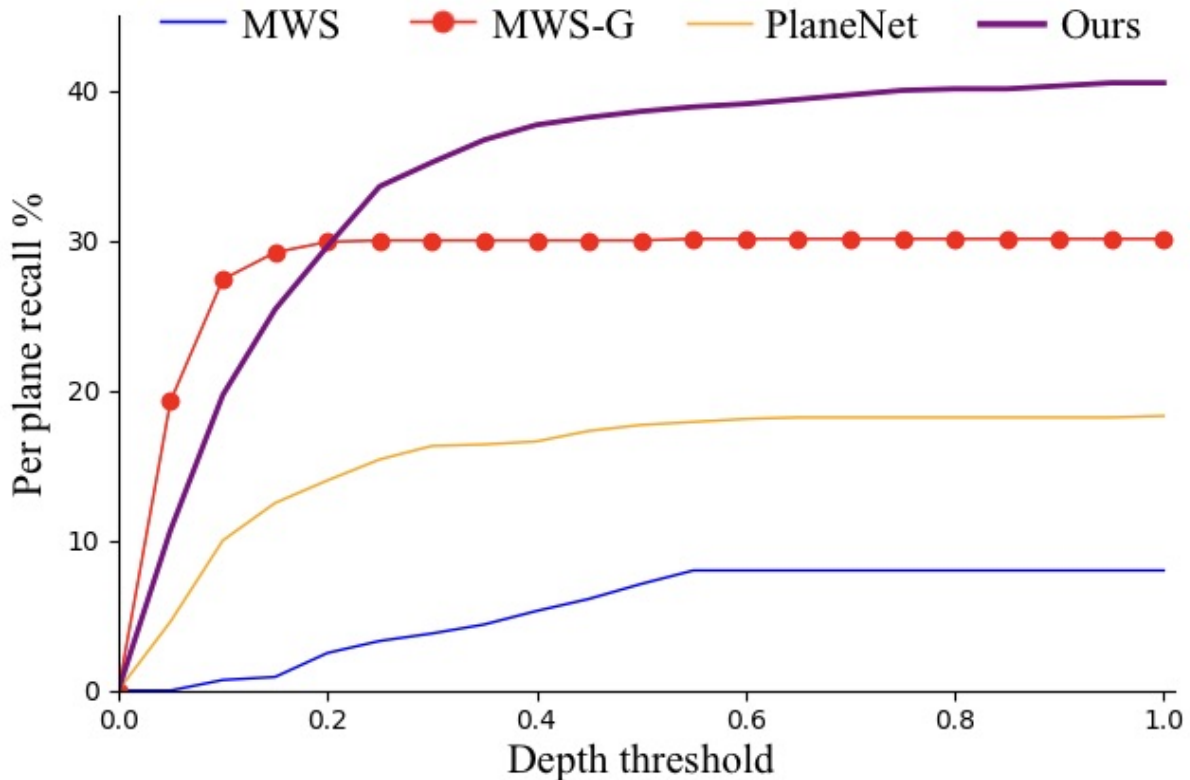


Figure 3.5: Plane-wise accuracy against baselines. PlaneRCNN outperforms all the competing methods except when the depth threshold is very small and MWS-G can fit 3D planes extremely accurately by utilizing the ground-truth depth values.

Manhattan World Stereo (MWS) [23], which is the most competitive traditional MRF-based approach as demonstrated in prior evaluations [48]. MWS requires a 3D point cloud as an input, and we either use the point cloud from the ground-truth 3D planes (MWS-G) or the point cloud inferred by our depthmap estimation module in the plane detection network (MWS). PlaneRecover [78] was originally trained with the assumption of at most 5 planes in an image. We find it difficult to train PlaneRecover successfully for cluttered indoor scenes by simply increasing the threshold. We believe that PlaneNet serves as a stronger competitor for the evaluation.

As demonstrated in Fig. 3.5, PlaneRCNN significantly outperforms all other methods, except when the depth threshold is small and MWS-G can fit planes extremely accurately with the ground-truth depth values. Nonetheless, even with ground-truth depth information, MWS-G fails in extracting planar regions robustly leading to lower recalls in general. Our results are superior also qualitatively as shown in Fig. 3.8.

3.4.3 Geometric accuracy

We propose a new metric in evaluating the quality of piecewise planar surface reconstruction by mixing the inferred depthmaps and the ground-truth plane segmentations. More precisely, we first generate a depthmap from our reconstruction by following the process in the warping loss evaluation (Sec. 3.2.3). Next, for every ground-truth planar segment, we convert depth values in the reconstructed depthmap to 3D points, fit a 3D plane by SVD, and normalize the plane coefficients to make the normal component into a unit vector. Finally, we compute the mean and the area-weighted mean of the parameter differences as the metric over the NYU dataset [65] for fair comparison. Table 3.1 shows that, with more flexible detection network, PlaneRCNN generalizes much better even without fine-tuning. PlaneRCNN also outperforms PlaneNet [48] in every metric after fine-tuning using the ground-truth depths from the NYU dataset.

3.4.4 Ablation studies

PlaneRCNN adds the following components to the Mask R-CNN [28] backbone: 1) the pixel-wise depth estimation network; 2) the anchor-based plane normal regression; 3) the warping loss module; and 4) the segmentation refinement network. To evaluate the contribution of each component, we measure performance changes while adding the components one by one. Following [78], we evaluate the plane segmentation quality by three clustering metrics:

Table 3.1: Geometric accuracy comparison over the NYUv2 dataset.

Method	PlaneNet [48]	Ours
w/o fine-tuning		
Rel	0.220	0.164
\log_{10}	0.114	0.077
<i>RMSE</i>	0.858	0.644
Param.	0.939	0.776
Param. (weighted)	0.771	0.641
w/ fine-tuning		
Rel	0.129	0.124
\log_{10}	0.079	0.073
<i>RMSE</i>	0.397	0.395
Param.	0.713	0.642
Param. (weighted)	0.532	0.505

variation of information (VOI), Rand index (RI), and segmentation covering (SC). To further assess the geometric accuracy, we compute the average precision (AP) with IOU threshold 0.5 and three different depth error thresholds $[0.3m, 0.6m, 0.9m]$. Larger value means higher quality for all the metrics except for VOI.

Table 3.2 shows that all the components have positive contribution to the final performance. Fig. 3.9 further highlights the contributions of the warping loss module and the segmentation refinement network qualitatively. The first example shows that the segmentation refinement network fills in gaps between adjacent planar regions, while the second example shows that the warping loss module improves reconstruction accuracy with the help from the second view.

3.4.5 Occlusion reasoning

A simple modification allows PlaneRCNN to infer occluded/invisible surfaces and reconstruct layered depthmap models. First, ground-truth layered depthmaps are constructed as follows.

Table 3.2: Ablation studies on the contributions of the four components in PlaneRCNN. Plane segmentation and detection metrics are calculated over the ScanNet dataset. PlaneNet represents the competing state-of-the-art.

Method	Plane segmentation			Plane detection		
	VOI ↓	RI	SC	AP ^{0.3m}	AP ^{0.6m}	AP ^{0.9m}
PlaneNet	2.142	0.797	0.692	0.156	0.178	0.182
Ours (basic)	2.113	0.851	0.719	0.269	0.329	0.355
Ours (depth)	2.041	0.856	0.752	0.352	0.376	0.386
Ours (depth + anch.)	2.021	0.855	0.761	0.352	0.378	0.392
Ours (depth + anch. + warp.)	1.990	0.855	0.766	0.365	0.384	0.401
Ours (depth + anch. + warp. + ref.)	1.809	0.880	0.810	0.365	0.386	0.405

In our original process, we fit planes to aligned 3D scans to obtain ground-truth 3D planar surfaces, then rasterize the planes to an image with a depth testing. We simply remove the depth testing and generate a "complete-mask" for each plane. Second, we add one more mask prediction module to PlaneRCNN to infer the complete-mask for each plane instance.

The key challenge for training the network with occlusion reasoning is to generate ground truth complete mask for supervision. Besides projecting 3D planes to the image domain without depth checking, we further complete the mask for layout structures based on the fact that layout planes are behind other geometries. First, we collect all planes which have layout labels (e.g., *wall* and *floor*), and compute the convexity and concavity between two planes in 3D space. Then for each combination of these planes, we compute the corresponding complete depthmap by using the greater depth value for two convex planes and using the smaller value for two concave ones. A complete depthmap is valid if 90% of the complete depthmap is behind the visible depthmap (with 0.2m tolerance to handle noise). We pick the valid complete depthmap which has the most support from visible regions of layout planes.

Fig. 3.10 shows the new view synthesis examples, in which the modified PlaneRCNN successfully infers occluded surfaces, for example, floor surfaces behind tables and chairs. Note that a

depthmap is rendered as a depth mesh model (i.e., a collection of small triangles) in the figure. The layered depthmap representation enables new applications such as artifacts-free view synthesis, better scene completion, and object removal [45, 72]. This experiment demonstrates yet another flexibility and potential of the proposed PlaneRCNN architecture.

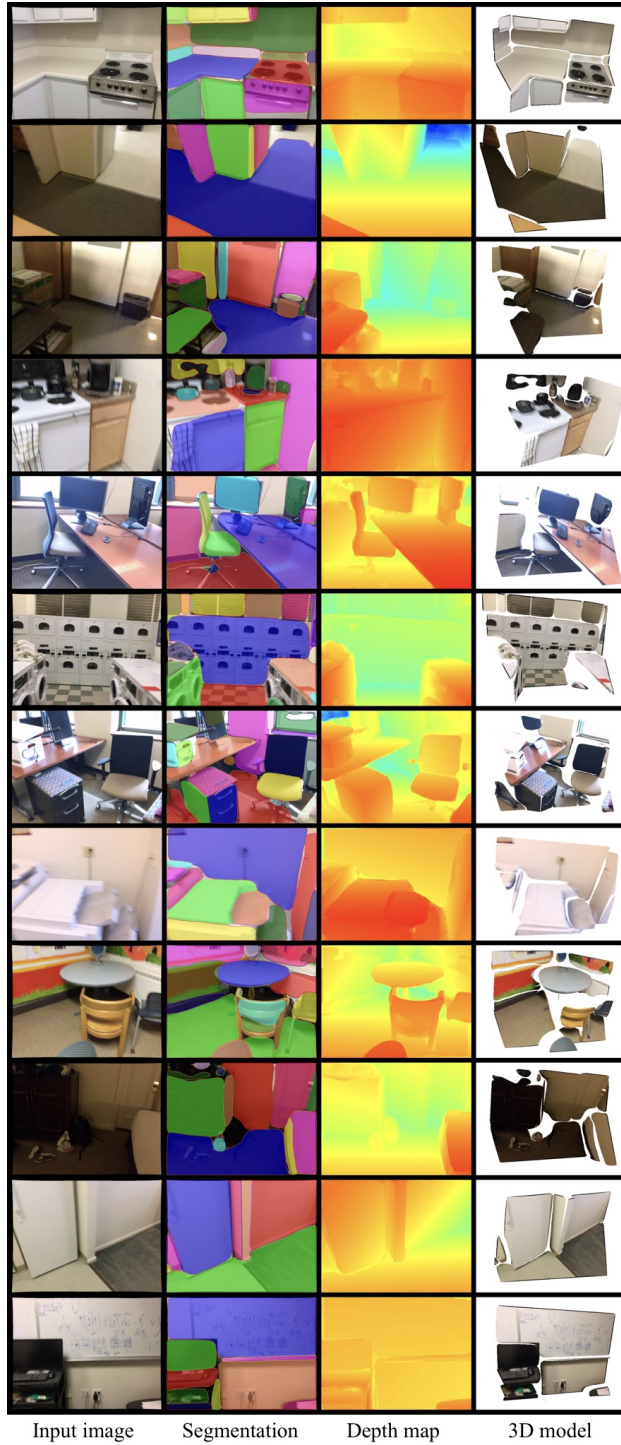


Figure 3.6: Piece-wise planar reconstruction results by PlaneRCNN. From left to right: input image, plane segmentation, depthmap reconstruction, and 3D rendering of our depthmap (rendered from a new view with -0.4m and 0.3m translation along x -axis and z -axis respectively and 10° rotation along both x -axis and z -axis).

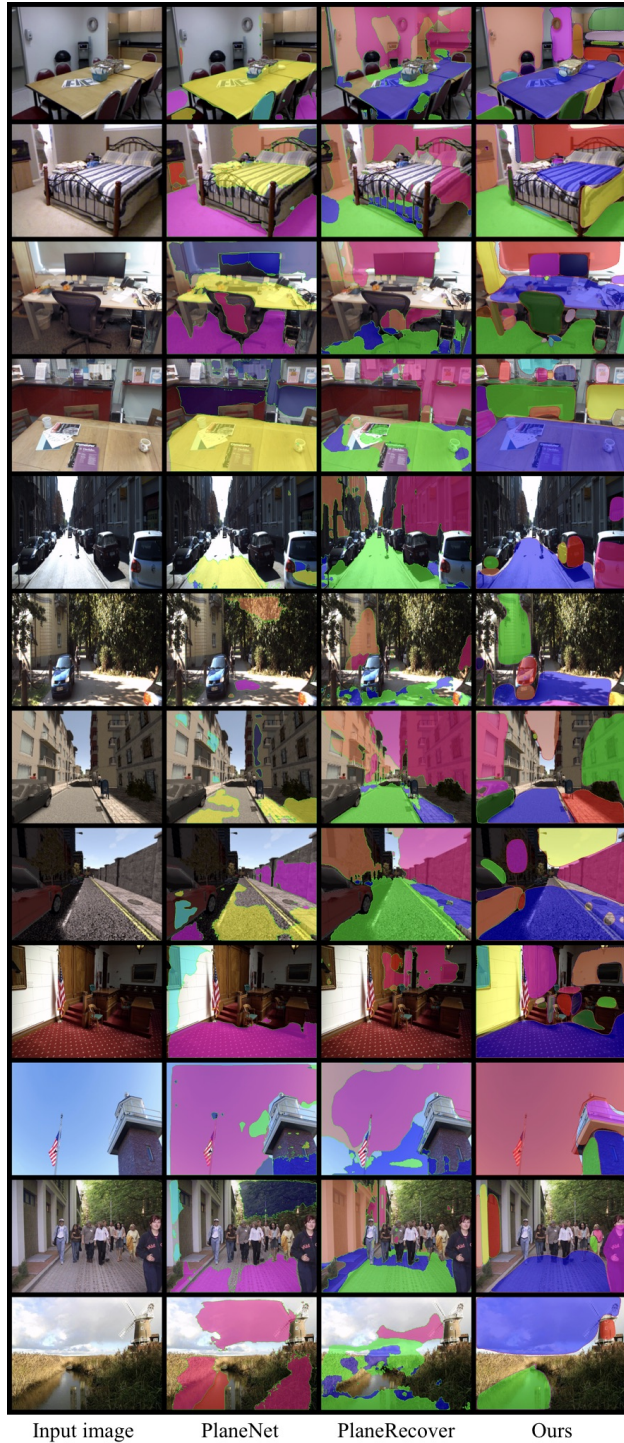


Figure 3.7: Plane segmentation results on unseen datasets without fine-tuning. From left to right: input image, PlaneNet [48] results, PlaneRecover [78] results, and ours. From top to the bottom, we show two examples from each dataset in the order of NYUv2 [65], 7-scenes [64], KITTI [26], SYNTHIA [62], Tank and Temple [38], and PhotoPopup [33].

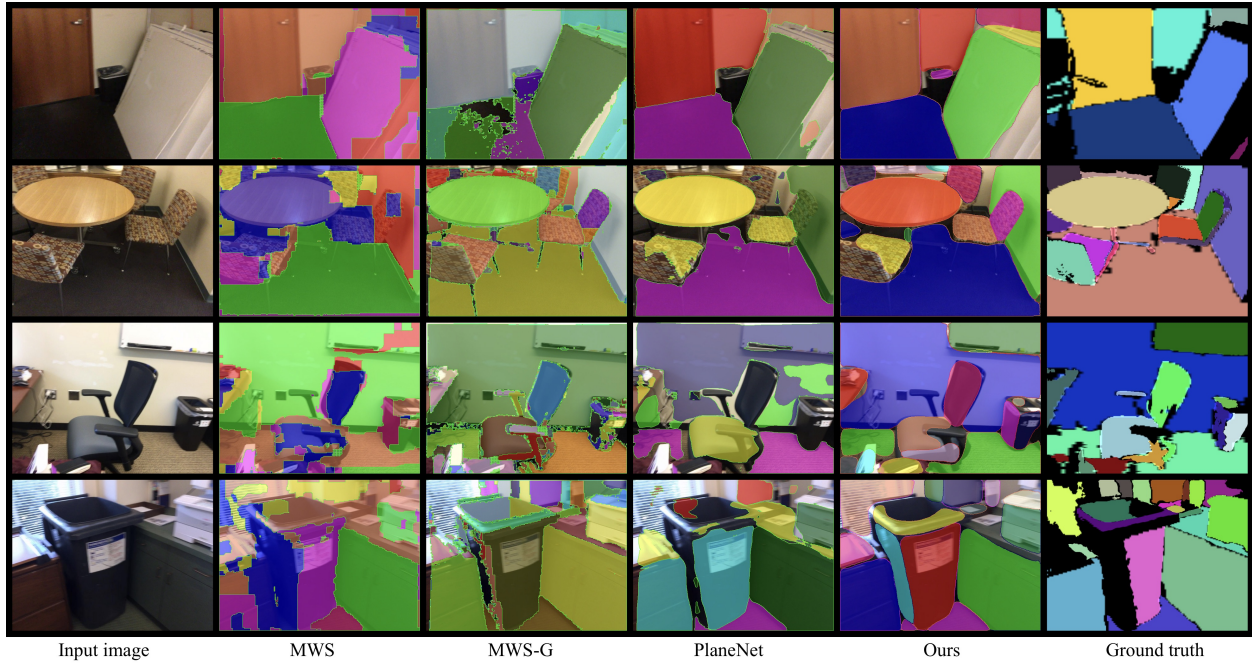


Figure 3.8: Plane segmentation comparisons. From left to right: 1) input image, 2) MWS with inferred depths, 3) MWS with ground-truth depths, 4) PlaneNet, 5) Ours, and 6) ground-truth.

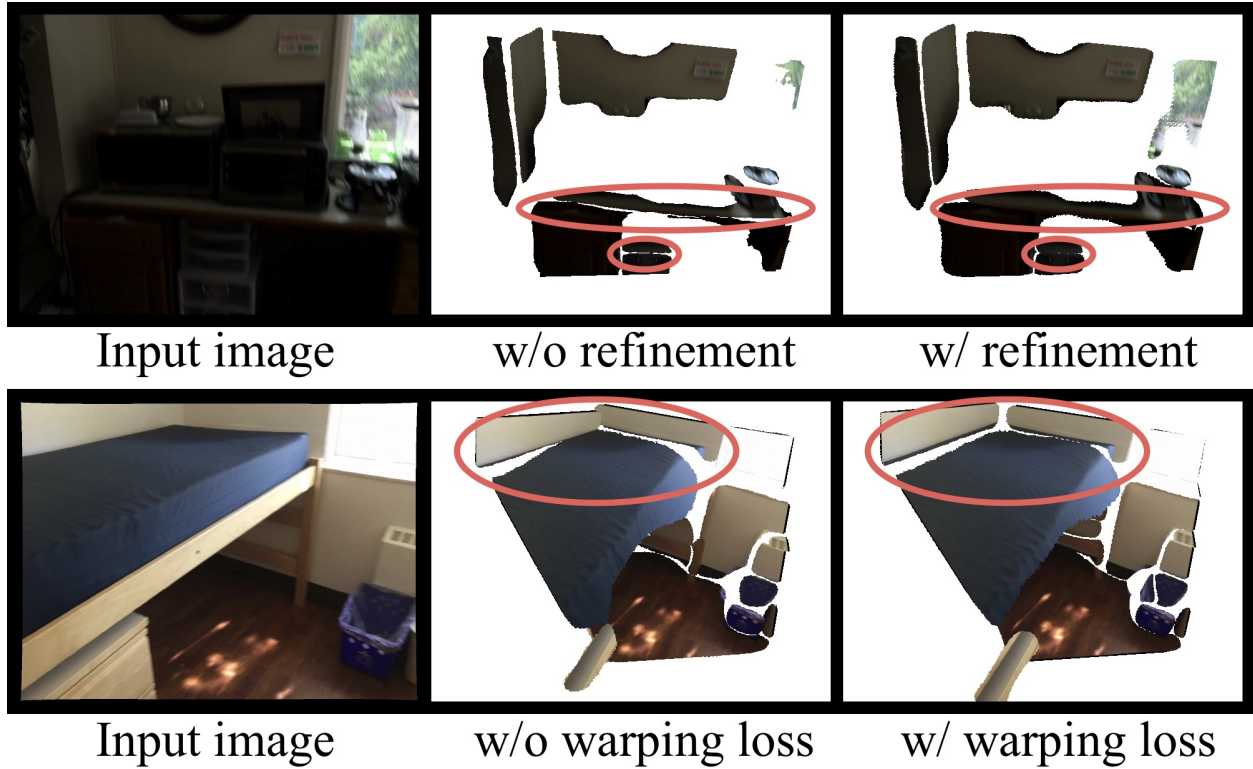


Figure 3.9: Effects of the surface refinement network and the warping loss module. Top: the segmentation refinement network narrows the gap between adjacent planes. Bottom: the warping loss helps to correct erroneous plane geometries from the second view.

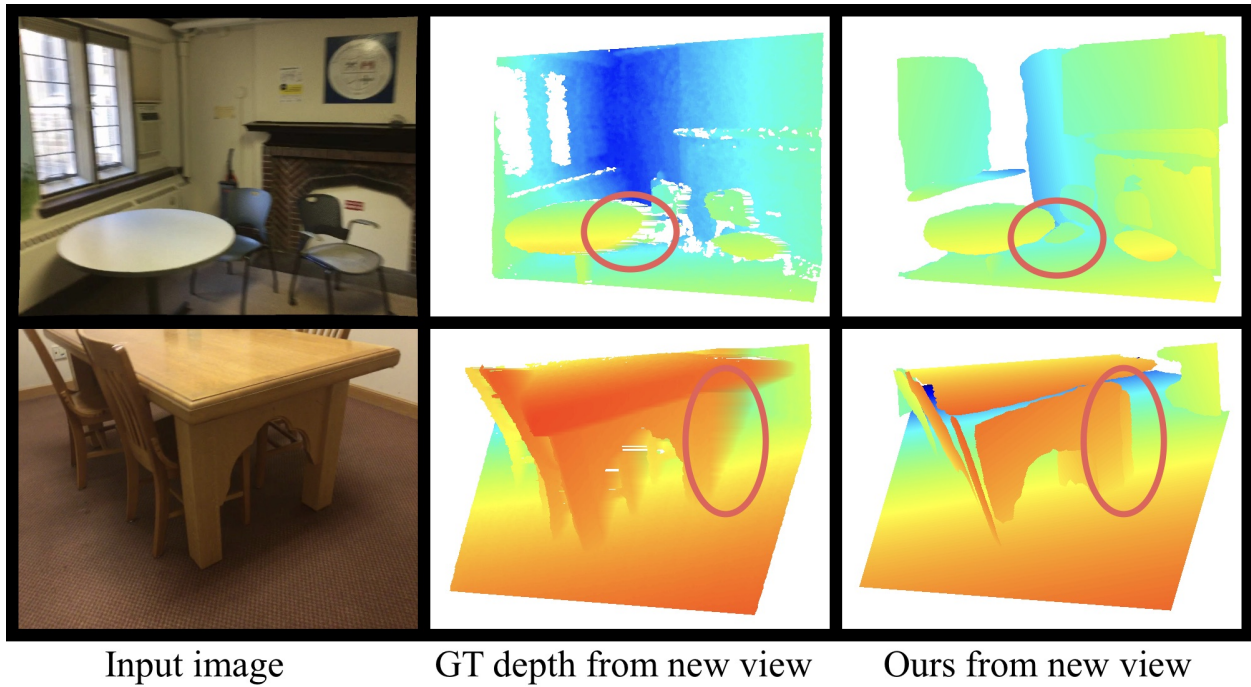


Figure 3.10: New view synthesis results with the layered depthmap models. A simple modification allows PlaneRCNN to also infer occluded surfaces and reconstruct layered depthmap models.

3.5 Discussions

This paper proposes PlaneRCNN, the first detection-based neural network for piecewise planar reconstruction from a single RGB image. PlaneRCNN learns to detect planar regions, regress plane parameters and instance masks, globally refine segmentation masks, and utilize a neighboring view during training for a performance boost. PlaneRCNN outperforms competing methods by a large margin based on our new benchmark with fine-grained plane annotations.

Chapter 4

Raster-to-Vector: Revisiting Floorplan Transformation

4.1 Introduction

This paper proposes a novel solution to the problem of raster-to-vector floorplan transformation. Existing techniques typically rely on a sequence of low-level image processing heuristics [27]. Our approach integrates a deep network and an integer programming through two novel intermediate floorplan representations. First, a deep network extracts low-level geometric and semantic information into a set of junctions (*i.e.*, points with types) as the first representation. For example, a wall structure is represented as a set of junctions of four different types, depending on the degree of its connections. Note that we assume Manhattan world and restrict candidate lines or boxes to be axis-aligned. Second, an integer programming (IP) aggregates junctions into a set of simple primitives (*e.g.*, walls as lines and icons as boxes). IP enforces higher-level constraints among primitives, for instance, a room as a chain

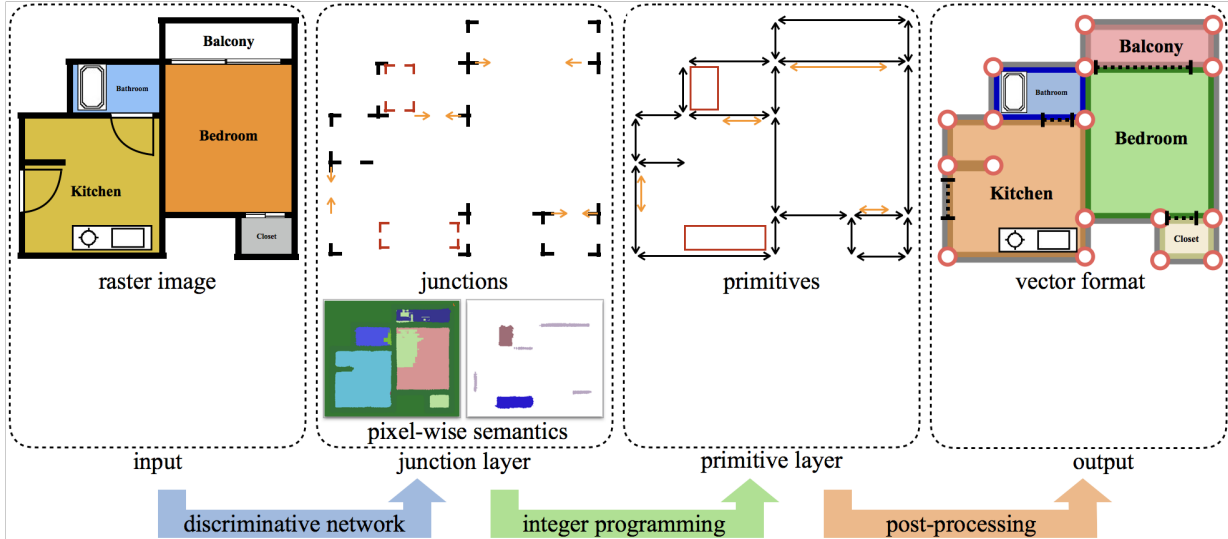


Figure 4.1: Our approach converts a floorplan image through two intermediate representation layers. A neural architecture first converts a floorplan image into a junction layer, where data is represented by a set of junctions (*i.e.*, points with types) or per-pixel semantic classification scores. An integer programming aggregates junctions into a set of primitives (*i.e.*, lines or boxes), while ensuring a topologically and geometrically consistent result. A simple post-processing can be used to produce the final vector format.

of wall primitives forming a closed loop. This ensures that a simple post-processing step can produce a complete vector representation,

We have manually annotated 870 floorplan images for training as well as for quantitative evaluation. The proposed algorithm significantly outperforms existing methods with around 90% precision and recall, getting closer to production-ready performance. We believe that this paper makes progress in understanding floorplans with a new approach achieving a significant performance boost, a large-scale benchmark for evaluating future algorithms, and a large corpus of vector floorplan data and popup 3D models, opening potentials for a new family of Computer Vision research.

4.2 Stratified floorplan representation

Instead of directly solving for the vector representation with high-level structure, we take a stratified approach and convert a floorplan image through two intermediate floorplan representations (See Fig. 5.1). This stratified approach effectively integrates the use of a Convolutional Neural Network (CNN) and Integer Programming (IP), where the network extracts low-level geometric and semantic information as junctions, and IP aggregates junctions into primitives while enforcing high-level structural constraints.

In the intermediate layers, the floorplan data are represented by three factors: walls, openings (doors or windows), or icons. Note that we do not have a factor corresponding to rooms, which will come out in the final post-processing step. Door and window symbols are usually indistinguishable, and are modeled as the same “openings”. In our system, we treat an opening as a window if one side of the opening is outside in the final vector format, except for the main door which sits next to the entrance. This section explains how we encode those factors through our intermediate layers, namely the junction layer and the primitive layer.

4.2.1 Junction layer

The first layer encodes factors as a set of junctions (*i.e.*, points with types).

- Wall structure is represented by a set of junctions where wall segments meet. There are four wall junction types, I-, L-, T-, and X-shaped, depending on the degrees of incident wall segments (*c.f.* Fig. 4.2). Considering orientations, there are in total 13 ($= 4 + 4 + 4 + 1$) types.
- An opening is a line represented by opening junctions at its two end-points. Considering rotational variations, there are 4 types of opening junctions.

- An icon is represented by an axis aligned bounding box, and hence, icon junctions consist of four types: top-left, top-right, bottom-left, or bottom-right.

The junction layer also has two types of per-pixel probability distribution maps over different semantic types. The first map distinguishes if a pixel belongs to a wall or a certain room type. The map is a probability distribution function (PDF) over 12 different classes, as there are 11 room types: *living-room*, *kitchen*, *bedroom*, *bathroom*, *restroom*, *washing-room*, *balcony*, *closet*, *corridor*, *pipe space*, or *outside*. The second map distinguishes if a pixel belongs to an opening, a certain icon type or *empty*. The map is a PDF over 10 different classes, as there are 8 icon types: *cooking counter*, *bathtub*, *toilet*, *washing basin*, *refrigerator*, *entrance mat*, *column*, or *stairs*. Note that a pixel can be both a bathroom (first map) and a bathtub icon (second map).

4.2.2 Primitive layer

The second layer encodes the three factors as low-level geometric primitives. A wall or an opening primitive is represented as a line, namely a valid pair of junctions. Similarly, an icon primitive is represented as a box, that is, a valid quadruple of junctions. Here, "validity" denotes if junctions can be connected given their orientations. Each primitive is also associated with some semantics information.

- A wall primitive is associated with room types at its both sides. The possible room types are the same as in the junction layer.
- An opening primitive, which is either a door or a window, does not have any semantics associated at this layer, as doors and windows are indistinguishable on our floorplan images.

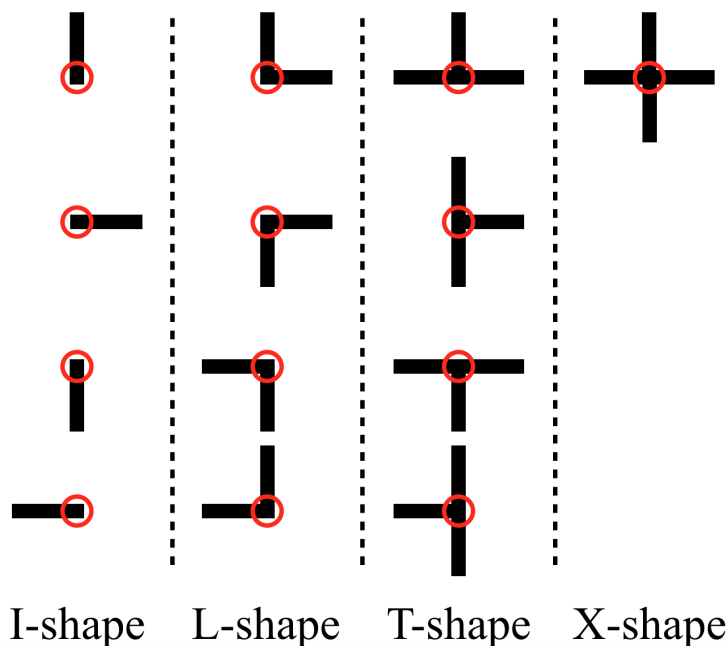


Figure 4.2: There are four wall junction types: I-, L-, T-, and X-shaped, depending on the degrees of incident wall segments. Considering orientations, we have in total 13 ($= 4+4+4+1$) types.

- An icon primitive is associated with one of the icon types, which are the same as in the junction layer.

Primitives must satisfy a variety of constraints so that a simple post-processing can extract a floorplan vector representation with high-level structure. For example, a bedroom must be represented by a set of wall primitives that form a closed-loop and have consistent room type annotation on one side. The constraints are enforced in solving the Integer Programming as explained in Section 4.3.2.

4.3 Raster to vector conversion

Our system consists of three steps. First, we employ a CNN to convert a raster floorplan image into the first junction layer (*i.e.*, junction maps and per-pixel room-classification scores).

Second, after generating primitive candidates from junctions based on simple heuristics, we use Integer Programming to select the right subset while enforcing high-level geometric and semantic constraints. Third, a simple post-processing is used to convert the floorplan data into the final vector-graphics representation. We now explain each step.

4.3.1 Junction layer conversion via a CNN

CNNs have been proven to be very powerful in extracting low-level information from images. Our junction and per-pixel classification representation allows straight-forward application of CNNs. We borrow the residual part of the detection network from [10], which modified ResNet-152 [32] to predict heatmaps at pixel-level. We drop their last deconvolution layer, and append three deconvolution layers in parallel, one for junction heatmap regression and two for per-pixel classifications. For junctions, there are at total $21(= 13 + 4 + 4)$ different types, and one heatmap is regressed for each type, where pixelwise sigmoid cross entropy loss is applied. For classification tasks, we use pixelwise softmax cross entropy loss. We train three branches jointly, and the final loss is a weighted summation with larger weight, 20, only for junction heatmap regression. Both the input and output have resolution 256×256 . Besides common data augmentation techniques like random cropping and color jittering, we also rotate the image with an angle randomly picked from 0, 90, 180, and 270. During inference, we threshold junction heatmaps with 0.4 (slightly lower than 0.5 to bring in more junction candidates for IP to choose), and apply non-maximum suppression to extract junctions.

While the network makes very good predictions of junction locations, it sometimes misclassifies junction types (*e.g.*, mis-classifies a L-shaped junction as T-shaped). To make the detection robust, we allow one mistake in the estimation of the degree. For example, for each detected T-shaped junction, we hallucinate two L-shaped junctions and one X-shaped junction at the same location. The integer programming will enforce later that at most one

of the junctions can be used. We found that mis-classification between I and L is rare, and perform the hallucination for all the other cases.

4.3.2 Primitive layer conversion via IP

Deep network makes very good predictions and simple heuristics suffice to extract primitive candidates (*i.e.*, walls, openings, and icons). Integer programming then finds the correct subset while enforcing various geometric and semantic constraints. With the small problem size, it takes around 2s to find the optimal solution to IP using Gurobi [2].

Primitive candidate generation

A wall primitive can be formed by two wall junctions if 1) they are axis-aligned with a tolerance of 10 pixels, and 2) their aligned orientation is compatible with the junction orientations. Similarly, two door junctions can form a door primitive if qualified. For icons, four axis-aligned junctions (top-left, top-right, bottom-right, and bottom-left) together form an icon primitive.

Integer programming

Integer Programming enforces geometric and semantic constraints among primitives to filter out spurious primitive candidates and guarantee properties of floorplan data, which must hold true. For instance, a bedroom must be surrounded by a set of walls forming a 1D loop, with a bedroom type associated with the correct side of each wall primitive.

Variable definition: We define indicator variables for junctions, primitives, and semantic types:

- $J_{wall}(j)$, $J_{open}(j)$, $J_{icon}(j)$ for junctions,
- $P_{wall}(p)$, $P_{open}(p)$, $P_{icon}(p)$ for primitives,
- $S_{wall}^L(p, s)$, $S_{wall}^R(p, s)$, $S_{icon}(p, s)$ for semantics.

j, p and s denote indexes for junctions, primitives and possible semantic types, respectively. For instance, $P_{open}(p)$ is an indicator variable encoding if the p_{th} opening primitive exists or not. Indicator variables for semantics employ one-hot encoding and have two indexes, a primitive index and a semantic type index. Lastly, a wall primitive is associated with two room types as semantics on its both sides. For instance, $S_{wall}^L(p, s)$ indicates if the p_{th} wall primitive has the s_{th} room type on the left hand side.

Objective function: The objective function for maximization is a linear combination of the junction and semantic indicator variables, where weights are defined as follows:

- The weight is 10 for all the junctions except for the hallucinated wall junctions, whose weight is set to -5 (See Section 4.3.1). In other words, we encourage the use of the primitives as much as possible, but discourage the use of the hallucinated ones.
- The weights for the semantic indicator variables are calculated based on the per-pixel classification scores in the junction layer. For an icon type indicator variable, the weight is simply the average icon type classification score inside the box. For a room type indicator variable associated with a wall primitive on one side, we use the average room type classification score in its neighborhood. A neighborhood is obtained by sweeping pixels on the wall primitives along its perpendicular direction (on the side of the room type variable). Each pixel is swept until it hit another wall primitive candidate.

We have not used primitive indicator variables in the objective function, as similar information has been already captured by the junction primitives.

Constraints: We enforce a variety of constraints either as linear equalities or linear inequalities.

- One-hot encoding constraints: When a wall primitive does not exist (i.e., $P_{wall}(p) = 0$), its wall semantic variables must be all zero. When it exists, one and only one semantic variable must be chosen. The same is true for icon primitives, yielding the following constraints.

$$P_{wall}(p) = \sum_s S_{wall}^L(p, s) = \sum_s S_{wall}^R(p, s),$$

$$P_{icon}(p) = \sum_s S_{icon}(p, s).$$

- Connectivity constraint: The degree (i.e., the number of connections) of a junction must match the number of incident primitives that are chosen. This applies to walls, openings and icons, and here we only show the constraint for the walls, where the summation is over all the wall primitives connected with the wall junction:

$$\{\# \text{ degree}\} J_{wall}(j) = \sum P_{wall}(p).$$

- Mutual exclusion constraints: Two primitives cannot be chosen when they are spatially close, in particular, within 10 pixels. We find every pair of such primitives and enforce that the sum of their indicator variables is at most 1. The same constraint is also applied to wall junctions to ensure that two wall junctions are not close and hallucinated junctions are not selected simultaneously with the original one.
- Loop constraints: Bedroom, bathroom, restroom, balcony, closet, pipe-space, and the exterior boundary must form a closed loop (allowing some walls that stick out). It turns out that this high-level rule can be enforced by local constraints at every wall

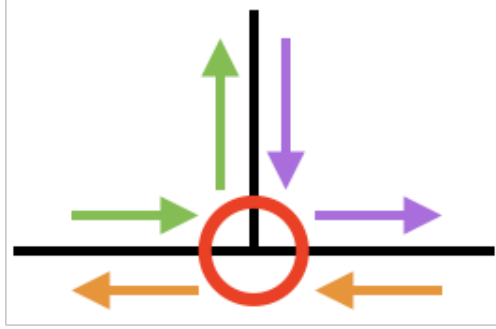


Figure 4.3: Loop constraints can be enforced locally at each junction. The room types must be the same for each pair of walls marked with the same color.

junction. We use a T-shaped wall junction to explain our idea in Fig. 4.3. Room types must be the same for a pair of walls with arrows of the same color in the figure.

- **Opening constraints:** An opening (a door or a window) must be on a wall. For each opening primitive, we find a list of wall primitives that contain the opening (parallel to the opening with distance smaller than 10 pixels) and enforce the following. Note that the right summation is over all the collected wall primitives.

$$P_{open}(p) \leq \sum P_{wall}(p).$$

4.3.3 Final data conversion

The IP output is close to the final representation with a few issues remaining. First, junctions are not well-aligned, because we allow some coordinate error when finding connections. The alignment issue can be simply fixed by averaging the junction coordinates along a straight line. The second issue is that doors are not sitting exactly on walls. To fix this issue, we move each door to align with its closest wall. The last issue is the missing of high-level room information, as room labels are currently associated with walls locally. To derive room information, we first find all the closed polygons formed by walls. If all the walls of a polygon

share the same room label, then polygon forms a room with that label. For a polygon with multiple labels, we further split the polygon by either horizontal lines or a vertical lines into sub-regions, and associate each sub-region with a room label. The detailed split algorithm is in Appendix B.2.

4.4 Evaluations

The section explains 1) our data collection process; 2) evaluation metrics; 3) quantitative evaluations; and 4) qualitative evaluations.

4.4.1 Annotating a Floorplan Dataset

To fully exploit data-hungry neural architecture, we have built a large-scale dataset with groundtruth for vector-graphics floorplan conversion, based on the LIFULL HOME’S dataset [4] which contains 5 million floorplan raster images. To create the groundtruth, we randomly sample 1,000 floorplan images and ask human subjects to annotate the geometric and semantic information for each floorplan image. An annotator can draw a line representing either a wall or an opening, draw a rectangle and pick an object type for each object, or attach a room label at a specific location. We then convert the annotated information to our representation. The conversion is straightforward, and please refer to Appendix B.1 for details. We perform automatic checks to make sure that all the geometric and semantic constraints discussed in Section 4.3.2 are satisfied. After this verification, we manually go through all the annotations to correct the remaining mistakes. After dropping images with poor qualities, we collect 870 groundtruth floorplan images. As a benchmark, 770 of them are used for network training, while the remaining 100 examples are served as test images.



Figure 4.4: Floorplan vectorization results. From left to right, an input floorplan image, reconstructed vector-graphics representation visualized by our custom renderer, and the corresponding popup 3D model. We have manually checked the correctness of each junction and primitive estimation, whose statistics are shown under each example.

4.4.2 Metrics

We compute the precision and recall scores for wall-junctions, opening primitives, icon primitives, and rooms. For a prediction to be correct, its distance to a target must be the smallest compared to the other prediction, and less than a threshold. For wall junctions, we use Euclidean distance, and threshold τ_w . For opening primitives, the distance between a prediction and a target is the larger value of Euclidean distances between two pairs of corresponding endpoints, and the threshold is τ_o . For objects and rooms, we calculate Intersection Over Union (IOU), and use $1 - IOU$ as distance (with threshold τ_o and τ_r respectively). We set $\tau_w = \tau_d = 0.01 \max(W, H)$, $\tau_o = 0.5$ and $\tau_r = 0.3$, where W and H denote the image resolution.

4.4.3 Quantitative evaluations

We evaluate our results on our benchmark images using the metrics mentioned in Section 4.4.2 (See Table 6.1). We have implemented the algorithm in [8] as a baseline. We have not implemented OCR for detecting rooms labels in their algorithm, as we could not find a free open-source OCR system that handles the mix of Japanese and English. However, we believe that the use of OCR is comparable to our deep-network based solution. Their method also ignores objects, so we only evaluate the wall and door detection in their method and show results in Table 6.1. The table also shows the performance of our approach with various algorithmic features removed for an ablation study.

To evaluate the effectiveness of each constraint in IP, we disable one constraint each time and record the performance. Note that we have not evaluated the one-hot encoding constraints and the connectivity constraints, since they are essential. Table 6.1 shows that when full IP is performed, we improve precision consistently over the results without IP. The mutual

Method	Wall Junction		Opening		Icon		Room	
	Acc.	Recall	Acc.	Recall	Acc.	Recall	Acc.	Recall
Ahmed <i>et al.</i> [8]	74.9	57.5	61.3	48.7	N/A	N/A	N/A	N/A
Ours (w/o IP)	70.7	95.1	67.9	91.4	22.3	77.4	80.9	78.5
Ours (w/o mutual exclusion)	92.8	91.7	68.5	91.1	22.0	76.2	82.8	87.5
Ours (w/o loop)	94.2	91.5	91.9	90.2	84.3	75.0	82.5	88.2
Ours (w/o opening)	94.6	91.7	91.7	90.1	84.0	74.8	84.3	88.3
Ours (w/ full IP)	94.7	91.7	91.9	90.2	84.0	74.6	84.5	88.4

Table 4.1: Quantitative evaluations based on our benchmark.

exclusion constraints introduce the competition between conflicting primitives, and thus filter out many false-positives. This filtering process improves all precisions by a large margin, with only small sacrifice in recall for openings and objects. On the contrary, the recall for rooms increases due to the more accurate room shapes formed by walls. The loop constraints improve the precision for rooms by 2%. This improvement is critical in some cases to ensure the visual quality of conversion. The opening constraints ensure that no opening appears without a wall.

4.4.4 Qualitative evaluations

Figure 5.4 shows an input floorplan image, the reconstructed vector representation visualized by our own renderer, and a popup 3D model for each example. In our rendering, a wall junction is highlighted as a disk with a red border, an opening primitive is shown with a black dashed line, an object primitive is shown as an icon with different styles, depending on the inferred semantics. We also change the background color of each room based on its type. The popup 3D model is generated by extruding wall primitive to a certain height, adding window or door textures at the location of opening primitives (an opening becomes a window, if it faces outside), and placing 3D objects models in the bounding box of icon primitives.

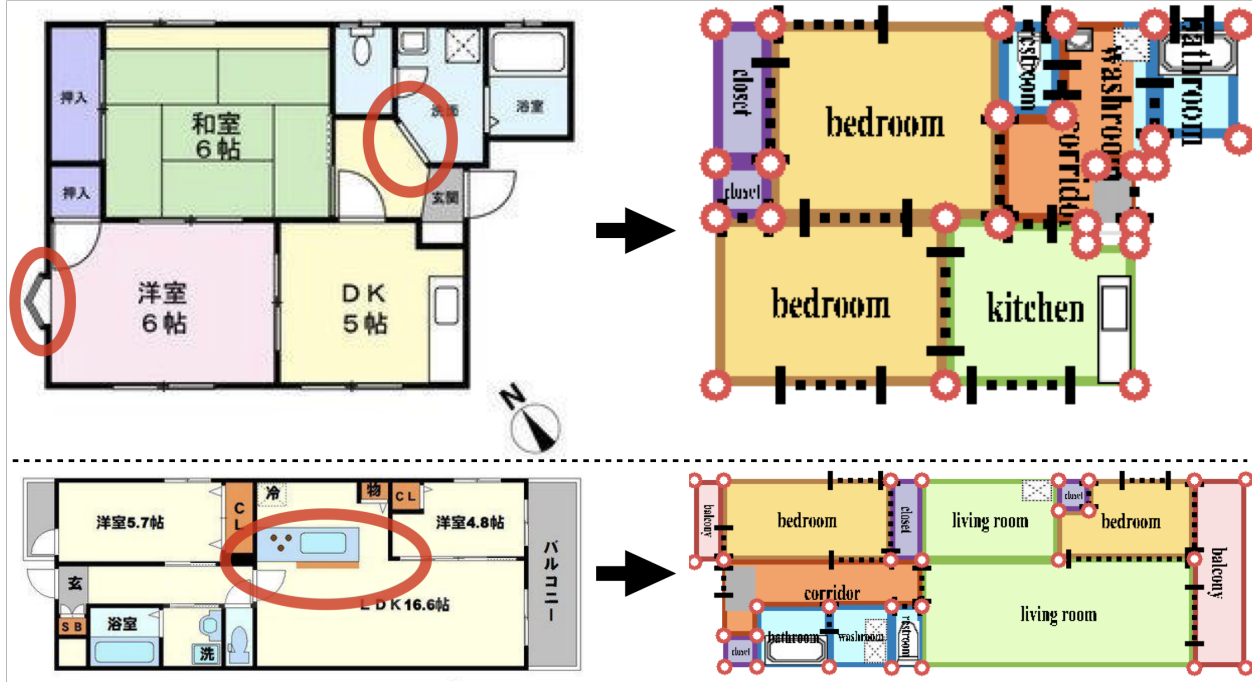


Figure 4.5: Typical failure cases. Relying on Manhattan assumption, our method is unable to detect walls which are neither horizontal nor vertical as shown in the first example. In the second example, our method puts a wall which does not exist.

We have manually verified the correctness of the floorplan data, which are shown by the three numbers for the wall junctions, opening primitives, icon primitives, and rooms. For example, (57/59/60) means that there are 59 target predictions to make, where our algorithm makes 60 predictions and collected 57 correct ones. Verified by the quantitative evaluation, our algorithm is able to recover near complete floorplan models despite of varying styles (e.g., black and white, colored background, or textured background mixing Japanese, English, and Chinese characters). We have run our algorithm on 100,000 raster images to recover their vector-graphics representation and 3D popup models. Though rare, there are some typical failure cases as shown in Fig. 4.5.

Lastly, we have evaluated the generalization capability of our system by processing floorplan images from other data sources. Figure 4.6 shows our results on two such examples: one from Rent3D database [49], and the other from Google image search with a search keyword

"floorplan". Despite the fact that these two images have very distinctive styles from examples in our dataset, our system has successfully reconstructed most walls and openings. Due to dramatic differences in styles, semantic information (i.e., icon types and room types) is often mis-recognized.

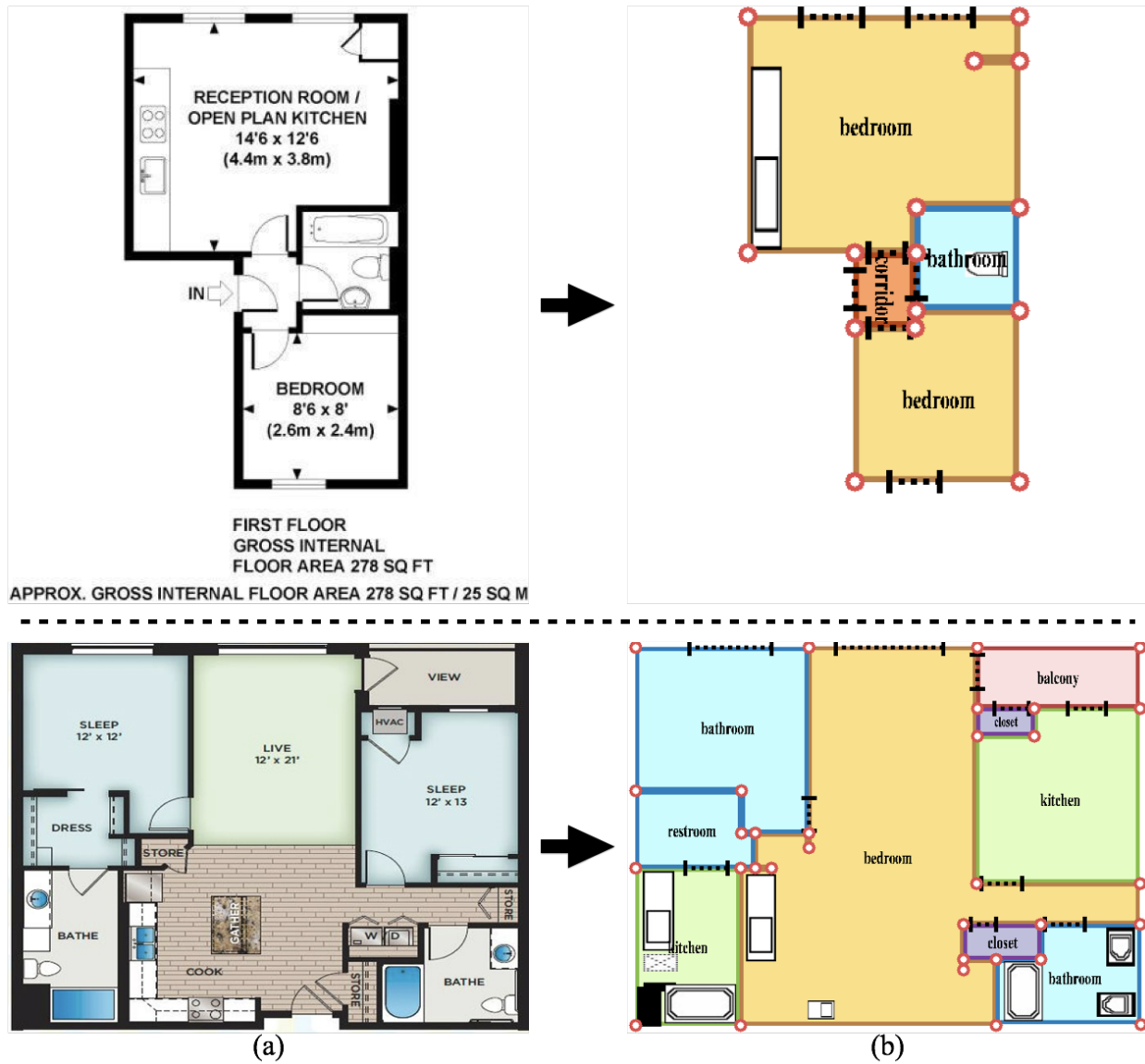


Figure 4.6: Floorplan vectorization results on an image in Rent3D (top) and an image from the web (bottom).

4.5 Discussions

We have presented a novel approach to the classical problem of converting a rasterized floorplan image into a vector-graphics representation. Instead of relying on low-level image processing heuristics, our approach invents two intermediate representations encoding both geometric and semantic information of a floorplan image. We employ a neural architecture to convert an input rasterized image into a set of junctions and per-pixel semantic classification scores as the first representation. Integer programming then aggregates junctions into line or box primitives as the second representation, while ensuring topologically and geometrically consistent result. We believe that the paper makes a key milestone in the literature with a novel approach achieving a significant performance boost, a large-scale benchmark for evaluation, and a large corpus of floorplan vector data and popup 3D models, opening potentials for a new family of big-data Computer Vision research.

Chapter 5

FloorNet: A Unified Framework for Floorplan Reconstruction from 3D Scans

5.1 Introduction

Reconstruction of the floorplan for an entire house or an apartment with multiple rooms poses fundamental challenges to existing techniques due to its large 3D extent. A standard approach projects 3D information onto a 2D lateral domain [34], losing the information of height. PointNet [56, 58] consumes 3D information directly but suffers from the lack of local neighborhood structures. A multi-view representation [57, 69] avoids explicit 3D space modeling, but has been mostly demonstrated for objects, rather than large scenes and complex camera motions. 3D Convolutional Neural Networks (CNNs) [60, 70] also show promising results but have been so far limited to objects or small-scale scenes.

This paper proposes a novel deep neural network (DNN) architecture FloorNet, which turns a RGBD video covering a large 3D space into pixel-wise predictions on floorplan geometry and

semantics, followed by an existing Integer Programming formulation [46] to recover vector-graphics floorplans. FloorNet consists of three DNN branches. The first branch employs PointNet with 3D points, exploiting the 3D information. The second branch uses a CNN with a 2D point density image in a top-down floorplan view, enhancing the local spatial reasoning. The third branch uses a CNN with RGB images, utilizing the full image information. The PointNet branch and the point-density branch exchange features between the 3D points and their corresponding cells in the top-down view. The image branch contributes deep image features into the corresponding cells in the top-down view. This hybrid DNN design exploits the best of all the architectures and effectively processes the full RGBD video covering a large 3D scene with complex camera motions.

We have created a benchmark for floorplan reconstruction by acquiring RGBD video streams for 155 residential houses or apartments with Google Tango phones and annotated their complete floorplan information including architectural structures, icons, and room types. Extensive qualitative and quantitative evaluations demonstrate the effectiveness of our approach over competing methods.

In summary, the main contributions of this paper are two-fold: 1) Novel hybrid DNN architecture for RGBD videos, which processes the 3D coordinates directly, models local spatial structures in the 2D domain, and incorporates the full image information; and 2) A new floorplan reconstruction benchmark with RGBD videos, where many indoor scene databases exist [16, 68, 5] but none tackles a vector-graphics reconstruction problem, which has immediate impact on digital mapping, real estate, or civil engineering applications.

5.2 FloorNet

The proposed FloorNet converts a RGBD video with camera poses into pixel-wise floorplan geometry and semantics information, which is an intermediate floorplan representation introduced by Liu et al. [46]. We first explain the intermediate representation for being self-contained, then provide the details.

5.2.1 Preliminaries

The intermediate representation consists of the geometry and the semantics information. The geometry part contains room-corners, object icon-corners, and door/window end-points, where the locations of each corner/point type are estimated by a 256×256 heatmap in the 2D floorplan image domain, followed by a standard non-maximum suppression. For example, a room corner is either I-, L-, T-, or X-shaped depending on the number of incident walls, making the total number of feature maps to be 13 considering their rotational variants. The semantics part is modeled as 1) 12 feature maps as a probability distribution function (PDF) over 12 room types, and 2) 8 feature maps as a PDF over 8 icon types. We follow their approach and use their Integer Programming formulation to reconstruct a floorplan from this representation at the end.

5.2.2 Triple-branch hybrid design

Floornet consists of three DNN branches. We employ existing DNN architectures in each branch without modifications. Our contribution lies in its hybrid design: how to combine them and share intermediate features (See Fig. 5.1).

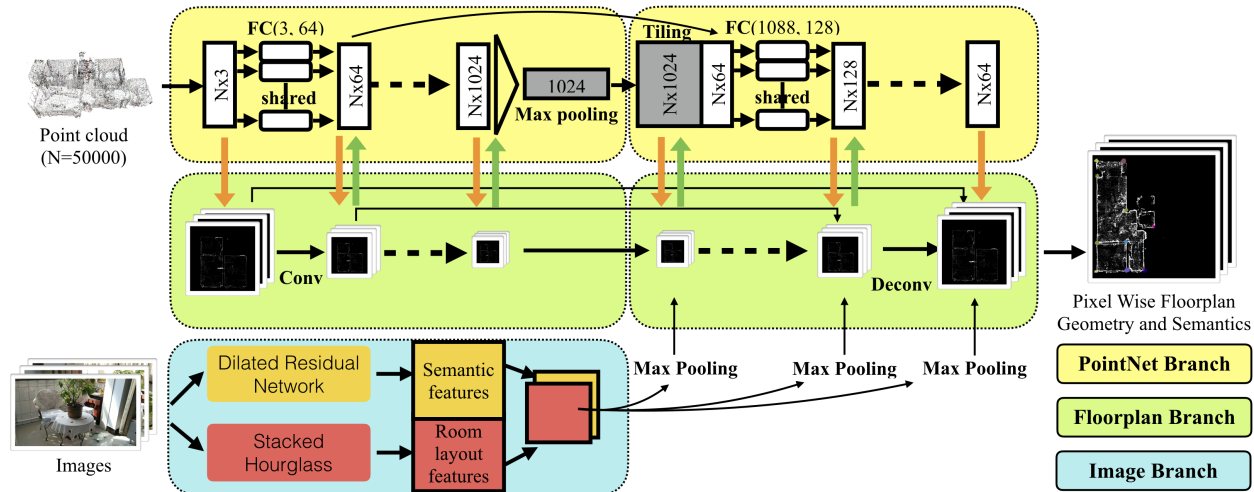


Figure 5.1: FloorNet consists of three DNN branches. The first branch uses PointNet [56] to directly consume 3D information. The second branch takes a top-down point density image in a floorplan domain with a fully convolutional network [53], and produces pixel-wise geometry and semantics information. The third branch produces deep image features by a dilated residual network trained on the semantic segmentation task [79] as well as a stacked hourglass CNN trained on the room layout estimation [55]. The PointNet branch and the floorplan branch exchanges intermediate features at every layer, while the image branch contributes deep image features into the decoding part of the floorplan branch. This hybrid DNN architecture effectively processes an input RGBD video with camera poses, covering a large 3D space.

PointNet Branch: The first branch is PointNet [56] with the original architecture except that each 3D point is represented by XYZ and RGB values without the normalized position. We randomly subsample 50,000 points for each data. We manually rectify the rotation and align the gravity direction with the Z-axis. We add translation to move the center of mass to the origin.

Floorplan Branch: The second branch is a fully convolutional network (FCN) [53] with skip connections between the encoder and the decoder, which takes a point-density image with RGB values in the top-down view. The RGB value in each cell is computed as the average over the 3D points. We compute a 2D axis-aligned bounding box of the Manhattan-rectified 3D points to define a rectangular floorplan domain, while ignoring the 2.5% outlier points

and expanding the rectangle by 5% in each of the four directions. The rectangle is placed in the middle of the 256×256 square image in which the geometry and semantics feature maps are produced. The input to the branch is a point-density image in the same domain.

Image Branch: The third branch computes deep image features through two CNN architectures: 1) Dilated residual network (DRN) [79] trained on semantic segmentation using the ScanNet dataset [16]; and 2) stacked hourglass CNN (SH) [55] trained on room layout estimation using the LSUN dataset [84].

5.2.3 Inter-branch feature sharing

Different branches learn features in different domains (3D points, the floorplan, and images). Figure 5.2 shows three inter-branch feature sharing by pooling and unpooling operations, based on the camera poses and 3D information.

PointNet to floorplan pooling: This pooling module takes features of unordered points from each layer of the PointNet branch and produces a 2D top-down feature map in the corresponding layer of the floorplan branch. The module simply projects 3D point features into cells in the floorplan feature map, then aggregates the features in each cell by either summation or maximum operation. We use the summation operation in the first three convolution layers to keep more information while taking the maximum in the rest layers to introduce competition. A constructed feature map has the same dimension as the feature map in the floorplan branch. We have explored several different aggregation schemes between the feature map from the pointnet branch and the feature map at the floorplan branch. We found that a sum-pooling (i.e., element-wise addition) works the best. The time complexity of the projection pooling module is linear in the number of 3D points.

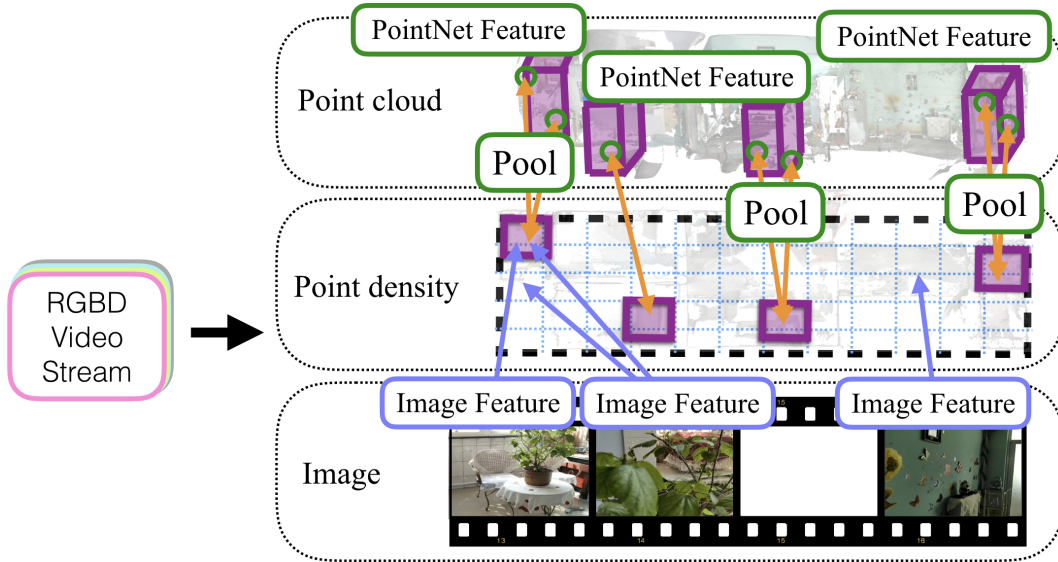


Figure 5.2: FloorNet shares features across branches to exploit the best of all the architectures. PointNet features at 3D points are pooled into corresponding 2D cells in the floorplan branch. Floorplan features at 2D cells are unpooled to the corresponding 3D points in the PointNet branch. Deep image features are pooled into corresponding 2D cells in the floorplan branch, based on the depthmap and the camera pose information.

Floorplan to PointNet unpooling: This module reverses the above pooling operation. It simply copies and adds a feature of the floorplan cell into each of the corresponding 3D points that project inside the cell. The time complexity is again linear in the number of points.

Image to floorplan pooling: The image branch produces two deep image features of dimensions $512 \times 32 \times 32$ and $256 \times 64 \times 64$ from DRN and SH for each video frame. We first unproject image features to 3D points by their depthmaps and camera poses, then apply the same 3D to floorplan pooling above. One modification is that we use max-pooling at all the layers in projecting 3D points onto the floorplan domain to be simple, instead of the mix of sum and max poolings. The reason is that we use pre-trained models for image feature encoding, and more complex hybrid pooling would have less effects. We conduct the image branch pooling for every 10 frames in the video sequence.

5.2.4 Loss functions

Our network outputs pixel-wise predictions on the floorplan geometry and semantics information in the same resolution 256×256 . For geometry heatmaps (i.e., room corners, object icon-corners, and door/window end-points), a sigmoid cross entropy loss is used. The ground-truth heatmap is prepared by putting a value of 1.0 inside a disk of radius 11 pixels around each ground-truth pixel. For semantic classification feature maps (i.e., room types and object icon types), a pixel-wise softmax cross entropy loss is used.

5.3 Floorplan reconstruction benchmark

This paper creates a benchmark for the vector-graphics floorplan reconstruction problem from RGBD videos with camera poses. We have acquired roughly two-hundreds 3D scans of residential units in the United States and China using Google Tango phones (Lenovo Phab 2 Pro and Asus ZenFone AR) (See Fig. 6.4). After manually removing poor quality scans, we have annotated the complete floorplan information for the remaining 155 scans: 1) room-corners as points, 2) walls as pairs of room-corners, 3) object icons and types as axis-aligned rectangles and classification labels, 4) doors and windows (i.e., openings) as line-segments on walls, and 5) room types as classification labels for polygonal areas enclosed by walls. The list of object types is $\{counter, bathtub, toilet, sink, sofa, cabinet, bed, table, refrigerator\}$. The list of room types is $\{living\ room, kitchen, bedroom, bathroom, closet, balcony, corridor, dining\ room\}$. Table 5.1 provides statistics of our data collections.

Reconstructed floorplans are evaluated on three different levels of geometric and semantic consistency with the ground-truth. We follow the work by Liu et al. [46] and define the low- and mid-level metrics as follows.

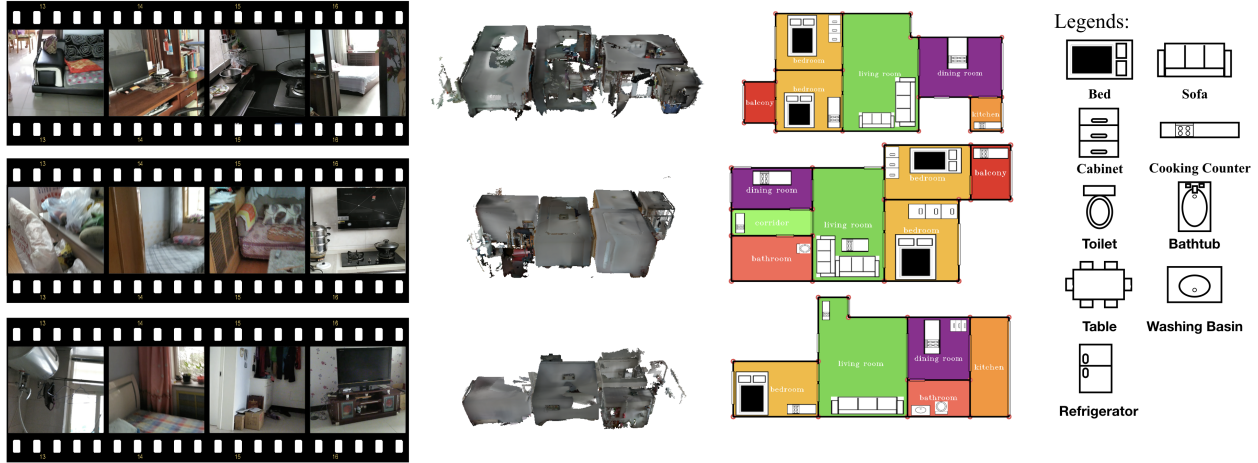


Figure 5.3: Floorplan reconstruction benchmark. From left to right: Subsampled video frames, colored 3D point clouds, and ground-truth floorplan data. The floorplan data is stored in a vector-graphics representation, which is visualized with a simple rendering engine (e.g., rooms are assigned different colors based on their types, and objects are shown as canonical icons).

Table 5.1: Dataset statistics. From left to right: the number of rooms, the number of icons, the number of openings (i.e., doors or windows), the number of room-corners, and the total area. The average and the standard deviation are reported for each entry.

	#room	#icon	#opening	#corner	area
Average	5.2	9.1	9.9	18.1	63.8[m ²]
Std	1.8	4.5	2.9	4.2	13.0[m ²]

- The low-level metric is the precision and recall of room corner detections. A corner detection is declared a success if its distance to the ground-truth is below 10 pixels and the closest among all the other room corners.
- The mid-level metric is the precision and recall of detected openings (i.e., doors and windows), object-icons, and rooms. The detection of an opening is declared a success if the largest distance of the corresponding end-points is less than 10 pixels. The detection of an object (resp. a room) is declared a success if the intersection-over-union (IOU) with the ground-truth is above 0.5 (resp. 0.7).

- Relationships of architectural components play crucial roles in evaluating indoor spaces. For example, one may look for apartments where bedrooms are not connected to a kitchen. A building code may enforce every bedroom to have a quick evacuation route to outside through windows or doors in case of fire. We introduce the high-level metric as the ratio of rooms that have the correct relationships with the neighboring rooms. More precisely, we declare that a room has a correct relationship if 1) it is connected to the correct set of rooms through doors, where two rooms are connected if their common walls contain at least one door, 2) the room has an IOU score larger than 0.5 with the corresponding ground-truth, and 3) the room has the correct room type.

5.4 Implementation details

5.4.1 DNN Training

Among the 155 scans we collected, we randomly sample 135 for training and leave 20 for testing. We perform data augmentation by random scaling and rotation every time we feed a training sample. First, we apply rescaling to the point-cloud and the annotation with a random factor uniformly sampled from a range $[0.5, 1.5]$. Second, we randomly apply the rotation around the z axis by either 0° , 90° , 180° , or 270° .

We have utilized the official code for the two image encoders DRN [79] and SH [55]. We pre-trained DRN on the semantic segmentation task with ScanNet database [16] and SH on the room layout estimation task with LSUN [84]. DRN and SH are fixed during the FloorNet training. We implemented the remaining DNN modules by ourselves in TensorFlow with the modern APIs, that is, PointNet [56] for the Pointnet branch and FCN [53] for the Floorplan branch.

Training of FloorNet takes around 2 hours with a TitanX GPU. We set the batch size to 6. FloorNet has three types of loss functions. To avoid overfitting with the icon loss, we trained icon-loss separately for at most 600 epochs with early-stopping based on the testing loss. The others losses are trained jointly for 600 epochs. [¶] The training consumes $81,000 = 135(\text{samples}) \times 600(\text{epochs})$ augmented training samples. It is initially to our surprise that FloorNet generalizes even from a small number of 3D scans. However, FloorNet makes pixel-wise low-level predictions. Each 3D scan contains about 10 object-icons, 10 openings, and a few dozen room corners, which probably lead to the good generalization performance together with data augmentation, where similar phenomena were observed by Liu et al. [46]

5.4.2 Enhancement heuristics

We augment the Integer Programming Formulation [46] with the following two enhancement heuristics to deal with more challenging input data (i.e., large-scale raw sensor data) and hence more noise in the network predictions.

Primitive candidate generation: Standard non-maximum suppression often detects multiple room corners around a single ground-truth. After thresholding the room-corner heatmap by a value 0.5, we simply extract the highest peak from each connected component, whose area is more than 5 pixels. To handle localization errors, we connect two room-corners and generate a wall candidate when their corresponding connected components overlap along X or Y direction. We do not augment junctions to keep the number of candidates tractable.

[¶] We considered synthetic dataset SUNCG [67] and real dataset Matterport3D [14] for training with the icon loss, while using their semantic segmentation information to produce icon annotations. However, the joint-training still experiences overfitting, while this simple early-stopping heuristic works well in our experiments.

Objective function: Wall and opening candidates are originally assigned uniform weights in the objective function [46]. We calculate the confidence of a wall (resp. opening) candidate by taking the average of the semantic heatmap scores of type "wall" along the line with width 7 pixels (resp. 5 pixels). We set the weight of each primitive by the confidence score minus 0.5, so that a primitive is encouraged to be chosen only when the confidence is at least 0.5.

5.5 Experiments

Figure 5.4 shows our reconstruction results on some representative examples. Our approach successfully recovers complex vector-graphics floorplan data including room geometries and their connectivities through doors. One of the major failure modes is in the icon detection, as object detection generally requires more training data than low-level geometry detection [46]. We believe that more training data will overcome this issue. Another typical failures come from missing room corners due to clutter or incomplete scanning. The successful reconstruction of a room requires successful detection of every room corner. This is a challenging problem and the introduction of higher level constraints may reveal a solution.

Figure 5.6 and Table 5.2 qualitatively and quantitatively compare our method against competing techniques, namely, OctNet [60], PointNet [56], and a few variants of our FloorNet. OctNet and PointNet represent state-of-the-art 3D DNNs. More precisely, we implement the voxel semantic segmentation network based on the official OctNet library,[‡] which takes 256x256x256 voxels as input and outputs 3D voxels of the same resolution. We then add three separate $5 \times 3 \times 3$ convolution layers with strides $4 \times 1 \times 1$ to predict the same pixel-wise geometry and semantics feature-maps with the same set of loss functions. PointNet is simply our FloorNet without the point density or the image input. Similarly, we construct a FloorNet

[‡]OctNet library: <https://github.com/griegler/octnet>

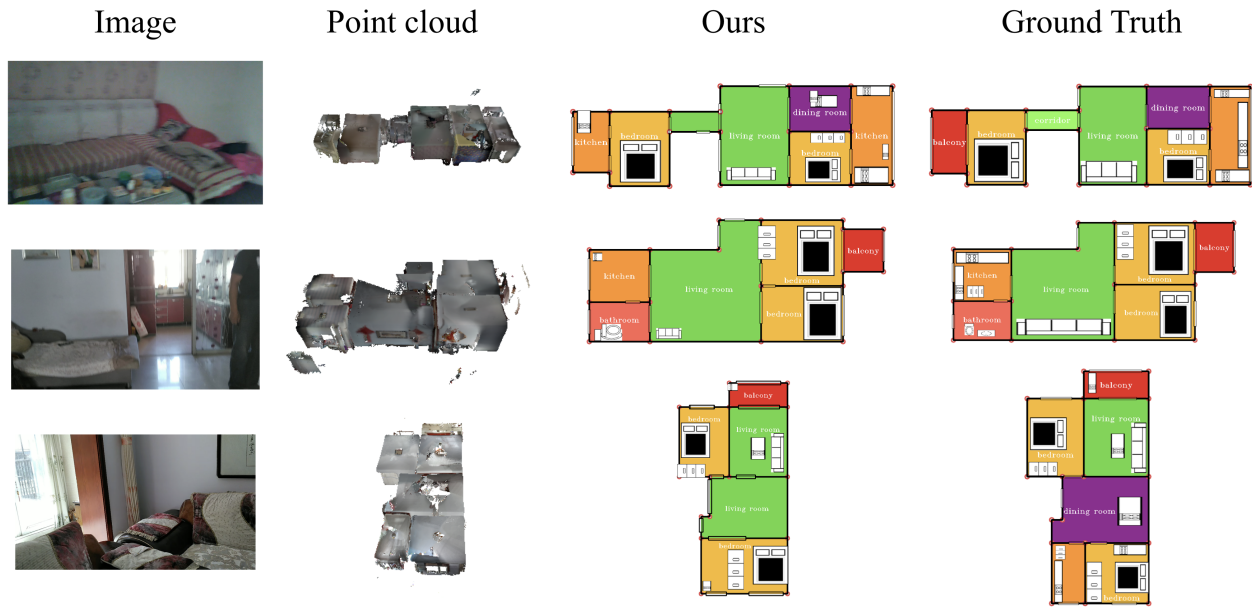


Figure 5.4: Floorplan reconstruction results.

variant by enabling only the 3D points (for the PointNet branch) or the point density image (for the floorplan branch) as the input.

The table shows that the floorplan branch is the most informative as it is the most natural representation for floorplan reconstruction task, while PointNet branch or Image branch alone does not work well. We also split the entire point clouds into $1m \times 1m$ blocks, train the PointNet-only model that makes predictions per block separately, followed by a simple merging. However, this performs much worse. OctNet performs reasonably well across low- to mid-level metrics, but does poorly on the high-level metric, where all the rooms and relevant doors must be reconstructed at high precision to report good numbers.

To further evaluate the effectiveness of the proposed FloorNet architecture, we conduct ablation studies by disabling each of the inter-branch pooling/unpooling operations. The bottom of Table 5.2 shows that the feature sharing overall leads to better results, especially for mid- to high-level metrics.

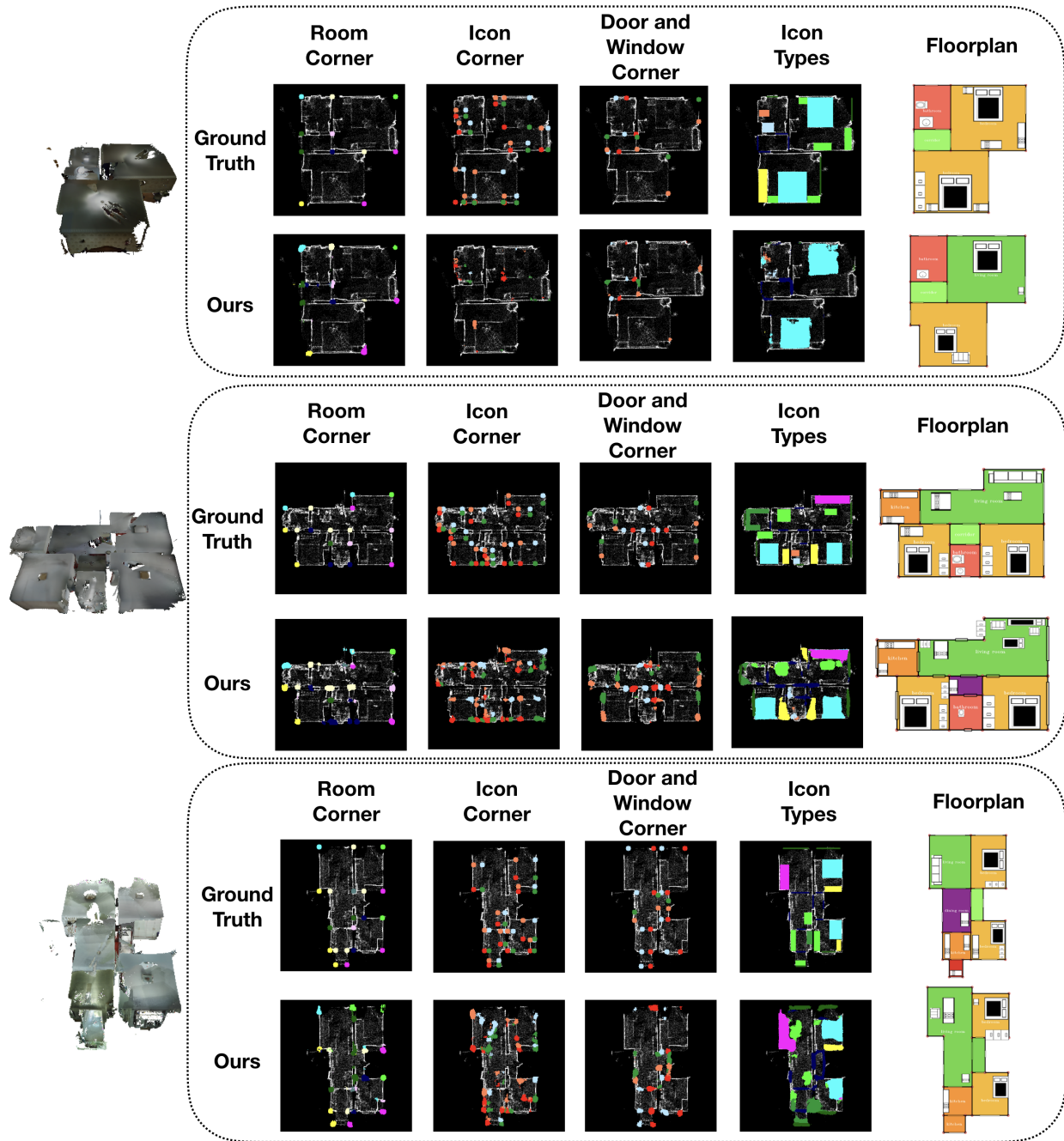


Figure 5.5: Intermediate results. For each example, we show raw outputs of the networks (room corners, icon corners, opening corners, and icon types) compared against the ground-truth. In the second example, we produce a fake room (blue color) at the top due to poor quality 3D points. In the third example, reconstructed rooms have inaccurate shapes near the bottom left again due to noisy 3D points, illustrating the challenge of our problem.

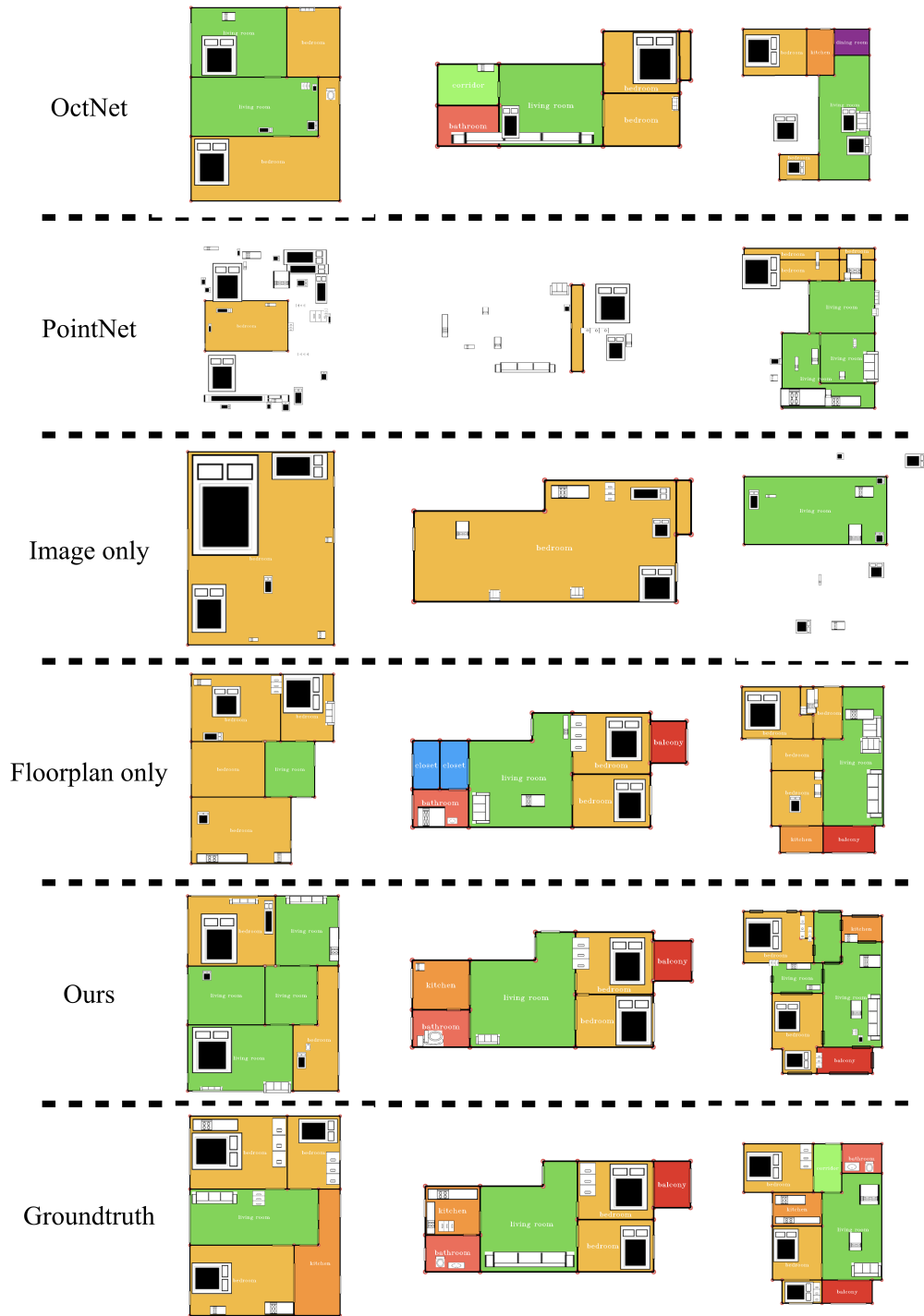


Figure 5.6: Qualitative comparisons against competing methods. The top is OctNet [60], a state-of-the-art 3D CNN architecture. The next three rows show variants of our FloorNet, where only one branch is enabled. FloorNet with all the branches overall produce more complete and accurate floorplans.

Table 5.2: Quantitative evaluations on low-, mid-, and high-level metrics against competing methods and our variants. The orange and cyan color indicates the best and the second best result for each entry.

	wall	door	icon	room	relation
PointNet [56]	25.8/42.5	11.5/38.7	22.5/27.9	27.0/40.2	5.0
Floorplan-branch	90.2/88.7	70.5/78.0	43.4/42.8	76.3/75.3	50.0
Image-branch	40.0/83.3	15.4/47.1	21.4/17.4	25.0/57.1	0.0
OctNet [60]	75.4/89.2	36.6/82.3	32.8/48.8	62.1/72.0	13.5
Ours w/o PointNet-Unpooling	92.6 /92.1	75.8/76.8	55.1/51.9	80.9/77.4	52.3
Ours w/o PointNet-Pooling	88.4/ 93.0	73.0/ 87.2	50.0/42.2	75.0/80.6	52.8
Ours w/o Image-Pooling	92.6 /89.7	77.1 /74.4	50.5/ 57.8	84.2 /83.1	56.8
Ours	92.1/92.8	76.7/80.2	56.1 / 57.8	83.6/ 85.2	56.8

Table 5.3: In the PointNet to floorplan inter-branch pooling, we use a mix of sum and max pooling in projecting 3D points onto the 2D floorplan domain. To validate this hybrid scheme, we have also evaluated the performance when only the max-pooling or the sum-pooling is used at all the layers.

Pooling Method	wall	door	icon	room	relationship
Max	88.0/89.9	70.9/86.6	59.1/47.8	76.3/77.3	47.0
Sum	88.3/97.0	69.6/85.9	55.6/53.4	76.3/82.9	52.3
Sum/Max (default)	92.1/92.8	76.7/80.2	56.1/57.8	83.6/85.2	56.8

Table 5.3 compares different inter-branch pooling/unpooling schemes for the PointNet to floorplan pooling. The table shows that the max operation in early layers loses too much information and leads to worse performance.

Finally, Figure 5.7 compares against a build-in Tango Navigator App [35], which generates a floorplan image real-time on the phone. Note that their system does not 1) produce room segmentations, 2) recognize room types, 3) detect objects, 4) recognize object types, or 5) produce CAD-quality geometry. Therefore, we quantitatively evaluate only the geometry information by measuring the line distances between the ground-truth walls and predicted walls. More precisely, we 1) sample 100 points from each wall line segment, 2) for each



Figure 5.7: Comparison against a commercial floorplan generator, Tango Navigator App. Top: Floorplan image from Tango. Bottom: Our results.

sampled point, find the closest one in the other line segment, and 3) compute the mean distance over all the sampled points and line segments. The average line distances are 2.72 [pixels] and 1.66 [pixels] for Tango Navigator App and our FloorNet, respectively. This is a surprising result, because our algorithm drops many confident line segments during Integer Programming, when the corresponding room is not reconstructed. On the other hand, it is an expected result as our approach utilizes all the geometry and image information.

5.6 Discussions

This paper proposes a novel DNN architecture FloorNet that reconstructs vector-graphics floorplans from RGBD videos with camera poses. FloorNet takes a hybrid approach and exploits the best of three DNN architectures to effectively process a RGBD video covering a large 3D space with complex camera motions. The paper also provides a new benchmark for a new vector-graphics reconstruction problem, which is missing in the recent indoor scene databases of Computer Vision. Two main future works are ahead of us. The first one is

to learn to enforce higher level constraints inside DNNs as opposed to inside a separate post-processing (e.g., Integer Programming). Learning high-level constraints likely require more training data and the second future work is to acquire more scans.

More than 90% of houses in North America do not have floorplans. We hope that this paper together with the benchmark will be an important step towards solving this challenging vector-graphics reconstruction problem, and enabling the reconstruction of a floorplan just by walking through a house with a smartphone. We publicly share our code and data to promote further research.

Despite the immediate impact in industrial applications, FloorNet has three major failure modes. First, as in any bottom-up process, missing corners in the detection phase automatically lead to missing walls and rooms in the final model. Second, false candidate primitives could lead to the reconstruction of extraneous walls and rooms. Third, to enable the usage of powerful IP, FloorNet needs to restrict the solution space to Manhattan scenes. To address these issues, we propose a top-down approach, dubbed FloorPlotter, to first segment room instances and then formulates an energy optimization problem that reconstructs a floorplan as multiple closed polygonal curves, one for each room. Our sound optimization strategy enables FloorPlotter to reconstruct floorplans with complex shapes and non-Manhattan structures.

Chapter 6

FloorPlotter: Towards Ultimate Inverse CAD System for Floorplans

6.1 Introduction

Architectural floorplans play a crucial role in designing, understanding, and remodeling indoor spaces. Automated floorplan reconstruction from raw sensor data is a major milestone in indoor mapping research. The core technical challenge lies in the inference of wall graph structure, whose topology is unknown and varies per example.

Computer Vision has made remarkable progress in the task of graph inference, for instance, human pose estimation [12] and hand tracking [81]. Unfortunately, the success has been limited to the cases of fixed known topology (e.g., a human has two arms). Inference of graph structure with unknown varying topology is still an open problem.

A popular approach to graph reconstruction is *primitive detection and selection* [24, 76, 54], for example, detecting corners, selecting subsets of corners to form edges, and selecting subsets of edges to form regions. The major problem of this bottom-up process is that it cannot recover from a single false-negative in an earlier stage (i.e., a missing primitive). The task becomes increasingly more difficult as the primitive space grows exponentially with their degrees of freedom, especially for non-Manhattan scenes which most existing methods do not handle [24, 11, 46, 47].

This paper seeks to make a breakthrough in the domain of floorplan reconstruction with two key ideas.

- First, we start from a top-down process of room segmentation via instance semantic segmentation technique [29]. The room segmentation reduces the floorplan graph inference into the reconstruction of multiple polygonal loops, one for each room. This reduction allows us to formulate floorplan reconstruction as sound energy optimization as opposed to a series of primitive detection and selection heuristics.
- Second, we employ *room-wise coordinate descent* strategy in optimizing the objective function. By exploiting the fact that the room topology is a simple loop, our formulation finds the (near-)optimal graph structure for each room one by one by dynamic programming, while enforcing consistency with the other rooms.

More concretely, the objective function in the energy optimization consists of three terms: 1) Data term, measuring the discrepancy with the input sensor data by deep neural networks; 2) consistency term, encouraging adjacent rooms to share corners and walls at the room boundaries; and 3) model complexity term, penalizing the number of corners in the graph. The optimization produces multiple loops, and simple graph merging operations are applied at the end to generate final floorplan graphs.

We have evaluated the proposed approach on production-quality RGBD scans of 527 apartments or houses, a few times larger than the current biggest database [47]. Our approach makes significant improvements over the current state-of-the-art [47]. We will share the code and data with the community.

This is a joint work with Jiacheng Chen, who implemented most of the system, while my role is to advise him.

6.2 FloorPlotter: System Overview

FloorPlotter turns aligned panorama RGBD images into a floorplan graph in three phases: room segmentation, room-aware floorplan reconstruction, and loop merging (See Figure 6.1). This section provides the system overview with minimal details. Unlike FloorNet [47], we focus only on the wall structures, where doors/windows, icons, and room semantics can be easily added later.

Room segmentation: After converting the input panorama scans into a 4-channel 256×256 point-density/normal image in a top-down view, we utilize instance semantic segmentation technique (Mask R-CNN [29]) to find room segments. Note that the point-density is simply the number of 3D points falling into the pixel, and the point-normal is the sum of the surface-normal vectors associated with the 3D points, normalized to a unit-length vector.

Room-aware floorplan reconstruction: Given a set of room segments, we formulate an optimization problem that reconstructs a floorplan graph as multiple polygonal loops, one for each room. Deep neural networks derive data terms in the objective. We propose a novel room-wise coordinate descent algorithm that directly optimizes the number and placement of corners while minimizing the objective.

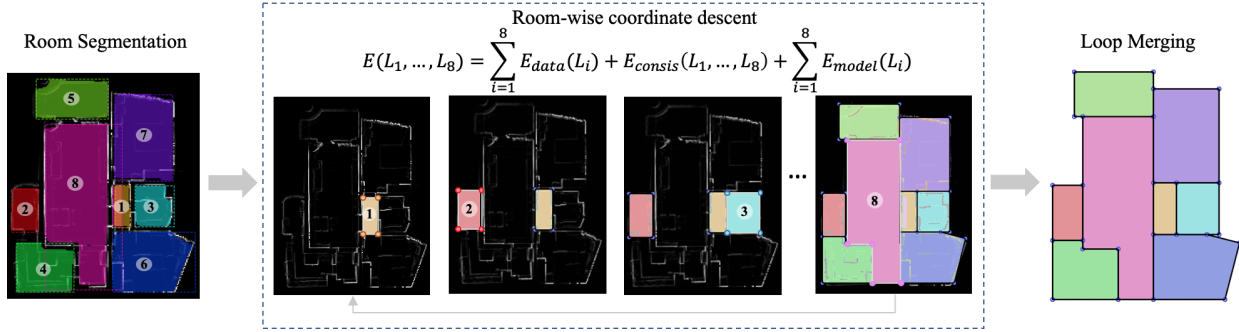


Figure 6.1: System overview: (Left) Mask-RCNN finds room segments from a top-down projection image consisting of point density and mean surface normal, allowing us to reconstruct a floorplan as multiple loops. (Middle) Room-wise coordinate descent optimizes room structures one by one by minimizing the sum of data, consistency, and model complexity terms. (Right) Simple graph merging operations combine loops into a floorplan graph structure.

Loop merging: Simple graph merging operations combine multiple polygonal loops into a final floorplan graph.

Room-aware floorplan reconstruction is the technical core of the paper, where Section 6.3 defines the problem formulation, and Section 6.4 presents the optimization algorithm. Room segmentation and loop merging are based on existing techniques, where Section 6.5 provides their algorithmic details and the remaining system specifications.

6.3 Room-aware floorplan reconstruction

The room segmentation (R_i) from Mask R-CNN allows us to reduce the floorplan graph inference into the reconstruction of multiple loops (L_i), one for each room. L_i is defined as a sequence of pixels at integer coordinates forming a polygonal curve with a loop topology.

Our problem is to minimize the following objective with respect to the set of loops \mathcal{L} :

$$\sum_{L_i \in \mathcal{L}} E_{data}(L_i) + E_{consis}(\mathcal{L}) + \sum_{L_i \in \mathcal{L}} E_{model}(L_i),$$

subject to L_i being a loop containing R_i inside.

Data term: E_{data} is a room-wise unary potential, measuring the discrepancy with the input sensor data over the set of pixels along each loop.

$$E_{data}(L_i) = \sum_{p \in \mathbb{C}(L_i)} \lambda_1 E_{data}^{\mathbb{C}}(p) + \sum_{p \in \mathbb{E}(L_i)} [\lambda_2 E_{data}^{\mathbb{E}}(p) + \lambda_3 E_{data}^{\mathbb{I}}(p)].$$

- $E_{data}^{\mathbb{C}}(p)$ is the penalty of placing a corner at pixel p (see Figure 6.2a), and hence, summed over all the corner pixels $\mathbb{C}(L_i)$ on L_i . The penalty is defined as one minus the pixel-wise corner likelihood. We estimate the corner likelihood map from a point density/normal image using Dilated Residual Networks (DRN) [80].
- $E_{data}^{\mathbb{E}}(p)$ is the penalty of placing an edge over a pixel p . The term is defined as one minus the pixel-wise edge likelihood (see Figure 6.2b), summed over all the edge pixels $\mathbb{E}(L_i)$ along L_i . We use Bresenham’s line algorithm to obtain edge pixels given corners. The same DRN estimates the edge likelihood.
- $E_{data}^{\mathbb{I}}(p)$ is also the penalty summed over the edge pixels, which enforces L_i not to pass through the room segment R_i . The term is a large constant if a pixel belongs to any of the room segments and 0 otherwise.

Consistency term: E_{consis} is a room-wise higher-order potential, encouraging loops to be consistent at the room boundaries (i.e., sharing corners and edges). We define the penalty to

be the number of pixels that are *used by* the corners (or edges) of all the loops together. For instance, if two corners are close to each other, this term suggests to move them to the same pixel so that penalty is imposed only once:

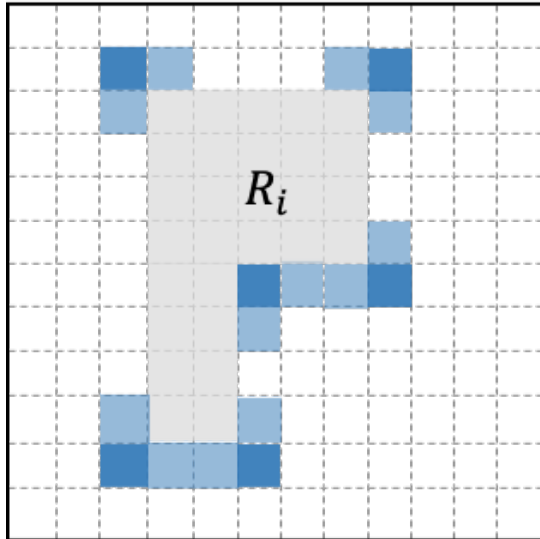
$$E_{consis}(\mathcal{L}) = \sum_p [\lambda_4 \mathbf{1}_C(p, \mathcal{L})] + \sum_p [\lambda_5 \mathbf{1}_E(p, \mathcal{L})]$$

The first term $\mathbf{1}_C(p, \mathcal{L})$ is an indicator function, which becomes 1 if a pixel (p) is a corner of at least one loop. Similarly, the second term is an indicator function for edges. See Figure 6.2 for the illustration over the toy examples.

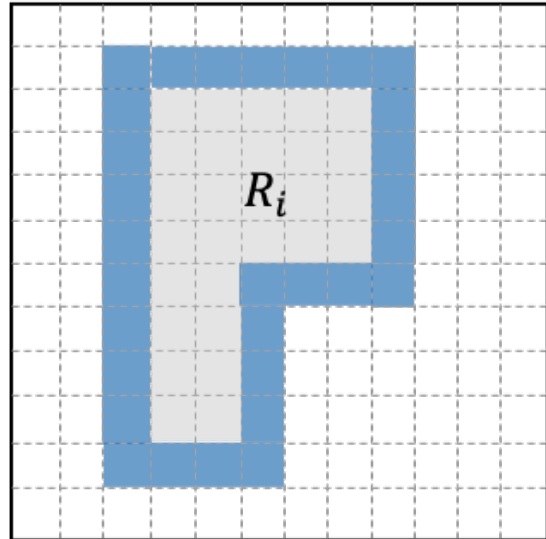
Model complexity term: E_{model} is the model complexity penalty, counting the number of corners in our loops, preferring compact shapes.

$$E_{model}(L_i) = \lambda_6 \{\# \text{ of corners in } L_i\}.$$

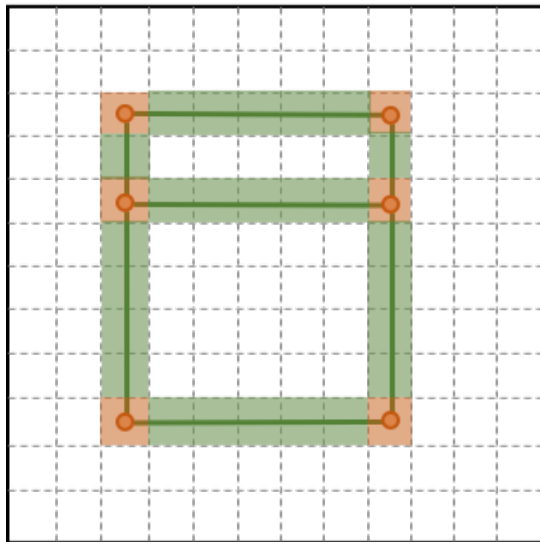
$\lambda_?$ are scalars defining the relative weights of the penalty terms. We found our system robust to these parameters and use the following setting throughout our experiments: $\lambda_1 = 0.2$, $\lambda_2 = 0.2$, $\lambda_3 = 100.0$, $\lambda_4 = 0.2$, $\lambda_5 = 0.1$, $\lambda_6 = 1.0$.



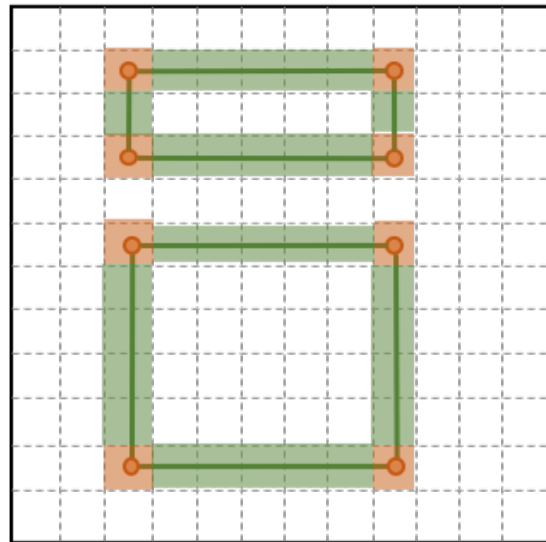
(a) E_{data}^c



(b) E_{data}^e



(c) $E_{consis} = \lambda_4 * 6 + \lambda_5 * 25$



(d) $E_{consis} = \lambda_4 * 8 + \lambda_5 * 30$

Figure 6.2: Illustration of data and consistency terms. E_{data}^c and E_{data}^e are defined based on corner and edge likelihood maps. Blue pixels indicate lower costs in these toy examples. E_{consis} counts the number of pixels used by room corners and room edges. When neighboring rooms share corners and edges as shown in (c), E_{consis} goes down.

6.4 Room-wise coordinate descent

The inspiration of our optimization strategy comes from a prior work, which solves a shortest path problem and reconstructs a floorplan as a loop [11]. This formulation considers every pixel as a node of a graph, encodes objectives into edge weights, and finds the shortest path as a loop.

Our problem solves for multiple loops over multiple rooms. We devise *room-wise coordinate descent strategy* that optimizes room shapes one by one by reducing a room-wise coordinate descent step into the same shortest path problem. We visit rooms in increasing order of their areas so that smaller/easier rooms are handled first. We repeat two rounds of optimization.

This section explains 1) Shortest path problem reduction; 2) Containment constraint satisfaction; and 3) Two approximation methods for speed-boost.

Shortest path problem reduction: The reduction process is straightforward, as our cost function is the summation of pixel-wise penalties and the number of corners. Without loss of generality, suppose we are optimizing L_1 while fixing the other loops. Our optimization problem is equivalent to solving a shortest path problem for R_1 with the following weight definition for each edge (e) (See the supplementary document for the derivation):

$$\sum_{p \in \mathbb{C}(e)} \frac{\lambda_1}{2} E_{data}^{\mathcal{C}}(p) + \sum_{p \in \mathbb{E}(e)} [\lambda_2 E_{data}^{\mathcal{E}}(p) + \lambda_3 E_{data}^{\mathcal{I}}(p)] + \sum_{p \in \mathbb{E}(e)} [\lambda_4 (1 - \mathbf{1}_{\mathcal{C}}(p, \mathcal{L} \setminus \{L_1\})) + \lambda_5 (1 - \mathbf{1}_{\mathcal{E}}(p, \mathcal{L} \setminus \{L_1\}))] + \lambda_6.$$

With abuse of notation, $\mathbb{C}(e)$ denotes the two pixels at the end-points of e , $\mathbb{E}(e)$ denotes the set of pixels along e obtained by Bresenham’s line algorithm, and $\mathcal{L} \setminus \{L_1\}$ denotes the set of loops excluding L_1 .

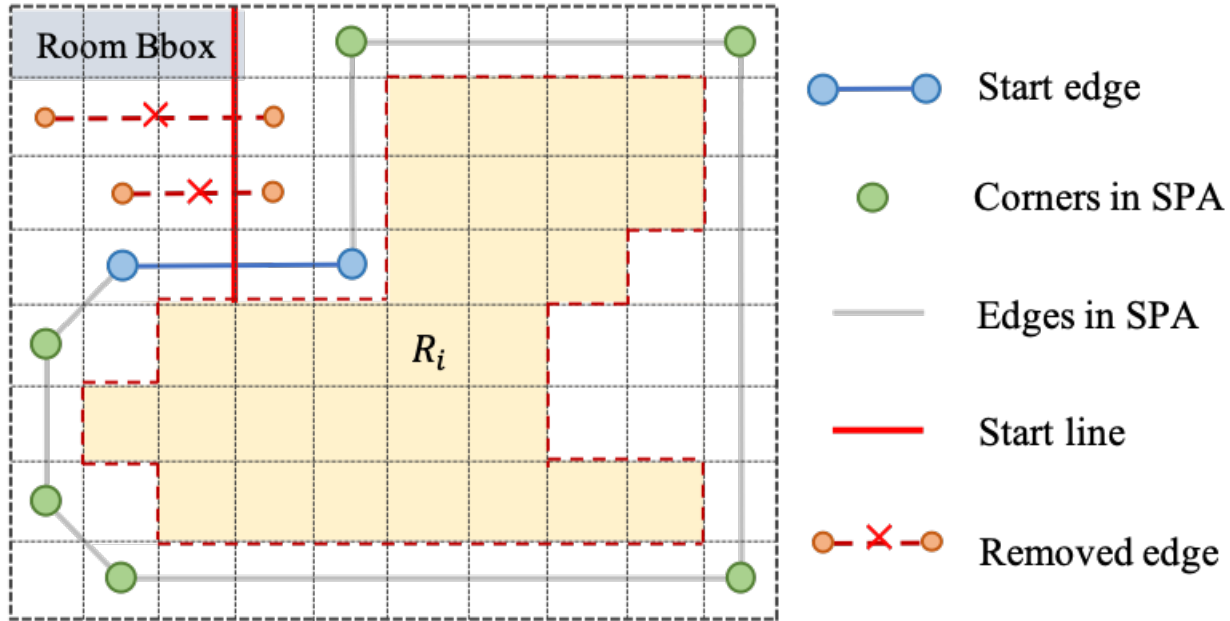


Figure 6.3: We solve a shortest path problem for each room, where cost functions are encoded into edge weights. In order to avoid a trivial solution (i.e., an empty graph) and enforce the path to go around the room segment (R_i), we first identify a start-edge that is a part of a room shape with high-confidence. Next, we draw a (red) start-line perpendicularly to split the domain. We prohibit crossing the start-line, assign a very high penalty for going through R_i , then solve for a shortest path that starts and ends at the two end-points of the start-edge.

Containment constraint satisfaction: Shortest path is a powerful formulation that searches for the optimal number and placement of corners with one caveat: An additional constraint is necessary to avoid a trivial solution (i.e., an empty loop). We use a heuristic similar in spirit to the prior work [11] to implement this constraint: “ L_i contains (or goes around) R_i ”. We refer the details to the supplementary document and here summarize the process.

First, we find corner candidates from the same corner likelihood map used for the data term (see Figure 6.3). Second, we look at the edge likelihood map to identify a good pair of corners forming the *start-edge* of the loop. Third, we draw a *start-line* that starts from the room mask (R_i) and passes through the start-edge perpendicularly at its middle point. Lastly, we

remove all the edges that intersect with the start-line to ensure that the path must go around R_i .

Note that fixing the start-edge to be part of the loop breaks the local optimality of our coordinate descent step, but works well in practice as it is not difficult to identify one wall segment with high confidence.

Bounding box approximation: We make two approximations in pruning edges to reduce the computational expenses of the shortest path algorithm (SPA), which is $O(N^2 \log N)$, where N is the number of pixels. The first method restricts the domain, as it is wasteful to run SPA over an entire image domain to reconstruct one room. Given a room mask R_i , we apply the binary dilation 10 times to expand the mask and find its axis-aligned bounding box with a 5-pixel margin, in which we solve SPA.

Dominant direction approximation: FloorPlotter goes beyond the conventional Manhattan assumption by allowing multiple Manhattan frames per room. We train the same DRN architecture to estimate the wall direction likelihoods in an increment of 10 degrees at every pixel. We perform a simple statistical analysis to extract four Manhattan frames globally, then assign its subset to each room. We allow edges only along the selected dominant directions with some tolerance on discretization errors (See the supplementary document for the details).

6.5 System Details

This section provides the details of the room segmentation and the loop merging algorithms with the remaining system specifications.

Room segmentation: Given a set of panorama RGBD scans where the Z axis is aligned with the gravity direction, we compute the tight axis-aligned bounding box of the points on the horizontal plane. We expand the rectangle by 2.5% in each of the four directions, apply non-uniform scaling into a 256×256 pixel grid, and compute the point density and normal in each pixel. The point density is the number of 3D points that fall inside the pixel, which we linearly scale to an image intensity so that the highest density becomes 255. The point normal is the sum of surface normal vectors associated with the 3D points in that pixel. We normalize to a unit length vector and map each coordinate in the range of $[-1.0, 1.0]$ to the intensity range $[0, 255]$.

We use the publicly available Mask R-CNN implementation [19] with the default hyperparameters except that we lower the detection threshold from 0.7 to 0.2. Given a segment from Mask R-CNN, we apply the binary erosion operation for 2 iterations with 8-connected neighborhood to obtain room segments (R_i).

Loop merging: We use simple graph merging operations to convert loops into the final floorplan graph structure. More concretely, we denote a contiguous set of colinear line segments as a *segment group*. We repeatedly identify a pair of parallel segment groups within 5 pixels and snap them into a new segment group at the middle point while merging corners. After applying the edge merging to all compatible pairs, we merge corners that are within 3 pixels.

System details: To estimate pixel-wise likelihoods. We use the official implementation of Dilated Residual Networks [80], which produces 32×32 feature maps. In order to produce an output in the same resolutions as the input, we add 3 extra layers of residual blocks [22] with transposed convolution of stride 2 to reach the resolution of 256×256 . For the corner likelihood supervision, we render each ground truth corner as a 7×7 disk. For the edge

likelihood and wall-direction supervision, we draw the edge mask and direction information with a width of 5 pixels.

6.6 Experiments

We have evaluated the proposed system on 527 sets of aligned panorama RGBD scans. The average numbers of 1) input 3D points for the point-density/normal image, 2) corners in the annotations, 3) wall segments in the annotations, and 4) rooms in the annotations are 432,552, 28.87, 35.88, and 7.73, respectively. Out of 4072 rooms, 489 rooms do not follow the primary Manhattan structure of the unit. Figure 6.4 shows four examples from our dataset.

527 units are split into 433 and 94 for training and testing, respectively. We make the test set more challenging than the training set on purpose for evaluations. 48 out of 94 testing units contain challenging non-Manhattan structure. Similarly, 199 out of 667 testing rooms follow non-Manhattan geometry.

We have implemented the proposed system in Python while using PyTorch as the DNN library. We have used a workstation equipped with an NVIDIA 1080Ti with 12GB GPU memory. We trained the Mask-RCNN for 70 epochs with a batch size of 1, and the DRNs for 35 epochs with a batch size of 4. The training of each DNN model takes at most a day. At test time, it takes about 5 minutes on average to process one apartment/house, where the bottleneck is the construction of the graph for shortest path problems (a CPU-intensive task).

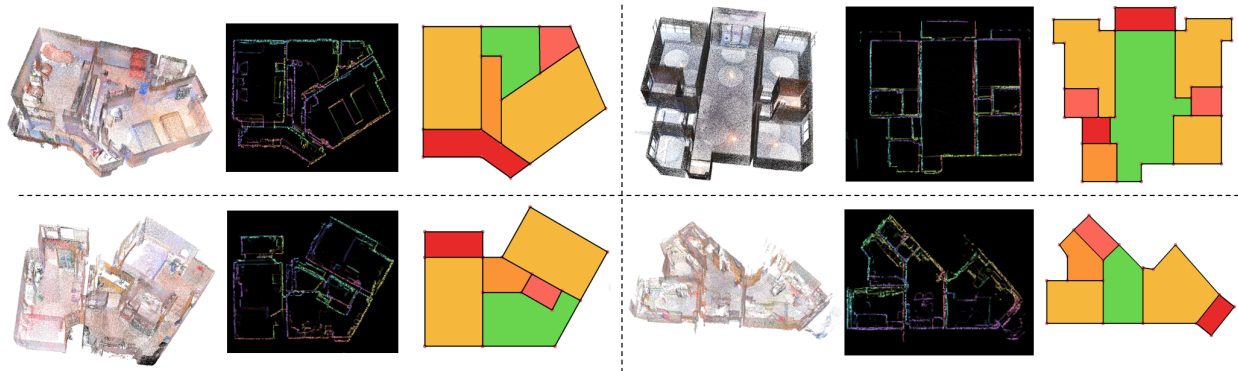


Figure 6.4: Our dataset offers production-level panorama RGBD scans for 527 houses/apartments. We convert each scan into a point density/normal image from a top-down view, which is the input to our system. We annotated floorplan structure as a 2D polygonal graph. Note that for point-density/normal images in the middle column, the intensity encodes the point density, and the hue/saturation encodes the 2D horizontal component of the mean surface normal.

Table 6.1: The main quantitative evaluation results. The colors cyan, orange, magenta represent the top three entries.

Method	Corner		Edge		Room		Room++	
	Prec.	Recall	Prec.	Recall	Prec.	Recall	Prec.	Recall
FloorNet [47]	95.0	76.6	94.8	76.8	81.2	72.1	42.3	37.5
Ours (w/o E_{data}, E_{consis})	84.4	80.4	82.3	79.8	75.1	61.3	23.3	22.0
Ours (w/o E_{consis})	93.9	82.3	89.2	81.2	83.8	81.7	49.4	48.5
Ours (1st-round)	94.6	82.8	89.4	81.7	83.9	81.8	49.5	48.7
Ours (2nd-round)	95.1	82.2	90.2	81.1	84.7	83.0	51.4	50.4

6.6.1 Qualitative evaluations

Figure 6.5 compares FloorPlotter against the current state-of-the-art floorplan reconstruction technique FloorNet [47] and the variants of our system. FloorNet follows a bottom-up process, where it first detects corners then uses Integer Programming to find valid connections between corners. FloorNet suffers from three failure modes: 1) Missing rooms due to missing corners in the first corner detection step; 2) Extraneous rooms coming from extraneous corner detections;

and 3) Broken non-Manhattan structures, which they cannot handle due to the excessive amount of search space in Integer Programming.

The right three columns show the variants of proposed FloorPlotter. The left does not have the consistency term and replaces the DNN based data term by the heuristic-based cost functions in the prior work [11]. Our overall formulation guarantees a room reconstruction at each detected room segment, producing reasonable results. On adding our DNN-based data term E_{data} (middle), per-room structure improves significantly. However, inconsistencies at the room boundaries are often noticeable. Lastly, with the addition of the consistency term (right), we see clean floorplan structures with consistent shared room boundaries.

Figure 6.6 illustrates the effect of room-wise coordinate descent over multiple rounds. Red ovals indicate challenging floorplan structure causing room overlaps or holes, which are resolved after the second round of the room-wise optimization.

6.6.2 Quantitative evaluations

We follow FloorNet [47] and define the following four metrics for the quantitative evaluations:

Corner precision/recall: We declare that a corner is successful reconstructed if there is a ground-truth room corner within 10 pixels. When multiple corners are detected around a single ground-truth corner, we only take the closest one as the correct reconstruction and treat the others as false-positives.

Edge precision/recall: We declare that an edge of a graph is successfully reconstructed if its two end-points pass the corner test described above and the corresponding edge belongs to the ground-truth.

Room precision/recall: We declare that a room is successfully reconstructed if 1) it does not overlap with any other room, and 2) there exists a room in the ground-truth with intersection-over-union (IOU) score more than 0.7. Note that this metric does not consider the positioning and sharing of corners and edges.

Room++ precision/recall: We declare that a room is successfully reconstructed in this metric, if the room is connected (i.e., sharing edges) to the correct set of successfully reconstructed rooms as in the ground-truth, besides passing the above two room conditions.

Table. 6.1 shows the main quantitative evaluations. Precision metrics on low-level primitives (i.e., corners and edges) are high for FloorNet as expected, because this task does not require high-level structural reasoning and the majority of the corners are easy ones (e.g., Manhattan corners). On the other hand, their recall metrics are low even for low-level primitives, because there often exist room corners that do not have ample 3D points due to occlusions where DNN based corner detection fails. FloorPlotter recovers such challenging corners through the room-level optimization process.

On room-level metrics, FloorPlotter is consistently better than FloorNet. Furthermore, the addition of the data and consistency terms also improve the room-level metrics. Finally, room-wise coordinate descent adds a further boost to the performance.


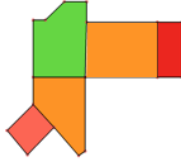
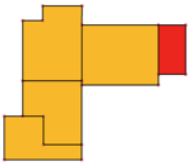
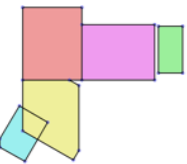
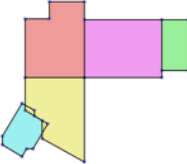
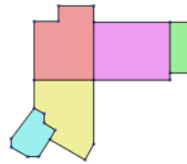
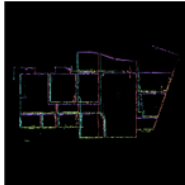





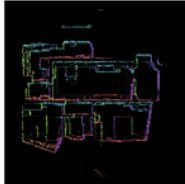





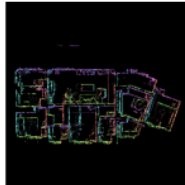





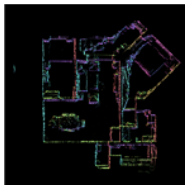
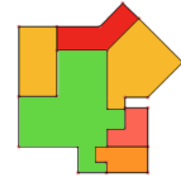
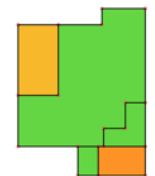
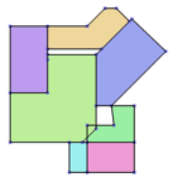
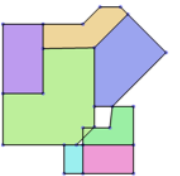
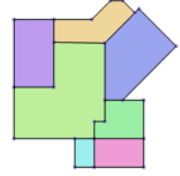


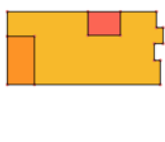
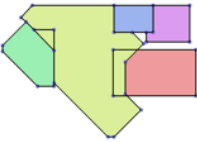
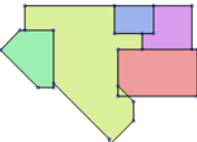
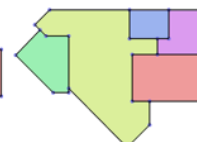


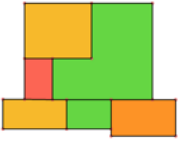
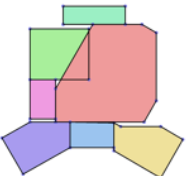
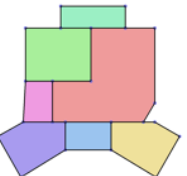
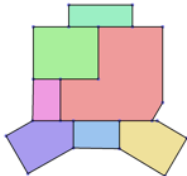
Point Cloud	Ground Truth	FloorNet	Ours (w/o E_{data}, E_{consis})	Ours (w/o E_{consis})	Ours
					
					
					
					
					
					
					

Figure 6.5: Qualitative comparisons against FloorNet [47] and the variants of our approach. We select hard non-Manhattan examples here to illustrate the reconstruction challenges in our dataset. For reconstructions by FloorPlotter variants, room colors are determined by corresponding room segments from Mask R-CNN. For the ground-truth and the FloorNet, colors are based on the room types.

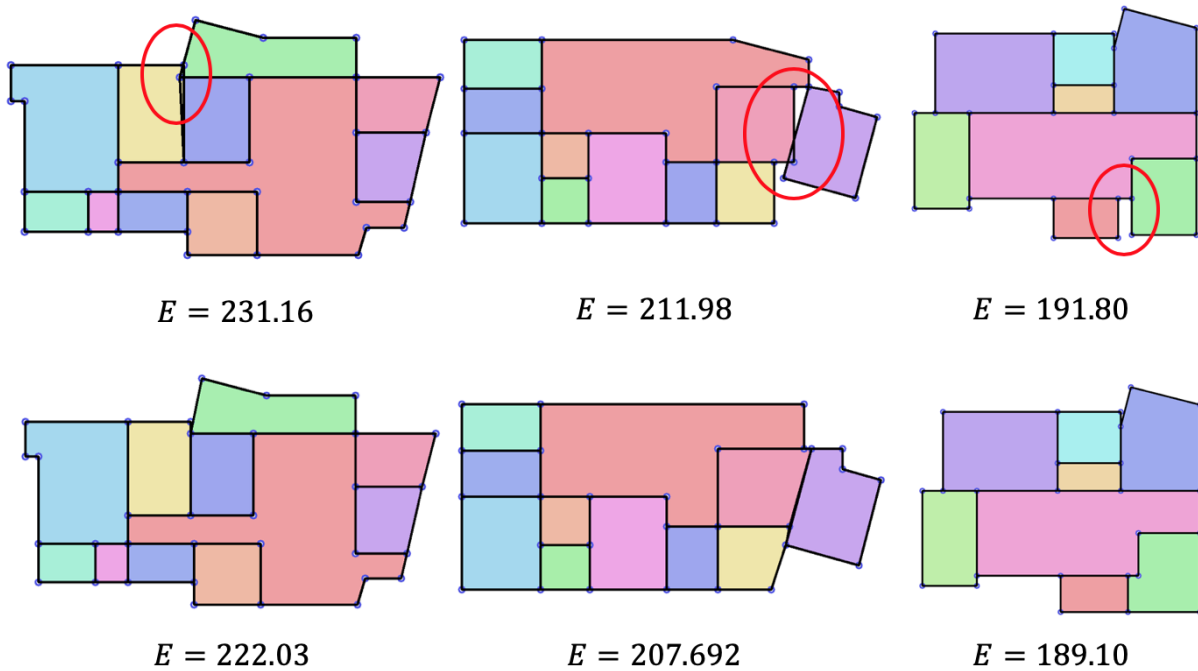


Figure 6.6: Multiple rounds of the coordinate descent fix mistakes at challenging floorplan structure. The top row shows the results after the first round of the coordinate descent optimization, and the bottom shows the results after the second round. We also show the total amount of energy after each round. Corresponding ground-truth annotations are found in Figure 6.5.

6.7 Discussions

FloorPlotter produces near-perfect results for Manhattan structures. The majority of the failures are concentrated on non-Manhattan cases. Quantitatively, our Room++ metrics are just slightly above 50. However, we would like to point out that our reconstructions are not terribly bad even in extremely challenging cases with poor Room++ metrics.

Look at the first example in Figure 6.7. Room++ precision and recall are both 0 with our reconstruction, while the reconstruction looks fairly reasonable. The reasons are threefold as marked by the numbers. 1) A small non-Manhattan room has wrong dominant directions in the pre-processing step, which makes it impossible for FloorPlotter to recover, and fails the IOU test; 2) Small details such as concave structures are hard to keep and the room fails the IOU test; 3) The room segmentation by Mask R-CNN makes a mistake for a very complex room, which is again impossible to recover. Once a single room fails, all the adjacent rooms automatically fail in the Room++ metric, leading to the zero precision and recall in this example.

We would like to also note that the input to our system is a single point-density/normal image from a top-down view. We have discarded the 3D information by projecting the points onto a 2D image. We have not utilized a set of high-resolution panorama RGB images. Like FloorNet [47], great opportunities arise in fusing such information to make the system more robust.

We believe that this paper sets a major milestone in indoor mapping research. The proposed system produces compelling results on production-quality challenging scenes in large quantities. We will publicly share our code and data to promote further research.



Figure 6.7: Typical failure modes. The left is the ground-truth annotation and the right is our result. Our system still makes mistakes for challenging non-Manhattan structures.

Chapter 7

Conclusion and Future Works

In this dissertation, we demonstrate our automated systems for piece-wise reconstruction (Chapter 2, 3), floorplan reconstruction (Chapter 4, 5, 6). By exploiting massive 3D data enabled by advance in data capturing techniques, we develop data-driven algorithms to learn shape priors, namely planarity and orthogonality, which are then used to represent 3D models in a structured way. Our reconstruction systems often consist of two key components: 1) a deep neural network, which learns to extract effective features in a data-driven manner, and 2) a domain-specific optimization, which is built upon deep features to find the optimal reconstruction while enforcing global constraints. With the application-ready representation and the production-level reconstruction quality, our structured modeling systems draw immediate industrial attentions.

While the proposed approaches have limitations, they open doors for new research in the domain of structured modeling. Potential future directions include:

Adding structured regularities to networks for end-to-end training. Our piece-wise planar reconstruction systems demonstrate the power of adding planarity as part of the

network to generate impressive reconstruction results, which satisfy the planarity constraint. However, it is still challenging to add many other structural regularities, such as connectivity (e.g., two wall corners are connect by a wall) and orthogonality (e.g., two walls are orthogonal to each other). While the current frameworks generate sound results for most cases, their robustness and generalizability could be further improved if we can directly optimize the final reconstruction results via end-to-end training. To achieve this, we can exploit new techniques which approximate global optimization using neural network layers [9]. Moreover, it would make a fundamental difference if we could convert some irregular constraints to some forms which can be represented as network output and train everything in an end-to-end fashion.

Exploring more structure regularities. Besides the most ubiquitous properties, planarity and orthogonality, there are other geometrical and semantical constraints to be utilized. Several representative scenarios revealing common regularities include: 1) the front face and a side face of a cabinet are always connected (connectivity), 2) a table supports objects atop (support relation), and 3) a nightstand usually sits besides a bed (semantical pair). Reconstructed 3D models which satisfy these constraints could better imitate real scenes and thus enable more applications.

Reconstructing holistic models of indoor scenes. Though important, both single image reconstruction and floorplan reconstruction recover only a partial model for a scene. The ultimate goal for 3D structured modeling is to represent the entire scene in a detailed yet structured way. To achieve this goal, we have to consider multiple views and address the challenge of learning correspondences.

Utilizing different input sources. Our floorplan reconstruction system, FloorNet, requires an RGBD stream, which is less common than RGB streams. It would be interesting to extend

FloorNet to handle RGB streams or even 1D laser scans which are captured by robot vacuum cleaners on a daily basis.

References

- [1] Autocad. <http://www.autodesk.com/products/autocad/overview>.
- [2] Gurobi. <http://www.gurobi.com>.
- [3] Homestyler. <http://www.homestyler.com>.
- [4] Lifull home's dataset. <https://www.nii.ac.jp/dsc/idr/lifull/homes.html>.
- [5] Matterport. <https://matterport.com/>.
- [6] Sketchup. <https://www.sketchup.com>.
- [7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [8] Sheraz Ahmed, Marcus Liwicki, Markus Weber, and Andreas Dengel. Improved automatic analysis of architectural floor plans. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 864–869. IEEE, 2011.

- [9] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017.
- [10] Adrian Bulat and Georgios Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In *European Conference on Computer Vision*, pages 717–732. Springer, 2016.
- [11] Ricardo Cabral and Yasutaka Furukawa. Piecewise planar and compact floorplan reconstruction from images. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 628–635. IEEE, 2014.
- [12] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7291–7299, 2017.
- [13] Ayan Chakrabarti, Jingyu Shao, and Greg Shakhnarovich. Depth from a single image by harmonizing overcomplete local network predictions. In *Advances in Neural Information Processing Systems*, pages 2658–2666, 2016.
- [14] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.
- [15] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.
- [16] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. *arXiv preprint arXiv:1702.04405*, 2017.

- [17] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.
- [18] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.
- [19] pytorch-mask-rcnn. <https://github.com/multimodallearning/pytorch-mask-rcnn>.
- [20] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. *arXiv preprint arXiv:1612.00603*, 2016.
- [21] David Ford Fouhey, Abhinav Gupta, and Martial Hebert. Unfolding an indoor origami world. In *European Conference on Computer Vision*, pages 687–702. Springer, 2014.
- [22] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.
- [23] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Manhattan-world stereo. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1422–1429. IEEE, 2009.
- [24] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Manhattan-world stereo. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 1422–1429, 2009.

- [25] David Gallup, Jan-Michael Frahm, and Marc Pollefeys. Piecewise planar and non-planar stereo for urban scene reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1418–1425. IEEE, 2010.
- [26] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
- [27] Lucile Gimenez, Jean-Laurent Hippolyte, Sylvain Robert, Frédéric Suard, and Khaldoun Zreik. Review: reconstruction of 3d building information models from 2d scanned plans. *Journal of Building Engineering*, 2:24–35, 2015.
- [28] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.
- [29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.
- [30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

- [33] Derek Hoiem, Alexei A Efros, and Martial Hebert. Automatic photo pop-up. *ACM transactions on graphics (TOG)*, 24(3):577–584, 2005.
- [34] Satoshi Ikehata, Hang Yang, and Yasutaka Furukawa. Structured indoor modeling. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1323–1331, 2015.
- [35] Google Inc. Project tango. <https://developers.google.com/tango/>.
- [36] Kevin Karsch, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, Hailin Jin, Rafael Fonte, Michael Sittig, and David Forsyth. Automatic scene inference for 3d object compositing. *ACM Trans. Graph.*, 33(3):32:1–32:15, June 2014.
- [37] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [38] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):78, 2017.
- [39] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.
- [40] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 239–248. IEEE, 2016.
- [41] Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation. *arXiv preprint arXiv:1703.06241*, 2017.
- [42] JC Lee, R Dugan, et al. Google project tango.

- [43] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017.
- [44] Chen Liu, Kihwan Kim, Jinwei Gu, Yasutaka Furukawa, and Jan Kautz. Planercnn: 3d plane detection and reconstruction from a single image. *arXiv preprint arXiv:1812.04072*, 2018.
- [45] Chen Liu, Pushmeet Kohli, and Yasutaka Furukawa. Layered scene decomposition via the occlusion-crf. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–173, 2016.
- [46] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2195–2203, 2017.
- [47] Chen Liu, Jiaye Wu, and Yasutaka Furukawa. Floornet: A unified framework for floorplan reconstruction from 3d scans. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 201–217, 2018.
- [48] Chen Liu, Jimei Yang, Duygu Ceylan, Ersin Yumer, and Yasutaka Furukawa. Planenet: Piece-wise planar reconstruction from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2579–2588, 2018.
- [49] Chenxi Liu, Alexander G Schwing, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Rent3d: Floor-plan priors for monocular layout estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3413–3421, 2015.
- [50] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian D Reid. Learning depth from single monocular images using deep convolutional neural fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(10):2024–2039, 2016.

- [51] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *arXiv preprint arXiv:1807.03247*, 2018.
- [52] Xiaobai Liu, Yibiao Zhao, and Song-Chun Zhu. Single-view 3d scene parsing by attributed grammar. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 684–691, 2014.
- [53] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [54] Aron Monszpart, Nicolas Mellado, Gabriel J Brostow, and Niloy J Mitra. Rapter: rebuilding man-made scenes with regular arrangements of planes. *ACM Trans. Graph.*, 34(4):103–1, 2015.
- [55] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. 2016.
- [56] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.
- [57] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, 2016.
- [58] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5105–5114, 2017.

- [59] Srikumar Ramalingam and Matthew Brand. Lifting 3d manhattan lines from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 497–504, 2013.
- [60] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. *arXiv preprint arXiv:1611.05009*, 2016.
- [61] 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*, pages 234–241. Springer, 2015.
- [62] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016.
- [63] Alexander G Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction for 3d indoor scene understanding. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2815–2822, 2012.
- [64] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.
- [65] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgb-d images. *Computer Vision–ECCV 2012*, pages 746–760, 2012.
- [66] Sudipta Sinha, Drew Steedly, and Rick Szeliski. Piecewise planar stereo for image-based rendering. 2009.

- [67] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *arXiv preprint arXiv:1611.08974*, 2016.
- [68] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [69] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.
- [70] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *arXiv preprint arXiv:1703.09438*, 2017.
- [71] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. *arXiv preprint arXiv:1612.00404*, 2016.
- [72] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3d scene inference via view synthesis. *arXiv preprint arXiv:1807.10264*, 2018.
- [73] Peng Wang, Xiaohui Shen, Bryan Russell, Scott Cohen, Brian Price, and Alan L Yuille. Surge: Surface regularized geometry estimation from a single image. In *Advances in Neural Information Processing Systems*, pages 172–180, 2016.
- [74] Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–547, 2015.

- [75] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [76] Jianxiong Xiao and Yasutaka Furukawa. Reconstructing the world’s museums. *International Journal of Computer Vision*, 110(3):243–258, 2014.
- [77] Dan Xu, Elisa Ricci, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. Multi-scale continuous crfs as sequential deep networks for monocular depth estimation. In *Proceedings of CVPR*, 2017.
- [78] Fengting Yang and Zihan Zhou. Recovering 3d planes from a single image via convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 85–100, 2018.
- [79] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. *arXiv preprint arXiv:1705.09914*, 2017.
- [80] Fisher Yu, Vladlen Koltun, and Thomas A. Funkhouser. Dilated residual networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 636–644, 2017.
- [81] Shanxin Yuan, Qi Ye, Bjorn Stenger, Siddhant Jain, and Tae-Kyun Kim. Bighand2. 2m benchmark: Hand pose dataset and state of the art analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4866–4874, 2017.
- [82] Lukas Zebedin, Joachim Bauer, Konrad Karner, and Horst Bischof. Fusion of feature-and area-based information for urban buildings modeling from aerial imagery. *Computer Vision–ECCV 2008*, pages 873–886, 2008.

- [83] Jian Zhang, Chen Kan, Alexander G Schwing, and Raquel Urtasun. Estimating the 3d layout of indoor scenes and its clutter from depth sensors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1273–1280, 2013.
- [84] Yinda Zhang, Fisher Yu, Shuran Song, Pingmei Xu, Ari Seff, and Jianxiong Xiao. Large-scale scene understanding challenge: Room layout estimation. *accessed on Sep*, 15, 2015.
- [85] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *arXiv preprint arXiv:1612.01105*, 2016.
- [86] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

Appendix A

Baselines for Piece-wise Planar Reconstruction

We explain our modifications to the three baseline methods to enable their executions on our problem setting for piece-wise planar reconstruction (i.e., a single image input).

NYU Toolbox is the first baseline [65]. We follow Wang *et al.* [73] and add the same extension to the toolbox as follows: If a plane occupies more than 90% pixels of a semantic mask, the entire semantic mask will be assigned to that plane.

Manhattan World Stereo (MWS) requires a 3D point cloud as the input [23]. Instead of using multi-view stereo, we un-project the input depthmap to obtain a point cloud. The data term in their MRF formulation requires the visibility information associated with multiple views. The information does not exist for our case, and we instead use the data term in NYU Toolbox [65]. This term utilizes pixel-wise surface normals, which are calculated from the depthmap.

Piece-wise Planar Stereo (PPS) extracts vanishing directions (instead of the three Manhattan axes) and generates plane hypotheses from every pair of the extracted directions. PPS also reconstructs 3D lines by multi-view stereo techniques and fits additional planes, whose step is impossible and excluded in our experiments. However, we do not expect much performance degradation, as our input is a dense depthmap and most of the line information are covered through vanishing line extraction. The input point-cloud is generated by un-projecting the input depthmap as in the case of MWS. The data term in the MRF formulation is again replaced by the one in NYU Toolbox.

Appendix B

Algorithm Details for Floorplan Transformation

B.1 Pre-processing Details

This section provides algorithmic details on how we obtain human annotations then converts them into our own representation for training.

Our primitive representation is expressive and enables our learning-based inversion. However, it is trivial for a human subject to explicitly denote the location and type of all the junctions, and associate each wall with room labels. To ease the human annotation process, we ask human subjects to draw a line representing either a wall or an opening, draw a rectangle, and pick an object type for each object, or attach a label at a specific location. We need to convert such annotations to our junction layer representation for training the network, and to final representation for evaluation. We convert the annotation information to junction layer representation as follows.

- Openings and objects primitives are directly annotated, and thus we simply read junctions from the annotation (either endpoints of an opening or corners of an icon).
- An annotated wall often consists of multiple wall primitives. Thus we determine a wall junction by looking at its local connections of walls (i.e., checking if a wall exists on its up, bottom, left, or right side).

The junction layer representation is sufficient for network training. For evaluation, we first convert the junction layer representation to get information for walls, openings and icons in the final representation. The only missing piece in the final representation is rooms. We split closed polygons (formed by walls) in a similar manner explained in Appendix B.2. Note that now we have room labels specified at certain locations instead of associated with each wall primitive. In this setting, we find a pair of facing walls which has one and only one room label attached in between, and assign this room label to the region between this pair of walls. The rest is the same with before.

B.2 Post-processing Details

This section provides algorithmic details on how we split a closed polygon whose walls are associated with more than one room-type. This information is not essential but used to change the color/texture of walls and the floor geometry in our 2D and 3D visualizations.

We allow certain rooms not to be a closed polygon (e.g., a kitchen and a living room appear in the same closed polygon) (see Fig. B.1 for an example). In such cases, with room labels associated with each wall (i.e., each edge of the polygon), it is unclear how to divide the closed polygon based on functionality. We address this issue based on a practical heuristic. For each pair of facing walls which have the same room label, we find the rectangle between these two walls, and assign the room label to the intersection region between the rectangle



Figure B.1: A floorplan example where kitchen and living room appear in the same closed polygon. A pair of facing walls (red line and purple line) with the same room label (e.g., living room) determines that pixels colored with green belong to living room.

and the closed polygon as shown in Fig. B.1. If a pixel is assigned with multiple room labels, then we pick the label with the highest priority, in the order of living room, kitchen, washing room, and corridor (other room types enforce loop constraints). For a pixel with no label assigned, we assign it to the label which appear in this polygon and has the highest priority. After assigning all the pixels inside the polygon with one room label, we find the connected components based on the assignment to be the final room regions.